The Port To AViiON*

Russell J. Clark†

Now that you have heard all about the features of the $500/MIPS, scalable 88K-architecture of the AViiON family, you probably want to know about the software. Specifically, you want to know what it would take to run your software on an AViiON system.

The good news is that if you use one of several hundred third-party software products, your application may already be supported – after a year of availability, Data General's AViiON system now hosts a plethora of applications software. And because of DG's compliance with the 88open Binary Compatibility Standard (BCS), the AViiON system now supports hundreds of applications certified by 88open.

But if you want to use your own software, you want to be aware of the issues and benefits of porting it to DG/UX. This article covers some of those topics and helps you determine what it takes to port your software to DG/UX.

1 Planning your Port

When exploring this porting project, it is important to take the same planned approach that you would follow if your were setting out to develop a product from scratch. It will, of course, be a much lesser task to port the application to DG/UX than it was to write it, but the preparation time should be deemed just as important.

As part of that preparation, examine both the source development environment and the target DG/UX environment. Included in these environments are the hardware architecture, operating system, and development tools used to build the application.

Your review of the application's source environment should identify the exact dependencies on that environment. Chances are good that a product developed solely for a single architecture and operating system has inherited some dependencies on that system. Such dependencies include things like data-type size, memory alignment, stack growth direction, vendor-specific library routines, filesystem layout, and communications protocols. For instance, some programs have hard coded dependencies on the size of the "int" data-type. On the 88K RISC processor, an "int" is 32 bits wide. Any code dependent upon a different "int" size will need to be modified. If you are not one of the original developers of the application being ported, try to review dependencies with someone who worked on the initial development.

If the application is currently runing on a UNIX system, find out whether the current system is based primarily on AT&T System V or BSD. Knowing the UNIX standard gives you information about possible implementation dependencies like signal handling, filesystem design, network protocols, and interprocess communication facilities.

After reviewing the source environment, spend some time reviewing the target environment. As part of the DG/UX documentation set, DG publishes a manual entitled "Porting Applications to the DG/UX System." It should be your primary tool when reviewing the features of DG/UX in preparation for a port. It covers the most common porting issues and addresses the differences between DG/UX and several other UNIX derivatives such as AT&T V.3, BSD 4.2, and 386/ix. This document also describes several hardware dependencies that affect software portability. Using this document, you should compare the dependencies you identified in the source environment to the features and characteristics of DG/UX.

If the application source environment is a UNIX system, you will probably find that most of the system requirements are supported by DG/UX. The developers designed DG/UX to be a true merged environment with support included for AT&T SVID, BSD, POSIX, ANSI, and BCS standards. This means that DG/UX can accommodate software developed on several different systems without major changes. In some areas where the standards conflict, such as AT&T and BSD line disciplines, DG/UX supports two different flavors of the programmer's interface. By knowing which interface an application is expecting, a programmer can

*This article appeared in AViiON News, Database Publications, Inc. March/April, 1990.

†Russell J. Clark is a consultant software engineer with DG's Industry Standard Applications group at Research Triangle Park, NC. He has worked extensively with vendors porting applications to DG/UX and is one of the original authors of the DG manual, "Porting Applications to the DG/UX System." Clark may be reached at DG, 62 TW Alexander Drive, RTP,NC, 27709.

select the mode that allows the software to run without modification.

2 High Level Language Support

While the language most commonly used with UNIX based applications is C, DG/UX supports many other high level languages including F77, Pascal, COBOL, PL/1, and BASIC. DG/UX also supports several debuggers including sdb, dbx, gdb, and Mxdb. These languages and tools are available on DG/UX and from several third-party vendors who have ported their products to AViiON. This broad availability of tools makes it easier to port even non-UNIX based software, since it may be possible to use the same compilers on DG/UX that were used on the native system. If your application is currently developed using a third-party language platform, you should research the availability of that platform on DG/UX.

A major portion of code for most applications is in the user interface. To facilitate this section of the port, DG/UX supports several different standard libraries. For system which use the X Window System on a workstation graphics terminal or an X-terminal, DG/UX supports the X11 Release 3 and OSF Motif libraries. Programs written to use X11 on other systems will be able to use the same X library routines on DG/UX. For applications that do not use graphics but instead interface to an asynchronous terminal, DG/UX supports the standard termcap and terminfo library routines for writing portable, terminal independent programs. If you are moving your application from a non-UNIX environment, you will want to consider enhancing your software to use either the X11 windowing interface or the portable terminfo routines. To do this, you will need to modify the software's terminal I/O code to use the routines defined in the X11 or curses libraries. For programs that are not terminal-independent, AViiON workstations and DG terminals provide support for the standard VT100 terminal definition.

As you begin the port to DG/UX, you need to plan how to deal with possible changes to the application source. Most changes that are made when doing a port turn out to be generic fixes that can be rolled back into the original source. An example of this is the addition of a type cast to coerce function arguments into the type expected by the function. Changes that are not generic should be handled with some type of conditional compilation mechanism such as the *#ifdef* directive supported by the C pre-processor and several other languages. You will also want to track your changes with a source code control mechanism such as sccs(1) or rcs(1), both of which are supported on DG/UX. This will allow you to track source changes and roll them into software revisions of the application.

As part of your determination of the porting effort, you should not overlook the testing and documentation of the product in the DG/UX environment. It is important that part of the porting schedule include plans for porting your automated test suites and running them on DG/UX. This should involve peripheral and configuration testing, including installation and execution in a network environment and on diskless workstations. You should also plan to enhance the software documentation with notes on installing and using the application on the DG/UX platform.

3 Some Examples

One example of a possible issue when porting to the AViiON system involves use of the NULL pointer. The NULL pointer is used in the C language to specify an uninitialized pointer. This is prone to misuse because on some machines the NULL pointer actually points to a zero memory value. On AViiON systems, the NULL pointer does not point to zero. The following illustrates misuse of the NULL pointer:

```
if (*p) ...
as a shorthand for:
if (p && *p) ...
```

The first case tests only to see if "p" points to zero. The second case correctly verifies that "p" is a valid pointer before attempting to use it.

A second architecture-specific issue deals with dependencies on the alignment of structures and unions. The AViiON systems running DG/UX conform to the alignment requirements of the 88open Object Compatibility Standard. This specifies that all structures and unions are aligned to the alignment of the most restrictive element. The following structure gives us an example of this:

```
struct doublealigned {
double d1; /* 8 bytes, 8 byte-aligned */
char c1;   /* 1 byte, 1 byte-aligned  */
char c2;   /* 1 byte, 1 byte-aligned  */
char c3;   /* 1 byte, 1 byte-aligned  */
} struct1, struct2;
```

The sum of the element sizes for this structure is 11 bytes. But, because this structure contains a "double" that is eight byte-aligned, the structure itself must be eight byte-aligned and the actual structure size is padded out to 16 bytes. This means that "struct2" is allocated at an offset of 16 from the location of "struct1".

Programs that are written for portability should not make assumptions about the exact alignment of structures in memory.

4 Go Forth and Port

Porting a software application to DG/UX allows you to make it available on the AViiON system and gives you the opportunity to enhance the application to use standard DG/UX features like X11. DG/UX provides integrated support of the UNIX standards to simplify the process of porting. DG provides the developer with tools and the commitment to support a wide range of software development products for the AViiON. Added together, you have what you need to outline the porting effort, complete the port, and make your application software a successful product on the AViiON system.