North American Response Centers

HP 3000 APPLICATION NOTE #17

OPTIMIZING VPLUS UTILIZATION

November 1, 1986
Document P/N 5958-5824/2644

HP 3000 APPLICATION NOTES *are published by the North American Response Centers twice a month and distributed with the Software Status Bulletin. These notes address topics, where the volume of calls received at the Centers indicates a need for addition to or consolidation of information available through HP support services. You may obtain previous notes (single copies only, please) by returning the attached Reader Comment Sheet listing their numbers.*

Note #	*Published*	*Topic*
1	*2/21/85*	*HP 3000 Printer Configuration Guide (superceeded by note #4)*
2	*10/15/85*	*Terminal Types for HP 3000 HPIB Computers (superceeded by note #13)*
3	*4/01/86*	*HP 3000 Plotter Configuration Guide*
4	*4/15/86*	*HP 3000 Printer Configuration Guide - Revised*
5	*5/01/86*	*MPE System Logfile Record Formats*
6	*5/15/86*	*HP 3000 Stack Operation*
7	*6/01/86*	*Cobol II/3000 Programs: Tracing Illegal Data*
8	*6/15/86*	*KSAM Topics: Cobol's Index I/O; File Data Integrity*
9	*7/01/86*	*Port Failures, Terminal Hangs, TERMDSM*
10	*7/15/86*	*Serial Printers - Configuration, Cabling, Muxes*
11	*8/01/86*	*System Configuration or System Table Related Errors*
12	*8/15/86*	*Pascal/3000 - Using Dynamic Variables*
13	*9/01/86*	*Terminal Types for HP 3000 HPIB Computers - Revised*
14	*9/15/86*	*Laser Printers - A Software and Hardware Overview*
15	*10/01/86*	*Fortran Language Considerations - A Guide to Common Problems*
16	*10/15/86*	*IMAGE: Updating to TurboImage & Improving Data Base Loads*

Optimizing VPLUS Utilization

Introduction

VPLUS/3000, Hewlett-Packard's standard terminal management software, finds wide use in applications ranging from manufacturing control to financial transaction processing. Offered as part of the HP 3000 Fundamental Operating Software (FOS), VPLUS provides programmers and designers access to the features of HP block-mode terminals, effectively insulating programs from the details of data communication and terminal control.

The Response Centers receive many questions regarding VPLUS optimization and program performance. This Note will address several topics related to these issues. There are three major areas where even small amounts of effort from the programmer can make significant differences in VPLUS performance:

1. Effective forms design
2. Stack use by VPLUS applications
3. Programs' utilization of forms and forms files

A proper balancing of effort in all three areas can help you make significant strides towards efficient use of VPLUS.

A working knowledge of FORMSPEC and "general principles" of forms file management is assumed, as is knowledge of the major VPLUS routines (e.g., VGETNEXTFORM, VSHOWFORM, VOPENTERM).

Definitions of Terms

In this Note we shall discuss what changes need to be made to the **largest form** in the forms file targeted as being troublesome. There are many valid definitions of "largest form". One is "that form in which the largest amount of data is transferred between the program and terminal". Another is "that form which contains the largest number of fields". A third is "the form which, when displayed on the terminal, uses up the most space on the screen". When the term "largest" is used, it will be further defined for its context.

There are also three acronyms used througout this paper. **FST** stands for **Field Status Table**, and is used by VPLUS to hold information about each field on the current form. **DBUF** and **IBUF** refer to the Data BUFfer and Input BUFfer used internally by VPLUS. Any program (through calls to VPUTBUFFER and VGETBUFFER) can manipulate the DBUF, while the IBUF is a reordering of the DBUF and is used internally by the VPLUS intrinsics for data transfer to and from the terminal.

Effective Forms Design

One of the easiest and least painful ways to help the performance of programs using VPLUS is to use foresight in designing the forms to be used by those programs. A few simple tricks applied in advance can save a lot of debugging and redesign later in the process.

One of the first things to keep in mind when designing forms is how much data the user needs to access at one time. Assuming there is enough information to fill the screen, does one want to show it all at once (taking a risk of overwhelming the user with too much data in one sitting and making data entry more

difficult due to a crowded and confusing screen), or put it out in small doses (possibly boring the user as they move through multiple forms to see all the data and making data entry that much longer)? Here, the old "space/time" tradeoff comes into play: by saving the space needed for the displaying of data (in the multiple-form scheme), more time is needed for showing all screens, while by displaying all the data at once, more space is needed but all necessary information is shown in one pass. One advantage to the mutiple-form method is that a user could stop viewing the screens of data once the needed data is displayed (assuming such an escape mechanism is designed into the program).

If you decide to use the one-screen approach, there are many things that you can do to avoid problems before they happen. One is to keep the amount of data (in number of bytes) equal between forms in the forms file. This will enable VPLUS to get the greatest amount of utilization from the DBUF and IBUF, since there would then be only a few instances where just portions of each were needed for I/O operations. Another is to keep the number of fields as equal as possible between forms. This will allow VPLUS to get the greatest amount of utilization from the FST and other internal tables connected with field manipulation on each form.

If some forms in the forms file are composed of many small fields, while there is also a form with one or two VERY large fields (for example, a program where general information is needed about a person, but specific comments may be made), you might consider changing the form with those large fields into a form with smaller fields. A good example of this is one where the entire screen is an unprotected field for text entry. That form would only need one entry from the FST (since there's only one field), but would require 80*24=1920 characters for the DBUF and IBUF. Changing this form to one which repeats, appends to itself, and is made of only one 80-character long field will allow the same functionality of the former form (with the exception of having to press ENTER at the end of each line), and actually be more flexible. This flexibility comes from the fact that now the user can input as many lines of text as are needed, whereas before they were restricted to 24 lines.

Similar things can be done to the form which has the largest number of fields. If the fields are some sort of tabular data, making another repeat/append-type form set can help. In this set, the first form could contain header information, while the second is a self-repeating and self-appending form containing the fields for each detail line.

Another trick to speed up the display of forms is the effective use of **forms families**. Forms families are sets of forms which share a common screen definitiion, but different field characteristics. Any time a forms family member is to be displayed, VPLUS checks to see if the last form displayed was another member of that same family. If it was, then instead of transmitting the entire form definition, VPLUS sends only those escape sequences needed to alter those fields that differ between the forms (e.g., from unprotected to display-only, from half-intensity inverse to underlined blinking, etc.). If all of the information to be displayed is similar in format, it's possible to use forms families to make minor changes in the display without repainting the entire screen. You can also use the VCHANGEFIELD intrinsic (released with version B.04.17 of VPLUS) to accomplish the same ends.

If your program accesses several forms files, you should seriously consider combining them into a single file. Not only does this save stack space (since VPLUS must maintain seperate control information -- including the DBUF and IBUF -- on the program's data stack for each open forms file), but it helps make maintenance of the forms much simpler. If the forms are scattered among multiple forms files, the forms files must be closed and reopened each time they are used, and much of the time used by the program will be tied up in this moving between forms files, rather than with the actual processing of the data.

Stack Use by VPLUS Applications

Some of what will be talked about in this section will contradict what was discussed in the prior section. Once again, it's the computer world's old "space/time" problem: optimizing the data space required for a program will most likely increase the execution time of the resulting application.

If a VPLUS application aborts with any of the typical indicators of a stack problem (STACK OVERFLOW; a call to VOPENFORMF fails with error codes 40, 41, 61, 62, 68, or 69), the *first* thing to do is to compile the forms file into a **fast forms file**. One part of the space reserved on a data stack is an area which contains the directory of all records in a forms file. By all records, we mean ALL records -- **source** records containing the raw, input form definitions, **code** records containing the escape sequences needed to paint those forms on the screen, as well as FORMSPEC-internal records needed to manipulate those forms. What goes into a fast forms file is only a few control records (to designate it as a fast forms file) and code records -- but *no* source records. Since there are fewer records in the file, the directory is smaller and less space is needed on the stack for it. As a result, you save a minimum of 800 words *per forms file* simply by using fast forms files. (We should also note that, in addition to the space savings, fast forms files are so named because they require much less disc I/O. Some benchmarks have shown as much as a 50% reducition in forms file I/Os when a fast forms file is used instead of a slow forms file.)

Other methods of space optimization may involve taking out some of the features incorporated in the original design of the application. **Local Forms Storage** (LFS) is very useful for minimizing data communications overhead involved in VPLUS, since by its use a forms definition may be sent to a terminal *once*, but there is a price to be paid on the stack. If LFS is being used, a directory of form names already downloaded to the terminal is kept on the stack to enable VPLUS to quickly discover whether a form has already been downloaded. Each entry in this table is 16 words long. Therefore, if LFS is enabled for a large number of local forms (via VOPENFORMF) disabling it will save 16*n words, where n is that number of "local" forms. (Local forms storage will be discussed in greater detail in a later section.)

One of the ergonomic features discussed in the forms design section comes into play now. Remember, VPLUS keeps two copies of the data buffers on your stack (the DBUF and IBUF), in addition to any buffer space allocated by your program. Minimizing, or standardizing, the amount of data that is transferred for each form will help control the sizes of those buffers. For example, suppose most forms in the forms file contain 20 2-character long fields, while one form contains 40 6-character long fields. Whereas most of the forms would only require the DBUF and IBUF to be 40 characters long (20 fields times 2 characters per field), VPLUS will allocate 240 characters for both buffers (40 times 6). Changing that 40-field form to a self-repeat/append containing 4 fields brings that form's requirements down to buffers 24 characters long. Notice that by making that change, VPLUS's stack requirements have just decreased by 432 words (216 words each for the DBUF and IBUF).

In the same vein, if most of the forms have close to the same number of fields and almost the same data buffer requirements, but one form has MANY more fields, a simple splitting of the form into either a self-repeatinf or multiple forms (if dealing with REPEAT/APPEND is not desirable) will decrease the space needed for the FST, as well as the other internal tables needed to manipulate fields.

If stack space is a problem, and function key labels (FKLs) are enabled, their elimination could save up to approximately 800 words. When enabled, space is reserved for two copies of FKLs: one copy for global FKLs to be used in the forms file, and another to be used for FKLs local to each form. For each copy, there are 16 bytes of storage reserved *per key label* for the messages associated with each, as well as information used by VPLUS to tell if FKL definitions have changed (via calls to the VSETKEYLABELS or VSETKEYLABEL routines) and if those changed values have yet been displayed.

Another feature of VPLUS that occupies a large amount of stack space is the use of INIT, FIELD, and FINISH phase edit specifications. FORMSPEC compiles these into object code meaningful only to the appropriate VPLUS routines (VINITFORM, VFIELDEDITS, VFINISHFORM) which act upon the copy of

the data brought into the IBUF by VREADFIELDS. Since each form can have up to 12000 words of code (which is treated on the stack as part of the form definition), either eliminating or simplifying edit specifications can save immense amounts of stack space.

All of these suggestions used in combination can save a program on the order of hundreds, if not thousands, of words of stack space. In most cases, simply using a fast forms file in combination with eliminating LFS will return more than enough stack space to allow processing to commence.

Programs' Utilization of Forms and Forms Files

Many programs lose processing time because in their original design no thought was given to the fact that a forms file is a "data file" to a program, as much as a KSAM file or IMAGE dataset. Every time a new form is referenced in a program, that form's definition must be retrieved from the forms file (involving at least one and possibly multiple disc I/O's). If FKLs are enabled in the program and defined for the form on the forms file, those FKL definitions must be retrieved.

Here, again judicious use of forms families can make a tremendous difference in execution speeds and throughput time. After 1 member of a forms family is displayed, each family member is retrieved from their forms file, but what actually happens is that only the fields that changed are repainted. In this case, we haven't saved much time when dealing with fast I/O devices (i.e., the discs), but we have saved a very large amount of time dealing with slower devices (i.e., the terminal).

If execution speed is a concern (how fast forms get painted, time needed for transmitting form definitions to the terminal, etc.), there are a few things to keep in mind. One is the use of Local Forms Storage on the 2624B, 2394A, and 2626A terminals. This feature allows you to download a form definition *once* to the terminal; every time that form is used thereafter, its definition is simply retrieved from terminal memory rather than being retransmitted from the computer. The 2626 terminal can store up to 4 form definitions, while the 2624 and 2394 can store up to 256 (this limit of 256 is a function of the complexity and size of the forms being stored, and might decrease as the size and complexity of the forms increases). If a form to be displayed on the terminal screen is not currently in terminal memory when VSHOWFORM is done it can be concurrently downloaded, and the form least recently accessed will be purged from the terminal memory. This brings up an interesting point on application (rather than forms file) design: inefficient use of forms (e.g., bouncing between many different forms) will significantly degrade the performance of the application as a whole.

Another terminal-specific feature which can be used is local edits, which provide high level access to the special features of the 2624B terminal. By defining a CONFIG phase in the processing specifications area of the Field Menu (in FORMSPEC) and selecting the `HP262X Family` in the Terminal Selection Menu, you may choose appropriate combinations of local edits for any or all fields in a form. The escape sequences needed to enable these edits are embedded by FORMSPEC into the compiled screen design and loaded into the terminal by VSHOWFORM when the form is displayed. The local edits allow the fields *in the terminal* (as opposed to in the form definition stored on the data stack of the program) to be designated as having certain characteristics, such as alphabetic only or numeric only. A word of caution: if you use forms families and local edits, be aware that edits for the *first* family member displayed in a sequence will be used, even if you switch to a different family member. This is due to the optimization VPLUS uses when displaying a different member of the same forms family.

In addition to LFS, creative use of repeating and appending forms can cause a significant speedup in processing time. If the application will generally ask for a given set of data (e.g., sales orders, student information, inventory, etc.), repeating and/or self-appending forms can be used, which will give two benefits. The first is that all terminal input and output is being done repeatedly through one form, the definition for which is read ONCE from the forms file then repeatedly reused. The second is that the user only has to learn one (or possibly two) forms for data input, simplifying their interaction with the application.