AIPS Memo 85

DDT Revised and AIPSMark$^{(93)}$ Measurements

Eric W. Greisen

February 9, 1994

Abstract

The AIPS certification and benchmarking package known as DDT has been revised to replace the obsolete self-calibration program ASCAL with CALIB. The window used for Clean in the large case was enlarged to encompass the full source and a number of other practical and cosmetic changes were made. In this Memo I present the results of performance tests made with the 15JAN94 release of AIPS using both the old and new versions of DDT on a variety of computer architectures. A new form of the "AIPS Mark" is defined and limitations to the accuracy of the images computed are discussed.

1 Introduction

The idea of an AIPS certification and benchmarking package was introduced by Wells *et al.* in 1985 ([1]). Since then, it has been the basis of an additional ten AIPS Memos (see References). The certification test, called "DDT" for "dirty-dozen test," is run regularly on the development version of AIPS to insure its continued correctness. The package has also been used in at least two formal computer procurements and in a number of other tests with commercial or financial implications. Given this importance, I felt that the test should not be heavily biased by the use of the self-calibration task ASCAL, which is no longer actively supported. In addition, I thought that the certification results were compromised by comparing images computed with recent versions of AIPS to master images computed with quite old versions.

The DDT package is based on the assumption that it is best to test things using real data with programs really used by our users. In this way, we determine that our most basic programs work over both time and architecture and measure the performance that our users will actually realize. The package consists of a collection of AIPS procedures designed to run the basic tasks UVSRT to sort data, UVMAP to Fourier transform *uv* data into images, APCLN to deconvolve these images, ASCAL (now CALIB) to apply the self-calibration process to improve the data, MX to re-image and re-deconvolve the data, and VTESS to deconvolve the images using an algorithm similar to maximum entropy. A number of other tasks are used in support of these. They are SUBIM to copy images, CCMRG to compress Clean component (source model) files, COMB to difference master and test images, UVDIF to compare master and test *uv* files, FITLD to read data from tape or disk, FITTP to write data to tape or disk, PRTAC to report the contents of the accounting file, and, of course AIPS itself to compile and run the procedures, compute some of the comparison results, and print the messages.

There are four different input data sets called SMALL, MEDIUM, LARGE, and HUGE with appropriate imaging parameters built into the procedures. They are chosen to allow us to test a wide range of computers and to compare machine overhead, computation, and input-output performance. The SMALL case has 8000 visibility samples and makes images 256 on a side with 2000 Clean components. The MEDIUM case has 13200 visibilities, images 512 on a side, and 5000 Clean components. The LARGE case jumps to 77500 visibilities, images 1024 on a side, and 15000 Clean components. The HUGE case has 908400 visibilities and images 2048 and 4096 on a side. In the HUGE case, rather than beginning at the beginning each time, the APCLN computations are restarted at 400000 and carried to a flux limit around 423000 components, while the MXCLN is restarted at 240000 and carried to about 285000 components. The HUGE test was intended for comparing large super computers and has been tried only sparingly. The LARGE case has been the work horse and the basis of all recent machine comparisons. Of course, when we began work on the DDT, the SMALL case was painfully slow on most of the machines of the day.

2 Revisions to DDT

This section should probably be entitled "no good deed goes unpunished" (de Sade's principle) since it appears rather as a saga of changes. The most important change, to substitute CALIB for ASCAL, was done first. But, when I ran it, CALIB did not work. It turned out that we have not been recording shifts correctly in image headers for a great many years. This seems not to have been significant when the model computations were done by direct Fourier transforms. However, CALIB uses a gridded modeling technique that depends on the components being exactly on grid cells. With the incorrectly recorded shifts, the components did not end up on cells and the modeling failed. Fortunately, the actual shift appears not only in the shift parameters in the header but also in the location of the reference pixel. This allowed us to construct a subroutine to correct the shift parameters in the old headers for the bug which we found (and corrected) when they do not agree with the location of the reference pixel. A minor bug in MX was also found which gave low level sine waves in the output Clean image in the unlikely case that a *uv* sample lay exactly on a row boundary. During all this testing, it was noted that the source in the LARGE case was not fully covered by the Clean box being used forcing me to increase it size.

Another improvement to DDT which has been needed for some time was support for reading and writing images to FITS disk rather than magnetic tape. This was added using the new adverb DDTDISK to specify the environment variable name for the disk area, with a blank name specifying magnetic tape. The disk file names are of the form DDT*SXXXXXX*, where *S* is the size of the problem (S, M, L, or H) and *XXXXXX* is the task name or other identifier (UVDATA, UVSRT, UVMAP, UVBEAM, APCLN, APRES, CALIB, MXMAP, MXBEAM, MXCLN, VTESS). Needless to say, this change exposed numerous hidden assumptions about magnetic tape throughout DDT.

Other corrections were also made. I switched the output to be be IEEE floating-point FITS files rather than 32-bit integer. I changed the ALLDEST calls to check all disks, not just the intended output disk. And I added the code for the HUGE test to the standard DDT run files.

A number of changes to other AIPS programs were prompted by this work. The task UVDIF was changed to detect differences due to visibilities being swapped in the sort ordering and to add these up but to print them only if something else is "wrong." Otherwise, the UVDIF outputs were filled with differences due to these swaps (*e.g.*, about 80 swaps in the LARGE case) which tended to obscure any real differences. The AIPS program was given two new history editing verbs to allow me to remove the dozen-or-so copies of the UVLOD history cards which have crept into our DDT files. The confusion caused by the new verbs led me to add a pseudo-verb to define verbs and to change the mechanism by which adverb values are set and retrieved by verbs. The DDT test uses the DOWAIT = TRUE mode in which the AIPS program periodically checks for task completion. At modern computer speeds, the old interval between checks (8 seconds) added a variable and significant overhead to the tests. This interval was reduced to 2 seconds to decrease its impact. Bugs handling long string variables were found in the AIPS verbs INPUTS and TYPE, both used by DDT. The corrections made forced a minor restructuring of the message handling in the procedures. The print routines were revised to put the host name in the title line of each page so that I could tell which machine had printed the output.

When testing the HUGE case, I found a number of errors in the DDT procedures themselves. When these were corrected, it was found that the images computed did not agree with the masters to an acceptable level. After some investigation, it was found that the gridded modeling routines (and several others) were using only the small "AP memory" (64K words) rather than the full memory (1280K words). This led to serious errors for very large images, but had some affect even on the LARGE images. Then, Bill Cotton found and corrected an error in fitting the dirty beam to determine the dimensions of the Clean beam. At the same time, it was found that some of the architectures had been using a smaller "AP memory" than that used by the Sun IPX and IBM. The Q routines and memory include files were restructured to support the larger

AP, but to allow the system parameter to restrict how much of that memory would actually be used. When checking the optimization levels used, I found that several whole libraries and the `APCLN` task were compiled in debug mode with no optimization whatsoever. These were changed to standard levels and most of *AIPS* rebuilt.

In investigating the minor changes in the LARGE images due to changes in the AP utilization, the need to provide the user with additional controls and feedback became apparent. In particular, it was found that, if too much of the *uv* plane is used to grid the data, then very high spatial frequency effects can occur during, and disrupt, the deconvolution. The adverb `GUARD` was added to all tasks which grid *uv* data to control how much of the *uv* plane is available for gridding. Messages were added to the tasks to report lost data more clearly and to warn of incautious gridding. The latter messages immediately appeared when the MEDIUM `DDT` test was run, forcing me to decrease the cell size used and to increase the Clean window correspondingly. The other user control added to the deconvolution tasks was the adverb `MAXPIXEL` to control how many image pixels are searched for Clean components during each minor cycle. This control was added to the `DDT` size-dependent parameters as well. Tests showed that rather small values, *e.g.*, 8000, give much faster results than the old default (20050) for the MEDIUM case. The LARGE case is, however, well served by the default.

The HUGE case revealed another problem with the code. The histogram used to determine which pixels are selected to be searched for components was too coarse for such large images. As a result, very few components were searched in each major cycle and a very large number of very expensive major cycles occurred. The histogram used was made much finer so that the algorithm can search nearly up to `MAXPIXEL` pixels in each major cycle.

Using `DDT` to test the effects of the size of the "AP memory" led to the discovery that some of the code stored addresses (fundamentally integers) in the floating-point AP memory. This was done to allow the memory of true array processors (which were 38 bit) to be rolled to disk (as 32-bit floating numbers). Fortunately, we no longer have true array processors and hence do not support rolling. When the addresses exceeded 16777216, IEEE floating-point could no longer store them accurately causing our gridding and other algorithms to fail. While changing the code back to use integer address storage, I noticed that the portion of the dirty beam used inside each major cycle was quite restricted, limiting the accuracy of the minor cycles. So I raised the maximum Clean patch allowed in all Clean tasks.

Many of these changes required the master images to be recomputed and the timing measurements to be repeated. (And the "punishment" continues ...)

3 Computers Tested

All computers tested for this Memo are owned, or on loan to, the NRAO, except for the Hewlett-Packard machine called langer. That is owned by the Jet Propulsion Laboratory in California and was made available to us via internet by Bill Langer. Table 1 gives the relevant details for the computers tested. (The order of machines in all tables is set by the AIPSMark$^{(93)}$ results reported here, beginning with the highest $A_m^{(93)}$.) The Gateway 486DX2-66V is a 486 personnal computer built by Gateway2000, but it is similar to a number of other 486's on the market. All machines were tested with all of the algorithmic improvements described above except those marked with an asterisk. The results for those three machines should be regarded as representative, but not precisely correct. Differences between revision levels of operating systems were not measured in general. Two lines are given for the Sun LX machine to show the improvement from SunOS 5.2 to 5.3. *AIPS* programs are not particularly "memory hungry," but some operating systems are. I found a 2.5% improvement on the Sun IPX when the memory was raised from 32 to 48 Mbytes. However, when the DEC Alpha memory was raised from 32 to 64 Mbytes there was a 21% improvement. On the Gateway, an increase of memory from 16 to 32 Mbytes improved performance on LARGE by 17%, but had no effect on SMALL. All tests (except the Convex) were run to xterm windows running under an X-Windows server. On the Gateway, the presence of the X server caused the small `DDT` to run 9% slower and the large `DDT` to run 6% slower when the PC had only 16 Megabytes of memory. The cost of the X-Windows was not measured on the other machines.

All machines, except the old Convex and the Gateway, use SCSI-bus disk drives. The Gateway uses disk

Computer		Name	OS	OS version	Memory
IBM RS/6000 (580)		rhesus	AIX	3.2.5	128
IBM RS/6000 (560)		ringtail	AIX	3.2.3	256
HP 9000/755		hptest	HP-UX	9.01	128
Sun 10/512MP	*	kochab	SunOS	5.2	112
DEC Alpha 3000/300		pongo	OSF/1	1.2	64
HP 9000/735	*	langer	HP-UX	9.01	128
IBM RS/6000 (530)		lemur	AIX	3.2.3	48
Sun IPX (4.1.2)		primate	SunOS	4.1.2	48
Convex C-1	*	yucca	Convex OS	9.0	64
Sun LX Solaris 2.3		tamarin	SunOS	5.3	24
Sun LX Solaris 2.2		tamarin	SunOS	5.2	24
Sun IPC Solaris		digit	SunOS	5.1	24
Sun IPC (4.1.2)		spica	SunOS	4.1.2	24
Gateway 486DX2-66V		tarsier	Linux	0.99.14	32
DECStation 3100		bonobo	ULTRIX	4.3	16

Table 1: Computer Systems Tested

local bus IDE which seems fairly well matched to the 486 cpu. However, when we used an improved version of the C compiler in version 14, the disks limited the improvement in total real times (*i.e.*, the cpu-to-real ratios decreased slightly). SCSI bus and disk technology has improved with time. The DECStation 3100 has a SCSI-1 bus and the oldest type of disks and the results to be presented below clearly suggest that its performance suffers as a consequence. The importance of disk speed is clearly shown by ringtail, which gave 14% better results with new high-speed disks even though it still uses only early SCSI-2 (5 Mbytes/sec) technology. The HP 755 and 735 are very similar computers except that the 755 had new large and empty disks and a fast SCSI-2 bus (10 Mbytes/sec) , while the 735 we tested had older SCSI-1 disks on which the file allocation was probably rather fragmented. The 755 outperformed the 735 by 68% on the new `DDT` test as a consequence. The Sun 10 also had fast SCSI-2 buses and disks. The IBM 580 is the first to have fast and wide SCSI-2 buses (20 Mbytes/sec) with the same fast disks as the IBM 560. In all cases, only local disks were used. I forgot about this once, and used a remote disk for messages only (neither data nor scratch files) from the `IPX`. The new `DDT`, which takes 4082 seconds with fully local disks, took 6139 seconds with remote messages. That is about 1 second per message. This suggests that $\mathcal{AIPS}$ users should not position disk 1 on some large public computer (as is the default many places), but, instead, position it either on the machine they are currently using, or on the machine they use the most.

The Fortran compilers used were all those supplied by the vendor except for the Gateway. In particular, this means we used DEC Fortran for the DECStation 3100 rather than the compiler from MIPS. We used $\mathcal{AIPS}$ "level 6" optimization for all Q routines (vector-oriented code) and $\mathcal{AIPS}$ "level 2" optimization for all other subroutines and tasks. The exceptions to this are the use of $\mathcal{AIPS}$ "level 0" for `AIPS` itself on the IBM, and for all non-Q subroutines on the DECStation. The compilers and their control options are summarized in Table 2. On the Gateway, we run `f2c` followed by the Gnu C compiler. On the IBM we also specify the compiler options `-ND16384 -NA65536 -qmaxmem=32768` to direct it to begin with larger internal memory and tables, beginning at optimization level 2. In general, the object modules were link edited in advance with all $\mathcal{AIPS}$ subroutines rather than at run time using our own shared libraries. On the HP 755 we found a performance improvement of 10% when we switched from shared to static libraries. Most of the system routines are provided from shared libraries however.

Computer	compiler level 0 control code	OPT2 adds	OPT6 adds
IBM RS/6000 (580)	`xlf -c -u -qfips -qcharlen=10000 -qextname`	`-O -Q`	`-O -Q -Pv`
IBM RS/6000 (560)	`xlf -c -u -qfips -qcharlen=10000 -qextname`	`-O -Q`	`-O -Q -Pv`
HP 9000/755	`f77 -c -a -u -v +ppu -K`	`+O2`	`+O2`
Sun 10/512MP	`/opt/SUNWspro/bin/f77 -c -ansi -u`	`-O4`	`-O4`
DEC Alpha 3000/300	`/bin/f77 -fpe3 -v -static -u -c -O0`	`-O2`	`-O4`
HP 9000/735	`f77 -c -a -u -v +ppu -K`	`+O2`	`+O2`
IBM RS/6000 (530)	`xlf -c -u -qfips -qcharlen=10000 -qextname`	`-O -Q`	`-O -Q -Pv`
Sun IPX (4.1.2)	`/usr/lang/f77 -c -ansi -u`	`-O2`	`-O4`
Convex C-1	`/usr/convex/fc -vn -72 -fi -c -O0`	`-O2`	`-O2`
Sun LX Solaris	`/opt/SUNWspro/bin/f77 -c -ansi -u`	`-O2`	`-O4`
Sun IPC Solaris	`/opt/SUNWspro/bin/f77 -c -ansi -u`	`-O2`	`-O4`
Sun IPC (4.1.2)	`/usr/lang/f77 -c -ansi -u`	`-O2`	`-O4`
Gateway 486DX2-66V	`f2c -ARw8 -Nn1604 -Nx400 | gcc -m486 -c -O0`	`-O2`	`-O2`
DECStation 3100	`f77 -fpe3 -v -static -c -O0`	`-O2`	`-O2`

Table 2: Fortran Compiler Parameters Used

4 Benchmark Results

The matter of most concern to users is the question of how long it will take to perform the full sequence of jobs needed to reduce their data. To express this simply, Glendenning and Hunt (1991, [10]) invented the concept of "AIPSmarks." They define the total run time of the `DDT` as the real time between the procedure initiation ("`RUN DDTEXEC`") and the (nearly) final print message (`PRINTING ANSWERS, ERRORS, OTHER IMPORTANT MESSAGES`). This time is easily determined from the messages printed at that final print message. From the LARGE test, they define the AIPSmark as

$$A_m \equiv \frac{5000}{T_{LARGE} - 0.6 \times T_{ASCAL}} ,$$

where T_{LARGE} is the total run time in seconds and T_{ASCAL} is the real time for the LARGE `ASCAL` step. The scaling factor (5000) was chosen to make the Convex C-1 approximately 1.0 AIPSmark. Glendenning and Hunt reported values for A_m of 2.08 for IBM RS/6000 model 550, 1.01 for Convex C-1, 0.69 for Sun SparcStation 2, and 0.36 for Sun IPC. In the time since then, it appears that we have improved the general optimization of $\mathcal{AIPS}$ over all architectures. I find (with considerable help from Dave Adler and Pat Murphy) the results for the old `DDT` test given in Table 3. Colin Lonsdale of MIT reports a similar result ($A_m = 2.26$) for his HP 735 (private communication). These results were computed essentially with the `15JUL93` release of $\mathcal{AIPS}$ before all of the software improvements described in the preceding section. This version of $\mathcal{AIPS}$ was not available on all architectures, leading to the blank lines in Table 3. In addition, the results for the IPX were obtained with "only" 32 Mbytes of memory.

With the revised `DDT`, I had hoped to define a new AIPSmark formula that would give approximately the same results as the old one. As a consideration of Table 3 would suggest, this turns out to be impossible. For example, the IBM is 5.17 times faster than the IPX at `ASCAL`, but only 2.94 times faster at everything else. Similarly, the Convex is 2.34 time faster at `ASCAL`, but only 1.05 times faster at everything else. Thus, not surprisingly, the relative machine performance depends on what the machine is being asked to perform. In particular, `ASCAL` performance depends almost exclusively on the evaluation of sines and cosines which are heavily optimized and vectorized by our special code on the Convex and which also appear very efficient with no special code on the IBM. The Convex loses its advantage when the code is less heavily vectorized and the IBM loses some of its advantage when the times are more dependent on disk input/output. The Sun 10/512MP, a fast machine at most things, is surprisingly slow to execute `ASCAL`. `ASCAL` also does very little input/output relative to its computations. Thus machines with fast cpu's appear fast on `ASCAL` even if they have poor disk performance (*i.e.*, the HP 735).

After consideration of the results of the revised `DDT` I have defined the AIPSMark$^{(93)}$ to be

$$A_m^{(93)} \equiv \frac{4000}{T_{LARGE}} ,$$

where T_{LARGE} is the total run time in seconds as defined by Glendenning and Hunt. The results of the tests are presented in Table 4. The total run times for the two Clean steps, the self-calibration step, the maximum entropy step, and the remainder of the LARGE test are given in the table to show the major contributors to the total run time. Note that the new AIPSMarks$^{(93)}$ are not grossly different from the old ones (except the HP 735) and that the new test is significantly faster than the old even with no correction for a single dominant task. The old Convex is still faster than the IPX when the code is optimized and uses single-precision floating point. However, when the code uses double precision floating-point arithmetic, even with vectorization, as in `CALIB`, or when the code is essentially scalar, as in the remainder of the test, the IPX is actually significantly faster.

To attempt to evaluate the affect of I/O on performance, we need to look at the cpu-to-real ratios achieved by the various machines. Furthermore, if a machine has a very fast cpu with modest I/O, then it may be more useful for multiple users/tasks than one with a slower cpu, better matched to the I/O speeds. Table 5 lists the cpu times and cpu-to-real ratios for the four most compute intensive tasks in the new LARGE `DDT`. The most noticeable thing in this table is the very high ratio of cpu to real times. The one exception is the HP 735, which has the fastest cpu times reported, with the lowest cpu-to-real ratios measured. A similar result was found by Colin Lonsdale of MIT (private communication). The IBM also appears to have a cpu that is faster than its I/O while the DECStation 3100 has a slow cpu (approximately an IPC), but even slower I/O.

Computer	T_{LARGE}	T_{ASCAL}	A_m
IBM RS/6000 (580)	1986	977	3.57
IBM RS/6000 (560)	2350	1219	3.09
HP 9000/755			
Sun 10/512MP	3726	2244	2.10
DEC Alpha 3000/300			
HP 9000/735	2898	1076	2.22
IBM RS/6000 (530)	4839	2470	1.49
Sun IPX (4.1.2)	9983	6309	0.81
Convex C-1	6197	2693	1.09
Sun LX Solaris 2.3			
Sun LX Solaris 2.2			
Sun IPC Solaris	18953	11581	0.42
Sun IPC (4.1.2)	19343	11503	0.40
Gateway 486DX2-66V			
DECStation 3100	17451	7561	0.39

Table 3: Old `DDT` AIPSmark Results

It is also interesting to look at the total run times for the SMALL and MEDIUM versions of `DDT` as well. These are given in Table 6 for the revised test. The SMALL test is dominated by overhead of various sorts while the LARGE test is rather more compute bound. We see that, as the computation load becomes heavier, the IBM becomes relatively more efficient than the IPX while the IPC becomes relatively less efficient. The Gateway outperforms the IPC in SMALL, but the IPC outperforms the Gateway in LARGE. The Convex handles overhead so poorly that even the IPC outperforms it on the SMALL test. The Sun 10/512MP is very good at handling overhead, perhaps taking advantage of its large memory and dual cpu's.

5 Certification Results

The version of *AIPS* running on the current computer is considered "certified" if the answers it computes are essentially the same as the "standard" answers computed at some initial time on some initial computer. The standard TEST mode of `DDT` runs `COMB` to difference the computed images with the standard images. The verb `IMSTAT` is then run on the difference images to give the maximum and rms differences. These are converted to "bits of accuracy" by dividing them into the maximum of the image and taking the logarithm base 2.

For the revised `DDT`, I computed the master images on my Sun IPX effectively with the `15JAN94` release of *AIPS*. I did this since we use Suns for much of our software development and, thus, wish to be most sensitive to changes in that environment. The IPC under both SunOS 4.1.2 and SunOS 5.1 (Solaris 2.1) and the LX and the Sun 10/512MP under SunOS 5.2 and 5.3 (Solaris 2.2 and 2.3) computed images identical to those of the IPX. The number of bits of agreement for the DEC, HP, IBM and Gateway machines are given in Table 7. The Convex is omitted because the tests were done before most of the code revisions described above. It is interesting to note that the number of bits has improved by a few for some of the tests, but is actually a bit worse for `UVMAP` and unchanged for `MX` Clean. Exact agreement is not expected since, even if the computers use fully IEEE conventions (and none do), they will still optimize the code differently and, hence, compute the numbers in different sequences. Some improvement was to be expected since we are now comparing images computed with similar computers on the same release of *AIPS*. The old test compares one set of images made with current computers and current *AIPS* to a set made with a Vax with array processor and a 4-year old version of *AIPS*. The small number of bits for the maximum of `MXCLN` is unfortunate, but appears to reflect the problem of using an adaptive algorithm for such tests. Once Clean chooses a different component in one computer from that chosen in another, the algorithm will cause the two to continue to diverge in which components are chosen. It is argued that an image with very good

Computer		$A_m^{(93)}$	T_{LARGE}	T_{APCLN}	T_{CALIB}	T_{MXCLN}	T_{VTESS}	T_{rest}
IBM RS/6000 (580)		3.62	1104	273	89	351	77	314
IBM RS/6000 (560)		3.29	1215	338	89	415	89	284
HP 9000/755		3.06	1306	375	100	491	87	233
Sun 10/512MP	*	2.45	1630	369	124	613	160	364
DEC Alpha 3000/300		2.10	1908	501	151	638	140	478
HP 9000/735	*	1.82	2199	304	373	712	184	626
IBM RS/6000 (530)		1.50	2674	687	207	908	234	638
Sun IPX (4.1.2)		1.01	3975	1041	311	1388	431	804
Convex C-1	*	0.99	4039	608	533	1173	300	1425
Sun LX Solaris 2.3		0.95	4225	1001	371	1422	481	950
Sun LX Solaris 2.2		0.91	4376	999	384	1459	520	1014
Sun IPC Solaris		0.56	7135	1830	529	2560	764	1452
Sun IPC (4.1.2)		0.54	7375	1980	563	2605	840	1387
Gateway 486DX2-66V		0.51	7780	2229	529	2950	858	1214
DECStation 3100		0.38	10626	2294	948	3130	1219	3035

Table 4: New `DDT` AIPSMark$^{(93)}$ Results

Computer	`APCLN`		`CALIB`		`MXCLN`		`VTESS`	
	ratio	cpu	ratio	cpu	ratio	cpu	ratio	cpu
IBM RS/6000 (580)	0.95	259	0.52	46	0.85	300	0.74	57
IBM RS/6000 (560)	0.96	323	0.71	64	0.90	374	0.82	73
HP 9000/755	0.81	303	0.50	50	0.75	369	0.68	59
Sun 10/512MP	0.98	360	0.78	96	0.89	544	0.84	134
DEC Alpha 3000/300	0.95	476	0.60	90	0.90	575	0.72	101
HP 9000/735	0.77	289	0.15	46	0.52	371	0.32	59
IBM RS/6000 (530)	0.94	644	0.61	126	0.82	748	0.61	143
Sun IPX (4.1.2)	0.97	1014	0.85	265	0.95	1314	0.90	389
Convex C-1	0.88	535	0.85	451	0.79	926	0.83	249
Sun LX Solaris 2.3	0.93	934	0.78	290	0.90	1281	0.86	413
Sun LX Solaris 2.2	0.92	921	0.78	298	0.89	1293	0.85	444
Sun IPC Solaris	0.96	1763	0.86	453	0.94	2399	0.92	701
Sun IPC (4.1.2)	0.91	1803	0.83	469	0.91	2372	0.88	736
Gateway 486DX2-66V	0.95	2119	0.84	447	0.94	2773	0.90	776
DECStation 3100	0.82	1815	0.59	472	0.81	2401	0.60	686

Table 5: New `DDT` LARGE Cpu Times and Cpu-to-real Ratios

Computer		T_{LARGE}	T_{MEDIUM}	T_{SMALL}
IBM RS/6000 (580)		1104	323	184
IBM RS/6000 (560)		1215	357	193
HP 9000/755		1306	337	205
Sun 10/512MP	*	1630	448	179
DEC Alpha 3000/300		1908	501	267
HP 9000/735	*	2199	546	200
IBM RS/6000 (530)		2674	683	328
Sun IPX (4.1.2)		3975	918	355
Convex C-1	*	4039	1319	770
Sun LX Solaris 2.3		4225	1018	405
Sun LX Solaris 2.2		4376	1135	453
Sun IPC Solaris		7135	1670	595
Sun IPC (4.1.2)		7375	1616	564
Gateway 486DX2-66V		7780	1579	508
DECStation 3100		10626	2399	849

Table 6: New `DDT` Run Times for 3 Sizes

Program	DECStation new DDT		DEC Alpha new DDT		HP 9000 new DDT		IBM 6000 new DDT		Gateway new DDT		IBM 6000 old DDT	
	max	rms	max	rms	max	rms	max	rms	max	rms	max	rms
`UVMAP`	15.6	21.6	16.8	22.6	14.0	20.1	13.3	15.8	13.3	15.8	12.7	17.3
`UVBEAM`	21.0	25.8	22.2	27.8	19.4	23.0	14.7	16.3	14.7	16.3	14.9	17.7
`APCLN`	9.8	16.5	12.2	17.2	11.4	17.0	9.8	16.5	9.8	16.5	10.8	16.8
`APRES`	14.1	21.4	16.3	23.0	13.5	20.0	16.3	23.2	16.3	23.2	14.2	20.6
`MXMAP`	16.4	22.6	18.4	23.8	15.7	21.1	14.1	15.2	13.9	15.2	10.9	13.0
`MXBEAM`	21.4	26.0	22.4	28.8	19.5	23.0	14.7	15.4	14.7	15.4	11.6	13.7
`MXCLN`	13.2	17.4	13.1	17.3	8.9	16.0	10.2	16.5	10.2	16.5	9.0	15.5
`VTESS`	22.2	30.5	22.2	30.7	22.2	30.0	21.7	30.0	22.2	30.1	18.6	27.2

Table 7: New LARGE `DDT` Agreement with Sun IPX in Bits

signal-to-noise ratio and a rather Clean dirty beam should constrain Clean to give quite similar results on two computers. It appears that 9 bits is the limit of that constraint despite the high quality of the data used in the LARGE test. The difference image is roughly a rotated rectangular array of positive and negative bumps of Clean beam size located only on source. The off-source regions are in excellent agreement (15+ bits).

6 Conclusions

The measurement of computer performance has a long and fairly inglorious history. Instructions per second and floating-point operations per second were early measures of performance. But pipelining of operations within the cpu, both for single computations and for computations on vectors, made these measures obsolete, or at least, of limited value. Performance measurement has shifted to the measurement of the speed of actual code execution. This began with the linpack routines and expanded into SPECmarks, or, more recently, separate integer and floating-point SPECmarks. Table 8 summarizes the published performances of the computers studied here, where the Sun 10/512MP numbers are on a per cpu basis. The Convex C-1 achieves 6.5 Mflops on the linpack tests. The column labeled R is the ratio of the SPECfp92 measured for the

Computer	Mips	Mflops	SPECmarks	SPECint92	SPECfp92	R	$A_m^{(93)}$	$AL_m^{(93)}$
IBM RS/6000 (580)							3.6	3.9
IBM RS/6000 (560)	80	31	92	48	97	4.6	3.3	3.1
HP 9000/755	124	40	147	81	150	7.1	3.1	
Sun 10/512MP	135	25		65	83	4.0	2.4	
DEC Alpha 3000/300				66	92	4.4	2.1	4.8
HP 9000/735	124	40	147	81	150	7.1	1.8	2.7
IBM RS/6000 (530)							1.5	1.5
Sun IPX (4.1.2)	28	4	24	20	21	1.0	1.0	1.0
Convex C-1	11	40					1.0	0.5
Sun LX Solaris 2.3				26	20	1.0	0.9	1.3
Sun LX Solaris 2.2				26	20	1.0	0.9	1.3
Sun IPC Solaris	16	2	14	14	11	0.5	0.6	
Sun IPC (4.1.2)	16	2	14	14	11	0.5	0.5	0.5
Gateway 486DX2-66V							0.5	0.8
DECStation 3100	14						0.4	1.0

Table 8: External Performance Measurements

computer to that of the IPX.

The last column in Table 8 is yet another measure of computer performance which I am introducing with the Memo. It is based on the total cpu time used by a new *AIPS* task called `RTIME` with an outer loop count of 50 and an inner loop count of 2000000. Table 9 lists the cpu times for the 6 main loops of the program, the total real and cpu times for the task, and the AIPSLoopMark$^{(93)}$ produced by dividing the total cpu time into 600 seconds. The first loop is a simple floating summation a floating constant:

$$Sum = Sum + D$$

The second loop sums the constant minus the current sum divided by another floating constant:

$$Sum = Sum - Sum/C + D$$

The third loop has the form

$$Sum = Sum - Sum/C + A(i) + A(2000001 - i)$$

where A is an array of dimension 2000000. The fourth, fifth, and sixth loops are similar to the first, second, and third, respectively, but entirely in integer form. There are a number of things to notice about the numbers. The Convex fully vectorizes only the first loop, but for that loop outperforms the next fastest machine by a factor of 8 in integer and 17 in floating. The Convex in the scalar loops is the slowest machine. The third loop is faster on IBMs than the second despite the apparently greater work to be done. A similar discrepancy occurs on Suns but for the integer rather than the floating loops.

It is clear that no single measurement can predict the performance of a computer system for a particular application, except for measuring it in that application. For example, the HP 735 achieved very high SPECfp92 ratings, but was just 5% faster than the IBM 560 in reported cpu times for the main tasks of the LARGE `DDT`, and was only 60% as fast in real time for the full job. What is important to the user is some weighted average of the cpu/cache, cpu-to-memory, and memory-to-disk speeds convolved with the ability of the compiler to optimize the user's code. This weighting is very much application specific. There are a few choices users can make to improve performance without rewriting the software. The most important is to use only local disks, even for *AIPS*' messages. The `MSGKILL` pseudoverb can also be used to reduce the traffic of messages to disk (with consequent loss of those messages of course). Jobs run faster when they use only the fastest disks even if they have to suffer some head contention using multiple files on the same disk. Placing the scratch files, for example, on a separate slower disk seems to reduce net performance. The

Computer	Loop 1	Loop 2	Loop 3	Loop 4	Loop 5	Loop 6	Total	Real	$AL_m^{(93)}$
IBM RS/6000 (580)	8.0	43	21	3.2	35	41	152	153	3.94
IBM RS/6000 (560)	10.0	54	26	4.0	44	51	191	194	3.15
HP 9000/755									
Sun 10/512MP									
DEC Alpha 3000/300	4.1	27	36	2.0	22	32	124	124	4.83
HP 9000/735	6.1	13	19	6.1	85	91	221	225	2.71
IBM RS/6000 (530)	20.2	109	53	8.1	89	103	384	391	1.56
Sun IPX (4.1.2)	25.5	97	148	22.9	166	138	601	608	1.00
Convex C-1	0.24	271	320	0.24	301	320	1217	1329	0.49
Sun LX Solaris 2.3	16.8	79	115	14.6	136	106	473	494	1.27
Sun LX Solaris 2.2	16.8	79	103	14.7	136	102	456	463	1.32
Sun IPC Solaris									
Sun IPC (4.1.2)	46.7	225	310	38.0	296	226	1148	1188	0.52
Gateway 486DX2-66V	48.4	177	268	37.2	106	160	800	802	0.75
DECStation 3100	16.4	92	161	6.1	120	195	597	769	1.01

Table 9: `RTIME` loop cpu times and AIPSLoopMarks$^{(93)}$

AIPS adverbs `OUTDISK` and `BADDISK` are used to control file palcement. Disks that are fragmented take longer to read and write. By clearing all unused data from the disk, *AIPS* users can improve performance. Of course, better algorithms, more memory, faster cpus, and, especially more and faster disks and I/O buses are the ultimate solution.

References

[1]Wells, Donald C., Fickling, Gary A., and Cotton, William D., 1985, "Certification and Benchmarking of AIPS on the VAX-8600," AIPS Memo No. 36, NRAO, Charlottesville, Virginia, June 24.

[2]Hilldrup, Kerry C., Wells, Donald C., and Cotton, William D., 1985, "Certification and Benchmarking of AIPS on the CONVEX C-1 and Alliant FX/8," AIPS Memo No. 38, NRAO, Charlottesville, Virginia, December 24.

[3]Wells, Donald C., Fickling, Gary A., and Cotton, William D., 1986, "Benchmarking AIPS on a VAX-8600 with FPS-120B Array Processor," AIPS Memo No. 44, NRAO, Charlottesville, Virginia, April 19.

[4]Calabretta, Mark and Rayner, Paul, 1986, "Benchmarking AIPS on a VAX 8600," AIPS Memo No. 48, NRAO, Charlottesville, Virginia, September 22.

[5]Hilldrup, Kerry C., 1989, "AIPS Benchmarks on the CLSC and PSC Cray X-MPs," AIPS Memo No. 58, NRAO, Charlottesville, Virginia, January 25.

[6]Greisen, Eric W. and Calabretta, Mark, 1990, "Installing AIPS on an IBM RISC SYS6000 and Performance Results for Convex C220 and Sun Sparc Computers," AIPS Memo No. 65, NRAO, Charlottesville, Virginia, July 16.

[7]Calabretta, Mark and May, Henrietta, 1990,"AIPS DDT Benchmark Results for Sun's SPARCStation 2GX (Sun 4/75)," AIPS Memo No. 67, NRAO, Charlottesville, Virginia, November 28.

[8]Murphy, Patrick P., 1991,"A Comparison of DDT Results for IBM RS/6000 and Convex C-1," AIPS Memo No. 71, NRAO, Charlottesville, Virginia, April 8.

[9]Langston, Glen, Murphy, Patrick P., and Schlemmer, Dean, 1991,"AIPS DDT History," AIPS Memo No. 73, NRAO, Charlottesville, Virginia, May 16.

[10]Glendenning, Brian and Hunt, Gareth, 1991, "15APR91 DDT Results on a Sun IPC, a Sun Sparcstation 2, a IBM RS/6000 Model 550, and a Convex C1," AIPS Memo No. 75, NRAO, Charlottesville, Virginia, September 23.

[11]Allen, Ernest and Langston, Glen, 1992,"Summary of DDT Accuracy Results," AIPS Memo No. 77, NRAO, Charlottesville, Virginia, September 3.