AIPS Memo 83

Dual Libraries and Binaries in AIPS

Patrick P. Murphy

December 14, 1992

Abstract

One of the most frustrating aspects of working with *AIPS* on a daily basis is dealing with the debug and/or optimization level built into the many libraries. There is inherently a conflict between the desire for high performance and the need to access symbolic debugging information, at least on many compilers. This is true on current versions of SunOS where one cannot use debug and optimize qualifiers together on compile or link commands. To address this problem, a scheme similar to one first proposed by Mark Calabretta ([1], [2]) has been implemented for the Sun-4 architecture in the 15APR93 test version of *AIPS* at both Charlottesville and AOC sites within NRAO. The rest of this memo describes in some detail the mechanisms used in this system.

1 Introduction

The basis of the system is to have two separate areas for object and/or shareable libraries: one that will hold the debug-capable, non-optimized versions of the object modules, the other for the optimized version with no symbolic debug information included. There are also two corresponding `LOAD` areas for the resulting binaries (executables in historic *AIPS* terminology). Several of the Unix shell scripts have to be modified to accomodate this change. The entire system is enabled or disabled based on the presence or absence of a single file: `$SYSLOCAL/DOTWOBIN`.

The new areas are pointed to by environment variables ("logicals") `LIBRDBG` and `LOADDBG`. For now, these are defined in the site-specific portion of the `AIPSASSN` files as directories parallel to the `LIBR` and `LOAD` areas, *e.g.*,, `$AIPS_VERSION/$ARCH/LIBRDBG`. If the shell scripts described below fail to find them and the `$SYSLOCAL/DOTWOBIN` file is present, the variables are dynamically defined and exported.

In the sections that follow, the behaviour described is for the most part that relating to files that reside *in the main AIPS directory tree*. Local files, *e.g.*,, those residing in a user's private directory, will still compile and link in the same way they always have. Throughout this document, the term "local" will be used to refer to such private files, and "system" to refer to files in the `$AIPS_VERSION` directory tree. The only significant changes users developing such "local" code really need to be aware of is the presence of a debug set of libraries, and the new command line option on `LIBS` (see below).

2 COMRPL – New Behaviour

The normal behaviour of `COMRPL` is to (a) compile a source code module into an object file, and (b) optionally move it to a holding area for eventual insertion into a library. Part (b) is not done for "local" code. The revised `COMRPL` procedure will compile any "system" module twice as follows.

First, any optimize/debug directives from the command line are removed, and any `PURGE` directive is overridden (the preprocessed source code will be needed for a second compilation). Thus, for these "system"

files, the debug/optimize settings are determined solely by the FCLEVEL.SH and OPTIMIZE.LIS files for Fortran modules, and by CCOPTS.SH for C language modules.

The first compilation done will produce an optimized, non-debug object file. There is an inherent assumption here that the settings in the various files mentioned in the previous paragraph will produce just this.

This second compilation takes the basic options (again, exclusive of command line directives) and overrides them with DEBUG, NOOPT, and NOPURGE. Then both this object file and the first one are moved to their respective directories under the LIBR and LIBRDBG areas. The procedures will give verbose details on how the object files are kept separate, but knowledge of these details is not necessary for an understanding of the logical behaviour of the script.

3 LIBS – New Command-line Option

The use of the LIBS shell script is solely to generate options files for "local" compilations and links. In the revised setup described here, it is necessary to be able to specify whether the debug or non-debug versions of the libraries are used. This is accomplished now via, *e.g.*,: LIBS $AIPPGM DEBUG or maybe LIBS $APGNOT -d. Either form of the option is case insensitive. If any form of this is detected, the procedure will generate a list with the LIBRDBG set of libraries. Otherwise the default action will be to use the LIBR libraries.

4 COMLNK – New Behaviour

This revised procedure is similar in operation to COMRPL. As with its cousin, the behaviour for "local" files is unchanged. For "system" files only, its behaviour depends on whether or not the DEBUG option is specified in the command line. If it is not, the behaviour of COMLNK is unchanged.

However, if DEBUG is specified for a "system" file, the program in question will be compiled *twice* just as modules are compiled twice in COMRPL, and the link will be performed twice also. The non-debug libraries are used with the first link, and that binary is moved to the regular LOAD area. The debug libraries are then used with the debug-compiled object file in the second link. The binary from this operation is moved to the LOADDBG area.

The alternate load areas may still be used in an environment where multiple TVs or Array Processors are supported. In such an installation, the naming convention for LOAD areas is extended to also include LOADDBG, *e.g.*,, LOADDBG2 will be $LOADDBG/ALT2 just as LOAD2 is currently defined as $LOAD/ALT2.

One feature not yet implemented but perhaps desirable for "system" debug load modules would be to remove both debug and nondebug versions of a given task any time a COMLNK (debug or not) is performed on that task. Otherwise it is possible for the LOADDBG area to become cluttered with obsolete versions of tasks.

5 Impact on Shared Libraries

The use of shared libraries in the Sun versions of *AIPS* has been controlled for some time by the presence or absence of the file $SYSLOCAL/USESHARED. This has not changed with the modifications described above, except that shared libraries will only be used for the optimized, non-debug case. They are then generated in addition to the regular static libraries.

The location and naming of these shared libraries has, however, changed. The static libraries will remain in the various subdirectories in the LIBR area, but now the shared libraries will be stored directly in LIBR and are named after the area in question. Thus, what was formerly $LIBR/APLSUB/SUBLIB.so will now be found in $LIBR/APLSUB.so. The rationale for this change is to permit the easy movement of the location of the AIPS_ROOT area without necessitating a rebuild of a shareable library *AIPS* installation. It may also facilitate a binary distribution of *AIPS*. Both of these are possible by adding the LIBR directory to the

SunOS environment variable `LD_LIBRARY_PATH`.

References

[1]"Disk usage, build time, and execution time for *AIPS* in SunOS under a variety of compilation modes", Mark Calabretta, Australia Telescope National Facility, 1991/Oct/04, on the *AIPS++* Tools and *AIPS* Bananas E-mail exploders.

[2]"Verification", Mark Calabretta, Australia Telescope National Facility, 1991/Oct/04, on the *AIPS++* Tools E-mail exploder. See also the follow-up messages.