Overview of Optimizing (On-Line and Batch)

Robert M. Green
Robelle Consulting Ltd.

SUMMARY

The performance of many HP3000 installations can often be improved significantly. There are general principles for delivering better response time to on-line users, and other principles to speed execution of production batch jobs. As long as users continue to consumer the extra horsepower of new HP3000 models by loading them with new applications, there will continue to be a need for optimizing knowledge and tools. And, if interest rates remain at current levels, many managers may not be able to upgrade to faster computers as soon as they would like.

CONTENTS

SECTION 1 HOW TO IMPROVE ON-LINE RESPONSE TIME

I have identified five general principles which help in optimizing the performance of on-line programs:

- Make each disc access count.
- Maximize the value of each "transaction."
- Minimize the run-time program "size."
- Avoid constant demands for execution.
- Optimize for the common events.

On a systems programming project, such as a data entry package or a text editor, you should be able to apply all five of these principles with good results. That is because systems software usually deals with MPE directly and most of the sources of slow response are under your control. Applications software, on the other hand, usually depends heavily upon data management sub-systems such as IMAGE and V/3000. The optimizing principles proposed here may not be as easy to apply when so many of the causes of slow response are beyond your control. However, there are still many ways in which you can apply the guidelines to application systems (monitoring program size, designing your database and laying out your CRT screens). Relying upon standard software not only increases your programmer productivity, it also provides an unexpected bonus: any improvements that the vendor makes in the data management tools will immediately improve the efficiency of your entire application system, with no re-programming or explicit "optimizing" on your part.

I. A. Make Each Disc Access Acount

Disc accesses are the most critical resource on the HP3000. The system is capable of performing about 30 disc transfers per second, and they must be shared among many competing "consumers." (This can increase to 58 per second under the best circumstances, and can degrade to 24 per second when randomly accessing a large file.) MPE IV can double the maximum disc throughput for multi-spindle systems by doing "look-ahead" seeks, but only for the Series II/Series III, not the Series 30/33/44.

The available disc accesses will be "spent" on several tasks:

- Virtual memory management (i.e., swapping).
- MPE housekeeping (logon, logoff, program load, etc.).
- Lineprinter spooling.
- Accesses to disc files and databases by user programs (the final payoff).

If the disc accesses are used up by overhead operations, there will not be sufficient left to provide quick response to on-line user transactions. Some examples of operations that consume disc accesses on the HP3000 are:

- Increasing the number of keys in a detail dataset, thus causing IMAGE to access an extra master dataset on each DBPUT. Also, making a field a key value means that a DBDELETE/DBPUT is required to change it (which is 10 times slower than a DBUPDATE).
- Increasing the program data stack by 5000 words, thus causing the MPE memory manager to perform extra, swapping disc accesses to find room in memory for the larger stack.
- Improperly segmenting the code of an active program, causing many absence traps to the memory manager to bring the code segments into main memory.
- Constantly logging on and off to switch accounts.
- Defining a database with a BLOCKMAX value of 2000 words, thus limiting IMAGE to about 13 data buffers in the extra data segment that is shared by all users of that database. With such a small number of buffers, there can be frequent buffer "thrashing." This effectively eliminates the benefits of record buffering for all users of the database, and greatly increases disc accessing.

Much of the remainder of this document is devoted to methods of "saving the precious resource — disc accesses."

I. B. Maximize the Value of Each "Transaction"

This principle used to read, "Maximize the Value of Each Terminal Read," but I have generalized it to "transaction" to take into account the prevalence of V/3000, DS, MTS and other "communications" tools. In the terms of MPE IV, a "transaction" begins when the user hits the 'return' key (or Enter) and ends when the user can type input characters again. This includes the time needed to read the fields from the terminal (or from another HP3000), to validate them, perform database lookups and updates, format and print the results, and issue the next "read" request.

Each time a program reads from the terminal, MPE suspends it and may swap it out of memory. When the operator hits the 'return' key, the input operation is terminated, and MPE must dispatch the user process again. If MPE has overlaid parts of the process, they must be swapped back into main memory again. Due to the overhead needed to dispatch a process, a process should get as much work done as possible before it suspends for the next terminal input.

The simplest way to program data entry applications is to prompt for and accept only one field of data at a time. This is also the least efficient way to do it. Since there is an unpredictable "pause" every time the user hits 'return' (depending upon the system load at the moment), consistently fast response cannot be guaranteed. The resulting delays are irritating to operators. They can never work up any input speed, because they never know when the computer is ready for the next input line. If response time and throughput are the only considerations, it is always preferable to keep the operator typing as long as possible before hitting the 'return' key. Multiple transactions per line should be allowed, with suitable separators, and multiple lines without a 'return' should be allowed. If you are using V/3000, the same principles applies: each high-volume transaction should be self-contained on a single form, rather than spread out over several different forms.

I. C. Minimize the Run-Time Program "Size"

The HP3000 is an ideal machine for optimizing because of the many hardware features available at run-time to minimize the effective size of the program. Even large application systems can be organized to consume only a small amount of main memory at any one time. Each executing process on the HP3000 consists of a single data segment called the "stack," several extra data segments for system storage, such as file buffers, and up to 63 code segments. All segments (code and data) are variable-length and can be swapped between disc and main memory.

Program code which is not logically segmented makes it harder for the memory manager to do its job, causing disc accesses to be used for unnecessary swaps. Proper code segmentation is a complex topic (more like an art than a science), but here is a simplified training course: write modular code; don't segment until you have 4000 words of code; isolate modules that seldom run; isolate modules that often run; aim for 4000 words per segment, and group modules by "time" rather than "function;" if you reach 63 segments, increase segment size, but keep active segments smaller than inactive ones.

Although every process is always executing in some code segment, the code segment does not belong to the process, because a single copy of the code is used by all processes that need it. Since code is shared, it does not increase as the number of users running a given program increases. Most of your optmizing should be directed to the data areas (which are duplicated for each user). A 3000 can provide good response to more terminals if most data segments are kept to a modest size (5000 to 10,000 words). To keep stacks small, declare most data variables "local" to each module (DYNAMIC in COBOL), and only use "global" storage (the mainline) for buffers and control values needed by all modules. Dynamic local storage is allocated on the top of the stack when the subroutine is entered, and is released automatically when the subroutine is left. This means that if the main program calls three large subroutines in succession, they all reuse the same space in the stack. The stack need only be large enough for the deepest nesting situation. By inserting explicit calls to the ZSIZE intrinsic, you can further reduce the average stack size of your program.

You can also minimize stack size by ensuring that constant data items (such as error messages and screen displays) are stored in code segments rather than in the data stack. Since constants are never modified, there is no logical reason that they should reside permanently in the data stack. By moving them to the code segment, one copy of them can be shared by all users running the program. In SPL, this is done by including =PB in a local array declaration or MOVEing a literal string into a buffer. In COBOL, constants can be moved to the code segment by DISPLAYing literal strings in place of declared data items. In FORTRAN, both FORMAT statements and DISPLAYed literals are stored in the code.

A frequently overlooked component of program "size" is the effect of calls to system subroutines (IMAGE, V/3000, etc.). These routines execute on the caller's stack, and the work they do is "charged" to the caller. In many simple on-line applications (dataset maintenance program, for example), 90% of the program's time and over 50% of the stack space will be controlled by IMAGE and V/3000. You should be aware of the likely impact of the calls that you make. Do you know how many disc accesses a particular call to DBPUT is going to consume? As an example of how ignoring the "extended size" of a program can impact response time, consider the following case:

An application with many functions can be implemented with one of two different strategies. The first, and simplest, strategy is to code the functions as separate programs and RUN them via a UDC (or CREATE them as son processes from a MENU program). Each function opens the databases (and forms-file, etc.) when you RUN it, and closes them before stopping.

The second strategy is to code each function as a subprogram that is passed in the previously opened databases (and forms-file, etc.) as a parameter from a mainline driver program. If the application requires frequent movement from function to function (performing only a few transactions in each function), the "process" strategy will be up to 100 times slower than the "subprogram" strategy. The resources required to RUN the programs, open the databases, close the databases, and perform other "overhead" operations will completely swamp the resources needed to perform the actual transactions.

I. D. Avoid Constant Demands for Execution

The HP3000 is a multi-programming, virtual-memory machine that depends for its effectiveness on a suitable mix of processes to execute. The physical size of code and data segments is only one factor in this "mix." The "size" of a program is not just the sum of its segment sizes; it is the product that results from multiplying physical size by the frequency and duration of demands for memory residence (i.e., how often, and for how long, the program executes). A given 3000 can support many more terminals if each one executes for one second every 30 seconds, rather than 60 seconds every two minutes. Each additional terminal that demands continuous execution (in high priority) makes it harder for MPE to respond quickly to the other terminals.

Here are some examples of the kind of operation that can destroy response time, if performed in high priority:

- EDIT/3000, a GATHER ALL of a 3000-line source file.
- V/3000, forms-file compiles done on four terminals at once.
- QUERY, a serial read of 100,000 records (or any application program that must read an entire dataset, because the required access path is not provided in the database).
- SORT, a sort of 50,000 records.
- COBOL, compiles done on four terminals at once.

You should first try to find a way to avoid these operations entirely. (Can you use QEDIT instead of EDIT/3000? Would a new search item in a dataset eliminate many serial searches, or could you use SUPRTOOL to reduce the search time? Are you compiling programs just to get a clean listing?)

After you have eliminated all of the "bad" operations that you can, the remainder should be banished to batch jobs that execute in lower priority (this works better in MPE IV than III). Since jobs can be "streamed" dynamically by programs, the on-line user can still request the high-overhead operations, but the system fulfills the request when it has the time. The major advantage of batch jobs is that they allow you to control the number of "bad" tasks that can run concurrently (set the JOB LIMIT to 1 for best terminal response).

I. E. Optimize for the Common Events

In any application where there is a large variation between the minimum and maximum load that a transaction can create, the program should be optimized around the most common size of transaction. If a program consists of 20 on-line functions, it is likely that four of them will be most frequently used. If so, your efforts should be directed toward optimizing these four functions; the other functions can be left as is. Because the HP3000 has code segmentation and dynamic stack allocation, it is possible for an efficient program to contain many inefficient modules, as long as these modules are seldom invoked.

Since MPE will be executing a great deal of the time, you should become competent at general system tuning. Learn to use TUNER, IOSTAT, and SYSINFO (and the new :TUNE command in MPE IV). Any improvement in the efficiency of the MPE "kernel" will improve the response time of all users.

You do not have infinite people-resources for optimizing, so you must focus your attention on the factors that will actually make a difference. There is no point in optimizing a program that is seldom run. The MPE logging facility collects a number of useful statistics that can be used to identify the commonly accessed programs and files on your system. Learn to use the contributed programs FILERPT and LOGDB (Orlando Swap). If you are using IMAGE transaction logging, the DBAUDIT/Robelle program will give you transaction totals by database, dataset, program, and user (total puts, deletes, updates, and opens). Such statistics help in isolating areas of concern.

You can optimize application programs around the average chain length for detail dataset paths (the contributed program DBLOADNG will give you this information). Suppose you need to process chains of entries from an IMAGE dataset. If your program only provides data buffers for a single entry, you will have to re-read each entry on the chain each time you need it (extra disc I/O!!). Or, if you provide room for the maximum chain length, the data stack will be larger than needed most of the time (the maximum chain length is often much larger than the average). The larger data stack may cause the system to overload, eliminating the benefits of keeping the records in your stack. You should provide space in the stack for slightly more than the average number of entries expected. This will optimize for the common event.

SECTION II
ON-LINE OPTIMIZING EXAMPLE: QEDIT

QEDIT is a text editor for the HP3000 that was developed by Robelle Consulting Ltd. The primary objective of QEDIT is to provide the fastest editing with the minimum system load. Other objectives include conservation of disc space, similarity to EDIT/3000 in command syntax, ability to recover the workfile following a system crash or program abort, and increased programmer productivity.

QEDIT is an alternative to a hardware upgrade for users who are doing program development on the same HP3000 that they are trying to use for on-line production. Every optimizing paper in recent years by an HP performance specialist has recommended avoiding EDIT/3000. They usually recommend the "textfile-masterfile" approach to program development. (You do not actually edit your source program; instead, you create a small "textfile" containing only the changes to your "masterfile," then merge the two files together at compile-time). QEDIT allows you to have "real" editing on your HP3000, with less overhead than the "textfile masterfile" method, and still give good response time to your end-user terminals.

II. A. QEDIT and "Disc Accesses"

In order to reduce disc accesses, QEDIT eliminates the overheads of the TEXT, KEEP and GATHER ALL commands of EDIT/3000. These three operations have the most drastic impact upon the response time of the other users. QEDIT attacks the problem of KEEPs by providing an interface library that fools the HP compilers into thinking that a QEDIT workfile is really a "card image" file. As a result, it is never necessary to KEEP a workfile before compiling it. Since KEEPs are rarely used, most TEXTs are eliminated. The LIST command was given the ability to display any file (e.g., /LIST DBRPT1.SOURCE), so that a TEXT would not be required just to look at a file. TEXT is only needed when you want to make a backup or duplicate copy of an existing file. Since most users choose to maintain their source code in QEDIT workfiles (they use less disc space), the TEXTing of workfiles is optimized (by using NOBUF, multi-record access) to be four to seven times faster than a normal TEXT of a card-image file. The GATHER ALL operation is slow because it makes a copy of the entire workfile in another file. QEDIT renumbers up to 12 times faster by doing without the file copy.

Disc accesses during interactive editing (add, delete, change, etc.) are minimized by packing as many contiguous lines as possible into each disc block. Leading and trailing blanks are removed from lines to save space. The resulting workfile is seldom over 50% of the size of a normal KEEP file, or 25% of the size of an EDIT/3000 K-file (workfile). Most QEDIT users maintain their source programs in workfile form, since this saves disc space, simplifies operations (there need be only one copy of each version of a source program), and provides optimum on-line performance.

QEDIT always accesses its workfile in NOBUF mode, and buffers all new lines in the data stack until a block is full before writing to the disc. Wherever possible in the coding of QEDIT, unnecessary disc transfers have been eliminated. For example, the workfile maintains only forward direction linkage pointers, which reduce the amount of disc I/O substantially. Results of a logging test show that reducing the size of the workfile and eliminating the need for TEXT/KEEP reduce disc accesses and CPU time by 70-90%.

II. B. QEDIT and "Transaction Value"

Like EDIT/3000, QEDIT allows either a single command per line (/ADD), or several commands on a line, separated by semi-colons (/LIST 5/10;M 6;D 5). The principle of maximizing transaction value has been applied with good results to the MODIFY command. In

EDIT/3000, several interactions may be needed to modify a line to your satisfaction. QEDIT allows you to perform as many character edits as you like on each transaction; many users can perform all of their changes in a single pass. For complex character editing, such as diagrams, version 3.0 of QEDIT will provide "visual" editing in block-mode.

II. C. QEDIT and "Program Size"

QEDIT is a comletely new program, written in highly structured and modular SPL. The code is carefully segmented, based on the knowledge of which SPL procedures are used together and most frequently. Only two code segments need be resident for basic editing, and the most common function (adding new lines) can be accomplished with only a single code segment present.

QEDIT uses a modest data stack (3200 words) and no extra data segments. The stack expands for certain commands (especially the MPE :HELP command), but QEDIT contracts it back to a normal size after these infrequent commands are done. All error messages are contained in the code, isolated in a separate code segment that need not be resident if you make no errors.

Use of CPU time is th eother dimension to program "size." QEDIT is written in efficient SPL and consumes only a small amount of CPU time (compared with the COBOL compiler, or even EDIT/3000). Because QEDIT does its own internal blocking and deblocking of records, it can reduce the CPU time used in the ile system by opening files with NOBUF/MR access.

II. D. QEDIT and "Constant Demands"

Most QEDIT commands are so fast that they are over before a serious strain has been placed on the host machine. For example, a 2000-line source program can be searched for a string in four seconds. For those operations that still are too much load, QEDIT provides the ability to switch priority subqueues dynamically. In fact, the system manager can dictate a maximum priority for compiles and other operations that cause heavy system load.

II. E. QEDIT and "Common Events"

The design of QEDIT is based on the fact that program editing is not completely random. When a programmer changes line 250, he is more likely to require access to lines 245 through 265 next, than to lines 670 through 710. This observation dictated the design of the indexing scheme for the QEDIT workfile. There are many examples of optimizing for the most common events in QEDIT:

- Each block of a QEDIT workfile holds a "screenful" of lines, with leading and trailing blanks eliminated.
- QEDIT has built-in commands to compile, PREP and RUN (since these functions are frequently used by programmers).
- QEDIT has a fast /SET RENUM command (it can renumber 600 lines per second), instead of a slow GATHER command.
- QEDIT can TEXT a workfile much faster than a KEEP file (since most text will end up in QEDIT workfiles).
- QEDIT can "undo" the DELETE command (because programmers are always deleting the wrong lines).

II. F. Results of Applying the Principles to QEDIT

In less than seven seconds, QEDIT can text 1000 lines, renumber them, and search for a string. Commands are 80% to 1200% faster than EDIT/3000, program size is cut in half, and disc I/O and CPU time are reduced by up to 90%. There are now more than 350 computers with QEDIT installed, in all parts of the world. Recently, we asked the QEDIT users what they would tell another user about QEDIT. Here are some of their answers:

> "If he's doing program development, he needs QEDIT." (Gerald Lewis, Applied Analysis, Inc.)
>
> "Would not live without it. $INCLUDEs in FORTRAN; one file or dataset per include-file." (Larry Simonsen, Valtek, Inc.)
>
> "Fantastic product." (Lewis Patterson, Birmingham-Southern College)
>
> "Buy it. The productivity advantages are tremendous and don't cost anything in machine load. The disc savings in a large (13 programmers) shop will pay for it." (Jim Dowling, Bose Corp.)
>
> "It's great. We usually get into QEDIT and just stay there for a whole session. Compiles and PREPs are very easy. I really like FIND, LIST, and BEFORE commands. QEDIT is very fast. It is great for programmers." (Larry Van Sickle, Cole & Van Sickle)
>
> "It's a tremendous tool and should be used by any medium-sized shop. I use it to produce an index of all source or job streams for an account." (Vaughn Daines, Deseret Mutual Benefit Assoc.)
>
> "QEDIT is the best editor I've used on the market. It makes a programmer extremely efficient and productive. In rewriting an existing system completely, the on-line compile, flexible commands, and savings of disc space all contributed to bringing the system up very rapidly." (Glenn Yokoshima, HP Corvallis)
>
> "Excellent product. Increases programmer productivity dramatically (morale too!)." (David T. Black, The John Henry Company)
>
> "FAST, convenient. No need to TEXT and KEEP. Somewhat dangerous for novice, because changes are made directly. [It worked well for us in] conversion of SPSS, BMDP, and other statistical packages to the

HP3000." (Khursh Ahmed, McMaster University)

"If you are writing a lot of programs, you should get QEDIT. It is much easier than EDITOR for this purpose. Program source files demand complex editing capabilities, which QEDIT has. I shudder to think of having to work on a 4000-statement SPL source using EDITOR rather than QEDIT." (Bud Beamguard, Merchandising Methods)

"Excellent product. Anyone using the HP editor more than 6 times per day (or more than 1 hour/day average) should not be without QEDIT!" (T. Larson, N. J. McAllister and Associates Ltd.)

"Easier to use than HP editor and much more efficient. I do not have to leave QEDIT to RUN, PREP." (Myron Murray, Northwest Nazarene College)

"Takes a great load off the mind (i.e., the "electronic brain"). There have been occasions when heavy editing would have killed our system if we had been using EDITOR." (Mike Millard, Okanagan Helicopters Ltd.)

"Very good product — works well in development environment. Compilation of source programs without leaving QEDIT is very nice for debugging." (David Edmunds, Quasar Systems Ltd.)

"Use it. It is so much better than HP editor that there is no comparison." (Ilmar Laasi, TXL Corp.)

"Fast text editor." (F. X. O'Sullivan, Foot-Joy, Inc.)

"In one word. Fantastic." (Tracy Koop, Systech, Inc.)

"Superb tool. Far better than EDIT/3000. Also, information about HP3000 that is supplied gratis is very useful." (James McDaniel, The UCS Group Ltd.)

"I would highly recommend it over EDIT/3000. In benchmarks and actual use, it has proven to be much less load on the computer. In a University environment, we have many students and faculty editing programs at one time. QEDIT allows us to run with a high session limit and still get decent batch turnaround." (Dan Abts, University of Wisconsin — La Crosse)

"QEDIT is an excellent product for the price, and is one of the easiest ways to increase programmer productivity. The LIST command has been invaluable for cross-referencing data items in COBOL source programs." (Mark Miller, Diversified Computer Systems of Colorado)

"Absolutely. QEDIT has allowed us to control the development of systems (requiring off-line compiles, audit trails for source modifications) while actually increasing programmer productivity." (Jean Robinson, Leaseway Information Systems, Inc.)

"Get it! It's great. Cheap at twice the price." (Willian Taylor, Aviation Power Supply, Inc.)

"QEDIT is THE ONLY text editor that you should use in a development environment." (Craig T. Hall, Info-tronic Systems, Inc.)

"Much better than HP's editor, well supported, well documented and continually improving. An excellent product. We activate QEDIT from our job file generator and activate SPOOK from QEDIT for editing and testing output and job streams." (Patrick Hurley, Port of Vancouver)

"Excellent — can do more than Editor, faster, and saves disc space. In searching for a specific literal, QEDIT finds them all in one command [e.g., LIST "literal"]." (Larry Penrod, Datafax Computer Services Ltd.)

"We could probably not operate if QEDIT were not available." (Winston Kriger, Houston Instruments)

"Buy it, or another computer (a second HP3000, of course)" (John Beckett, Southern Missionary College)

"Best software package I've bought for our shop." (James Runde, Furman University)

SECTION III
HOW TO INCREASE BATCH THROUGHPUT

By a "batch job" I mean a large, high-volume, long-running task, such as a month-end payroll or financial report. Why is there any problem with this type of task? Because the batch job is only a poor, neglected cousin of the on-line session. "On-line" is "with it," new, Silicon Valley, exciting; "batch" is old, ordinary, IBM, and boring. The best people and most of the development resources have been dedicated to improving the on-line attributes of the HP3000. The result is predictable: batch jobs are beginning to clog many HP3000 processors. The overnight jobs are not completing overnight and the month-end jobs seem never to complete.

The methods for maximizing the throughput of a single batch job are not the same as for maximizing the response time of a large number of on-line users. The biggest difference: for an on-line application, it is seldom economical to optimize CPU usage. There isn't enough repetition to amount to much CPU time. But, a batch process may repeat a given section of code 100,000 or a million times. CPU time matters.

I have identified five general principles for increasing batch throughput. Not surprisingly, they differ significantly from the principles used to improve on-line response time:

- Bypass Inefficient Code (CPU hogs).
- Transfer More Information Per Disc Access.
- Increase Program Size to Save Disc Accesses.

- Remove Structure to Save Unneeded Disc Accesses.
- Add Structure for Frequent Events.

For each optimizing principle, there are three different tactics you can apply, with three levels of complexity and cost:

- Changes in the Data Storage (simplest and cheapest, since no programming changes are needed).
- Simple Coding Changes (still inexpensive, since these are "mechanical" changes which do not require re-thinking of the entire application).
- Changes to the Application Logic (the most complex and expensive, since the entire application may have to be re-designed).

III. A. Bypass Inefficient Code (CPU hogs)

Elimination of inefficient code is the simplest way to produce big throughput improvements, assuming that you can find any code to eliminate that is inefficient (or more general-purpose than needed).

For a number of reasons, IMAGE is usually more efficient than KSAM as a data management method. If you don't need "indexed sequential" as your primary access method, convert from KSAM files to IMAGE datasets. Or, if you don't need "keyed" access to the data, convert all the way from a data management subsystem to an MPE flat file, and use sequential searches. The more powerful the data access method, the more CPU time is required to maintain it.

Bypassing inefficient code is simply a matter of recoding parts of programs to substitute an efficient alternative for an existing method that is known to have poor performance. For example, the MPE file system is CPU-bound when handling buffered files, so converting to NOBUF access will save considerable CPU time (you transfer blocks and handle your own records). In IMAGE, use the "*" or "@" field list instead of a list of field names. In COBOL, re-compile your COBOL68 programs with the COBOL-II compiler and they will run faster. The FORTRAN formatter is a notorious "CPU hog"; either bypass it completely or learn its secrets. The third-party software tool, APG/3000 (application profile generator), should be helpful in identifying the portions of an application where the CPU time is spent (APG was written by Kim Leeper of Wick Hill Associates). Once APG has identified the key section of code, you might want to recode it in SPL/3000 for maximum efficiency.

As is usually the case, the biggest improvements are obtained by re-evaluating the logic of the application. For example, you should periodically check the distribution of all reports to see if anyone is reading them. If not, don't run the job at all — that is an infinite performance gain.

III. B. Transfer More Information Per Disc Access

Besides CPU time, the other major limit on throughput is the access speed of the discs. One way to transfer more information per disc access is to build files with larger blocksizes. The "block" is the unit of physical transfer for the file. A larger blocksize means that you move more records per revolution of the disc. However, there is a trade-off: increased buffer space and impact on other users. In on-line applications, you usually want a small blocksize. Below, I will explain NOBUF/MR access, which is a technique that allows you to "have your cake and eat it, too!"

Another way to transfer more useful information per disc access is to ensure that the data is organized so the records that are usually required together are in the same disc block. Rick Bergquist's DBLOADNG program (contributed library) reports on the internal efficiency of IMAGE datasets. For example, if it shows that the work orders for a given part are randomly dispersed throughout a detail dataset (necessitating numerous disc accesses), you can ensure that they will be stored contiguously by doing a DBUNLOAD/DBLOAD (assuming that part number is the primary path into work orders). For master datasets, DBLOADNG shows you how often you can find a specific entry with only a single disc read (the ideal). If DBLOADNG shows that multiple disc reads are often needed for a certain dataset, you may be able to correct the situation by increasing the capacity of the dataset to a larger prime number or by changing the data type and/or internal structure of the key field.

Don't overlook the obvious either. If you can compress the size of an entry by using a more efficient data type (Z10 converted to J2 saves six bytes per field), you can pack more entries into each block and thus reduce the number of disc accesses to retrieve a specific entry.

You can often increase the "average information value" of each disc access by re-thinking your application. For example, suppose you must store transactions in a database in order to provide some daily reports, many monthly reports, a year-end report, and an occasional historical report covering several years. If you store all transactions in a single dataset, the daily jobs will probably take three hours to find, sort, and total 100 transactions. Why not put today's transactions in a separate dataset and transfer them to the monthly dataset after the daily jobs are run? When the monthly reports are completed, you can move the data to a yearly dataset, and so on. This is called "isolating data by frequency of access." The fewer records you have to search to find the ones you want, the more information you are retrieving per access.

It is theoretically possible to transfer more information per second by reducing the average time per disc access. Typically, you attempt to improve the "head locality" (i.e., keep the moving "heads" of each disc drive in the vicinity of the data that you will need next).

Although it is hard to prove, it does seem that using device classes to keep spooling on a different drive from databases, for example, does improve batch throughput. Under MPE IV, you can also spread "virtual memory" among several discs. The next "logical step" is to place masters and details on separate drives. However, in all tests that I have run with actual datasets and actual programs, there was no consistent difference in performance between having the datasets on the same drive or on different drives. The dynamics of disc accessing on the HP3000 are very complex. Unless you have the time to do a RELOAD afterwards, don't move files around; the moving process itself (:STORE and :RESTORE) may fragment the disc space and eliminate the potential benefit of spreading the files. Remember Green's Law: "The disc heads are never where you think they are."

You can also improve overall batch throughput by recovering wasted disc accesses. The disc drives revolve at a fixed speed, whether you access them or not. Any disc revolution that does not transfer useful data is wasted. Multiprogramming attempts to use these wasted accesses by maintaining a queue of waiting tasks. Unfortunately, maximum throughput under MPE III coincided with JOB LIMIT = ONE (no multiprogramming!). Under MPE IV, however, I have obtained a 25% decrease in elapsed time on the Series III by running two or three jobs concurrently. Try it.

III. C. Increase Program Size to Save Disc Accesses

In on-line optimizing, we are always trying to reduce the size of the program (code, data, and CPU usage), so as to allow the system to provide good response time to more users at once. In batch optimizing, we do not want better response time (we won't be running 36 batch jobs at a time, so we don't have to worry about mix); we want better throughput. Since most of the on-line tricks actually make the program slightly slower, we should avoid them. Batch tricks usually consist of trading off a larger program size for a faster elapsed time.

You can often save disc accesses by storing data in larger "chunks," keeping more data in memory at any time. Larger blocks will accomplish this, as will extra buffers. MPE file buffers can be increased above the default of two via :FILE, but doing so actually appears to degrade throughput. KSAM key-block buffers are increased via :FILE (:FILE xx;DEV=,,yy :MNS where xx is the KSAM data file and yy is the number of key-block buffers), which will help for empty files (KSAM cannot deduce how many buffers it will need unless the B-tree already exists). IMAGE buffers are increased via the BUFFSPECS command of DBUTIL; this can be effective for a stand-alone batch job, but only if it works with a large number of blocks concurrently (i.e., puts and deletes to complex datasets with many paths).

Pierre Senant of COGELOG (the developer of ASK/3000) has an ingenious method for "increasing program size" dramatically. He has implemented "memory files." An entire file is copied in main memory and kept there. For a small file that is frequently accessed (e.g., a master dataset containing only a few edit codes that must be applied to many transactions), Pierre's method should save enormous numbers of disc accesses.

NOBUF access to files was mentioned above as a way to save CPU time. If you use NOBUF with MR access, you can save disc accesses also, but at the cost of a larger data stack. MR stands for "multi-record," and gives you the ability to transfer multiple blocks per access, instead of just one block. With a large enough buffer, you will reduce the number of disc accesses dramatically.

Since multi-block access is faster only if each block is an exact multiple of 128 words in length, you should always select a recordsize and blockfactor such that the resulting blocksize (recordsize times blockfactor) is evenly divisible by 128 words. The resulting blocksize need not be large; it need only be a multiple of 128 (i.e., 256, 384, 512, . . .). As I promised earlier, here is your way to have the best of both worlds. Build your files with 512-word blocks (i.e., 4 times 128, 8 times 64, 16 times 32) for on-line use, and redefine the blocksize to 8192 words in batch programs via NOBUF/MR access.

For a "stand-alone batch" job, you may as well set MAXDATA to 30,000 words. This allows sorts to complete with maximum speed and provides other opportunities for optimization. With a larger stack you can keep small master datasets in the stack (e.g., a table of transaction codes). When you have exhausted the 30,000 words of your data stack, there are always extra data segments, which can be thought of as "fast, small files."

Re-evaluate your view of the data. Databases are usually set up to make life easy for the on-line user (rightly). Their organization may not be optimum for batch processing. In order to provide numerous enquiry paths, a single word order may be scattered in pieces among seven different datasets, and may require up to 20 calls to DBFIND and DBGET for assembly. In a batch job, if you are going to have to re-assemble the same order many times, it may be more efficient to define a huge, temporary record for the entire order, assemble it once, and write it to a temporary file. Then you can sort the temporary-file record numbers in numerous ways, and retrieve an entire order with a single disc read whenever you need it. Of course, this wastes disc space (temporarily) and increases your program size.

III. D. Remove Structure to Save Unneeded Disc Accesses

"Structure" for data means organization, lack of randomness, and the ability to quickly find selected groups of records. It takes work to maintain a "structure," and the more structure there is, the more work (CPU time and disc accesses) it takes.

Study your data structures critically. Can you reduce the number of keys in a record? A serial search may be the fastest way to get the data. Can you eliminate a sorted path? Overall, the application may be faster if you sort each chain in the stack after reading it from the dataset (Ken Lessey's SKIPPER package has this capability), but only if you don't use the COBOL SORT verb.

Another type of "structure" is consistency. IMAGE is a robust data management system because it writes all dirty data blocks back to the disc before terminating each intrinsic call. You can make IMAGE faster, but less robust, if you call DBCONTROL to defer disc writes (only after a backup). Another IMAGE idea: don't use DBDELETE during production batch jobs. Just flag deleted records with DBUPDATE and DBDELETE them later, when no one is waiting for any reports. When you can, use a DBUPDATE in place of DBDELETE and DBPUT.

For KSAM, if you are planning to sort the records after you retrieve them, use "chronological access" (FREADC) instead of default access (FREAD). Default KSAM access is via the primary key; KSAM must jump all over the disc to get the records for you in this sorted order, just so you can re-sort them in another order! Also for KSAM, try to keep only one key (no alternate keys), do not allow duplicates (much more complex), and avoid changing key values of records.

I am grateful to Alfredo Rego for pointing out a useful way to "eliminate structure" from IMAGE. When you are loading a large master dataset, use a Mode-8 DBGET prior to the DBPUT in order to find out if the new entry will be a primary entry or a secondary entry. Load only primaries on the first pass, then go back and load the secondaries on a second pass. This effectively turns off the IMAGE mechanism known as "migrating secondaries," which although essential, is time-consuming when filling an entire dataset.

III. E. Add Structure for Frequent Events

I saved this for last because it is one of the most powerful ideas. Batch tasks usually repeat certain key steps numerous times. Batch tasks have patterns of repetition in them. If you make that key step faster by adding structure to it, or re-structure the application so that "like-steps" are handled together, you can make the whole task faster. Extra structure (code complexity or data complexity) is justified in the most frequent operations of batch processing.

Check your data structures for patterns that you could capitalize on. For example, if you have a file of transactions to edit and post to the data base, could the task be made faster if the file were sorted by transaction type (only do validation of the transaction type when it changes) or by customer number (only validate the customer number against the database when it changes)?

Here are more examples of adding structure. If you sort by the primary key before loading a KSAM file, you can often cut the overall time in half. When erasing an IMAGE detail dataset, sort the record numbers by the key field that has the longest average chain length and delete the records in that order. When loading a detail dataset with long sorted chains, first sort by the key field and the sort field. In all of these examples, throughput is increased by adding code structure to match the structure of the data.

If you frequently require partial-key searches on IMAGE records, use an auxiliary KSAM file (or a sorted flat file and a binary search) to give you "indexed-sequential" access, rather than only serial access, to your IMAGE dataset. (Mark Trasko's IMSAM product enhances IMAGE by adding an indexed-sequential access method to the other access methods of IMAGE.)

If you have used many IMAGE calls to find a specific record, remember its record number. Then, when you need to update it, you can retrieve it quickly with a Mode 4 DBGET (directed read), instead of doing the expensive search all over again. If certain totals must be recalculated each month, why not re-design the database so that they are saved until needed again? If something takes work to calculate, check whether you will need it again.

The general principle is: look for patterns of repetition and add structure to match those patterns.

SECTION IV.
BATCH OPTIMIZING EXAMPLE: SUPRTOOL

SUPRTOOL is a utility program for the HP3000 that was developed by Robelle Consulting Ltd. The objectives of SUPRTOOL are to provide a single, consistent, fast tool for doing sequential tasks, whether in production batch processing, file maintenance, or ad hoc debugging. Example tasks that SUPRTOOL can handle are: copying files, extracting selected records from IMAGE datasets (and MPE files and KSAM files), sorting records that have been extracted, deleting records, and loading records into IMAGE datasets and KSAM files. SUPRTOOL can't do everything yet, but we are adding new capabilities to it regularly (the most recent enhancements are a LIST command to do formatted record dumps and an EXTRACT command to select fields from within records). SUPRTOOL embodies many of the batch optimizing ideas discussed in the previous section of this document.

IV. A. SUPRTOOL and "Bypassing Inefficient Code"

By doing NOBUF deblocking of records, SUPRTOOL saves enough CPU time to reduce the elapsed time of serial operations visibly. For MPE files, NOBUF is now fairly commonplace (although it still isn't the default mode in FCOPY — SUPRTOOL is 6 to 34 times faster in copying ordinary files). Where SUPRTOOL goes beyond ordinary tools is in extending NOBUF access to KSAM files (a non-trivial task) and to IMAGE datasets (very carefully). By making only a few "large" calls to the FREAD intrinsic, instead of many "small" calls to DBGET (each of which must access two extra data segments, look up the dataset name in a hash table, re-check user access security, and then extract a single record), SUPRTOOL quickly cruises through even enormous datasets with only a minimal consumption of CPU time.

For example, here is a comparison of SUPRTOOL and QUERY, selecting records from a detail dataset containing 60,971 current entries which are spread throughout a capacity of 129,704 entries.

```
SUPRTOOL/Robelle                   QUERY/3000
>BASE ACTIVE.DATA,5                >DEFINE
>GET LNITEM                        DATA-BASE =>>ACTIVE.DATA
>IF ORD-QTY>10000                  >FIND LNITEM.ORD-QTY>10000
>XEQ                               USING SERIAL READ
IN=60971. OUT=14479.               14479 ENTRIES QUALIFIED
CPU-SEC=56. WALL-SEC=133.          (CPU-SEC=520. WALL-SEC=763.)
```

Notice that SUPRTOOL used 1/9th as much CPU time and 1/6th as much elapsed time. And, the QUERY FIND command only builds a file of record numbers; to print the 14,479 records, QUERY must retrieve each one from the dataset again. SUPRTOOL creates an output disc file containing the actual record images, not the record numbers. With suitable prompting, SUPRTOOL can do this task even faster (see below for the BUFFER command).

IV. B. SUPRTOOL and "Transferring More Information"

SUPRTOOL transfers more information per disc access by doing multi-block transfers between the disc and the data stack in main memory. If records are 32 words long and stored as four per block (for a blocksize of 128 words), reading multiple blocks can make a big difference. For 20,000 records, one block at a time requires 5000 disc accesses. Using a 4096-word buffer and reading 32 blocks at a time reduces the number of disc accesses to 157!

SUPRTOOL has an option (SET STAT,ON) that prints detailed statistics after each task, so that you can see how it was done and where the processing time was spent. For example, suppose you want a formatted dump in octal and ASCII of all the records from the file described above for the order "228878SU." Below are the commands and times for SUPRTOOL and FCOPY:

```
FCOPY/3000
>FROM=SUMMRY;TO=*SUPRLIST;SUBSET="228878SU",1;OCTAL;CHAR
EOF FOUND IN FROMFILE AFTER RECORD 19999
3 RECORDS PROCESSED *** 0 ERRORS
(CPU-SEC=78. WALL-SEC=114.)

SUPRTOOL/Robelle
>SET STAT,ON
>DEFINE A,1,8
>IN SUMMRY
>LIST
>IF A="228878SU"
XEQ
IN=20000. OUT=3. CPU-SEC=11. WALL-SEC=16.

     ** OVERALL TIMING **
CPU milliseconds:              10854
Elapsed milliseconds:          16254
     ** INPUT **
Input buffer (wds):            4096
Input record len (wds):        32
Input logical dev:             12
Input FREAD calls:             157
Input time (ms):               6304
Input records/block:           4
Input blocks/buffer:           32
```

Notice that SUPRTOOL was using its default buffer size of 4096 words. FCOPY had to make 5000 disc transfers, while SUPRTOOL only had to make 157. That is one of the reasons why SUPRTOOL finished in 1/7th the time and used 1/7th the CPU time.

IV. C. SUPRTOOL and "Increasing Program Size"

SUPRTOOL gets a great deal of its performance edge by doing its own deblocking: allocating a large buffer within its data stack, reading directly from the disc into the buffer, and extracting the records from the blocks manually. SUPRTOOL trades a larger program size for a faster elapsed time. But you don't need to stop with the 4096-word buffer that SUPRTOOL normally allocates. Using the BUFFER command, you can instruct SUPRTOOL to work with buffers of up to 14,336 words and observe the results with SET STAT,ON. Here is the same selective file-dump that took 16 seconds with 4096-word buffers, done with 8192-word buffers:

```
SUPRTOOL/Robelle
>BUFFER 8192
>IN SUMMRY
>LIST
>IF A="228878SU"
>XEQ
CPU-SEC=10. WALL-SEC=13.  [An additional savings of 3 seconds]
```

By combining SUPRTOOL with IMAGE, you can have small data blocks for on-line access and large data blocks for batch sequential access. Here is the same database extract as done above (in the QUERY vs. SUPRTOOL test). Instead of using 4096-word buffers, we will increase the buffer space to 14,336 words:

```
SUPRTOOL/Robelle
>BUFFER 14336
>BASE ACTIVE.DATA,5
>GET LNITEM
>IF ORD-QTY>10000
>XEQ
IN=60971. OUT=14479. CPU-SEC=46. WALL-SEC=104. [Saved 29 sec.]
```

IV. D. SUPRTOOL and "Removing Structure"

SUPRTOOL can optimize batch operations by "removing structure." NOBUF deblocking of MPE files and IMAGE datasets provides faster serial access by saving CPU time and reading larger chunks of data, but NOBUF deblocking of KSAM files does that and more: it also eliminates structure. When you read a KSAM file serially by default, the KSAM data management system does not return the records to you in "physical" sequence; it returns them to you "structured" by the primary key value, and this takes work — a lot of work.

KSAM must search through the primary B-tree to find the sequence of the key values, and must then retrieve the specific blocks that contain each records. Quite often, logically adjacent records may not be physically adjacent; in the worst case, each logical record requires at least one physical block read. The SUPRTOOL NOBUF access to KSAM files cuts through all of this and returns the raw records to you in physical order; the savings in time can be impressive and, if you are planning to sort the records anyway, there is no loss of function. SUPRTOOL only removes the structure that you were not going to use.

Another example of removing structure in SUPRTOOL is the SET DEFER,ON command. When used in conjunction with the PUT or DELETE commands, the DEFER option causes SUPRTOOL to put IMAGE into output-deferred mode (via a call to DBCONTROL). Normally, IMAGE maintains a consistent and robust "structure" in the database after every intrinsic call. If you are planning to make a large number of database changes and can afford to store the database to tape first, you may be able to cut the elapsed time in half (or more) by leaving the physical database in an inconsistent state after intrinsic calls. (DBCONTROL makes the database consistent again when you are done.)

Here is an example use of SUPRTOOL to find all work orders that are completed (status="X") and old (dated prior to June 1st, 1982), delete them from the dataset, sort them by customer number and work-order number, and write them to a new disc file. SET DEFER,ON is used to make the DELETE command faster:

```
SUPRTOOL/Robelle
>BASE FLOOR.DATA
>GET WORKORDER
>IF WO-STATUS="X" AND WO-DATE<820601
>DELETE
>SORT CUSTOMER-NUM;SORT WORKORDER-NUM
>OUTPUT WO8206
>SET DEFER,ON
>XEQ
```

Another way to look at SUPRTOOL is as follows: if a serial search is fast enough, you may not need to have an official IMAGE "path" in order to retrieve the records you need. On the Series III, SUPRTOOL selects records at a rate of two seconds per 1000 sectors of data.

IV. E. SUPRTOOL and "Adding Structure"

SUPRTOOL can optimize batch tasks by "adding structure" to data. One way to add structure is to sort data. Experiments have shown that sorting records into key sequence can cut the time to load a large KSAM file in half. SUPRTOOL easily reorganizes existing KSAM files by extracting the good records, sorting them by the primary key field, erasing the KSAM file, and writing the sorted records back into it — all in one pass.

You can also add "structure" to raw data by defining a record structure for it (QUERY can access IMAGE entries because they have a structure defined by the schema). Normally, regular MPE files are not thought of as having the same kind of record structure as IMAGE datasets. Why is this so? Because you cannot access the fields of the file's records by name in tools such as FCOPY, even if the structure exists. In SUPRTOOL, you can.

If you use SUPRTOOL to archive old entries from IMAGE datasets to MPE disc or tape files, you can later do selective extracts, sorts, and formatted dumps on those MPE files, using exactly the same field names as you did when the entries were in the database. (In fact, you can even put selected records back into a temporary database with the same structure and run QUERY reports on them.) Here is how SUPRTOOL associates structure with raw MPE files:

```
SUPRTOOL/Robelle
>BASE FLOOR
>INPUT WO8206 = WORKORDER          [implied record structure!]
>IF CUSTOMER-NUM="Z85626"
>LIST
>XEQ
```

And, since SUPRTOOL has access to the IMAGE database that the entries originally came from, SUPRTOOL can still format the entries on the lineprinter with appropriate field names and data conversions (similar to REPORT ALL in QUERY).

IV. F. Results of Applying Batch Rules to SUPRTOOL

Just before completing this paper, we sent a questionnaire to the users of SUPRTOOL, asking them what they would tell other HP3000 sites about SUPRTOOL. Here are their replies:

"I always recommend SUPRTOOL with any new system. Without programming, I duplicated a master file from one application to another application. I set up a job stream to do this on a weekly basis (i.e., purge the old dataset entries and add the new dataset entries easily). SUPRTOOL creates files with different selection criteria to feed the same program." (Terry Warns, B P L Corp.)

"An essential package for efficient operation of a system. Most of our job streams include a SUPRTOOL function." (Vaughn Daines, Deseret Mutual Benefit Assoc.)

"Excellent. We had an application that serially dumped a dataset of 185,000 records (4 hours) and then sorted the 114-byte records in 6 hours (provided we had the disc space needed). We changed to SUPRTOOL with the OUTPUT NUM,KEY option and a modified program using DBGET mode 4 and maximum BUFFSPECS. The result was 4 hours altogether." (Bobby Borrameo, HP Japan)

"SUPRTOOL is an excellent utility for copying standard MPE files and databases very quickly . . . extracting and sorting records from a database (i.e., 40,000 records of 60,000), copying files across the DS line (much quicker than FCOPY), copying tape to disc and disc to tape." (Dave Bartlet, HP Canada)

"We couldn't operate without it. We are a heavy KSAM user and SUPRTOOL has cut our batch processing by at least 1/3." (Jim Bonner, MacMillan Bloedel Alabama)

"All sorts of marvelous things. [SUPRTOOL] is really nice (and fast) to copy a database for test pruposes or to make minor changes (instead of DBUNLOAD/LOAD) — even major changes, using a program to reformat the SUPRTOOL-created file." (Susan Healy, Mitchell Bros. Truck Lines)

"Just last night I told a friend that, after working with different sorts on IBM (DPD- and GSD-level machines), Burroughs sorts, and even HP sorts, SUPRTOOL is the best sort tool I have ever used." (Robert Apgood, Whitney-Fidalgo Seafoods)

"Get it. Runs much faster than SORT. Cheap at twice the cost." (Willian Taylor, Aviation Power Supply, Inc.)

"Fast and functional. SUPRTOOL is deeply embedded in our applications, most extracts are done with SUPRTOOL. Ad hoc inquiries [via SUPRTOOL], involving pattern matching on our customer file, extract the appropriate keys, which are then passed to the report program." (Patrick Hurley, Port of Vancouver)

"SUPRTOOL is a product which no shop that uses IMAGE and does batch report generation should be without. By changing certain reports to use SUPRTOOL instead of traditional selection techniques, a savings of 60% in CPU and wall time was obtained." (Vladimir Volokh, VSI/Aerospace Group)

"SUPRTOOL is a great timesaver when used with BASIC (or RPG) to modify IMAGE datasets and place them in another dataset or the same dataset." (John Denault, Datafax Computer Services, Inc.)