NASA TECHNICAL MEMORANDUM

NASA TM X-3512

NASA TM X-3512

EFFECT OF VIRTUAL MEMORY ON EFFICIENT SOLUTION OF TWO MODEL PROBLEMS

Jules J. Lambiotte, Jr.

Langley Research Center

Hampton, Va. 23665

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION • WASHINGTON, D. C. • JULY 1977

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM X-3512		

4. Title and Subtitle	5. Report Date
EFFECT OF VIRTUAL MEMORY ON EFFICIENT SOLUTION OF TWO MODEL PROBLEMS	July 1977
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
Jules J. Lambiotte, Jr.	L-11490
	10. Work Unit No.
9. Performing Organization Name and Address	505-15-37-04
NASA Langley Research Center Hampton, VA 23665	11. Contract or Grant No.
	13. Type of Report and Period Covered
12. Sponsoring Agency Name and Address	Technical Memorandum
National Aeronautics and Space Administration Washington, DC 20546	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

Computers with virtual memory architecture allow programs to be written as if they were small enough to be contained in memory. Two types of problems are investigated to show that this luxury can lead to quite inefficient performance if the programer does not interact strongly with the characteristics of the operating system when developing the program. The two problems considered are the simultaneous solution of a large linear system of equations by Gaussian elimination and a model three-dimensional finite-difference problem. The Control Data STAR-100 computer runs are made to demonstrate the inefficiencies of programing the problems in the manner one would naturally do if the problems were, indeed, small enough to be contained in memory. Program redesigns are presented which achieve large improvements in performance through changes in the computational procedure and the data base arrangement.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Virtual memory Page faulting Linear equations Data management Vector computers	Unclassified - Unlimited Subject Category 61

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price*
Unclassified	Unclassified	12	$3.50

* For sale by the National Technical Information Service, Springfield, Virginia 22161

EFFECT OF VIRTUAL MEMORY ON EFFICIENT SOLUTION OF TWO MODEL PROBLEMS

Jules J. Lambiotte, Jr.
Langley Research Center

SUMMARY

Computers with virtual memory architecture allow programs to be written as if they were small enough to be contained in memory. Two types of problems are investigated to show that this luxury can lead to quite inefficient performance if the programer does not interact strongly with the characteristics of the operating system when developing the program.

The two problems considered are the simultaneous solution of a large linear system of equations by Gaussian elimination and a model three-dimensional finite-difference problem. The Control Data STAR-100 computer runs are made to demonstrate the inefficiencies of programing the problems in the manner one would naturally do if the problems were, indeed, small enough to be contained in memory. Program redesigns are presented which achieve large improvements in performance through changes in the computational procedure and the data base arrangement.

INTRODUCTION

The introduction of virtual memory into computer architecture can be viewed as a mixed blessing. On the one hand, it eliminates the requirement that the programer explicitly bring data from backing store into fast memory. The operating system does this automatically whenever a piece of data which does not reside in fast memory is referenced; this is particularly helpful where "small" problems are ultimately made "big" by changing a few parameters in the program. In this situation, the tedious job of adding new software becomes unnecessary. On the other hand, it is easy to demonstrate programs which are quite inefficient in a virtual memory system once they become large.

The inefficiency can stem from several sources:

(1) Careless programing: Often minor changes in the order in which data are accessed and quantities are computed can have significant effects in reducing paging.

(2) Use of an inefficient algorithm: The most popular algorithm for a problem which is small enough to be contained in memory can be quite inferior to an algorithm carefully designed for a virtual memory system.

(3) Inefficient data base layout: The natural way to organize data in a program can be quite inefficient in that it can require, depending upon the page size selected, a large amount of unneeded data to be brought into memory. The effect is also, of course, to force data out that may be needed in a subsequent calculation.

This report describes the effect of virtual memory upon the efficient implementation of two types of problems:

(1) The solution of $Ax = b$ by Gaussian elimination

(2) The solution of partial differential equations by means of finite-difference equations

The first problem demonstrates the advantage of using an algorithm designed specifically for a virtual memory system. The second problem demonstrates the effect on paging of an inefficient (but natural) data base layout. Instances of minor programing changes which can cause significant paging reduction are pointed out in the discussion of both problems.

The computer model used is the Control Data STAR-100 computer. The difficulties a virtual system can present are particularly evident herein because of the limited selection of page size and because the speed of the STAR-100 central processing unit (CPU) makes inefficiency in the input/output (I/O) all the more noticeable. Timings from programs executed on this computer are given to quantify the concepts in this report.

STAR-100 VIRTUAL MEMORY SYSTEM

The Control Data STAR-100 computer at the Langley Research Center has a main memory of 524 288 64-bit words. The two page sizes are "small" pages containing 512 64-bit words and "large" pages containing 65 536 words (128 small pages). Main memory contains eight large pages and since the system requires nearly a large page, the user has about seven large pages of code and data in memory at any one time.

When a reference is made to an address which is not contained on a page currently in memory, the system will automatically replace one of the current pages with the desired page. This data movement is termed a "page fault." The page to be removed is selected by the system using a procedure called the "least recently used" (LRU) paging procedure. It determines that the page to be removed from memory is the one which has had the longest time since it has been referenced.

The choice of page size is very important and can affect program performance dramatically. The time required to transfer the two different size pages of data from disk to memory is analogous to computation on the STAR-100 with short and long vectors. There is a start-up time for the disk to locate and position itself to transmit the needed page. This start-up is the same for both small and large pages. Thus, a large page of 65 536 words of data is brought in with only one start-up, whereas there are 128 start-ups if small pages are used. Numerical experiments, discussed later, show that large pages are transmitted at about 200 000 words per second, whereas small pages are transmitted at about 10 000 words per second. Although it is clear that it is desirable to obtain the faster transfer rates associated with large pages, this report demonstrates that it is necessary to consider the interaction of the page size with the com-

putational algorithm, the data base arrangement, and the paging algorithm in order to achieve an efficient implementation.

SOLUTION OF SIMULTANEOUS EQUATIONS

The solution to the $N \times N$ system of equations $Ax = b$ can be obtained through the factorization of the $N \times N$ matrix A as $A = LU$ where L is a unit lower triangular matrix and U is an upper triangular matrix. Then, solution x is obtained by solving $Ly = b$ and $Ux = y$. There are many variants to this procedure which is commonly called Gaussian elimination. One such factorization procedure is described herein. The pivoting strategy which is usually included for numerical stability has been omitted but is discussed later in the report.

Algorithm A1:

```
(a)     For J←1,2,...,N-1

(b)         For I←J+1,J+2,...,N
               A(I,J)←A(I,J)/A(J,J)

(c)         For K←J+1,J+2,...,N

(d)             For I←J+1,J+2,...,N
                    A(I,K)←A(I,K)-A(I,J)*A(J,K)
```

When describing the algorithm, it is said that at step (b), the Jth column of L is computed, and at step (d), the Kth column of A is modified. Note that $L(I,J)$, $I = J + 1, J + 2, \ldots, N$, and $U(J,K)$, $K = J, J + 1, \ldots, N$, are stored back into the corresponding locations of A. The matrices A, L, and U are all stored columnwise and the vectorization is accomplished by collapsing the DO loops at step (b) and at step (d). For more details, see reference 1.

The paging for this algorithm can be determined quite easily since the data are accessed in a sequential manner which allows the interaction with the LRU paging procedure to be predicted. To derive equation (1), assume N^2 exceeds S, the available storage, and that N equals the page size. In this case, every column of A resides on a different page. Now, referring to algorithm A1 when $J = 1$, column 1 of L is computed and used to modify columns 2 to N. Assume M columns of A will fit into memory. Hence, an attempt to modify column $M + 1$ causes a page fault and the LRU paging procedure selects the page containing column 2 to be removed. Similarly, when column $M + 2$ is modified, column 3 is removed. After all $N - 1$ columns have been modified, the algorithm proceeds to $J = 2$ and references columns 2 to N, sequentially. It is clear that each reference to a column to be modified requires a page fault. At the Jth step, there are $N - J + 1$ columns referenced. Each reference gives a page fault except that there are no page faults for the last S/N steps since during those steps the submatrix of A fits into memory. The number of column references is

$$\sum_{i=1}^{N-(S/N)} (N - i + 1) \approx \frac{N^2 - (S^2/N^2)}{2}$$

This quantity can be converted to page faults for general N by

$$F_1 = p \frac{N^2 - (S^2/N^2)}{2} \qquad (1)$$

where p is the number of pages required for each column. The value of F_1 can be extraordinarily large; for example, for $N = 1000$, $S = 450\ 000$, and $p = 1000/65536$, $F_1 = 12\ 170$ large page faults. This would require about 60 minutes of I/O time for a problem which would require only about 45 seconds of CPU time on the STAR-100.

Consider another approach to Gaussian elimination. Since the multipliers $L(I,J)$ are saved at each step, one could compute M columns of L without ever referencing columns $M + 1$, $M + 2,\ldots,$ N and then do M modifications to column $M + 1$, then M modifications to $M + 2$, and so forth. The advantage of this technique is that after computing M columns of L and modifying columns $M + 1$, $M + 2,\ldots,$ N, M steps of Gaussian elimination have been completed and there are only N references to columns not in memory; however, the usual algorithm would have made $(NM - M^2)/2$ at this point. To describe the procedure, partition N into ℓ blocks, each consisting of M columns.

Algorithm A2:

```
(a)          For II←0,1,...,ℓ-1
        C FACTOR THE NEXT M COLUMNS

(b)            For J←II*M+1,II*M+2,...,II*M+M

(c)                For I←J+1,J+2,...,N
                      A(I,J)←A(I,J)/A(J,J)

(d)                For K←J+1,J+2,...,II*M+M

(e)                  For I←J+1,J+2,...,N
                        A(I,K)←A(I,K)-A(I,J)*A(J,K)

        C MODIFY THE REMAINING COLUMNS OF  A

(f)            For K←(II+1)*M+1,(II+1)*M+2,...,N

(g)                For J←II*M+1,II*M+2,...,II*M+M

(h)                  For I←J+1,J+2,...,N
                        A(I,K)←A(I,K)-A(I,J)*A(J,K)
```

Note that steps (b) to (e) are similar to the usual algorithm except that only M columns of L are being computed; steps (f) to (h) then apply all of those

M modifications to each of the remaining columns. This algorithm is similar in intent to an algorithm described by Pavkovich (ref. 2), which was designed for use with magnetic tape as a backing store for systems too large to be contained in central memory.

The number of page faults for algorithm A2 is sensitive to the choice of M. The value of M should be chosen as large as possible but under the constraint that no portion of the M columns in the block factored in steps (b) to (e) will be paged out during the subsequent modification of the remaining columns in steps (f) to (h). This implies that the choice of M should leave room for two pages of columns to be modified so that the LRU paging procedure will remove a page of previously modified columns when it needs to bring in the next page of columns. Roughly, this says that for large pages, NM should be about 250 000 or less.

Under these assumptions, compute the page faults F_M as follows: Let $\ell = N/M$ be the number of blocks. At the ith step of ℓ, there are $N - M(i - 1)$ column references requiring paging. Hence,

$$F_M = p \sum_{i=1}^{\ell} \left[N - M(i - 1)\right] = p \frac{N + M}{2} \ell \tag{2}$$

Using equation (1) to compare F_1 and F_M gives

$$\frac{F_1}{F_M} = \frac{N^2 - (S^2/N^2)}{(N + M)\ell} \tag{3}$$

Now, assume $S \approx NM$,

$$\frac{F_1}{F_M} \approx N \frac{\ell - 1}{\ell^2} \approx M \tag{4}$$

Table I shows some paging results of runs for the two algorithms on the STAR-100. The page faults are for the entire test program which includes page faults not associated with the kernel algorithms described here. However, the great difference between the two algorithms is clearly demonstrated. (There is a factor of 30 at $N = 750$.)

For simplicity, the pivoting strategy for the algorithms has been omitted, but mention should be made about its implementation which directly affects the number of page faults. The point is more simply made with reference to algorithm A1. Prior to step (b) for algorithm A1, one must search column J, diagonal and below, for the maximum element in absolute value. If this occurs in row IP, then rows J and IP are interchanged. Note that, however, if one performs the row interchange prior to proceeding to steps (b) to (d), twice the number of page faults given by F_1 are generated since the interchange requires indexing through the entire array A. The correct way to proceed is to interchange elements J and IP of column K between steps (c) and (d). This forces column K into memory. Then modify column K as in step (d) and proceed to col-

umn K + 1. The value for F_1 is still valid with this approach and similar modifications can be made for algorithm A2.

TABLE I.- COMPARISON OF PAGE FAULTING IN REGULAR AND BLOCK GAUSSIAN ELIMINATION

N	Algorithm A1 (regular)	Algorithm A2 (block)	Block size
	Large page faults	Large page faults	
600	15	15	
625	17	17	
650	17	18	400
700	691	37	300
750	1350	45	300
800		50	300
850		65	250
900		72	250

SOLVING LARGE FINITE-DIFFERENCE PROBLEMS

One of the most efficient uses of the STAR-100 computer is the solution of partial differential equations by finite-difference techniques in which an explicit approximation to the equations is made. This use leads to an implementation in which the vector lengths can be as large as the number of mesh points; thus, efficient use is made of the vector instructions of the computer. However, if the problem is programed as one might naturally do for a problem size which does not exceed available memory, extreme paging inefficiencies can result if the problem size is expanded so that its data base cannot be contained in memory.

A reasonable I/O goal in program design would be that data be referenced so that once the data are paged into memory during a particular iteration, all computations for that iteration involving data on that page are performed before it is paged out again. If this is achieved, then the total number of words paged in per iteration is the size of the data base. However, it is easy to demonstrate that for the STAR-100 large page size, the usual data base arrangement can lead to far more I/O activity than this. To illustrate, consider as a model problem a three-dimensional finite-difference grid consisting of NLAY planes each of size NSZ with NV variables V_I (I = 1,2,...,NV) required at each point with the data flow for one iteration or time step shown in figure 1. The usual data base (UDB) design is indicated in figure 2 where NSZ = 10 000, NLAY = 50, and NV = 20. The 500 000 words reserved for each variable require about eight large pages and, as shown in the figure, each variable at a level L resides on a separate large page. Since only seven large pages can be in memory at any one time, any computation for V_I which involves seven or more other variables must result in extra paging. The paging increases as a function of the

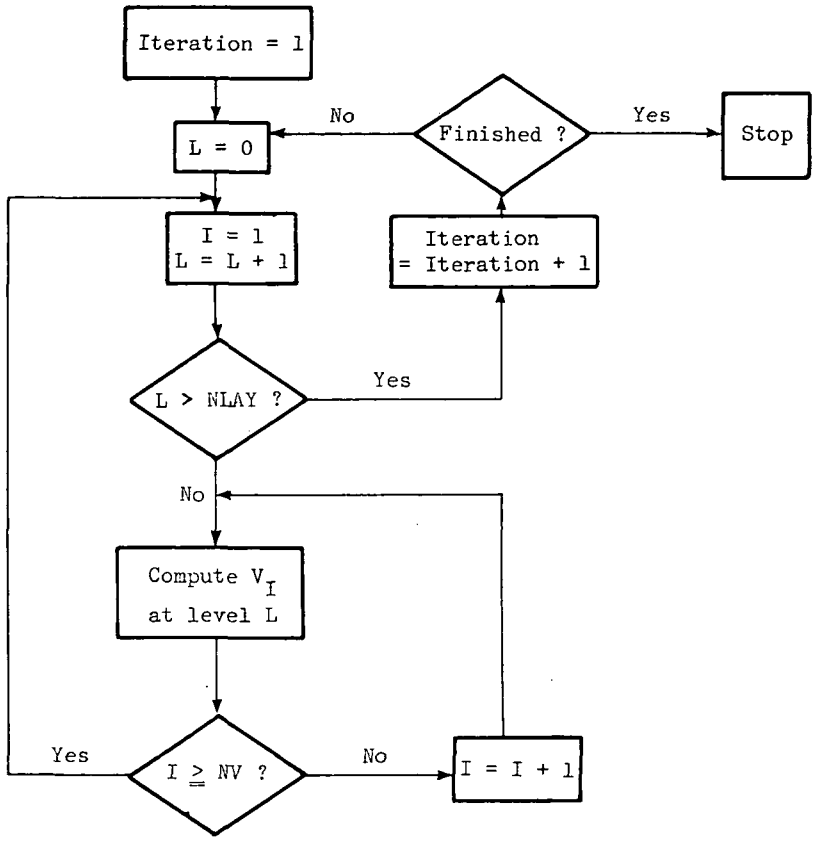

Figure 1.- Model problem program flow.

number of variables involved in the computation and the number of times each is referenced. In fact, when computing V_I at level L, it is possible for each reference to any of the other variables at that level to require a page fault.

The excessive paging results from the fact that whereas only the 10 000 words associated with some V_I at level L are required, an additional 55 536 unusable words are paged in also. These additional words affect the I/O time in two ways:

(1) They increase the I/O time because the unneeded pieces of data represent about 85 percent of the data transferred during each page fault.

(2) More importantly, they increase the subsequent I/O activity since the unwanted data reduce the effective size of memory to only 7*NSZ locations.

A solution to the problem is to have an interleaved data base (IDB) design where V_1 at level 1 is stored, followed by V_2 at level 1, followed by V_3 at level 1, and so forth, until all NV variables at level 1 are exhausted. Then level 2 data are stored and so forth. For the model problem, for instance, the 200 000 memory locations required for all variables at a particular level would require only about three large pages and, unlike the UDB design, once paged in

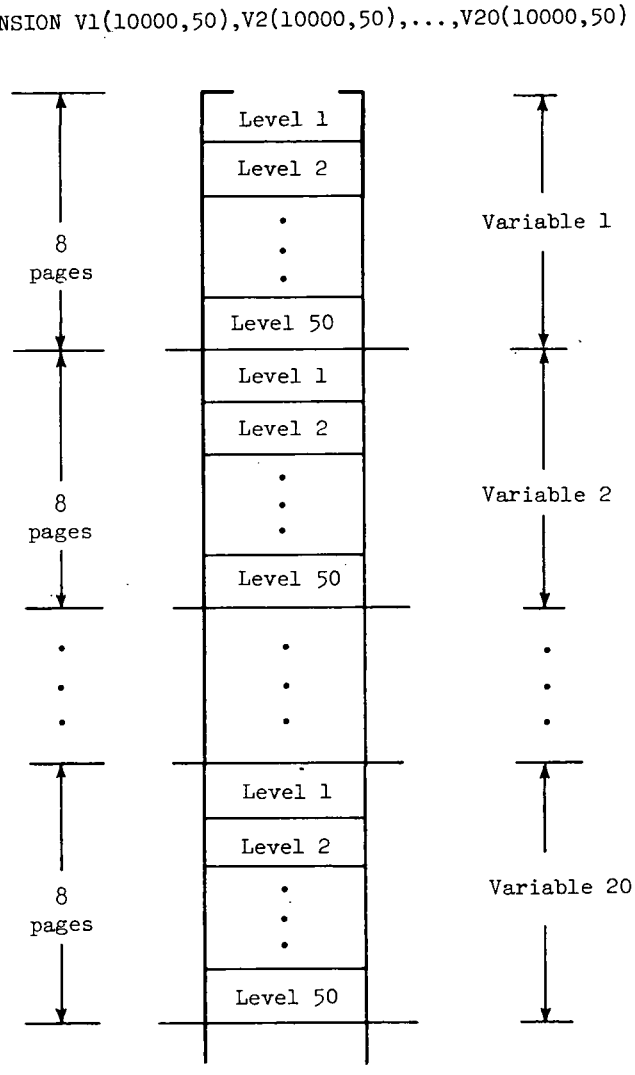

Figure 2.- Usual data base for NV = 20 and NLAY = 50.

would remain in no matter how much computation is required with each variable at that level. This arrangement is illustrated in figure 3. The difficulty with using this approach lies in the implementation. How does one conveniently store and reference the data with this organization? In FORTRAN, the desired data storage can be obtained by the DIMENSION statement (for the model problem)

DIMENSION A(10000,20,50)

Then, if the programer wishes to reference $V_I(J,L)$, he specifies A(J,I,L). Although this approach does work, there are several drawbacks:

(1) This type of reference is awkward and makes the program hard to write and even harder to read. For instance, the name of an array such as DENS(J,L) for density makes debugging much easier and is more meaningful later than a reference such as A(J,I,L).

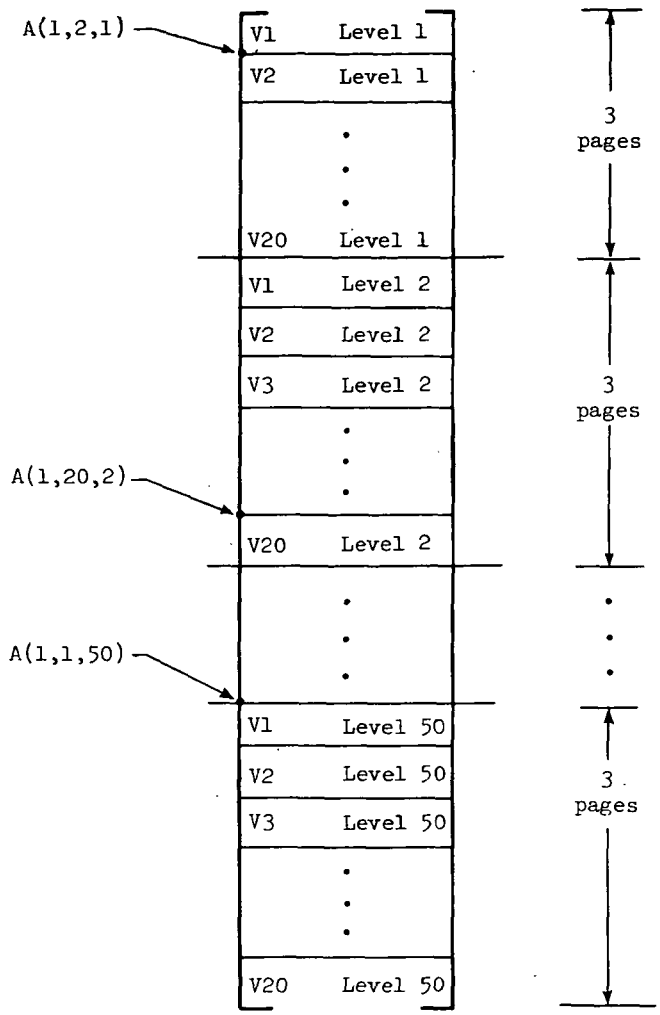

Figure 3.- Interleaved data base for NV = 20 and NLAY = 50.

(2) Many problems start out "small" and become "large" through the change of a few parameters. A programer not expecting the problem to become large is not likely to go to the extra effort of using the interleaved approach.

(3) A program being converted to the vector computer may convert quite simply if its usual data base layout is kept intact, but it requires complete rewriting if the interleaved approach is used.

From a user standpoint, the simplest solution would be a compiler construct which would allow a group of arrays to be dimensioned and referenced in the usual way but to be stored internally in an interleaved fashion. No such construct in any compiler is known to be available at this time. An approach which is less simple for the user, but perhaps more realistic, is to dynamically assign a more meaningfully named variable to the appropriate portion of array A. For instance, STAR FORTRAN allows the use of a descriptor to specify a vector

starting at a designated location within an array. Hence, if V_5 is density, one might write

```
DIMENSION A(10000,20,50)
DESCRIPTOR DENS
DO 1 L=1,50
  ASSIGN DENS,A(1,5,L;10000)
  etc.
```

Now, meaningful code can be written in terms of DENS.

In order to test the effectiveness of the IDB design, some numerical experimentation was carried out on the STAR-100. Three variations of a program, which follows the logic of the flow chart in figure 1, were executed for NLAY = 15, NV = 20, and various values for NSZ. Program A used the UDB design (fig. 2) with large pages; program B was the same as program A except small pages were used; program C used the IDB layout with large pages (fig. 3). The results for a time step are given in table II. Program A gave the poorest results. It exhibited the extraordinary increase in paging and elapsed time predicted for it. Program B had much less I/O activity because of its smaller page sizes but nevertheless was inferior to the interleaved approach in program C because of the poorer transfer time associated with small pages. The IDB layout has both the fast transfer associated with large pages and a moderate amount of I/O activity so that it proved superior to the other two. (In the largest cases run, the elapsed time for program C is better than that for program A by a factor of approximately 110 and better than that for program B by a factor of approximately 8.)

TABLE II.- PAGING STATISTICS FOR THREE-DIMENSIONAL MODEL PROBLEM

NSZ (total data base)	Program A (UDB) large pages			Program B (UDB) small pages			Program C (IDB) large pages		
	CPU time, sec	Elapsed time, sec	Page faults	CPU time, sec	Elapsed time, sec	Page faults	CPU time, sec	Elapsed time, sec	Page faults
1500 (450 000)	0.22	0.23	0	0.39	0.40	0	0.22	0.23	0
1600 (480 000)	.25	347.10	1899	.40	5.09	102	.23	3.11	8
1700 (510 000)				.44	17.57	333	.25	3.12	8
1800 (540 000)				.46	26.02	471	.26	3.44	9

The number of large pages faulted for in program C (8 or 9) does reflect the total data base for the problem. It is reasonable to ask, "Is it possible to have the I/O activity reflect the amount by which the data base exceeds memory?" The answer for the IDB arrangement is yes. Because the data are accessed in such a predictable fashion, it is known reasonably well what pages are in memory at any time. For instance, for NSZ = 1700, a data base of about eight large pages is required. When layers L = 14 or 15 are to be processed, a page fault occurs for that data. The LRU algorithm removes the page containing L = 1 and L = 2. Now, on the second iteration, if it is possible to go through the NLAY levels in reverse order as it would be for this model problem, there will not be a page fault until level 2 is reached. Here, the LRU algorithm will remove L = 14 and 15. Hence, if one proceeds forward and then backwards through the data base, the I/O activity reflects only the one page by which the data base exceeds available memory. Table III includes the time for the alternating IDB approach and repeats the IDB data from table II. The improvement factors of 110 and 8, mentioned earlier, increase to approximately 575 and 26, respectively.

TABLE III.- PAGING STATISTICS FOR FORWARD-BACKWARD AND FORWARD-ONLY ACCESS OF IDB DESIGN

NSZ (total data base)	Forward-backward access			Forward-only access		
	CPU time, sec	Elapsed time, sec	Large page faults	CPU time, sec	Elapsed time, sec	Large page faults
1600 (480 000)	0.23	0.60	1	0.23	3.11	8
1700 (510 000)	.25	.60	1	.25	3.12	8
1800 (540 000)	.26	.99	2	.26	3.44	9

The effective use of virtual memory requires that the programer make the most use of the data while the data are in memory. This includes program design features which may not be nearly as dramatic as reorganizing the data base as described previously. Some very minor changes can have important effects. For instance, in the model three-dimensional problem, if the boundary equations are not evaluated until after all the interior equations, twice as much paging results as in a program which evaluates the boundaries at level L immediately after evaluating the interior equations at level L.

CONCLUDING REMARKS

Two problems have been investigated to demonstrate that some very natural programing practices can result in catastrophic program performance in a virtual memory operating environment. The fact that the programer does not have to explicitly perform the input/output (I/O) from backing store can lull him into the belief that his usual programing procedures will suffice regardless of problem size.

The usual Gaussian elimination algorithm pointed out the need to redesign the calculations so that once a block of data is brought into memory, as much computation as possible is performed with it before it is paged out. The block Gaussian elimination program exhibited improvements in page faulting of a factor as high as 30.

The three-dimensional finite-difference model problem pointed out the inadequacy of the limited page sizes available on the Control Data STAR-100 computer. In order to take advantage of the high transfer rates associated with large pages, it was necessary to interleave the data base. This arrangement can make programing somewhat awkward, but use of descriptors in STAR FORTRAN can reduce this problem. Although the overly large page size of 65 536 words forces the redesign of the data base, the resulting interleaved data base (IDB) is quite efficient since very large blocks of data are involved each time an I/O operation is required. For the model problem, the elapsed time for the IDB design exhibited an improvement over that for the UDB (usual data base) design by a factor of approximately 110 when the latter used large pages and a factor of approximately 8 when small pages were used. These factors increased to approximately 575 and 26 when the interleaved data base was accessed in a backward direction on alternate iterations.

These algorithms are believed to have some relevance for problems which do not exceed memory if the operating system is run in a multiprograming mode. The interaction of several programs sharing memory can have the effect of reducing the effective size of memory available for each program. Hence, it is important that the algorithms for these small problems have efficient paging characteristics also.

Langley Research Center
National Aeronautics and Space Administration
Hampton, VA 23665
May 11, 1977

REFERENCES

1. Howser, Lona M.; and Lambiotte, Jules J., Jr.: STAR Adaptation for Two Algorithms Used on Serial Computers. NASA TM X-3003, 1974.

2. Pavkovich, John M.: The Solution of Large Systems of Algebraic Equations. Tech. Rep. No. 33 (Contract Nonr-225(37)), Comput. Sci. Div., Stanford Univ., Dec. 6, 1963. (Available from DDC as AD 427 753.)

NASA-Langley, 1977 L-11490

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
WASHINGTON, D.C. 20546

OFFICIAL BUSINESS
PENALTY FOR PRIVATE USE $300

SPECIAL FOURTH-CLASS RATE
BOOK

POSTAGE AND FEES PAID
NATIONAL AERONAUTICS AND
SPACE ADMINISTRATION
451

U.S.MAIL

POSTMASTER: If Undeliverable (Section 158 Postal Manual) Do Not Return

"The aeronautical and space activities of the United States shall be conducted so as to contribute . . . to the expansion of human knowledge of phenomena in the atmosphere and space. The Administration shall provide for the widest practicable and appropriate dissemination of information concerning its activities and the results thereof."

—NATIONAL AERONAUTICS AND SPACE ACT OF 1958

NASA SCIENTIFIC AND TECHNICAL PUBLICATIONS

TECHNICAL REPORTS: Scientific and technical information considered important, complete, and a lasting contribution to existing knowledge.

TECHNICAL NOTES: Information less broad in scope but nevertheless of importance as a contribution to existing knowledge.

TECHNICAL MEMORANDUMS: Information receiving limited distribution because of preliminary data, security classification, or other reasons. Also includes conference proceedings with either limited or unlimited distribution.

CONTRACTOR REPORTS: Scientific and technical information generated under a NASA contract or grant and considered an important contribution to existing knowledge.

TECHNICAL TRANSLATIONS: Information published in a foreign language considered to merit NASA distribution in English.

SPECIAL PUBLICATIONS: Information derived from or of value to NASA activities. Publications include final reports of major projects, monographs, data compilations, handbooks, sourcebooks, and special bibliographies.

TECHNOLOGY UTILIZATION PUBLICATIONS: Information on technology used by NASA that may be of particular interest in commercial and other non-aerospace applications. Publications include Tech Briefs, Technology Utilization Reports and Technology Surveys.

Details on the availability of these publications may be obtained from:

SCIENTIFIC AND TECHNICAL INFORMATION OFFICE

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

Washington, D.C. 20546