# A Set of Benchmarks for Measuring IRAF System Performance

*Doug Tody*

National Optical Astronomy Observatories*
March 1986
(Revised July 1987)

*ABSTRACT*

This paper presents a set of benchmarks for measuring the performance of IRAF as installed on a particular host system. The benchmarks serve two purposes: [1] they provide an objective means of comparing the performance of different IRAF host systems, and [2] the benchmarks may be repeated as part of the IRAF installation procedure to verify that the expected performance is actually being achieved. While the benchmarks chosen are sometimes complex, i.e., at the level of actual applications programs and therefore difficult to interpret in detail, some effort has been made to measure all the important performance characteristics of the host system. These include the raw cpu speed, the floating point processing speed, the i/o bandwidth to disk, and a number of characteristics of the host operating system as well, e.g., the efficiency of common system calls, the interactive response of the system, and the response of the system to loading. The benchmarks are discussed in detail along with instructions for benchmarking a new system, followed by tabulated results of the benchmarks for a number of IRAF host machines.

*Operated by the Association of Universities for Research in Astronomy, Inc. under cooperative agreement with the National Science Foundation.

**Contents**


1. **Introduction** ..... 1

2. **What is Measured** ..... 1

3. **The Benchmarks** ..... 2
   - 3.1. Host Level Benchmarks ..... 3
     - 3.1.1. CL Startup/Shutdown [CLSS] ..... 3
     - 3.1.2. Mkpkg (verify) [MKPKGV] ..... 3
     - 3.1.3. Mkpkg (compile) [MKPKGC] ..... 3
   - 3.2. IRAF Applications Benchmarks ..... 4
     - 3.2.1. Mkhelpdb [MKHDB] ..... 4
     - 3.2.2. Sequential Image Operators [IMADD, IMSTAT, etc.] ..... 4
     - 3.2.3. Image Load [IMLOAD,IMLOADF] ..... 5
     - 3.2.4. Image Transpose [IMTRAN] ..... 5
   - 3.3. Specialized Benchmarks ..... 5
     - 3.3.1. Subprocess Connect/Disconnect [SUBPR] ..... 6
     - 3.3.2. IPC Overhead [IPCO] ..... 6
     - 3.3.3. IPC Bandwidth [IPCB] ..... 6
     - 3.3.4. Foreign Task Execution [FORTSK] ..... 6
     - 3.3.5. Binary File I/O [WBIN,RBIN,RRBIN] ..... 7
     - 3.3.6. Text File I/O [WTEXT,RTEXT] ..... 7
     - 3.3.7. Network I/O [NWBIN,NRBIN,etc.] ..... 7
     - 3.3.8. Task, IMIO, GIO Overhead [PLOTS] ..... 8
     - 3.3.9. System Loading [2USER,4USER] ..... 8

4. **Interpreting the Benchmark Results** ..... 9

**Appendix A: IRAF Version 2.5 Benchmarks**

**Appendix B: IRAF Version 2.2 Benchmarks**

# A Set of Benchmarks for Measuring IRAF System Performance

*Doug Tody*

National Optical Astronomy Observatories*
March 1986
(Revised July 1987)

## 1. Introduction

This set of benchmarks has been prepared with a number of purposes in mind. Firstly, the benchmarks may be run after installing IRAF on a new system to verify that the performance expected for that machine is actually being achieved. In general, this cannot be taken for granted since the performance actually achieved on a particular system may depend upon how the system is configured and tuned. Secondly, the benchmarks may be run to compare the performance of different IRAF hosts, or to track the system performance over a period of time as improvements are made, both to IRAF and to the host system. Lastly, the benchmarks provide a metric which can be used to tune the host system.

All too often, the only benchmarks run on a system are those which test the execution time of optimized code generated by the host Fortran compiler. This is primarily a hardware benchmark and secondarily a test of the Fortran optimizer. An example of this type of test is the famous Linpack benchmark.

The numerical execution speed test is an important benchmark but it tests only one of the many factors contributing to the overall performance of the system as perceived by the user. In interactive use other factors are often more important, e.g., the time required to spawn or communicate with a subprocess, the time required to access a file, the response of the system as the number of users (or processes) increases, and so on. While the quality of optimized code is significant for cpu intensive batch processing, other factors are often more important for sophisticated interactive applications.

The benchmarks described here are designed to test, as fully as possible, the major factors contributing to the overall performance of the IRAF system on a particular host. A major factor in the timings of each benchmark is of course the IRAF system itself, but comparisons of different hosts are nonetheless possible since the code is virtually identical on all hosts (the applications and VOS are in fact identical on all hosts). The IRAF kernel (OS interface) is coded differently for each host operating system, but the functions performed by the kernel are identical on each host, and since the kernel is a very "thin" layer the kernel code itself is almost always a negligible factor in the final timings.

The IRAF version number, host operating system and associated version number, and the host computer hardware configuration are all important in interpreting the results of the benchmarks, and should always be recorded.

## 2. What is Measured

Each benchmark measures two quantities, the total cpu time required to execute the benchmark, and the total (wall) clock time required to execute the benchmark. If the clock time measurement is to be of any value the benchmarks must be run on a single user system. Given this "best time" measurement and some idea of how the system responds to loading, it is not difficult to estimate the performance to be expected on a loaded system.

The total cpu time required to execute a benchmark consists of the "user" time plus the "system" time. The "user" time is the cpu time spent executing the instructions comprising the

*Operated by the Association of Universities for Research in Astronomy, Inc. under cooperative agreement with the National Science Foundation.

user (IRAF) program, i.e., any instructions in procedures linked directly into the process being executed. The "system" time is the cpu time spent in kernel mode executing the system services called by the user program. On some systems there is no distinction between the two types of timings, with the system time either being included in the measured cpu time, or omitted from the timings. If the benchmark involves several concurrent processes no cpu time measurement of the subprocesses may be possible on some systems.

When possible we give both measurements, while in some cases only the user time is given, or only the sum of the user and system times. The cpu time measurements are therefore only directly comparable between different operating systems for the simpler benchmarks, in particular those which make few system calls. The cpu measurements given *are* accurate for the same operating system (e.g., some version of UNIX) running on different hosts, and may be used to compare such systems. Reliable comparisions between different operating systems are also possible, but only if one thoroughly understands what is going on.

The clock time measurement includes both the user and system times, plus the time spent waiting for i/o. Any minor system daemon processes executing while the benchmarks are being run may bias the clock time measurement slightly, but since these are a constant part of the host environment it is fair to include them in the timings. Major system daemons which run infrequently (e.g., the print symbiont in VMS) should invalidate the benchmark.

Assuming an otherwise idle system, a comparison of the cpu and clock times tells whether the benchmark was cpu bound or i/o bound. Those benchmarks involving compiled IRAF tasks do not include the process startup and pagein times (these are measured by a different benchmark), hence the task should be run once before running the benchmark to connect the subprocess and page in the memory used by the task. A good procedure to follow is to run each benchmark once to start the process, and then repeat the benchmark three times, averaging the results. If inconsistent results are obtained further iterations and/or monitoring of the host system are called for until a consistent result is achieved.

Many benchmarks depend upon disk performance as well as compute cycles. For such a benchmark to be a meaningful measure of the i/o bandwidth of the system it is essential that no other users (or batch jobs) be competing for disk seeks on the disk used for the test file. There are subtle things to watch out for in this regard, for example, if the machine is in a VMS cluster or on a local area network, processes on other nodes may be accessing the local disk, yet will not show up on a user login or process list on the local node. It is always desirable to repeat each test several times or on several different disk devices, to ensure that no outside requests were being serviced while the benchmark was being run. If the system has disk monitoring utilities use these to find an idle disk before running any benchmarks which do heavy i/o.

Beware of disks which are nearly full; the maximum achievable i/o bandwidth may fall off rapidly as a disk fills up, due to disk fragmentation (the file must be stored in little pieces scattered all over the physical disk). Similarly, many systems (VMS, AOS/VS, V7 and Sys V UNIX, but not Berkeley UNIX) suffer from disk fragmentation problems that gradually worsen as a files system ages, requiring that the disk periodically be backed off onto tape and then restored to render the files and free spaces as contiguous as possible. In some cases, disk fragmentation can cause the maximum achievable i/o bandwidth to degrade by an order of magnitude. For example, on a VMS system one can use `COPY/CONTIGUOUS` to render files contiguous (e.g., this can be done on all the executables in `[IRAF.BIN]` after installing the system, to speed process pagein times). If the copy fails for a large file even though there is substantial free space left on the disk, the disk is badly fragmented.

## 3. The Benchmarks

Instructions are given for running each benchmark, and the operations performed by each benchmark are briefly described. The system characteristics measured by the benchmark are briefly discussed. A short mnemonic name is associated with each benchmark to identify it in the tables given in the appendices, tabulating the results for actual host machines.

### 3.1. Host Level Benchmarks

The benchmarks discussed in this section are run at the host system level. The examples are given for the UNIX cshell, under the assumption that a host dependent example is better than none at all. These commands must be translated by the user to run the benchmarks on a different system (hint: use `SHOW STATUS` or a stop watch to measure wall clock times on a VMS host).

#### 3.1.1. CL Startup/Shutdown [CLSS]

Go to the CL login directory (any directory containing a `LOGIN.CL` file), mark the time (the method by which this is done is system dependent), and startup the CL. Enter the "logout" command while the CL is starting up so that the CL will not be idle (with the clock running) while the command is being entered. Mark the final cpu and clock time and compute the difference.

```
% time cl
logout
```

This is a complex benchmark but one which is of obvious importance to the IRAF user. The benchmark is probably dominated by the cpu time required to start up the CL, i.e., start up the CL process, initialize the i/o system, initialize the environment, interpret the CL startup file, interpret the user LOGIN.CL file, connect and disconnect the x_system.e subprocess, and so on. Most of the remaining time is the overhead of the host operating system for the process spawns, page faults, file accesses, and so on. *Do not use a customized* `LOGIN.CL` *file when running this benchmark*, or the timings will almost certainly be affected.

#### 3.1.2. Mkpkg (verify) [MKPKGV]

Go to the PKG directory and enter the (host system equivalent of the) following command. The method by which the total cpu and clock times are computed is system dependent.

```
% cd $iraf/pkg
% time mkpkg -n
```

This benchmark does a "no execute" make-package of the entire PKG suite of applications and systems packages. This tests primarily the speed with which the host system can read directories, resolve pathnames, and return directory information for files. Since the PKG directory tree is continually growing, this benchmark is only useful for comparing the same version of IRAF run on different hosts, or the same version of IRAF on the same host at different times.

#### 3.1.3. Mkpkg (compile) [MKPKGC]

Go to the directory "iraf$pkg/bench/xctest" and enter the (host system equivalents of the) following commands. The method by which the total cpu and clock times are computed is system dependent. Only the **mkpkg** command should be timed.

```
% cd $iraf/pkg/bench/xctest
% mkpkg clean         # delete old library, etc., if present
% time mkpkg
% mkpkg clean         # delete newly created binaries
```

This tests the time required to compile and link a small IRAF package. The timings reflect the time required to preprocess, compile, optimize, and assemble each module and insert it into the package library, then link the package executable. The host operating system overhead for the process spawns, page faults, etc. is also a major factor. If the host system provides a shared library facility this will significantly affect the link time, hence the benchmark should be run linking both with and without shared libraries to make a fair comparison to other systems.

Linking against a large library is fastest if the library is topologically sorted and stored contiguously on disk.

### 3.2. IRAF Applications Benchmarks

The benchmarks discussed in this section are run from within the IRAF environment, using only standard IRAF applications tasks. The cpu and clock times of any (compiled) IRAF task may be measured by prefixing the task name with a $ when the command is entered into the CL, as shown in the examples. The significance of the cpu time measurement is not precisely defined for all systems. On a UNIX host, it is the "user" cpu time used by the task. On a VMS host, there does not appear to be any distinction between the user and system times (probably because the system services execute in the context of the calling process), hence the cpu time given probably includes both, but probably excludes the time for any services executing in ancillary processes, e.g., for RMS.

#### 3.2.1. Mkhelpdb [MKHDB]

The **mkhelpdb** task is in the **softools** package. The function of the task is to scan the tree of ".hd" help-directory files and compile the binary help database.

```
cl> softools
cl> $mkhelpdb
```

This benchmark tests the speed of the host files system and the efficiency of the host system services and text file i/o, as well as the global optimization of the Fortran compiler and the MIPS rating of the host machine. Since the size of the help database varies with each version of IRAF, this benchmark is only useful for comparing the same version of IRAF run on different hosts, or the same version run on a single host at different times. Note than any additions to the base IRAF system (e.g., SDAS) will increase the size of the help database and affect the timings.

#### 3.2.2. Sequential Image Operators [IMADDS,IMADDR,IMSTATR,IMSHIFTR]

These benchmarks measure the time required by typical image operations. All tests should be performed on 512 square test images created with the **imdebug** package. The **images** and **imdebug** packages should be loaded. Enter the following commands to create the test images.

```
cl> mktest pix.s s 2 "512 512"
cl> mktest pix.r r 2 "512 512"
```

The following benchmarks should be run on these test images. Delete the output images after each benchmark is run. If you enter the commands shown once, the command can be repeated by typing ^ followed by return. Each benchmark should be run several times, discarding the first timing and averaging the remaining timings for the final result.

```
[IMADDS]    cl> $imarith pix.s + 5 pix2.s; imdel pix2.s
[IMADDR]    cl> $imarith pix.r + 5 pix2.r; imdel pix2.r
[IMSTATR]   cl> $imstat pix.r
[IMSHIFTR]  cl> $imshift pix.r pix2.r .33 .44 interp=spline3
```

The IMADD benchmarks test the efficiency of the image i/o system, including binary file i/o, and provide an indication of how long a simple disk to disk image operation takes on the system in question. This benchmark should be i/o bound on most systems. The IMSTATR and IMSHIFTR benchmarks are normally cpu bound, and test primarily the speed of the host cpu and floating point unit, and the quality of the code generated by the host Fortran compiler. Note that the IMSHIFTR benchmark employs a true two dimensional bicubic spline, hence the timings are a factor of 4 greater than one would expect if a one dimensional interpolator were used

to shift the two dimensional image.

### 3.2.3. Image Load [IMLOAD,IMLOADF]

To run the image load benchmarks, first load the **tv** package and display something to get the x_display.e process into the process cache. Run the following two benchmarks, displaying the test image PIX.S (this image contains a test pattern of no interest).

```
[IMLOAD]    cl> $display pix.s 1
[IMLOADF]   cl> $display pix.s 1 zt=none
```

The IMLOAD benchmark measures how long it takes for a normal image load on the host system, including the automatic determination of the greyscale mapping, and the time required to map and clip the image pixels into the 8 bits (or whatever) displayable by the image display. This benchmark measures primarily the cpu speed and i/o bandwidth of the host system. The IMLOADF benchmark eliminates the cpu intensive greyscale transformation, yielding the minimum image display time for the host system.

### 3.2.4. Image Transpose [IMTRAN]

To run this benchmark, transpose the image PIX.S, placing the output in a new image.

```
cl> $imtran pix.s pix2.s
```

This benchmark tests the ability of a process to grab a large amount of physical memory (large working set), and the speed with which the host system can service random rather than sequential file access requests. The user working set should be large enough to avoid excessive page faulting.

## 3.3. Specialized Benchmarks

The next few benchmarks are implemented as tasks in the **bench** package, located in the directory "pkg$bench". This package is not installed as a predefined package as the standard IRAF packages are. Since this package is used infrequently the binaries may have been deleted; if the file x_bench.e is not present in the *bench* directory, rebuild it as follows:

```
cl> cd pkg$bench
cl> mkpkg
```

To load the package, enter the following commands. It is not necessary to *cd* to the bench directory to load or run the package.

```
cl> task $bench = "pkg$bench/bench.cl"
cl> bench
```

This defines the following benchmark tasks. There are no manual pages for these tasks; the only documentation is what you are reading.

| | |
|---|---|
| FORTASK | - foreign task execution |
| GETPAR | - get parameter; tests IPC overhead |
| PLOTS | - make line plots from an image |
| RBIN | - read binary file; tests FIO bandwidth |
| RRBIN | - raw (unbuffered) binary file read |
| RTEXT | - read text file; tests text file i/o speed |
| SUBPROC | - subprocess connect/disconnect |
| WBIN | - write binary file; tests FIO bandwidth |
| WIPC | - write to IPC; tests IPC bandwidth |
| WTEXT | - write text file; tests text file i/o speed |

### 3.3.1. Subprocess Connect/Disconnect [SUBPR]

To run the SUBPR benchmark, enter the following command. This will connect and disconnect the x_images.e subprocess 10 times. Difference the starting and final times printed as the task output to get the results of the benchmark. The cpu time measurement may be meaningless (very small) on some systems.

```
cl> subproc 10
```

This benchmark measures the time required to connect and disconnect an IRAF subprocess. This includes not only the host time required to spawn and later shutdown a process, but also the time required by the IRAF VOS to set up the IPC channels, initialize the VOS i/o system, initialize the environment in the subprocess, and so on. A portion of the subprocess must be paged into memory to execute all this initialization code. The host system overhead to spawn a subprocess and fault in a portion of its address space is a major factor in this benchmark.

### 3.3.2. IPC Overhead [IPCO]

The **getpar** task is a compiled task in x_bench.e. The task will fetch the value of a CL parameter 100 times.

```
cl> $getpar 100
```

Since each parameter access consists of a request sent to the CL by the subprocess, followed by a response from the CL process, with a negligible amount of data being transferred in each call, this tests the IPC overhead.

### 3.3.3. IPC Bandwidth [IPCB]

To run this benchmark enter the following command. The **wipc** task is a compiled task in x_bench.e.

```
cl> $wipc 1E6 > dev$null
```

This writes approximately 1 Mb of binary data via IPC to the CL, which discards the data (writes it to the null file via FIO). Since no actual disk file i/o is involved, this tests the efficiency of the IRAF pseudofile i/o system and of the host system IPC facility.

### 3.3.4. Foreign Task Execution [FORTSK]

To run this benchmark enter the following command. The **fortask** task is a CL script task in the **bench** package.

```
cl> fortask 10
```

This benchmark executes the standard IRAF foreign task **rmbin** (one of the bootstrap utilities) 10 times. The task is called with no arguments and does nothing other than execute, print out its "usage" message, and shut down. This tests the time required to execute a host system task

from within the IRAF environment. Only the clock time measurement is meaningful.

### 3.3.5. Binary File I/O [WBIN,RBIN,RRBIN]

To run these benchmarks, make sure the **bench** package is loaded, and enter the following commands. The **wbin**, **rbin** and **rrbin** tasks are compiled tasks in x_bench.e. A binary file named BINFILE is created in the current directory by WBIN, and should be deleted after the benchmark has been run. Each benchmark should be run at least twice before recording the time and moving on to the next benchmark. Successive calls to WBIN will automatically delete the file and write a new one.

*NOTE:* it is wise to create the test file on a files system which has a lot of free space available, to avoid disk fragmentation problems. Also, if the host system has two or more different types of disk drives (or disk controllers or bus types), you may wish to run the benchmark separately for each drive.

```
cl> $wbin binfile 5E6
cl> $rbin binfile
cl> $rrbin binfile
cl> delete binfile           # (not part of the benchmark)
```

These benchmarks measure the time required to write and then read a binary disk file approximately 5 Mb in size. This benchmark measures the binary file i/o bandwidth of the FIO interface (for sequential i/o). In WBIN and RBIN the common buffered READ and WRITE requests are used, hence some memory to memory copying is included in the overhead measured by the benchmark. A large FIO buffer is used to minimize disk seeks and synchronization delays; somewhat faster timings might be possible by increasing the size of the buffer (this is not a user controllable option, and is not possible on all host systems). The RRBIN benchmark uses ZARDBF to read the file in chunks of 32768 bytes, giving an estimate of the maximum i/o bandwidth for the system.

### 3.3.6. Text File I/O [WTEXT,RTEXT]

To run these benchmarks, load the **bench** package, and then enter the following commands. The **wtext** and **rtext** tasks are compiled tasks in x_bench.e. A text file named TEXTFILE is created in the current directory by WTEXT, and should be deleted after the benchmarks have been run. Successive calls to WTEXT will automatically delete the file and write a new one.

```
cl> $wtext textfile 1E6
cl> $rtext textfile
cl> delete textfile      # (not part of the benchmark)
```

These benchmarks measure the time required to write and then read a text disk file approximately one megabyte in size (15,625 64 character lines). This benchmark measures the efficiency with which the system can sequentially read and write text files. Since text file i/o requires the system to pack and unpack records, text i/o tends to be cpu bound.

### 3.3.7. Network I/O [NWBIN,NRBIN,NWNULL,NWTEXT,NRTEXT]

These benchmarks are equivalent to the binary and text file benchmarks just discussed, except that the binary and text files are acccessed on a remote node via the IRAF network interface. The calling sequences are identical except that an IRAF network filename is given instead of referencing a file in the current directory. For example, the following commands would be entered to run the network binary file benchmarks on node LYRA (the node name and filename are site dependent).

```
cl> $wbin lyra!/tmp3/binfile 5E6      [NWBIN]
cl> $rbin lyra!/tmp3/binfile          [NRBIN]
cl> $wbin lyra!/dev/null 5E6          [NWNULL]
cl> delete lyra!/tmp3/binfile
```

The text file benchmarks are equivalent with the obvious changes, i.e., substitute "text" for "bin", "textfile" for "binfile", and omit the null textfile benchmark. The type of network interface used (TCP/IP, DECNET, etc.), and the characteristics of the remote node should be recorded.

These benchmarks test the bandwidth of the IRAF network interfaces for binary and text files, as well as the limiting speed of the network itself (NWNULL). The binary file benchmarks should be i/o bound. NWBIN should outperform NRBIN since a network write is a pipelined operation, whereas a network read is (currently) a synchronous operation. Text file access may be either cpu or i/o bound depending upon the relative speeds of the network and host cpus. The IRAF network interface buffers textfile i/o to minimize the number of network packets and maximize the i/o bandwidth.

### 3.3.8. Task, IMIO, GIO Overhead [PLOTS]

The **plots** task is a CL script task which calls the **prow** task repeatedly to plot the same line of an image. The graphics output is discarded (directed to the null file) rather than plotted since otherwise the results of the benchmark would be dominated by the plotting speed of the graphics terminal.

```
cl> plots pix.s 10
```

This is a complex benchmark. The benchmark measures the overhead of task (not process) execution and the overhead of the IMIO and GIO subsystems, as well as the speed with which IPC can be used to pass parameters to a task and return the GIO graphics metacode to the CL.

The **prow** task is all overhead and is not normally used to interactively plot image lines (**implot** is what is normally used), but it is a good task to use for a benchmark since it exercises the subsystems most commonly used in scientific tasks. The **prow** task has a couple dozen parameters (mostly hidden), must open the image to read the image line to be plotted on every call, and must open the GIO graphics device on every call as well.

### 3.3.9. System Loading [2USER,4USER]

This benchmark attempts to measure the response of the system as the load increases. This is done by running large **plots** jobs on several terminals and then repeating the 10 plots **plots** benchmark. For example, to run the 2USER benchmark, login on a second terminal and enter the following command, and then repeat the PLOTS benchmark discussed in the last section. Be sure to use a different login or login directory for each "user", to avoid concurrency problems, e.g., when reading the input image or updating parameter files.

```
cl> plots pix.s 9999
```

Theoretically, the timings should be approximately .5 (2USER) and .25 (4USER) as fast as when the PLOTS benchmark was run on a single user system, assuming that cpu time is the limiting resource and that a single job is cpu bound. In a case where there is more than one limiting resource, e.g., disk seeks as well as cpu cycles, performance will fall off more rapidly. If, on the other hand, a single user process does not keep the system busy, e.g., because synchronous i/o is used, performance will fall off less rapidly. If the system unexpectedly runs out of some critical system resource, e.g., physical memory or some internal OS buffer space, performance may be much worse than expected.

If the multiuser performance is poorer than expected it may be possible to improve the system performance significantly once the reason for the poor performance is understood. If disk seeks are the problem it may be possible to distribute the load more evenly over the

available disks. If the performance decays linearly as more users are added and then gets really bad, it is probably because some critical system resource has run out. Use the system monitoring tools provided with the host operating system to try to identify the critical resource. It may be possible to modify the system tuning parameters to fix the problem, once the critical resource has been identified.

## 4. Interpreting the Benchmark Results

Many factors determine the timings obtained when the benchmarks are run on a system. These factors include all of the following:

- The hardware configuration, e.g., cpu used, clock speed, availability of floating point hardware, type of floating point hardware, amount of memory, number and type of disks, degree of fragmentation of the disks, bus bandwidth, disk controller bandwidth, memory controller bandwidth for memory mapped DMA transfers, and so on.
- The host operating system, including the version number, tuning parameters, user quotas, working set size, files system parameters, Fortran compiler characteristics, level of optimization used to compile IRAF, and so on.
- The version of IRAF being run. On a VMS system, are the images "installed" to permit shared memory and reduce physical memory usage? Were the programs compiled with the code optimizer, and if so, what compiler options were used? Are shared libraries used if available on the host system?
- Other activity in the system when the benchmarks were run. If there were no other users on the machine at the time, how about batch jobs? If the machine is on a cluster or network, were other nodes accessing the same disks? How many other processes were running on the local node? Ideally, the benchmarks should be run on an otherwise idle system, else the results may be meaningless or next to impossible to interpret. Given some idea of how the host system responds to loading, it is possible to estimate how a timing will scale as the system is loaded, but the reverse operation is much more difficult.

Because so many factors contribute to the results of a benchmark, it can be difficult to draw firm conclusions from any benchmark, no matter how simple. The hardware and software in modern computer systems is so complicated that it is difficult even for an expert with a detailed knowledge and understanding of the full system to explain in detail where the time is going, even when running the simplest benchmark. On some recent message based multiprocessor systems it is probably impossible to fully comprehend what is going on at any given time, even if one fully understands how the system works, because of the dynamic nature of such systems.

Despite these difficulties, the benchmarks do provide a coarse measure of the relative performance of different host systems, as well as some indication of the efficiency of the IRAF VOS. The benchmarks are designed to measure the performance of the *host system* (both hardware and software) in a number of important areas, all of which play a role in determining the suitability of a system for scientific data processing. The benchmarks are *not* designed to measure the efficiency of the IRAF software itself (except parts of the VOS), e.g., there is no measure of the time taken by the CL to compile and execute a script, no measure of the speed of the median algorithm or of an image transpose, and so on. These timings are also important, of course, but should be measured separately. Also, measurements of the efficiency of individual applications programs are much less critical than the performance criteria dealt with here, since it is relatively easy to optimize an inefficient or poorly designed applications program, even a complex one like the CL, but there is generally little one can do about the host system.

The timings for the benchmarks for a number of host systems are given in the appendices which follow. Sometimes there will be more than one set of benchmarks for a given host system, e.g., because the system provided two or more disks or floating point options with different levels of performance. The notes at the end of each set of benchmarks are intended to document any special features or problems of the host system which may have affected the results. In general we did not bother to record things like system tuning parameters, working set, page faults, etc., unless these were considered an important factor in the benchmarks. In particular, few IRAF programs page fault other than during process startup, hence this is rarely a signficant factor when running these benchmarks (except possibly in IMTRAN).

Detailed results for each configuration of each host system are presented on separate pages in the Appendices. A summary table showing the results of selected benchmarks for all host systems at once is also provided. The system characteristic or characteristics principally measured by each benchmark is noted in the table below. This is only approximate, e.g., the MIPS rating is a significant factor in all but the most i/o bound benchmarks.

| *benchmark* | *responsiveness* | *mips* | *flops* | *i/o* |
|---|---|---|---|---|
| CLSS | • | | | |
| MKPKGV | • | | | |
| MKHDB | • | • | | |
| PLOTS | • | • | | |
| IMADDS | | • | | • |
| IMADDR | | | • | • |
| IMSTATR | | | • | |
| IMSHIFTR | | | • | |
| IMTRAN | | | | • |
| WBIN | | | | • |
| RBIN | | | | • |

By *responsiveness* we refer to the interactive response of the system as perceived by the user. A system with a good interactive response will do all the little things very fast, e.g., directory listings, image header listings, plotting from an image, loading new packages, starting up a new process, and so on. Machines which score high in this area will seem fast to the user, whereas machines which score poorly will *seem* slow, sometimes frustratingly slow, even though they may score high in the areas of floating point performance, or i/o bandwidth. The interactive response of a system obviously depends upon the MIPS rating of the system (see below), but an often more significant factor is the design and computational complexity of the host operating system itself, in particular the time taken by the host operating system to execute system calls. Any system which spends a large fraction of its time in kernel mode will probably have poor interactive response. The response of the system to loading is also very important, i.e., if the system has trouble with load balancing as the number of users (or processes) increases, response will become increasingly erratic until the interactive response is hopelessly poor.

The MIPS column refers to the raw speed of the system when executing arbitrary code containing a mixture of various types of instructions, but little floating point, i/o, or system calls. A machine with a high MIPS rating will have a fast cpu, e.g., a fast clock rate, fast memory access time, large cache memory, and so on, as well as a good optimizing Fortran compiler. Assuming good compilers, the MIPS rating is primarily a measure of the hardware speed of the host machine, but all of the MIPS related benchmarks presented here also make a significant number of system calls (MKHDB, for example, does a lot of files accesses and text file i/o), hence it is not that simple. Perhaps a completely cpu bound pure-MIPS benchmark should be added to our suite of benchmarks (the MIPS rating of every machine is generally well known, however).

The FLOPS column identifies those benchmarks which do a significant amount of floating point computation. The IMSHIFTR and IMSTATR benchmarks in particular are heavily into floating point. These benchmarks measure the single precision floating point speed of the host system hardware, as well as the effectiveness of do-loop optimization by the host Fortran compiler. The degree of optimization provided by the Fortran compiler can affect the timing of these benchmarks by up to a factor of two. Note that the sample is very small, and if a compiler fails to optimize the inner loop of one of these benchmark programs, the situation may be reversed when running some other benchmark. Any reasonable Fortran compiler should be able to optimize the inner loop of the IMADDR benchmark, so the CPU timing for this benchmark is a good measure of the hardware floating point speed, if one allows for do-loop overhead, memory i/o, and the system calls necessary to access the image on disk.

The I/O column identifies those benchmarks which are i/o bound and which therefore provide some indication of the i/o bandwidth of the host system. The i/o bandwidth actually achieved in these benchmarks depends upon many factors, the most important of which are the host operating system software (files system data structures and i/o software, disk drivers, etc.) and the host system hardware, i.e., disk type, disk controller type, bus bandwidth, and DMA memory controller bandwidth. Note that asynchronous i/o is not currently used in these benchmarks, hence higher transfer rates are probably possible in special cases (on a busy system all i/o is asynchronous at the host system level anyway). Large transfers are used to minimize disk seeks and synchronization delays, hence the benchmarks should provide a good measure of the realistically achievable host i/o bandwidth.