LOOK/3000
A New Real-Time System Performance Monitoring Tool

Kim D. Leeper
Wick Hill Associates Limited

INTRODUCTION

All programs resemble one another. This might seem to be a rash statement but let us examine the facts. Programs were designed to perform the same task over and over again; in order to do this, one must design the program to iterate through a set of data. This is true even on interactive programs. Each screen could be thought of as an input step in preparation for future major loop in the system. This paper will deal with how to identify these loops without having seen the application code.

If the reader directs his/her attention to Figure 1, s/he will see a generic application flowchart. The application in question has five loops labelled A, B, C, D and E. The number of times each loop is executed is also noted beside each loop. In order to transform this generic application flowchart into a program, the chart must be turned into a linear diagram. A linear diagram is required because a computer executes a single thread of instructions. A programmer's job is to be able to translate the two dimension flowchart into a linear series of instructions. This diagram may be seen in Figure 2.

If we were to examine the execution of this generic program over a period of time, we would find that the amount of time spent at a given location of memory would be proportional to the number of times we executed the corresponding program loop. This is graphically demonstrated in the time graph of Figure 3.

Some obscure law of computing, probably one of Murphy's Laws, tells us that the linear diagram as described above is going to be too long to fit inside the physical constraints of the computer we are programming for. This restriction presents us with an interesting problem — how to divide up the program so we can execute it on our machine. On the HP3000 this act of dividing the program up is called segmentation. Before delving into segmentation in depth, let us examine a mathematical/graphical explanation of the subject of locality.

What is Locality?

Locality is a measure of how well segmented your program is. I define it as the ratio of the number of internal PCALs to the number of external PCALs on a percentage basis. In equation form it would be the following:

$$\frac{\text{number of internal PCALs in a segment}}{\text{number of external PCALs in the same segment}}$$

The word PCAL stands for procedure call. It is the instruction generated by the language compiler when, in your application program, a CALL is made to a library routine in COBOL, a SECTION is PERFORMed or a named procedure is executed in SPL. See Figure 4 for visual assistance in understanding this concept.

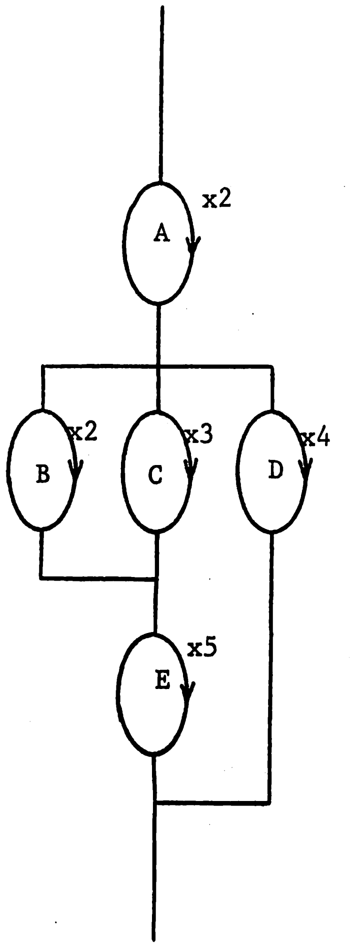
Why is Locality Important?

I am sure that you have all heard the following comment on segmentation: "Once in a segment stay there; Once out stay out." This comment is very appropriate to the HP3000 because of the significant difference in execution time between an internal PCAL and an external PCAL. Once you get into a segment, the program is advised to stay there because internal PCALs are twice as fast in execution compared with external PCALs. An internal PCAL/EXIT pair takes about 13 micro-seconds to execute on a Series II/III. An external PCAL/EXIT pair takes approximately 27 micro-seconds to execute on a Series II/III if the target code segment is in memory. If the code segment has to be read externally from the disk, then a disk access has to be added which brings the execution time up to 35-45 milli-seconds. This is obviously a significant time difference.

It is now possible to appreciate that ill-advised segmentation can have a significant adverse effect on the performance of the application program in question. You should aim to program for a maximum number of internal PCALs per segment. If you don't, you will waste time. To optimize your system from the time point of view, you must therefore segment your program appropriately.

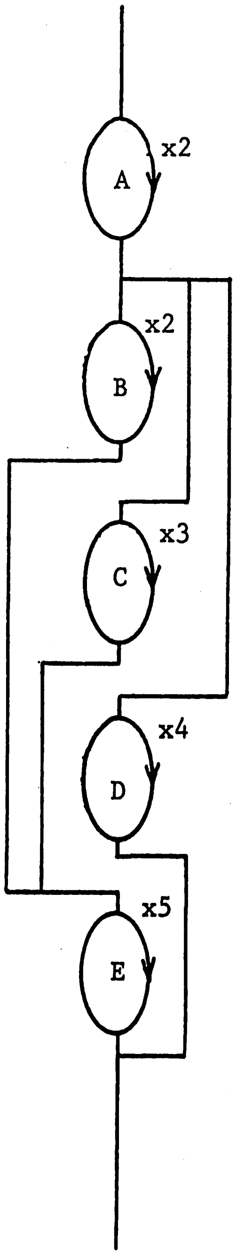
How Do You Identify If Your Program Is Appropriately Segmented?

One can use four different methods to identify if your program is properly segmented. The methods are as follows:



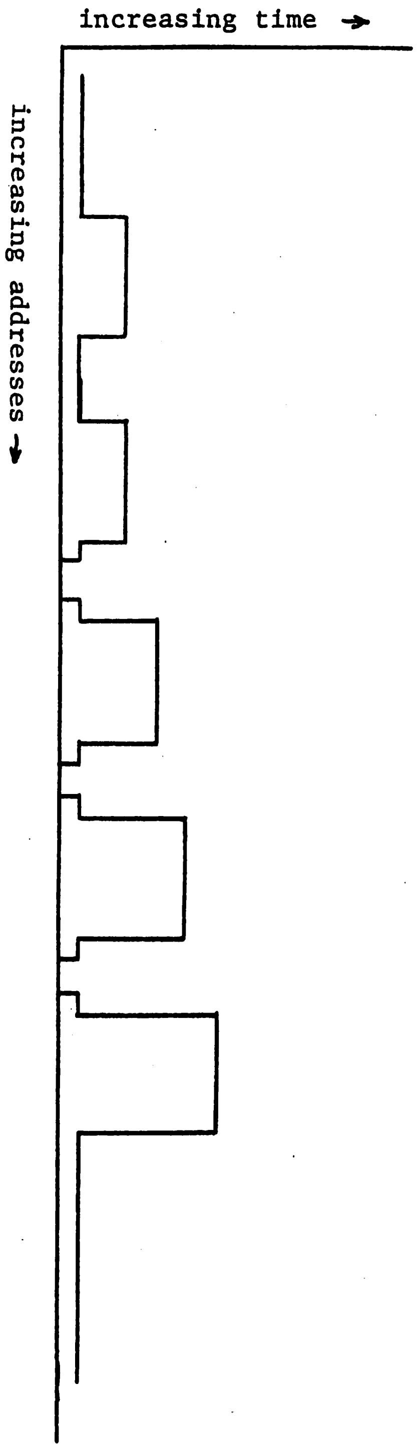
generic application
flowchart

Figure 1



linear flow
diagram

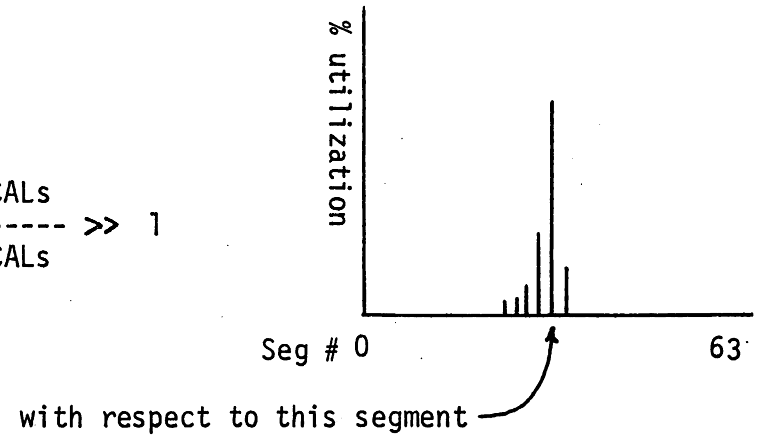
Figure 2



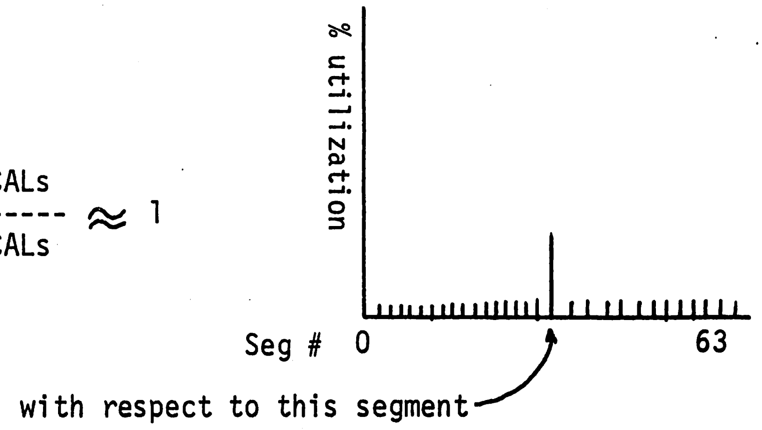
time graph

Figure 3

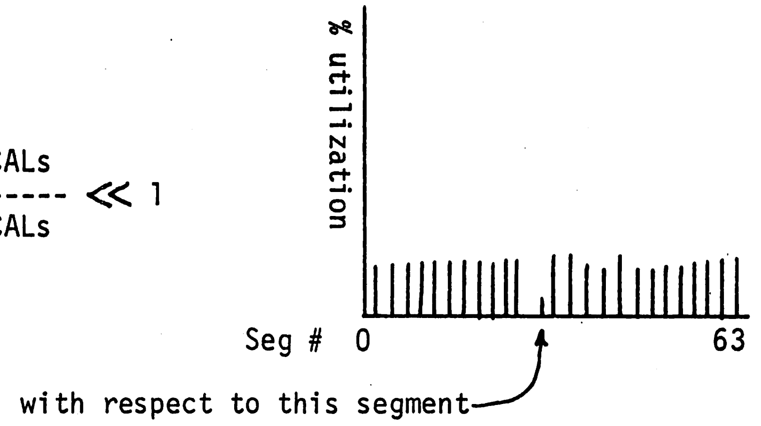
$$\frac{\text{internal PCALs}}{\text{external PCALs}} \gg 1$$



$$\frac{\text{internal PCALs}}{\text{external PCALs}} \approx 1$$



$$\frac{\text{internal PCALs}}{\text{external PCALs}} \ll 1$$



Locality Diagrams

Figure 4

1. use PROGSTAT out of the contributed library
2. use your eye
3. use programmer placed counters in the code
4. use LOOK/3000

PROGSTAT is an interesting program designed to provide a picture of how a program is segmented. It produces a list of external system references that the program under examination calls. It gives a separate count of external segment references that are satisfied within the program file itself. It also gives the segment lengths both actual and in a graphic form so one can balance the code lengths. PROGSTAT, unfortunately, does not provide a count of internal calls per segment so the designer may not calculate the locality profile as described earlier. PROGSTAT does not provide enough information to properly resegment one application program.

The oldest method available to the designer to segment his/her application code is to use the eye. This technique is dependent upon the experience of the individual using it. It is prone to error. The technique is time consuming. It is always biased. The designer might not realize that the data flow occuring in real life is not the way s/he imagines it to be. If the designer does not correctly segment the program for the flow of data emanating from the user, then the program might as well not be segmented. But how does the designer determine the character of the data flow? This brings us to the next method.

Placing counters in the code is a way of gathering information to help in the determination of appropriate segmentation. By judicious placement of the counters, we may determine an execution profile to assist us in proper segmentation. However, this technique is fraught with problems as well. The counters need to be initialized; they will need debugging; they will require stack space, and they will interfere with execution. The worst problem they introduce is that in a tight application program, placing counters in the code might require resegmentation just to get the program to run.

If we can use this method, we do gather some information as to how the data flow is causing the program to execute in a particular number of segments. The data from the counters could be used to draw some locality diagrams to assist in the resegmentation process.

The last method of determining if your program is segmented correctly is to use a system called LOOK/3000. This is a software tool provided by Wick Hill Associates Limited. LOOK produces the locality diagrams as shown on the next few pages. The biggest advantage LOOK provides is to allow the designer to watch the way the application is really being used with real data in real time, so when s/he resegments the application s/he has some assurance that the segmentation corresponds to the way the program is really being used. See Figures 5-9 regarding the displays LOOK produces.

Figure 5 is the display which allows the user to pick out the program/process that s/he wishes to examine in depth. This display is called the "SPECIFY PROGRAM SCREEN." The user identifies the process that s/he wants to examine by noting the PIN of that process. The PIN is known as the process identification number and is the number by which MPE manages your program. This PIN is entered and LOOK starts to acquire data regarding the process so identified. Every 10 seconds the display is updated. After the user enters the PIN of the process to be examined, the next screen is shown.

The next screen is called the "SEGMENT MAP SCREEN." This screen may be seen in Figure 6. It is the Locality Diagram of the program which was identified in the previous screen. This display helps the user to gain an understanding of the interaction between segments that make up the program under examination. This screen will be updated every 10 seconds. The user is now required to choose which segment s/he wishes to examine more closely. In this example segment number 5 was chosen.

The display shown next is Figure 7. It is called the "UNIQUE SEGMENT SCREEN." It is an overall map of CPU activity in this segment. In this example the user was only able to get two data points in this segment. Had the user waited longer, s/he would have acquired more data regarding segment number 5. The display shows that since the user chose to examine this segment, the CPU has executed two instructions in this segment. The first instruction is located between %1000 and %1377. The second instruction is located between %2400 and %2777. The user may choose any location to

```
FILE                                USER                                STACK        NUM  CPU
NAME                                NAME                        Q  J/S#  SIZE   PIN# SEGS TIME
------------------------------ -----------------  - ---- ----- ---- ---- ----
ENTRY    .PUB       .SYS        SIMON    .GOLUB     C S389  4924    16    1  144
ORDRNTRY.SKORDS    .GOLUB      SIMON    .GOLUB     C S388 17036    20   42  0
ORDRNTRY.SKORDS    .GOLUB      SIMON    .GOLUB     C S364 17164    22   42  271
C.I.<LISTF                    ) SIMON    .GOLUB     C S407  3036    24    0  0
SAR1350  .PUB       .GOLUB      SIMON    .GOLUB     D J212  3612    27    2  4643
APG3     .PUB       .WHA        MANAGER .WHA       C S406 23684    30    4  413
C.I.<REPLY 30,7               ) MANAGER .SYS       C S380  2140    32    0  0
ENTRY    .PUB       .SYS        SIMON    .GOLUB     C S403  4924    33    1  62
```

Figure 5

```
PROGRAM NAME = ORDRNTRY.SKORDS   .GOLUB            TOTAL NUMBER OF SEGMENTS = 42

SEGMENT NUMBER = %05                               LENGTH OF SEGMENT = %003130

ADDRESS = %002400

THIS PROGRAM WAS RUN ON A SERIES III

        100|             *
         95|             *
         90|             *
         85|             *
         80|             *
         75|             *
         70|             *
         65|             *
% TIME   60|             *
SPENT    55|             *
AT       50|             *
ADDRESS  45|             *
         40|             *
         35|             *
         30|             *
         25|             *
         20|             *
         15|             *
         10|             *
          5|             *
LOCATION IN+-----------------------------------------------------------------
SEGMENT ->  0000000011111111222222223333333344444444555555556666666677777777
  002400 +  0123456701234567012345670123456701234567012345670123456701234567

TOTAL NUMBER OF SAMPLES IS 1
```

Figure 6

```
PROGRAM NAME = ORDRNTRY.SKORDS   .GOLUB            TOTAL NUMBER OF SEGMENTS = 42

SEGMENT NUMBER = %05                               LENGTH OF SEGMENT = %003130

ADDRESS = %002400

THIS PROGRAM WAS RUN ON A SERIES III

        100|
         95|
         90|
         85|
         80|
         75|
         70|
         65|
% TIME   60|
SPENT    55|
AT       50|  *  *
ADDRESS  45|  *  *
         40|  *  *
         35|  *  *
         30|  *  *
         25|  *  *
         20|  *  *
         15|  *  *
         10|  *  *
          5|  *  *
           +-----------------------------------------------------------------
LOCATION IN                 111111111111111122222222222222223333333333333333
SEGMENT  -> 0011223344556677001122334455667700112233445566770011223344556677
 X 100      0404040404040404040404040404040404040404040404040404040404040404

TOTAL NUMBER OF SAMPLES IS 2
```

Figure 7

```
PROGRAM NAME = ORDRNTRY.SKORDS  .GOLUB             TOTAL NUMBER OF SEGMENTS = 42

SEGMENT NUMBER = %05                               LENGTH OF SEGMENT = %003130

ADDRESS = %002400

THIS PROGRAM WAS RUN ON A SERIES III

        100|              *
         95|              *
         90|              *
         85|              *
         80|              *
         75|              *
         70|              *
         65|              *
% TIME   60|              *
SPENT    55|              *
AT       50|              *
ADDRESS  45|              *
         40|              *
         35|              *
         30|              *
         25|              *
         20|              *
         15|              *
         10|              *
          5|              *
           +---------------------------------------------------------
LOCATION IN       0011223344556677        0011223344556677   IN
SEGMENT -> 002000+0404040404040404 003000+0404040404040404   OCTAL
                  0000000000000000        0000000000000000

 TOTAL NUMBER OF SAMPLES IS 1
```

Figure 8

```
PROGRAM NAME = ORDRNTRY.SKORDS  .GOLUB             TOTAL NUMBER OF SEGMENTS = 42

SEGMENT NUMBER = %05                               LENGTH OF SEGMENT = %003130

ADDRESS = %002400

THIS PROGRAM WAS RUN ON A SERIES III

        100|
         95|
         90|
         85|
         80|
         75|
         70|
         65|
% TIME   60|
SPENT    55|
IN       50|
SEGMENT  45|
         40|                                     *
         35|                                     *
         30|                                     *
         25|                                     *
         20|     *                               *
         15|     *           *                   *
         10|     * *         *                   *
          5|     * *   *     *                  **
           +----------------------------------------------------------------
SEGMENT #           1       2       3       4       5       6       7
FROM PMAP-> 012345670123456701234567012345670123456701234567012345670123456

 TOTAL NUMBER OF SAMPLES IS 18
```

Figure 9

look at but in this example s/he chose to examine the octal location 2400. This value is input and the user is now shown a more detailed display located around the location %2400.

This more detailed screen is called the "KILO-WORD WINDOW ON SEGMENT." Figure 8 shows this display very clearly. This display allows the user to determine if s/he wishes to continue examining this location in greater detail or whether s/he wants to move to another. The next screen shows this location at an expanded scale.

The new display is called the "CENTI-WORD WINDOW ON SEGMENT" screen. It gives such a detailed view of what is happening in the segment that the user may read off the actual address of where LOOK caught your application program. This is shown graphically in Figure 9. The address of the data point is %2415. This is where LOOK found the application program. If one examined the program over a longer period of time, then one could obviously get a much better picture of where the program under examination is spending its time.

By judicious use of LOOK/3000, one may locate down to the instruction address, where the application program is spending its time. Once this fact has been discovered what can be done about it?

ACTION PLAN

Three things can be done in order to improve the locality characteristics of an application program. These are as follows:

1. recode parts of the application code more efficiently
2. duplicate code modes by making them internal PCALs with judicious use of INCLUDES or COPYLIBS
3. resegment your application code

Where do we direct our attention to begin with? This is where the proprietary software tool called LOOK/3000 is invaluable. LOOK displays clearly where the application program is spending a large percentage of its execution time. As other authors have noted, programs spend 90% of their time executing 10% of the code. The trick is to identify which portion of the code you are spending your time in.

Once the offending part of the code is identified you could recode that part of your application. You might find your code executing a particular DBGET in the application program. Closer examination reveals the DBGET is acting on a data with a sorted chain that you thought had been removed months ago. You modify the schema to remove the sorted chain and the program now has a different profile because the application is not waiting for IMAGE to read down the chain.

Alternatively one could find the application code constantly calling another segment in your application. Closer examination would possibly indicate that the routine in constant use is a small one. The decision could be made to put this routine in a COPYLIB or INCLUDE file that could be inserted at compile time by the language translation. This action would make the code segment slightly larger but would remove an external PCAL.

LOOK/3000 can also help in the process of resegmentation of an existing program with an outdated design, i.e., where the application usage has changed.

CONCLUSION

As this paper has shown, poor segmentation on the HP3000 is an important contributor to poor application performance. We have seen various techniques to identify where a program is spending its time. We have seen the transformation between locality diagrams and segmentation. Various techniques have been offered regarding the evaluation of segmentation of a given application program. The most straightforward way of determining where you are spending your execution time is to use LOOK/3000. This is a proprietary software tool available from Wick Hill Associates, Ltd.

REFERENCES

1. Author unknown, "Segmentation for Maximum Efficiency of System Tape Programs," Communicator Number 5.
2. Author unknown, "Segmentation in COBOL," Communicator Number 12.
3. Author unknown, "Software Optimization Through Segmentation," Proceedings of IUG, February 1975.
4. Robert Green, "Principals for Optimizing Performance of On-Line Programs," HPGSUG Vol. II, No. 2, 1978.
5. Jim Squires and Ed Splinter, "System Performance Measurement and Optimization," Proceedings of IUG, November 1978.
6. Gerry Wade, "Programming for Survival," Proceedings of IUG, November 1978.
7. Rodney V. Smith, "Application Design for the HP3000," Proceedings of SCRUG, September 1980.
8. Robert Green, "HP3000/Optimizing Batch Jobs," Proceedings of IUG, April 1981.
9. Author unknown, "Application Design Course," HP Part # 22808A, November 1980.
10. Author unknown, "Application Design and Optimization for the HP3000," SE reference document, June 1978.
11. Jon W. Henderson, "Design and Segmentation Techniques for Large SPL Programs," Proceedings of IUG, February 1980.