AIPS MEMO NO. 29

CPG MEMO NO. 25

Array Processor Memory Size

W. D. Cotton

I. Introduction

In many of the recent discussions of future NRAO computing plans there has been an implicit assumption that large memories (and address spaces) will make a significant increase in array processor preformance. In this memo I will investigate the effects of array processor memory size on tasks currently available in AIPS and then suggest possible improvments that could be made using other methods explicitly designed for large memory size machines. Since AIPS has developed using FPS array processors which are limited to a 64 kword address space, many of the methods used are designed to circumvent the more serious effects of the small AP memory size.

There are two kinds of problems caused by small memory size. First and most serious is the threshold type of problem in which an operation cannot be done with less than a minimum memory size. An example of this is gridding. A finite number of rows in the grid must be kept in memory; the gridding routine in MX using an FPS AP with 64 kwords can only grid rows 4096 pixels long if the usual 7 rows are required by the convolving function support width.

The second kind of problem due to small memory size is inefficiency. (With some thought threshold problems can usually be changed to this class.) The inefficiency is usually the result of doing an operation a piece at a time and using the host memory or disk as temporary storage. An example of this is the disk based two dimensional FFT used in AIPS. If the transform does not all fit in the AP memory a two pass method is employed. In the first pass as many rows as will fit into the AP are read in, the rows are FFTed and the result is partially transposed and written to the host disk. The second pass reads in the partially transposed blocks, completes the transpose, FFTs the columns and sends the result to the host. This operation is inefficient because it requires an extra round trip for the data between the AP and the host.

II. Timings for the AIPS task MX.

In the following section I will evaluate the cost of the inefficiencies imposed on AIPS by a small AP memory size. For the purposes of comparision I will use timings on the VAX for the new AIPS routine MX. MX is a good task to use for this purpose for two reasons 1) it is our fastest method of producing deconvolved images and 2) it does most of the operations we currently do in the AP (FFT, gridding, model computation etc.). Since we cannot make our AP bigger, the tests were done telling MX that the AP was smaller. Table 1 shows the results of three runs of MX doing an identical

CLEAN on a 256 X 256 image CLEANing a 240 X 240 region.

Table 1
MX timings (256x256 CLEAN)

AP size	# maj. cycles	REAL(sec)	CPU(sec)	Real ratio	CPU ratio
64 k	5	438	145.12	1	1
32 k	6	483	166.22	1.10	1.14
16 k	6	473	159.02	1.08	1.10

1. FFT timing

The principle effect shown in Table 1 is that for the smaller AP sizes an extra major cycle is required. It is important to note that the FFT is done entirely in the AP for the 64k case but requires a disk based transpose for smaller APs. The effects of this are not noticible in the overall timing. Table 2 shows the average times in the three runs in Table 1 for gridding, FFTing and correction the image.

Table 2
Map (grid, FFT, correct) time

AP size	REAL(sec)	CPU(sec)	Real ratio	CPU ratio
64 k	30.8	13.2	1	1
32 k	33.8	14.1	1.10	1.07
16 k	33.0	13.8	1.07	1.05

Table 2 shows that the disk based transpose does add a small amount to the time for making the maps although the AP size does not seem to matter once the size is smaller than that needed for the entire image. In fact the smaller size seems to give slightly better results. Table 2 is consistent with timing measurments for just the FFT which show 7 sec Real time (3.92 sec CPU) for the 64 k AP and 10 sec Real time (4.77 sec CPU) for the 16 k AP.

In order to evaluate the time used by different portions of the FFT routines, a special program was written which does FFTs on a zero filled file. Three versions of the AIPS routine DSKFFT (with associated versions of PASS1 and PASS2) were written to time three operations: 1) Full FFT, 2) an FFT omitting all calls to the array processor and 3) an FFT omitting all calls to the disk I/O routines. The latter two functions will, of course, produce nothing of interest but will allow the determination of the cost of functions omitted.

In addition to the three operations times, maps of three different sizes were FFTed; 256X256, 512X512 and 1024X1024. The 256x256 transform was done entirely in the AP memory and the two larger transforms used a disk based transpose. The FFTs were half plane complex to

full plane real as are most of the FFTs in APCLN and MX.

Since use of the array processor is not included in the CPU charges and disk head and other I/O conflicts will slow down the operation both real and CPU times used are of interest. Both real and CPU times are sensitive to the current loading on the machine the timing test were done in different usage environments. Table 3 shows the results for a moderately busy machine and Table 4 shows the timings in an otherwise emtpy machine. For reference, the last column in Tables 3 and 4 give the time required for the AP to do the FFT (FFTs plus two swaps of each word).

Table 3
Moderately Busy VAX Timings, Real (CPU) in sec.

Image size	Full	No AP	No Disk	AP time
256X256	6.5 (3.81)	2.2 (0.33)	4.5 (3.41)	0.51
512X512	25.5 (10.9)	14.0 (1.75)	16.5 (8.56)	2.13
1024X1024	85. (24.9)	57. (6.15)	47. (19.3)	8.94

Table 4
Empty VAX Timings, Real (CPU) in sec.

Image size	Full	No AP	No Disk	AP time
256X256	6.3 (3.78)	2.5 (0.36)	5.0 (3.39)	0.51
512X512	25.0(10.0)	14.0(1.69)	15.0(8.03)	2.13
1024X1024	83. (24.8)	56. (6.15)	47. (19.2)	8.94

There are several results from Tables 3 and 4. The most important is that the total real time to do the FFT is about ten times the time it takes the array processor to do the FFT and shuffle the data. Thus there is plenty of room for improvment. It is interesting to note that the timings do not seem to be affected much by moderate use. This is probably due, in part, to the fact that the FFT routines spend at least half of their time waiting on either the disk or AP I/O.

A second result from Tables 3 and 4 is that most of the CPU time used is in talking to the AP. This could be reduced by chaining AP calls through use of the Vector Function Chainer.

A third result is that AP operations and disk operations for FFTs requiring a disk transpose take about the same amount of real time. This results in an approximately 50% increase in the time to do an FFT if a disk based transposed is used.

2. In-AP CLEAN timing

The results given above indicate that the speed of MX is not seriously affected by the AP memory size in

doing the FFT. The FFT itself is about 40 % slower but when diluted by the other operations being done this effect becomes relatively insignificant. What did appear to slow the program was the increase in the number of major cycles imposed by the limited number of residuals and limited beam patch that could be put into the AP. The number of major cycles required appears to be a rather weak function of AP memory size. If the size of the AP is sufficiently large that all of the residual points and the entire beam can be put into the AP, a classical CLEAN can be done and the operation takes only one major cycle. Reducing the number of cycles will significantly speed up the CLEAN; unless the in-AP CLEAN gets too expensive.

Numerous timing tests have shown that the in-AP CLEAN takes about 3 microsec per map residual point per component. Table 5 shows the time required to find 1000 components using different numbers of residual points. It should be noted that the process of finding CLEAN components is computation limited rather than I/O rate limited and the times quoted in Table 5 are quite close to the times based on counting cycles in the microcode.

Table 5
Time to find 1000 components

Size of residual image in AP	Real time per 1000 comp.
128 X 128	49 sec = 0.8 min
256 X 256	197 3.3
512 X 512	786 13.1
1024 X 1024	3146 52.4

It is evident from Table 5 that, if many more than about 128**2 residual points are used (about the limit for FPS APs), the time to find the components will dominate the process. In many cases it would be faster to search fewer residual points and do more major cycles than to try to do the entire CLEAN in one cycle. Other reasons that multiple cycles are desirable are that MX removes aliased sidelobes and can CLEAN nearly the full field if multiple cycles are done. The ability to do full field CLEANs allows MX to make images a factor of 4 smaller than the UVMAP - APCLN combination and accounts for most of the speed of MX.

III. Suggestions for Improvement

The above timing numbers suggest that simply adding a larger AP to a system running the current version of AIPS would not improve its preformance significantly and in fact in the case of CLEAN the total run times could increase. In this section I will discuss various possible methods to improve the current system in order of increasing expense (and hopefully payoff).

1. Improvments to AIPS with an FPS AP.

Currently in AIPS there are a number of routines in which modest restructuring could allow the number of calls to the AP to be reduced by means of a Vector Function Chainer (VFC) routine. Each call to the AP takes a minimum of 3 milliseconds on the VAX-FPS combination and reducing the number of calls could improve performance. Most of AIPS I/O is done in the double buffered mode and calls to the AP are overlapped with disk I/O. This disk I/O is in general more expensive and the improvment made from reducing the number of AP calls might not be very noticible in an otherwise empty machine but probably would be noticible in a busy machine.

2. Large memory methods.

If AIPS were provided with array processors with memory sizes modestly (say 10 times) larger than the FPS AP120B devices then methods could be developed which could make explicit use of the large size. One (and the only) specific example that comes to mind is an improvment to the gridding routines. If the entire grid to be transformed could be kept in memory, the data would not have to be sorted. This would probably not make the gridding run much faster but the sort step could be omitted.

The effect of eliminating sorting can be estimated from the AIPS accounting files. Table 6 gives the fraction of the real and CPU times used in sorting on various NRAO systems expressed both as a fraction of all AIPS processing and as a fraction of the most expensive single operation (APCLN+PHCLN for VLA processing and fringe fitting for VLBI).

Table 6
Fraction of Time Used for Sorting

	Total		Largest	
System	Real	CPU	Real	CPU
CVAX::-VLB	0.013	0.043	0.24	0.18
CVAX::-VLA	0.056	0.023	0.07	0.07
AIPS::	0.042	0.120	0.34	0.45
VAX3::	0.031	0.093	0.22	0.29
Modcomp	0.018	0.084	0.14	0.24

These numbers indicate that sorting takes on the order of 10% of the of the current AIPS processing and uses 20 to 40% of the time used by the largest single operation. Eliminating sorting in our current systems would be a significant but not major savings. However, future improvements in mapping, CLEANing etc. would make sorting a more critical item unless there are comparable improvments in sorting. In fact, the effects of MX on the system performance are not reflected in Table 6 and in many cases it is a factor of 2 faster than the UVMAP - APCLN combination. Fred Schwab points out the large memory could be used to make sorting more efficient.

The VLBA may further complicate this analysis. The VLBA will, in the observing modes currently envisioned, produce large amounts of data but may often require relatively small images (512 or 1024 pixels**2). Since sorting is done more frequently for VLB than VLA data (more self cal), the sort time may become a significant portion of the computing load. In this case the full gridding capability becomes more important.

3. Parallel processing

There are a significant number of data processing applications in which parallel processors, ie. a number of APs, can be easily ganged onto the same problem. The multiple field capability of MX is one such case as is the general problem of spectral line data. Gridding, transforming and CLEANing multiple fields or spectral channels are very parallel operations and could easily use a number of APs. With multiple APs the problems of communication with the host may be more difficult.

4. Faster I/O and APs.

A relatively simple but costly way to get better preformance than the current VAX-FPS combination used for AIPS is to use machines with faster I/O and faster APs. The faster I/O is the more critical since, in most cases, the FPS array processor runs significantly faster than the VAX can keep it fed.

The importance of I/O speed is demonstrated by a recent timing comparison between the VAX and the MODCOMP in Charlottesville on an identical run of APCLN; the MODCOMP ran about 20 percent faster in both real and CPU time. The MODCOMP has a slower CPU but faster I/O than the VAX.

5. Fast disk on the AP

Since the current bottleneck in the system seems to be getting data from the host disk to the AP and back again, putting a fast disk on the AP would eliminate one link in this system. If the AP could talk to the disk drive fast enough we could probably achieve much closer to the theoretical speed of the AP. In the framework of AIPS this would be done by putting much or all of each AP

task into the AP. This could in principle be done on the Numerix or Star AP and perhaps others.

It should be noted that putting much more of the computing load into the AP requires getting much of the AIPS system to run inside the AP. This requires that the operating system inside the AP be as flexible as those in general purpose computers. At present we have no independent verification that this is the case.

In order to determine the I/O bandwidth to keep the AP busy, I have used the quoted times for various operations in an FPS AP 120B and then computed the I/O rate necessary to read from and return to disk the values for and the results from the relevant operations. These values are tabulated in Table 7.

Table 7
I/O Rates to Keep an FPS AP 120B Busy.

Operation	I/O rate Mbyte/sec
Griding data	1.0
Add two vectors	14.8
Complex FFT (1 dim)	
512	3.0
1024	2.8
2048	2.5

Note: The rate quoted for gridding does not include reading out the final grid.

Obviously different operations require different I/O rates but in general a rate on the order of a few Mbytes/sec is necessary for an AP the speed of an FPS AP 120B. Faster APs need correspondingly faster I/O.

6. Huge AP memory

If the array processor had a truely huge memory, perhaps 100's of megabytes, the scratch files used in all of the current AIPS AP tasks could be eliminated and input data, as well as intermediate results could be stored in memory. Of course, the task would have to be entirely in the AP. Such an arrangment would eliminate all communication with the host and all disk I/O and would be limited strictly by the speed of the array processor. Implementation of this option awaits future hardware developments.

IV. Concluding Thoughts

The fundamental, inescapable conclusion of the results presented above is that modest changes (factors of 10 to 20 or less) in the memory size of our current array processors will not have a strong influence on the overall time it takes to process data in AIPS. I can see no reason that this result does not also apply to processing radio interferometry data in general. For example, increasing the AP memory size from the present 64 kwords for the FPS APs to a megaword or so would only make modest improvements; in fact, the CLEANing programs might have to be modified to use less than the full AP memory to keep them from actually slowing down.

One possibility, which could not be tested with the current hardware, is that with a very large memory size significant improvements could be made with modifications to the current methods. For example, if the AP were large enough that all FFTs could be done entirely in memory for two processes simultaneously, then data for one FFT could be read in or out of memory while another FFT was being processed. Also with a memory this size, the entire uv grid could be kept in memory and sorting could be eliminated. If 2048X2048 maps were typical then the memory required for the data arrays for this senario would be at least 8 megaword (32 megabyte). Such a scheme might be able to double the throughput of a computer system.