NATIONAL RADIO ASTRONOMY OBSERVATORY
VERY LARGE ARRAY PROGRAM

VLA COMPUTER MEMORANDUM NO. 144

Post-Processing - Phase I: Technical Memorandum

The Beginnings of NIPS

Eric W. Greisen

March 28, 1978

NIPS, short for NRAO Image Processing System, is a collection of software designed to run on the Modcomp IV/25 in Charlottesville. It will drive a variety of output and processing devices including the Dicomed, the TEK 4012, and the line printer (Centronics) which we already possess and a television-based image processor (probably from I^2S), an array processor (probably from Floating Point Systems), and a printer/plotter (probably a Varian) which we expect to get sometime this year. The disks attached to the Modcomp are two 1.25M-word cartridge disks (one demountable, both use 100 words/sector) and a 12.5M-word demountable disk (@ 128 words/sector). However, procurement of at least two 84M-word disks (@ 128 words/sector) has already begin.

The purpose of this software/hardware collection has not been defined in suitable detail and there is strong pressure to make it be everything for everybody. As I see it, however, the system will deal first with continuum interferometric (i.e., VLA and Green Bank Interferometer) maps. The programming effort will concentrate initially on user communication and basic display methods and routines. When these are well grounded we may then develop the numerical processing and advanced display routines to which we've made allusions in earlier memoranda. Although we must be careful to avoid excluding spectral-line and non-interferometric maps in these early stages, we can't give them special attention until the third stage in this development. In the

fourth (or perhaps third) stage, we will add uv (Fourier transform) data to the data base and develop uv-plane displays and routines involving transformations between uv and map planes.

This memorandum is devoted principally to a detailed discussion of the first stage of this development process. I present here my current ideas, some of which are already coded, for various technical aspects of NIPS. This version of this memo is not intended for general distribution within the scientific/administrative community. Instead, it is a request for comments - both positive and constructively critical - from the computer-oriented community. The program is not so developed that it cannot undergo major changes. Furthermore, I have tried to design it in a modular, pointer-passing manner which ought to minimize the disruptions arising from alterations.

1. Operating system: I am using MAX IV, Rev A.0 at the moment. Rev A is out of date and contains a variety of errors. In particular, (1) the ROLLER doesn't work, (2) the FORTRAN run-time package (~8K bytes) is not re-entrant and, hence, must be attached to each FORTRAN task, and (3) several processors contain rather annoying errors. These problems have been fixed, I believe, in Rev. C. However, Rev. C is based upon Modcomp's new, flexible File Manager System. This development is described with great detail and no organization in Modcomp's latest manuals. The needed overview would have to come from Modcomp's courses on the subject. Unfortunately, these courses aren't offered in the Spring quarter! Thus, until I can understand File Manager, I will continue to use Rev. A. It

is my impression that the ROLLER is useful in efficient systems only when long background tasks must make room infrequently for urgent, but quick, real-time tasks. We shouldn't have such an environment. Thus, we avoid an inefficient, clogged system by omitting the ROLLER at the expense of limiting the number of concurrent tasks. The waste of core for extra copies of the FORTRAN run-time package will become a problem since we have only 192K bytes of core. The processor errors may be circumvented.

2. Program language: The structure of Modcomp's subroutine packages, processors, and executive services provides a strong inducement for the use of Assembly Language. However, the political (but nonetheless reasonable and perhaps practical) desire to create exportable programs encourages the use of Fortran. I feel that a mixture of the two languages is acceptable. Fortran will be used for the main algorithmic routines. Assembly will be used in small subroutines supporting the Fortran routines and in areas in which FORTRAN is just too cumbersome (principally I/O). In fact, the 5500 lines of code generated to date are all in Assembly but they deal predominantly with menus (I/O, parsing), file management, and bulk data movement for which FORTRAN would be extremely inefficient.

3. Program structure - general remarks: Part 1: The program package can be regarded as consisting of roughly four parts. The first part, NINIT, is active when the system first comes up. It initializes global common including setting data set size information into data base management areas. Thus, only NINIT need change when more disc becomes available. NINIT also talks to the user to obtain the time and user ID

needed to assign a user block in the default file (see below). Finally, NINIT activates NIPS and self destructs.

Part 2: The main user communication task is NIPS. This task handles menu preparation, menu communication, parameter interpretation, a large part of data base management, program start up, and, occasionally, even carries out the requested operation itself. Along the way, NIPS provides user assistance and error checking services. NIPS is a heavily overlayed program in which the root segment contains many subroutines which handle common processing needs and a main routine which deals with category and operation selection menus. Conceptually, each overlay segment will be a menu processing subroutine peculiar to the selected operation. Since the form of these segments is at the heart of the package and since I've done most of my work to date in this area, I will describe these segments in considerable detail.

We may identify three types of operations. Type I includes very quick operations done in the NIPS overlay segment itself and interactive tasks which require the TEK 4023 and, hence, which require NIPS to suspend itself. Type I operations are, thus, never found to be running upon a new execution of a NIPS overlay. Type II operations are done by separate tasks which are normally quick and are not conducive to user interference other than orderly murder. Type III operations are lengthy, done by separate tasks, and conducive to interuption, parameter alteration, and resumption. The full menu overlay process is then:

1. Test program activity bit located in Global common (Types II and III). If active, ask user if he wishes to kill (Types II and III) or to change (Type III) the currently

active program. If user says kill, turn off activity bit and abort program. If user says change, go to change-menu process (see #12 below).

2. Get selected set of default values and format into menu.
3. Write menu to TEK4023, provide HELP if required, read parameters back.
4. Parse and interpret parameters including error checking.
5. Locate requested maps including preparing space for any maps to be added to the data base.
6. Test program activity bit (Types II and III). If active, ask user whether to wait for it to finish or to abort menu routine. Act on user's answer.
7. Place parameter and map location data in global common words assigned to task.
8. Update current and, if requested, user defaults.
9. If operation done by separate task, set program activity bit and activate the task. Otherwise do the operation.
10. If operation done by a separate task which requires the TEK4023, then suspend NIPS.
11. Return to root segment to get a new operation.

12. Change mode: less certain since none coded but maybe-

    a. Set bit telling program to stop for the moment.

    b. NIPS suspends until bit reset.

    c. Program at logical suspension point has placed data indicating its progress into Global common, turned off active bit, turned off stop bit and suspended itself until active bit set.

    d. NIPS overlay resumes, formatting current values and progress data into a change mode menu.

    e. Do steps 3, 4, 5, 7, and 8 above (in change mode version).

    f. Set program activity bit on.

    g. Return to root segment.

The change mode described above allows the user to change his program at whatever iteration is occurring when step 12a is executed. An additional and more useful change mode can be developed, however. In this version, the user, at program start-up or the previous change, specifies some event (e.g., some number of iterations) after which the program is to wait for the change mode to be entered. The program could tell the user when it starts to wait using the line printer and then remind the user by repeating the message periodically using a timer. A prime example of a changeable program is CLEAN. It is, for example, entirely reasonable to loosen the restrictions on the search area after some number of iterations.

Part 3 of the program package consists of those tasks which prepare displays for user interaction. These tasks place their output on devices from which cursor position data may be read, e.g., the television and the TEK4012. They also place information in global common to allow the association of cursor position with sky coordinates and true map intensities. I am being vague here because the program structure depends on the complexity of the routines needed to communicate to the device. A possible scenario is as follows: For the TEK4012, we have a task or tasks to draw countour and profile maps and to leave the necessary map data in common. Other tasks may change the picture on the screen (e.g., to plot a single profile), but won't alter the commons. Cursor reading should be simple enough to allow even fairly straight computational tasks to read the cursor. However, for the television, I expect a single task to handle all direct communication to the device. This task would handle data scaling, anotation (e.g., ticks, contour, profile plot), intensity alteration (e.g., pseudo coloring), and similar operations. This task would be invoked by interactive graphics tasks (perhaps as overlays to NIPS just for map display manipulation) and by interactive versions of computational tasks.

Part 4 of the program package consists of all those tasks which do the real work on the data. These tasks are started by NIPS, use parameters placed by NIPS in global common, and communicate with the rest of the programs via activity bits in the regular and the database management global commons.

4. Default management: A default file will be maintained on cartridge disk. The file has space for 1600-word default blocks assigned to "current", "system", and 20 users. A directory in the file records the user ID and date last used for the 20 users. When a user logs onto the package, the program NINIT searches this directory for the user's ID. If it is found then the corresponding default block is reassigned to the user and he recovers those parameter values which he chose to store during his last run(s) on NIPS. If the user's ID is not found in the directory, then the oldest default block is reset to the system values and assigned to the user. The current defaults are initialized, by NINIT, to the user defaults. Each time that a menu is read, the parameters entered are used to update the current defaults. The user's defaults are also updated if he so requests (using a blank which will appear on all menus). When the user selects an operation, he may specify that his own or the initial system defaults be used. If he specifies neither, then the current defaults are used.

A service program to create the default file and to add new parameters to each block has been written.

5. Menu management: A menu consists of 19 lines of text information including protected, non-transmittable fields used for descriptions and unprotected, transmittable fields used for parameters. The menu forms are stored on disk with the system defaults already filled in. When an operation is selected, the associated menu number is found via look-up table in the main part of NIPS. After the menu is read into core from disk, the NIPS overlay fills out the menu form with the default values (found using the specified default block) and transmits the completed menu to the TEK4023. Note that this process means that the later parts

of the NIPS overlay need not differentiate between default and new, user-entered values. When the user transmits the completed menu back to the computer, only the transmittable parts plus control characters are actually transmitted. A subroutine converts the control characters and any non-transmittable fields to blanks before the NIPS overlays attempt to parse the answers. Should an error be found, the overlay will ring a bell, write the word error, and cause the erroneous field to switch to black characters on a white field. The menu read is then reinitiated. Since users have a tendency to ask for things they don't want, each menu will have a parameter which will, if selected, cause NIPS to restart at the beginning.

If a user becomes confused concerning the meaning of a parameter, he may obtain additional information about it by first positioning the cursor within the parameter area and then pressing the ESC and] keys. This key sequence causes the cursor position to be returned to the reading subroutine. REDMEN recognizes the request by the leading control character, calls HELP, and finally reinitiates the menu read. The HELP subroutine uses the menu number as the entry to a look up table to find the record number within the HELP disk file of the pointer block assigned to the menu. These pointer blocks contain lists of the positions of the parameters on the screen and the associated record numbers for 5-line messages about the parameters. Thus, the transmitted cursor position leads to a message which is displayed just below the menu on the TEK4023 screen. Finally, the cursor is repositioned to the input location.

6. Data base management: I will describe the data base management (DBM) scheme which I plan to use for maps. Although I presume that a similar DBM scheme will apply to uv data, it is premature to worry about that

problem. I have set aside a map header file, part of a DBM file, half of a page of global common, and N (≤5) large data files to handle the map DBM.

Maps may be identified by the input source name, map number, and map type. In addition they may be identified by a number (called NIPS#) which represents their sequential position on input to the NIPS and also the record number of the header in the header file. A 12-word manager block is used for each map to contain these identifiers plus information on map size, map location on disk (block offset and file number), and current map activity. This last parameter is used for map deletion and to prevent tasks from using the map in a conflicting manner (e.g., to prevent one task from changing the map while another tries to read it). Eight of these manager blocks appear in each record of the DBM file. The first 100 words of the global common COMDBM contain the "in use" record from this DBM file. Words 100-124 of COMDBM contain information on the data files including name, next available record number, and maximum number of records. COMDBM also contains counters of the total number of maps, the number of tasks using the current DBM-file record, and the number of changes made in this record. Only one subroutine is used to access the DBM file. This routine first checks that other versions of itself are not already reading or writing the DBM file. Then it waits until all tasks are through using the current DBM record, rechecks the I/O activity, saves the current record (if needed), gets the new one, increments the in-use counter, and returns. The calling programs must make prompt use of the DBM data, increment the change

counter when appropriate, and then decrement the in-use counter releasing COMDBM for use by tasks requiring other records in the DBM file. This DBM scheme certainly contains the potential for serious logjams. However, since the number of simultaneous tasks is severely limited in the Modcomp IV and since most tasks won't need the DBM file for very long, I believe that this DBM scheme is usable.

7. Formats: The map header format planned for NIPS is listed in Appendix A. I believe that all needed parameters are present, but there are a lot of vacant words if I've forgotten some. You will note that the format is generally compact and uses no floating point. For use on the Modcomp at the moment, data will be on unlabeled, 1600-bpi tapes with none of the control words associated with standard tape formats (e.g. IBM's VB format, the VLA's uv-plane export format). Each row of the map will constitute a separate tape record and the data will be in half-word integers. The data are normally oriented such that the first datum is associated with the north-east (upper left) corner of the map.

In the future we may wish to add to the tape format the control words and record structure now used in VLA uv-plane export tape format. Since the latter is the logical uv-plane data format to use on the Modcomp, it seems reasonable to make this addition while we also add uv-plane routines to NIPS.

8. The IBM 360: We already have a large body of software available on the IBM 360/65. These routines are described in User's Guide for VLA Data Reduction in Charlottesville by Eric W. Greisen and Fred Schwab

(Users' Manual Series #29). The programs are used to edit, sort, calibrate, map, and display data received from either the synchronous or asynchronous subsystems of the VLA. We will continue to develop and maintain these packages. At the moment, we are working on new algorithms for antenna-based calibration, for delay beam correction, and for model fitting in both the uv and map planes. In addition we are developing a new group of map making, processing and displaying routines based on the new format discussed above. Although these routines will not offer new algorithms, they will offer enhanced capabilities through allowing for rectangular maps and rotated coordinates. I believe that these programs are important and that they will continue to be used long after NIPS is a functional system. I believe that the NIPS program will be used mostly to acquaint the user with his data and to help him decide what to do with them. The main work will still be done in the multi-user, batch environment of the 360.

9. Personnel: I wish to give here an outline of my feelings about the roles of our current personnel.

a. Bob Burns: overall supervision with emphasis on purchasing, public and administration relations, and integration with the rest of NRAO.

b. Eric Greisen: detailed supervision, project design, application programming particularly on the Modcomp.

c. Fred Schwab: mathematical studies, algorithm development including application software.

d. Betty Stobie (currently available roughly half time for this project): application programming on the Modcomp.

e. Claude Williams: user support, documentation development, application programming on the 360.

f. Scientific Staff: assist with program debugging, advise on desired/required capabilities.

This group is small enough to make all of us overworked, but large enough to require some formal organization. For these reasons, I would like to request that the scientific staff direct their suggestions principally to me and in writing. (Informal notes on scrap paper will do.) This will assist in maintaining a cohesive group effort and in providing as many of the needed programs as possible.

10. Current programs: At present, the NIPS system consists of some service programs (to build the menu, help, and default files), a variety of detailed menu handling subroutines, and a few menu overlays and/or programs in the data movement area. The current, as yet untested, capabilities are:

1. Move maps from tape to disk with subarray selection.
2. Move maps from disk to tape.
3. Flag/unflag maps.
4. List disk contents on terminal and/or printer.
5. Compress map data sets.

I am in the process of testing these programs and of determining what needs to be fixed in our TEK4023 terminal. The next area for program development is display on the TEK4012.

In conclusion I would like to mention a philosophical point. The NIPS system should, eventually, be of use for interferometric maps (Green Bank interferometer, VLA, VLBI, and other arrays) and for single-dish maps. It should be able to handle spectral line as well as continuum data and it should process calibrated visibility data as well as map data. Because the ultimate intentions are general, I have dropped VLA from the title of this report and I use the program name NIPS (NRAO Image Processing System). However, since we are responsible for the VLA batch system, since the needs of the VLA in the image processing area are pressing, and since a lot of the software developed for VLA image handling will apply to other images as well, our work (and this memo) will be concentrated on VLA problems for the moment.

I have given in some detail a review of my thinking on the NIPS system. I will appreciate hearing your reactions to these plans.

APPENDIX A: Map Header Format

WORD	PARAMETER	NOTES
0-3	Source name (ASCII characters left justified)	
4	Source numeric qualifier	
5	Map number (<0 = > deleted)	
6	process level/parameter	(1)
7	product/band	(2)
8-9	frequency (kHz)	
10-11	velocity (m/s)	
12	units/coordinates	(3)
13	gain (used to normalize map values)	(4)
14	minimum map value (normalized)	
15	maximum map value (normalized)	
16	X dimension (N * 64 recommended)	
17	Y dimension (any value ok)	
18-19	X array spacing (m/s or 0".00001 at field center)	
20-21	Y array spacing (0".00001 at field center)	
22	map rotation (0.1 degrees CCW)	
23	minimum X array point of valid data (all of Y assumed ok)	
24	maximum X array point of valid data	
25	NRAO User number	
26-27	right ascension (1950.0 current phase center, 31-bit fraction of circle)	
28-29	declination (1950.0 current phase center, 31-bit fraction of circle)	
30-31	X of phase center (0.01 array locations)	
32-33	Y of phase center (0.01 array locations)	
34-35	clean beam: major axis (0".00001)	
36-37	clean beam: minor axis (0".00001)	
38	clean beam: position angle (0°.01 CCW from N to major)	
39	number iterations used in CLEAN	
40	map creation time: DD + 256*MM (local time)	
41	map creation time: year (all 4 digits)	
42	map creation time: MM + 256*HH (in IAT, can go to 29 hours)	
43	observation date: MJAD	
44	observation date: DD + 256*MM (local time)	
45	observation date: year (all 4 digits)	
46-47	right ascension (1950.0, original phase center, 31-bit fraction of circle)	
48-49	declination (1950.0, original phase center, 31-bit fraction of circle)	
50-79	?	
80-99	user comment (ASCII characters, left justified)	

APPENDIX A (continued)

FORMAT NOTES

(1) Word 6:

process level uses bits 0-7 as

bit #	off	on
0	dirty	clean
1	FFT	MEM
2	data	model
3	--	dish beam correction
4	--	delay beam correction
5	--	FFT convolution correction
6		?
7		?

The low-order byte contains the code for the type of data found in the map. Currently assigned codes are

0	reserved to imply all	7	$X = 0.5 * \tan^{-1}(U,Q)$
1	beam	8	fractional polarization P/I
2	Stokes I	9	spectral index
3	" Q	10	continuum (used for line work)
4	" U	11	optical depth
5	" V	12	velocity
6	$P = \sqrt{Q*Q+U*U}$	13	blanking

(2) Word 7:

"product" in higher-order byte current has assigned values:

0	all	3	points
1	normal	4	residual
2	components		

and, at this time, would be useful only with CLEAN.

"band" in the low-order byte has values 1 to 5 to imply L, C, U, K, X, and S, respectively.

APPENDIX A (continued)

(3) Word 12:

"units" in upper byte has assigned values:

0	all	3	none (as optical depth)
1	Jy/beam area	4	degrees (as angle)
2	Kelvins	5	km s^{-1}

"coordinates" use lowest nibble for Y, next higher for X with assigned values:

0	right ascension	3	galactic longitude
1	declination	4	galactic latitude
2	velocity		

(4) Word 13: "gain".

The true map value is obtained from the recorded (16-bit integer) value by multiplying by 2.0**GAIN.