MAKING SHORT SHRIFT OF SORTS

Charles Sullivan
RunningMate Software
3001 I Street
Sacramento, California 95816

INTRODUCTION

This discussion examines the variables that affect sort performance. Benchmarks will be presented which compare both the hardware and the software of the various HP3000 systems. An attempt is made to compare HP3000 sort performance to the Digital Equipment VAX 11/780.

ABOUT THE BENCHMARKS

Most of the benchmarks shown here were run several times to ensure that the results presented are reasonable. Unless otherwise noted, the Series 48 and Series 70 had MPE disc caching turned on, the Series 48 had three megabytes of main memory, the Series 70 had eight, and the Series 950 had thirty-two. Each system had two or three disc drives which were a combination of 7933H and 7937H discs. The Series 950 was running the 1.0 release of MPE XL and the other two were using MPE V, either UB-Delta-1 or V-Delta-1.

MPE DISC CACHING

MPE disc caching, as implemented by MPE V software, has an interesting effect on sort performance. When there is plenty of stack space for the sort intrinsics, disc caching actually degrades performance. But when stack space becomes relatively scarce, disc caching speeds up sort performance, sometimes very dramatically.

For these tests, whose results are shown on the next page, the parameters for MPE disc caching, when turned on, were: sequential fetch quantum = 96 sectors; random fetch quantum = 16 sectors; block on write = no. The computer was a Series 70 and 100,000 records were processed.

TABLE 1: EFFECT OF DISC CACHING ON SORT PERFORMANCE

CPU TIME DATA	CPU seconds	
	MPE disc caching	
Words available for Sort/V workspace	OFF	ON
24,400	128	163
20,000	134	181
16,000	143	204
12,000	159	234
8,000	190	309
4,000	312	568

WALL TIME DATA	Elapsed seconds	
	MPE disc caching	
Words available for Sort/V workspace	OFF	ON
24,400	346	366
20,000	392	417
16,000	444	490
12,000	584	498
8,000	838	770
4,000	1886	1053

PROCESSOR COMPARISON

As you would expect, the more powerful the CPU, the quicker the sort. At least this is true between the MPE V computers. But the Series 950 is quite a contrast to the Series 70. A sort running with no competing jobs will usually finish sooner on the Series 950, but will consume more CPU resources. One supposes that the "fault" for this lies in the software of the 950, rather than in the hardware.

TABLE 2: EFFECT OF SYSTEM PROCESSOR ON PERFORMANCE

CPU TIME DATA

Number of 128-byte records	CPU Minutes		
	Series 48	Series 70	Series 950
10,000	0.74	0.22	
20,000	1.61	0.49	
30,000	2.52	0.76	
40,000	3.51	1.07	
50,000	4.48	1.40	2.02
60,000	5.45	1.72	
70,000	6.54	2.03	
80,000	7.49	2.35	
90,000	8.54	2.67	
100,000	9.56	3.01	4.20
200,000		7.00	9.21
300,000		10.54	14.15
400,000		14.60	19.83
500,000		18.05	25.08

WALL TIME DATA

Number of 128-byte records	Total Elapsed Minutes		
	Series 48	Series 70	Series 950
10,000	1.20	0.58	
20,000	2.61	1.24	
30,000	4.10	1.95	
40,000	5.66	3.00	
50,000	7.35	3.85	2.18
60,000	9.09	4.69	
70,000	10.70	5.64	
80,000	12.42	6.46	
90,000	13.94	7.05	
100,000	15.65	7.73	5.28
200,000		17.20	18.04
300,000		29.58	27.13
400,000		39.95	36.51
500,000		50.56	45.97

ALGORITHM COMPARISON

Hewlett-Packard's standard sort package uses Floyd's Treesort algorithm while SortMatePlus uses Singleton's Quickersort variant. Another, equally important, consideration is that HP's sort confines itself to the user's stack while SortMate uses extra data segments.

For these tests, 100,000 128-byte records were sorted with MPE disc caching turned on. The sort was initialized by calling SORTINIT, the records were sent to the sort with the SORTINPUT procedure, and records were retrieved by the SORTOUTPUT procedure.

TABLE 3: EFFECT OF ALGORITHM ON SORT PERFORMANCE

CPU TIME DATA		CPU Seconds	
Processor	Work Space	HP Sort	SortMatePlus
Series 48	16K words	638	380
Series 48	25K words	523	380
Series 70	16K words	204	114
Series 70	24K words	163	114

WALL TIME DATA		Total Elapsed Seconds	
Processor	Work Space	HP Sort	SortMatePlus
Series 48	16K words	1034	482
Series 48	25K words	800	482
Series 70	16K words	490	189
Series 70	24K words	366	189

THE VAX, THE HP3000, AND BACK-END PROCESSORS

For better or worse, Digital Equipment's VAX 11/780 has become an industry-standard reference point. For years, it was stated that the VAX 11/780 was rated at about one million instructions per second (1 MIP). [I use the MIP only because it is a widely-used way to compare different computers.] Now most observers believe that the 11/780 executes at about 0.5 MIP in a commercial processing environment. The upshot of all this is that the HP3000 Series 68 and 70, which were always considered merely the equal of the VAX 11/780, can now be seen as clearly superior machines.

The VAX benchmarks are taken from the February 1, 1986 issue of Computer Design, *Database Accelerator System Relieves Sorting Bottlenecks,* by Walter A. Foley. Mr. Foley is president of Accel Technologies (San Diego). Accel makes the DBA 1000, a specialized sorting machine which can be used to off-load a host processor. In the following benchmarks, the DBA 1000 was attached to the VAX 11/780 via Ethernet. According to Mr. Foley, the Ethernet connection was not a bottleneck in the test, rather it was the speed of the VAX file system which prevented even better results for the DBA 1000 benchmark.

TABLE 4: VAX 11/780 VS. HP3000 SORT PERFORMANCE

HOST CPU TIME DATA	Host CPU seconds		
	Number of 20-byte records		
Sort Environment	50,000	250,000	500,000
VAX 11/780	100	500	1150
VAX 11/780 and DBA 1000	20	30	50
HP3000/70 [Sort/V]	48	270	609
HP3000/70 [SortMate]	29	161	323
HP3000/950 [Sort/XL]	27	153	312

WALL TIME DATA	Total Elapsed seconds		
	Number of 20-byte records		
Sort Environment	50,000	250,000	500,000
VAX 11/780	550	2600	5700
VAX 11/780 and DBA 1000	120	225	600
HP3000/70 [Sort/V]	69	429	900
HP3000/70 [SortMate]	35	205	413
HP3000/950 [Sort/XL]	27	177	480

SORTING PECULIARITIES AND TIPS

Sort/V grabs all its necessary resources when you call SORTINIT. If your system is out of disc space, you will know immediately. This is better than having to wait for two hours before finding out that you need to free up some

more disc sectors. However, this has an unwanted side-effect which is caused by the way the file system allocates disc file extents. When you allocate all the extents for a file when it is created, all the extents must reside on a single disc drive. Therefore, allocating all extents at once will increase the probability of having your job flushed because you are "out of disc space." To eliminate this problem with the Hewlett-Packard sort, simply issue the following file equation:

```
:FILE SORTSCR;DEV=,32,1
```

The COBOL compiler, probably for simplicity and reliability, opens files for buffered access. This means that the sort intrinsics will probably find enough room on the stack for their work area, but it also means that a COBOL file-to-file sort will almost always run slower than necessary. (SortMatePlus, which replaces the sort intrinsics, will attempt to re-open buffered files for MR-nobuff access. This can lead to a measurable speed increase.) If you can, you should remove sorts from within COBOL programs and use a sort utility program such as SORT.PUB.SYS or SortMate.

Sort/V allows you to alter the collating sequence. If you need to sort upper- and lower-case letters properly, using an alternate collating sequence is necessary. Here is how you do it.

```
:RUN SORT.PUB.SYS
>DATA IS ASCII SEQUENCE IS ASCII
>ALTSEQ MERGE "A-Z" WITH "a-z"
```

Pretty simple, no? However, using an alternate collating sequence does slow down the sort process, but that's another story.

ANOTHER STORY

When developing SortMatePlus, I needed to allow for alternate collating sequences. There is a powerful machine instruction which is tailor-made for just such a purpose: the "compare translated strings" [CMPT] instruction (which is probably used by Sort/V). After about 4 hours of puzzlement and growing frustration, I concluded that CMPT does not work in split-stack mode. Knowing that I would encounter difficulties trying to convince Hewlett-Packard to modify the microcode on thirty thousand installed computers, I began writing a software routine that emulates the CMPT instruction.

On the next page you will find a program which makes use of the final software routine which emulates the CMPT machine instruction. Notice how cumbersome (and incomprehensible) it appears.

```
$CONTROL USLINIT,NOLIST
BEGIN
  INTEGER INDEX,
          KEY'POSITION:=0,   << starting position of key >>
          KEY'LENGTH:=10;    << byte length of key >>
  BYTE ARRAY BYTERECORD1(0:9):="CHARLIE001";
  BYTE ARRAY BYTERECORD2(0:9):="CHARLIE002";
  BYTE ARRAY TRANSLATIONTABLE(0:255);

  << Initialize the translation table >>

  FOR INDEX:=0 UNTIL 255 DO TRANSLATIONTABLE(INDEX):=INDEX;

  ASSEMBLE (LDX KEY'FIRSTPOSITION;
            LRA BYTERECORD1,I,X;
            LRA BYTERECORD2,I,X);
  TOS := KEY'LENGTH;
  GOTO ENTRYPOINT;
LOOP:
  ASSEMBLE (DABZ EQUAL;  LDXI 1;  LRA S-2,I,X;  STOR S-3;
            LRA S-1,I,X; STOR S-2);
ENTRYPOINT:
  ASSEMBLE (CMPB 0);
  IF = THEN GOTO EQUAL;
  ASSEMBLE (LDB S-2,I;  STAX,NOP;  LDB TRANSLATIONTABLE,I,X;
            LDB S-2,I;  STAX,NOP;  LDB TRANSLATIONTABLE,I,X;
            CMP,NOP);
  IF = THEN GOTO LOOP
  ELSE IF < THEN BEGIN  << record1 < record2 >>  END
  ELSE IF > THEN BEGIN  << record1 > record2 >>  END
  ELSE
    BEGIN
EQUAL:  << record1 = record2 >>
    END;
  ASSEMBLE (SUBS 3);  << must delete words left on stack >>
END.
```

Now an example of a program which uses the CMPT machine instruction. Notice how clear and simple the code appears in contrast to the emulation code above.

```
$CONTROL USLINIT,NOLIST
BEGIN
  INTEGER INDEX,
          KEY'POSITION:=0,   << starting position of key >>
          KEY'LENGTH:=10;    << byte length of key >>
  BYTE ARRAY BYTERECORD1(0:9):="CHARLIE001";
  BYTE ARRAY BYTERECORD2(0:9):="CHARLIE002";
  BYTE ARRAY TRANSLATIONTABLE(0:255);

  << Initialize the translation table >>

  FOR INDEX:=0 UNTIL 255 DO TRANSLATIONTABLE(INDEX):=INDEX;

  TOS := @TRANSLATIONTABLE;
  TOS := @BYTERECORD1(KEY'FIRSTPOSITION);
  TOS := KEY'LENGTH;
  TOS := @BYTERECORD2(KEY'FIRSTPOSITION);
  TOS := KEY'LENGTH;
  ASSEMBLE (CON %20477, %7);  << creates the CMPT code >>
  IF < THEN BEGIN  << record2 < record1 >>  END
  ELSE IF > THEN BEGIN  << record2 > record1 >>  END
  ELSE BEGIN  << record2 = record1 >>  END;
END.
```

The surprising fact is this: the machine-level CMPT instruction is faster than the software routine in only one case--when the first characters are not equivalent. The software routine gains its efficienty because it only goes to the translation table when necessary; the CMPT code goes to the translation table for every comparison, even when it is obviously unnecessary. For example, there is no need to go to the table if string1 is "CHARLIE" and string2 is also "CHARLIE". Similarly, you do not need to go to the translation table until the 10th byte if the first nine bytes are exactly the same.

Still, you might be wondering which way is better in reality. If, during a sort, 75% of the comparisons need to examine only one byte, then the CMPT instruction will probably be faster. So here is some "real" data to examine. I extracted all the keys from a master dataset where each key was 12 bytes long. I sorted on the first 10 bytes so some of the keys were "duplicates." These 23,553 records required 375,453 comparisons before they were sorted. Here is how the comparisons were broken down:

Comparisons decided by the 1st byte	114,383
Comparisons decided by the 2nd byte	86,971
Comparisons decided by the 3rd byte	65,911
Comparisons decided by the 4th byte	28,593
Comparisons decided by the 5th byte	50,666
Comparisons decided by the 6th byte	10,152
Comparisons decided by the 7th byte	2,197
Comparisons decided by the 8th byte	492
Comparisons decided by the 9th byte	153
Comparisons decided by the 10th byte	2,984
Comparisons in which keys were equal	13,005

About 70% of the comparisons needed to examine more than one byte, so in this case, using the CMPT instruction would be slower than using the software routine. A large proportion (probably 90%) of sorts encountered in practice will run faster with the software routine.