AIPS MEMO NO. 30

CPG MEMO NO. 26

AIPS and Array Processors

Bill Cotton & Don Wells

National Radio Astronomy Observatory
Edgemont Road, Charlottesville, VA 22901
(804)296-0211, FTS=938-1271, TWX=510-587-5482

2 December 1983

1.0 INTRODUCTION

Our Astronomical Image Processing System (AIPS) is critically dependent on AP computing power to deliver acceptable performance in large-scale image processing algorithms. We currently use Floating Point Systems AP120B APs. The fundamental specifications of AIPS include an insistance on portability. We have demonstrated a high degree of portability to a variety of CPUs, operating systems, and peripheral devices. The peripheral devices include image displays, graphics terminals, printers and plotters. It now appears to be desirable that we provide some degree of portability for our AP code.

When supermini hosts cost $200K it seemed quite reasonable to pay almost $100K for an AP. But we are in the process of porting our code to supermicros which are nearly as powerful as the superminis but which have prices much nearer to $20K than to $200K. In this new situation the prices of traditional APs seem disproportionate. FPS have recently lowered their prices substantially (the 5000 series) but even their current prices will tend to inhibit the installation of 5000-series APs in supermicro systems. And there is another problem: FPS appears to be reluctant to support the 5000 series for supermicros. Thus we think it would be a good thing if we had an AP option available for lower price and with support for a variety of supermicros.

Recently NRAO personnel have been actively considering a variety of CPUs and APs with performance much higher than that of the VAX780/120B combination. The question of how best to port our AIPS applications to such systems is a critical element (indeed, it is possibly the <u>most</u> important element) of such discussions.

For these reasons it is desirable that we provide a discussion of how AIPS utilizes the 120B and how it provides a software emulation when a 120B is not available. This memo is intended to fill that need. Our main conclusion is that our concept of emulating the 120B in software (the "pseudo-AP" library) suggests that:

> other APs and supercomputers can be interfaced to AIPS most easily if they have a software interface which emulates the 120B.

2.0 AN OVERVIEW OF AIPS AND ITS OPERATION

AIPS (Astronomical Image Processing System) is a body of programs and subroutines which are the main image-forming, image-processing, and image-analysis software for the aperture synthesis radio interferometers operated by the National Radio Astronomy Observatory. The system is coded in a dialect which is an "extended subset" of Fortran-66. By this we mean that certain standard syntactic constructs of Fortran-66 (e.g., assigned-go-to) are not used, and that certain nonstandard extensions (e.g., ENCODE/DECODE and INCLUDE) are used. A discussion of the full syntactic specifications of the AIPS dialect is beyond the scope of this memo. The principle motivation for the specifications is to assure portability of the code to the maximum number of CPUs and operating systems.

One of the programs of the system is called AIPS and it incorporates the command language processor which talks to the user's terminal. The command language is called POPS (People-Oriented Parsing System). It is a fairly powerful algebraic programming language. The syntax might be approximately described as a mixture of subsets of Algol, Pascal, and PL/I. In addition to all of the usual baggage of a programming language (e.g., SIN/ATAN/LOG/SQRT/..., string functions, etc.) it has a variety of operators wired into it to execute the frequently performed operations of an image processing system, such as listing file directories, deleting files, and manipulating the digital image display.

Extensive image processing operations are done in AIPS by other programs, which are called "tasks" by the AIPS programmers. When a task is initiated it receives input parameters from program AIPS. These parameters are POPS variables in program AIPS and one of the major duties of AIPS is to facilitate the listing and modification of these parameters. When the user is satisfied with the values of the parameters (AIPS users call these the "inputs") he issues the POPS command GO which initiates the task as a detached subprocess and passes the inputs to it. When the task has begun and acquired the inputs it allows program AIPS to resume talking to the user. The task is then free to execute in the background, often for periods of tens of minutes, even hours. The AIPS user can initiate more tasks. It is not uncommon for a single user to have three tasks in progress at any moment. There is a prohibition against initiating a second copy of the same task while the first copy is still executing.

Two or more users may be executing copies of program AIPS and spawning tasks. Thus multiple copies of the same task may be in execution at the same time, one for each user. It is typical for a VAX/780 to support two AIPS users fairly well, and five tasks in action at the same time is typical.

3.0 AIPS USE OF ARRAY PROCESSORS

A number of tasks (currently 15) utilize the FPS 120B array processor when one is available. The behavior of these tasks is completely analogous to CPU-only tasks with one exception. That exception is that there is only one AP normally and so only one AP task can be executing at any moment. The other AP tasks are in a wait state trying to obtain rights to the AP. At periodic intervals (about 5 minutes) the active AP task copies the contents of the AP to a scratch file and gives up the AP for a brief interval. Other AP tasks which are waiting then contend for the AP. We have a simple priority/queuing algorithm which assures that the primary AIPS user's AP tasks have priority over secondary users in a statistical sense. Thus AP tasks share the use of the AP, with execution time quanta of about 5 minutes.

AIPS tasks which use the array processor do so as a pipelined vector arithmetic unit; that is, data are passed from the host to the array processor, some operation is done on the data and the result is returned to the host. The operation frequently involves several calls to AP routines and may make use of function chaining to reduce some of the overhead of talking to the AP. In many cases the host computer serves only to pass data from the disks to the array processor and invoke the AP routines.

Because AIPS AP tasks were developed using FPS AP120B array processors the way in which the AP is called uses FPS conventions. Basically this means that the data are specified by a memory address and an increment. The data are passed to and from the AP in separate calls and there are occasional synchronization calls to the AP. Also, we have adopted the packing scheme used by FPS for real to complex FFTs of storing the real part of the n+1 st complex value in the imaginary part of the first complex value. This convention allows in-place FFTs and the input and output of a real to complex FFT to use the same memory or disk space.

At present we make use of microcoded routines in the FPS standard libraries (BAALIB, BABLIB, UTLLIB, SYMLIB and APFLIB) and a set of microcoded and Vector Function Chainer routines developed by NRAO. The NRAO developed microcode routines and Vector Function Chainer routines are listed in Appendix A.

In order to allow the use of AIPS on machines without FPS APs we have developed the concept of a pseudo array processor in which a FORTRAN common is used as the array processor memory and FORTRAN or assembly language routines operate on data in this common. For the pseudo array processor there are FORTRAN (or assembler) routines which perform all of the functions of a true array processor. In this fashion the main programs do not know or care, except in the most subtle ways, if they are using a true AP or a pseudo AP. In this implementation we only need one version of a program and can determine whether it uses a true AP or the pseudo AP by link editing it with the appropriate library. In practice we maintain both versions of our AP tasks on our development machines. A list of the pseudo AP routines is given in Appendix B.

In several ways we have generalized our code beyond what is necessary for an FPS AP120B. First, the size of the AP memory is read from a disk file and is not assumed to be limited to 64 Kwords. Most of our programs make use of this information and can make use of any and all AP memory available. Second, the AP memory addresses sent to the AP routines are in fact 32 bit addresses with the two lowest bytes first. This provides both the unsigned 16 bit integers needed for FPS machines and up to 31 bits for other machines. There is the complication that on some host computers the order of the bytes may need to be rearranged for non-FPS APs.

4.0 USING NEW APS WITH AIPS

The strongest requirement for a new array processor to be used with AIPS is that it be made to emulate an FPS AP120B. At the present time this means that there needs to be a library of routines duplicating the names, call arguments and functionality of the pseudo AP routines given in Appendix B. Any implementation of an array processor which could not be done in this way would mean redeveloping the existing applications programs to use the new AP and would significantly increase the maintainance required for AP programs. It is beyond our current ability and desire to support an AP which cannot emulate the current functions of our FPS AP120Bs. Note that the task of installing AIPS on a new AP is essentially defined as reproducing the pseudo-AP library for the new AP by incorporating all necessary functions (allocating an AP, loading routines etc.) into the equivalent of the appropriate pseudo-AP routines.

It is desirable that it be relatively easy to develop and maintain microcode for any new AP which we support for AIPS. First we must convert our own microcode routines to run on the new array processor and secondly, further development of AP microcode would have to be done for two, probably very different, APs and this continuing cost needs to be held to a minimum.

A final desirable property of a new AP for AIPS is that it have potential for improvement in the future over the current systems. This would include the ability to access a disk drive from the AP and a relatively sophisticated operating system in the AP. AIPS Memo no. 29 ("Array Processor Memory Size", 29 November 1983) analyzes the performance of AIPS useage of an AP and suggests possible ways to speed up the system.

5.0 AN ALTERNATE APPROACH

The current implementation of AIPS is primarily on superminis with anticipated support on supermicros. The CPU speed of these machines is sufficiently slow that they require an array processor to make them interesting AIPS machines. The performance of these computer-AP combinations is, at the moment, limited primarily by the overhead of the host talking to the AP and the additional data transfers.

An alternate approach is to use a computer whose CPU is fast enough to be interesting without an array processor, i.e. a "supercomputer". The pseudo-AP concept currently implemented in AIPS is probably readily adaptable to a supercomputer, especially if an optmizing Fortran compiler is available. If necessary, critical pseudo-AP routines could be recoded into assembly language to optimize the performance. The high degree of machine dependency isolation in AIPS should allow relatively easy installation of AIPS AP tasks on a supercomputer.

If the operating system on the supercomputer does not support multitasking then we might not be able to use the AIPS program itself on the supercomputer. (The "AIPS" program implements our "POPS" command language processor which is our interactive user interface.) In this case it might be necessary to implement only the AIPS array processor tasks. This solution would be less desirable from a users point of view but would probably allow a relatively rapid and efficient implementation of existing software on the supercomputer. Thus, image processing calculations probably could get done, and even be implemented fairly quickly, but flexible control of the processing operations would be hard to obtain. The best environment for an AIPS implementation will be a multitasking timesharing system. A precise description of the minimum specifications for the host operating system is beyond the scope of this memo, but a good indication is provided by the statement that both VMS and UNIX provide comfortable environments for AIPS. The ideal supercomputer OS for an AIPS implementation is UNIX because we already have an AIPS implementation for UNIX and probably would not need to do any new work to get AIPS running. Of course we would probably want to fine-tune the implementation for optimum performance but astronomers could do astronomy with the computer while such development work was in progress.

APPENDIX A

NRAO AIPS microcode and VFC routines

A list of the NRAO microcode routines is given in the following table:

NRAO microcode routines

APGRD1	IMULT
HIST (*)	IADD
CSQTRN	ISUB
CVCMUL	CVSDIV
MAXMIN	CVSMS
VIDIV	APGRD3
CVJADD	GRDMIX
PHSROT	CVMMAX
BOXSUM	APGRD4
APGRD2	APINTP
VTRANS	DIRADD
IMOD	CLNSUB

(*) HIST, although written by NRAO, emulates a microcode routine which is provided by FPS in a library which NRAO has not purchased.

Vector function chainer routines written at NRAO are listed in the following table:

NRAO Vector Function Chainer Routines

APlFIN.VFC;2	APlGRD.VFC;8	APGRID.VFC;19
APRFT.VFC;9	FINGRD.VFC;13	GRDCC.VFC;5
GRDFIN.VFC;34	GRIDAP.VFC;10	MCALC.VFC;5
MULCLN.VFC;3	PTDIV.VFC;4	PTSUB.VFC;5
SEARCH.VFC;5	TPVF1.VFC;6	TPVF2.VFC;6
UVINTP.VFC;8	XXPTS.VFC;6	

Listings of the source versions of these routines may be obtained by the following VMS command:

```
PRINT/HEAD CVAX::UMA0:[AIPS.15JAN84.FPS.SUB]WDC.AP,*.VFC
```

APPENDIX B

AIPS Pseudo-AP Routines.

The following table lists the AIPS pseudo-AP FORTRAN routines:

AIPS Pseudo-AP FORTRAN routines

APlFIN.FOR;1	APlGRD.FOR;7	APCOM.FOR;7
APGET.FOR;6	APGET2.FOR;4	APGRDl.FOR;6
APGRD2.FOR;8	APGRD3.FOR;1	APGRD4.FOR;4
APGRID.FOR;12	APGSP.FOR;5	APINIT.FOR;5
APINTP.FOR;1	APPUT.FOR;5	APPUT2.FOR;4
APRFT.FOR;4	APRLSE.FOR;3	APWAIT.FOR;3
APWD.FOR;3	APWR.FOR;3	ARRAY.FOR;46
BAKSUB.FOR;5	BOXSUM.FOR;4	BPCOM.FOR;3
BPINIT.FOR;6	BPRLSE.FOR;3	CFFT.FOR;4
CLNSUB.FOR;4	CRVMUL.FOR;4	CSQTRN.FOR;5
CVCMUL.FOR;5	CVCONJ.FOR;4	CVEXP.FOR;5
CVJADD.FOR;5	CVMAGS.FOR;4	CVMMAX.FOR;3
CVMOV.FOR;7	CVMUL.FOR;4	CVSDIV.FOR;8
CVSMS.FOR;5	DIRADD.FOR;4	GRDCC.FOR;1
GRDFIN.FOR;7	GRDMIX.FOR;3	GRIDAP.FOR;9
HIST.FOR;6	LVGT.FOR;4	MAKMSK.FOR;5
MAXMIN.FOR;5	MAXV.FOR;5	MCALC.FOR;6
MINV.FOR;6	MOVEl.FOR;5	MTRANS.FOR;5
MULCLN.FOR;3	PHSROT.FOR;4	POLAR.FOR;4
PTDIV.FOR;6	PTFAZ.FOR;3	PTSUB.FOR;8
RECT.FOR;4	RFFT.FOR;6	RLIEJ4.FOR;5
RLINJ4.FOR;5	SEARCH.FOR;7	SQMUL.FOR;4
SUBl.FOR;4	SVE.FOR;4	SVESQ.FOR;2
TESTR.FOR;3	UVINTP.FOR;1	VABS.FOR;3
VADD.FOR;4	VCLIP.FOR;4	VCLR.FOR;3
VCOS.FOR;3	VDIV.FOR;5	VEXP.FOR;4
VFILL.FOR;6	VFIX.FOR;4	VFLT.FOR;4
VIDIV.FOR;4	VLN.FOR;2	VMA.FOR;5
VMOV.FOR;5	VMUL.FOR;4	VNEG.FOR;4
VRVRS.FOR;4	VSADD.FOR;3	VSIN.FOR;3
VSMA.FOR;4	VSMAFX.FOR;3	VSMSA.FOR;3
VSMUL.FOR;4	VSQ.FOR;4	VSQRT.FOR;3
VSUB.FOR;4	VSWAP.FOR;5	VTRANS.FOR;7
VTSMUL.FOR;4	XFOUR.FOR;3	XFOUR.MAR;7 (*)
XXPTS.FOR;10		

(*) XFOUR.MAR is functionally equivalent to XFOUR.FOR. It is a more efficient assembly language version for VAXes under VMS.

Listings of the source code of these routines may be obtained by:

```
PRINT/HEAD CVAX::UMA0:[AIPS.15JAN84.PSAP.SUB]*.FOR
```