The E2062A Numeric Compiler For HP BASIC For Windows

v1.0 / 31 jan 96 / gvg

* The HP E2062A Numeric Compiler for HP BASIC for Windows allows you to optimize the performance of SUBs in your HP BASIC programs. It is an HPBW equivalent to the compiler available for HP BASIC workstations.

Like the HP BASIC compiler, the Numeric Compiler does not generate stand-alone executable programs; it simply takes SUBs and turns them into CSUBs that operate within the HPBW interpreter. (These are quite distinct from the CSUBs that you can produce with the E2061 CSUB toolkit that are external DLLs and are not executed from within the HPBW environment.) You can only compile SUBs; user-defined functions cannot be compiled.

* The Numeric Compiler is installed as a simple program under Windows; it is easy to use.

The first thing to do is to create the program with the SUBs you want to compile; you must first pre-run the program and STORE it (not SAVE it) as a PROG file.

When you run the Numeric Compiler, it asks you for the name of the program:

```
Source filename [.BAS]:
```

The ".BAS" simply means that will be the extension assumed if none is specfied; it's just as easy to enter the entire program name, such as "MYTEST.PRG". You then get:

```
Output filename [MYTEST.CSB]:
```

You can then either accept the default name or type in a new one. Finally, the Numeric Compiler provides you with the prompt:

```
SUB Name [MYTEST]:
```

This asks you the name of the SUB you want to compile ... MYTEST is just a guess on the part of the Numeric Compiler, you can enter what you like. If you just have one or have several SUBs and want to compile them all, just enter ALL ... and the program will be compiled, while the Numeric Compiler issues a status display. When you do a LOAD on the output file from HPBW, it will have the SUB converted to a CSUB.

* To determine the performance improvements available by use of the Numeric Compiler, I ran two benchmark programs -- one to perform a prime-number sieve test, the other to load up a set of arrays with values and then search for the maxima and minima in them. The details are not important -- the program listings are at the end of this article if you're interested; the essential thing is that they are both computation-intensive programs in which the computations can be isolated in a single SUB call.

For purposes of comparison, I ran the sieve and mat-search on three systems in both uncompiled and compiled form:

% An 82324B Measurement Coprocessor (MCP) with 16.7 MHz 68030 and FPP. This is a traditional HP BASIC engine and is included in these tests for speed comparison; compilation of the test SUBs was done with the traditional HP BASIC Compiler.

% An HP Vectra 386/25 with a floating-point unit.

% An HP Vectra XP/60 (60 MHz Pentium).

The results are as follows in seconds:

	sieve		mat-search	
	uncompiled	compiled	uncompiled	compiled
82324	6.88	1.5	17.48	9.95
386/25	12.96	0.27	29.99	3.46
XP/60	1.35	0.055*	3.40	0.27

Note that the 0.055 marked in the table above is the limit for timing on a PC ... it may have actually been faster than that.

For the test, if we assign the time for executing the uncompiled version of each test on a 386/25 the value of 1, the relative speed is:

	sieve		mat-search	
	uncompiled	compiled	uncompiled	compiled
82324	1.9	8.6	1.7	3.0
386/25	1.0	48.0	1.0	8.7
XP/60	9.6	235.0	8.8	111.1

The reason for the greater improvement in the sieve test than in the mat-search test is likely because the sieve test does a greater number of small operations ... the operations in the mat-search test are more complicated and so harder to optimize.

Note that the program has to be designed to take advantage of compilation! You have to consolidate the time-critical math operations in a SUB and avoid doing I/O or calls to uncompiled routines in that sub. If you just compile a few SUBs at random you are not likely to see much performance improvement.

* The benchmark programs are shown below:

```
10    ! ****************************************************************
20    !
30    ! Sieve Benchmark Test
40    !
70    ! This test program provides a simple basic-math
80    ! benchmark by finding the highest prime less than
90    ! a value.  Note that due to various restrictions
100   ! you can't test much above 16,000.
110   !
120   ! ****************************************************************
130   !
160   CLEAR SCREEN
170   REAL T1,T2
180   INTEGER Maxnum,Maxprime
190   Maxnum=16000
200   !
210   PRINT "Searching for highest prime less than: ";Maxnum
220   T1=TIMEDATE
230   CALL Sieve(Maxnum,Maxprime)
240   T2=TIMEDATE
250   PRINT "Highest prime: ";Maxprime
260   PRINT USING "K,3D.2D";"Time in seconds: ";T2-T1
270   !
280   END
290   !
300   SUB Sieve(INTEGER Maxnum,Maxprime)
310   !
320     INTEGER Firstprime,Lastprime,Factor,Multiple
330     Firstprime=2
340     ALLOCATE INTEGER S(Firstprime:Maxnum)
350     MAT S= (1)
360   !
370     FOR Factor=Firstprime TO Maxnum
380       IF S(Factor)=1 THEN
390         Maxprime=Factor
400         FOR Multiple=Factor+Factor TO Maxnum STEP Factor
410           IF S(Multiple)=1 THEN S(Multiple)=0
420         NEXT Multiple
430       END IF
440     NEXT Factor
450   !
460   SUBEND
```

```
470   !
480   ! ********************** That's All, Folks! ********************


10    ! **************************************************************
20    !
30    !  MAT SEARCH Test Program
40    !
70    !  This program benchmarks a matrix search required for a customer
80    !  application.
90    !
100   !  In the application, the customer reads 360 samples for 102 sine
110   !  waveforms (sample per degree) into a matrix, determines the min &
120   !  max of each waveform, and the determines the minimum and maximum
130   !  of all the mins and the minimum and maximum of all the maxes.
140   !
150   !  This is simulated here by loading up a 360x102 (INTEGER) matrix,
160   !  scanning through each of the 102 columns to find the min and max,
170   !  loading these values into two 102-element arrays, & then searching
180   !  through each of these arrays for their min and max.  The MAT SEARCH
190   !  command is used to do the search.
200   !
210   !  The test is timed between the start of the scan and the end of the
220   !  scans.
240   !
250   ! ***************************************************************
320   !
350   CLEAR SCREEN
360   REAL T0,T1
370   INTEGER Max_max,Max_min,Min_max,Min_min
380   !
390   PRINT "Beginning test, please wait."
400   !
410   T0=TIMEDATE
420   CALL Test(Max_max,Max_min,Min_max,Min_min)
430   T1=TIMEDATE
440   !
450   PRINT "Max_max = ";Max_max
460   PRINT "Max_min = ";Max_min
470   PRINT "Min_max = ";Min_max
480   PRINT "Min_min = ";Min_min
490   !
500   PRINT
510   PRINT USING "K,3D.2D";"Time = ",T1-T0
520   PRINT
530   PRINT "Done!"
540   !
550   END
560   !
590   SUB Test(INTEGER Max_max,Max_min,Min_max,Min_min)
600   !
610     INTEGER N,M,Sines(1:360,1:102),Maxes(1:102),Mins(1:102)
```

```
620   !
630   ! Fill up sines matrix with random values.
640   !
650     RANDOMIZE (TIMEDATE)
660   !
670     FOR N=1 TO 360
680       FOR M=1 TO 102
690         Sines(N,M)=INT((-2*INT(RND*2)+1)*(RND*32767))
700       NEXT M
710     NEXT N
720   !
730   ! Scan through Sines array to get min and max values.
740   !
750     FOR N=1 TO 102
760       MAT SEARCH Sines(*,N),MAX;Maxes(N)
770       MAT SEARCH Sines(*,N),MIN;Mins(N)
780     NEXT N
790   !
800   ! Scan through Maxes & Mins arrays for maxes and mins.
810   !
820     MAT SEARCH Maxes(*),MAX;Max_max
830     MAT SEARCH Maxes(*),MIN;Max_min
840   !
850     MAT SEARCH Mins(*),MAX;Min_max
860     MAT SEARCH Mins(*),MIN;Min_min
870   !
880   SUBEND
890   !
900   ! ******************** That's All, Folks! *******************
```

[<>]