System Benchmarking

By Dr. T.S. Kelso

March/April 1996

Throughout the development of this column, we have been faced with two inescapable conclusions: satellite tracking can be computationally intensive and requires careful attention to ensure accurate results. These conclusions are inescapable regardless of whether you are developing your own satellite tracking software or using someone else's.

As a result of these conclusions, it becomes important at some point to consider the issue of benchmarks to assess certain performance characteristics of any satellite tracking system. As we shall see, benchmarks can be used for a variety of purposes: evaluating the performance of a particular hardware configuration, operating system, programming language, or specific satellite tracking application. Which consideration—speed or accuracy—is most important to you will depend upon your needs, but we will address several approaches to assess each.

Which benchmark is best to use depends heavily on the application. In the area of satellite tracking, we would expect the best benchmarks to be floating-point intensive, that is, to make heavy use of real calculations as opposed to integer calculations. We would also expect them to make heavy use of trigonometric and other mathematical functions. While we could use existing standard benchmarks, the drawback to this approach is that often it is not clear what the basis of the benchmark calculation is and, hence, how appropriate it is to assessing the performance of a particular class of applications.

This column (and the next) will endeavor to provide you with a suite of benchmarks that are not only simple but will also allow you to assess performance across the spectrum of operations. We will begin by developing a set of benchmarks with known solutions. The easiest of these is something known as the Savage benchmark.

The Savage benchmark is a particularly good benchmark for the fields of satellite tracking and astronomy. The reason it is so good is that it relies entirely on the repeated use of mathematical functions which yield a known (and easily calculable) numerical result. This benchmark is based on the use of matching inverse mathematical functions. Sample code, written for Borland Pascal 7.0, is provided in figure 1 below. Note that units from the SGP4 Pascal Library have been used to time the calculation.

```
Program Savage_Benchmark;
{$N+,E+}
  Uses CRT,Support,SGP_Time,SGP_Init;

type
  test_type = extended;

var
  i,j             : word;
  a               : test_type;
  start,stop,dt   : double;
  time1,time2     : time_set;

Function Tan(arg : test_type) : test_type;
  begin
  Tan := Sin(arg)/Cos(arg);
  end; {Function Tan}

BEGIN

  Program_Initialize('SAVAGE');

  Get_Current_Time(time1);

{*** Begin main program loop ***}
  for j := 0 to 9 do
    begin
    a := 1;
    for i := 1 to 2499 >do
      a := tan(arctan(exp(ln(sqrt(a*a))))) + 1;
    end; {for j}
{*** End main program loop ***}

  Get_Current_Time(time2);

{** Calculate elapsed time **}
  with time1 do
    start := Julian_Date_of_Year(yr) + DOY(yr,mo,dy)
           + Fraction_of_Day(hr,mi,se,hu);
  with time2 do
    stop  := Julian_Date_of_Year(yr) + DOY(yr,mo,dy)
           + Fraction_of_Day(hr,mi,se,hu);
  dt := 86400*(stop - start);

{** Output elapsed time and result **}
  GotoXY(31,12);
  Write('Elapsed time ',dt:8:2,' seconds');
  GotoXY(31,13);
  Write('Answer is ',a:19:14);

  Program_End;

END.
```

Figure 1. Pascal Code for the Savage Benchmark

The heart of the calculation consists of taking the variable a, starting with a value of one, and incrementing it by one 2,499 times until a = 2,500. To make things more interesting, though, we evaluate a using a set of three matching inverse functions: square and square root, exponential and logarithm, and tangent and arc tangent. The result of each pair of matching inverse functions ***should*** be the original value of a and the result of all the calculations should equal exactly 2,500. Of course, due to limitations of the hardware, operating system, and programming language the result will not yield exactly the expected result. How close we get to the approved solution and how quickly we can calculate it are the two dimensions of this benchmark. Table 1 shows the results for a range of systems in current use for differing levels of numerical precision.

Table 1. Savage Benchmark Results for Current Systems

Precision		386DX-33[a]	486DX2-66[b]	Pentium 90[c]	Pentium 133[d]
Single	***Time***	6.70s	1.21s	0.55s	0.38s
(4 bytes, 7-8 digits)	***Result***	2476.51342773437500	Same	Same	Same
Real	***Time***	7.47s	1.43s	0.61s	0.38s
(6 bytes, 11-12 digits)	***Result***	2499.99997197091579	Same	Same	Same

[a]33 MHz 386DX running MS-DOS 6.20 with a 387 coprocessor.
[b]66 MHz 486DX2 running Windows 95 in MS-DOS mode.
[c]90 MHz Pentium running Windows 95 in MS-DOS mode.
[d]133 MHz Pentium running Windows 95 in MS-DOS mode.

Precision		386DX-33[a]	486DX2-66[b]	Pentium 90[c]	Pentium 133[d]
Double	***Time***	6.81s	1.26s	0.55s	0.38s
(8 bytes, 15-16 digits)	***Result***	2500.00000000117734	Same	Same	Same
Extended	***Time***	6.86s	1.27s	0.55s	0.38s
(10 bytes, 19-20 digits)	***Result***	2500.00000000001267	2499.99999999999981	Same	Same

[a]33 MHz 386DX running MS-DOS 6.20 with a 387 coprocessor.
[b]66 MHz 486DX2 running Windows 95 in MS-DOS mode.
[c]90 MHz Pentium running Windows 95 in MS-DOS mode.
[d]133 MHz Pentium running Windows 95 in MS-DOS mode.

The results are actually rather illuminating. Even the slowest machine beats the time of a Cray X-MP/24 from a decade ago.[1] And while the accuracy isn't quite as high as with the Cray, the adoption of standards for numerical processing has resulted in consistent results for standard data types. Finally, the use of numeric coprocessors allows high precision (certainly compared to the single-precision results) for very little additional time.

How might this benchmark be used? Well, because of its simplicity—relying entirely on number crunching—it can be used to demonstrate anything from the 18-fold improvement in performance going from a 386DX-33 to a Pentium 133, to the difference in programming languages (e.g., Pascal vs. C), or even the differences between operating systems (e.g., Windows 95 vs. Unix). The Savage benchmark is a straightforward way of assessing the kind of computational speed and accuracy required for satellite tracking precisely because it avoids measuring things like disk throughput or video performance.

Let's put aside the issue of speed for a moment and address the issue of accuracy. Of course, we've been assessing accuracy throughout the history of this column. Each time we've presented the theory behind a particular aspect of satellite tracking, we've followed up with a specific numerical example to ensure you can implement the theory from start to finish. These examples are usually fairly simple because we're only looking at a small piece of the larger picture. However, as we pull these smaller pieces together, it becomes increasingly important to be able to assess the accuracy of the resulting complex procedures. We do this through the use of standard test cases.

An example of a standard test case would be to provide element sets for a particular orbital model (e.g., NORAD two-line element sets for the SGP4 orbital model) and the output from a known correct implementation of the orbital model. An example of such test cases is included in the appendix of ***Spacetrack Report Number 3***.[2] The sample test cases in this report include an element set for one near-earth and one deep-space satellite and the resulting SGP4 state vectors (ECI position and velocity) at points over a specific time interval. These test cases can be used to verify the proper implementation of the SGP4 (near-earth) and SDP4 (deep-space) portions of the current NORAD orbital model for a particular satellite tracking application.

For example, we can verify the implementation of the SGP4 model used by ***TrakStar*** by running the code in figure 2 with the data in ***Spacetrack Report Number 3*** (included in the code).

```
Program SGP4_Test;
{$N+}
  Uses CRT,SGP_Intf,
       SGP_Init,SGP_Conv,
       SGP_Math,SGP_Time,
       SGP4SDP4;

var
  satnumber,interval,i : integer;
  delta,tsince,k1,k2   : double;
  pos,vel              : vector;

BEGIN

  sat_data[1,1] := '1 88888U          80275.98708465  .00073094  13844-3  66816-4 0    8 ';
  sat_data[1,2] := '2 88888  72.8435 115.9689 0086731  52.6988 110.5714 16.05824518  105 ';
  sat_data[2,1] := '1 11801U          80230.29629788  .01431103  00000-0  14311-1       ';
  sat_data[2,2] := '2 11801  46.7916 230.4354 7318036  47.4722  10.4117  2.28537848      ';

  delta := 360;
  for satnumber := 1 to 2 do
    begin
    ClrScr;
    Writeln(sat_data[satnumber,1]);
    Writeln(sat_data[satnumber,2]);
    Writeln;
    Writeln('     TSINCE              X                Y                Z');
    GotoXY(1,12);
    Writeln('                       XDOT             YDOT             ZDOT');
    Convert_Satellite_Data(satnumber);
    for interval := 0 to 4 do
      begin
      tsince := interval * delta;
      if ideep = 0 then
        begin
        GotoXY(1,4);
        Write('SGP4');
        SGP4(tsince,iflag,pos,vel);
        end {if SGP4}
      else
        begin
        GotoXY(1,4);
        Write('SDP4');
        SDP4(tsince,iflag,pos,vel);
        end; {else SDP4}
      Convert_Sat_State(pos,vel);
      GotoXY(1,6+interval);
      Writeln(tsince:16:8,pos[1]:17:8,pos[2]:17:8,pos[3]:17:8);
      GotoXY(1,14+interval);
      Writeln('                ',vel[1]:17:8,vel[2]:17:8,vel[3]:17:8);
      end; {for int}
    repeat until keypressed;
    end; {for satnumber}

END.
```

Figure 2. SGP4 Test Program

The results show agreement at the meter-level in position and millimeter/second-level in velocity. Most of the disparity comes from refinements to the constants used in the model together with modifications to the code since its initial release. Obviously, it is important to have good current test cases to use to verify your software. Perhaps not so obvious is the need to have a more diverse set of orbital elements to test against. Such a set would go further toward testing all aspects of the complicated SGP4 model and provide better confidence in a particular implementation.

Now that we have at least a basic means of assessing the overall accuracy of a particular implementation of an orbital model, let's return to the question of speed. Our ultimate benchmark would be to run a standard set of orbital elements—for satellites in various orbits—over a specific interval and count how many state vectors can be computed per unit time. That is exactly what SGP4-BM does. The code in figure 3 is run with the NORAD two-line orbital elements in figure 4 to calculate the state vectors for each satellite at one-minute intervals for the entire day of 1993 March 11. By measuring how long it takes to do these 14,400 calculations, we come up with a figure of how many SGP4 calculations per minute are computed.

```
Program SGP4_Benchmark;
{$N+,E+}
  Uses CRT,Support,
       SGP_Init,SGP_In,
       SGP_Time,SGP_Math,
       SGP_Conv,SGP4SDP4;

var
  i,j             : word;
  nr_sats         : word;
  start,stop,
  jtime,jt,dt     : double;
  time1,time2     : time_set;
  sat_pos,sat_vel : vector;

BEGIN

  Program_Initialize('SGP4-BM');
  nr_sats := Input_Satellite_Data('BM.TLE');

  jtime := Julian_Date_of_Epoch(93070.00000000);
  dt := 1/1440;
  Get_Current_Time(time1);
  for i := 0 to 1439 do
    begin
    jt := jtime + i*dt;
    for j := 1 to nr_sats do
      begin
      Convert_Satellite_Data(j);
      SGP(jt,sat_pos,sat_vel);
      end; {for j}
    end; {for i}
  Get_Current_Time(time2);
  with time1 do
    start := Julian_Date_of_Year(yr) + DOY(yr,mo,dy)
           + Fraction_of_Day(hr,mi,se,hu);
  with time2 do
    stop  := Julian_Date_of_Year(yr) + DOY(yr,mo,dy)
           + Fraction_of_Day(hr,mi,se,hu);
  dt := 1440*(stop - start);
  GotoXY(31,12);
  Write(((i+1)*j/dt):3:1,' SGP4/SDP4 calculations per minute');

  Program_End;

END.
```

Figure 3. SGP4 Benchmark (SGP4-BM.PAS)

```
LAGEOS
1 08820U 76039  A 93 69.12582531  .00000003  00000-0  99999-4 0  5849
2 08820 109.8630 336.6223 0043970  23.8633 336.4101  6.38664583137514
LandSat 5
1 14780U 84 21  A 93 71.16209231  .00000492  00000-0  11946-3 0  5006
2 14780  98.2246 133.3985 0003724 251.8708 108.2062 14.57106319480190
GPS-0009
1 15039U 84 59  A 93 68.04144710 -.00000007  00000-0  99999-4 0  8021
2 15039  63.6933  55.6304 0040393 227.8733 131.7914  2.00565485 64016
Mir
1 16609U 86 17  A 93 71.36410459  .00025269  00000-0  31433-3 0  9324
2 16609  51.6219 345.5680 0002409 120.8834 239.2635 15.59788436404037
GOES 7
1 17561U 87 22  A 93 71.23788560 -.00000018  00000-0  99999-4 0  3250
2 17561   0.5235  82.3851 0003474 317.7319 102.9778  1.00272013  5345
GRO
1 21225U 91 27  B 93 70.66224332  .00044046  00000-0  34280-3 0  8399
2 21225  28.4496 286.8479 0005651 140.2221 219.8776 15.70247193109885
NOAA 12
1 21263U 91 32  A 93 70.80172638  .00000350  00000-0  17506-3 0  5225
2 21263  98.6674 102.5932 0013574 133.3761 226.8548 14.22215020 94807
TOPEX
1 22076U 92 52  A 93 70.75194823 -.00000001  00000-0  99999-4 0  1952
2 22076  66.0404 171.5267 0007615 271.2903  88.7237 12.80930695 27266
LAGEOS II
1 22195U 92 70  B 93 70.21699884 -.00000004  00000-0  99999-4 0   415
2 22195  52.6682  26.2785 0137366 275.1033  83.2951  6.47293616  8973
GPS BII-16
1 22231U 92 79  A 93 69.49138886  .00000011  00000-0  99999-4 0   454
2 22231  54.8230 217.5279 0039349 297.7681  61.7678  2.00553950  2203
```

Figure 4. Element Sets for SGP4 Benchmark (BM.TLE)

For the four systems we looked at earlier, the results are compiled in table 2 below.

Table 2. SGP4 Benchmark Results

	386DX-33	486DX2-66	Pentium 90	Pentium 133
Calculations per minute	9,285.3	43,221.6	113,089.1	167,441.2

It should be obvious that this benchmark is best suited to determining the number-crunching ability of any satellite tracking system since that is exactly what it tests. Interestingly enough, though, we see the same 18-fold improvement between the 386DX-33 and the Pentium 133 with SGP4-BM as we did with the much simpler Savage benchmark.

If you think we haven't quite finished our discussion of satellite tracking benchmarks yet, you're absolutely right. While we've looked at system benchmarks, we haven't even addressed the need for benchmarking against real-world data. System benchmarks can only verify that our application is consistent with existing models but cannot validate that the application actually works. Next time we'll look at some real-world data sets and see how to test an application against that data using ***TrakStar*** as an example. We will also discuss various types of real-world data which may be used for this purpose, depending on your requirements.

As always, if you have questions or comments on this column, feel free to send me e-mail at TS.Kelso@celestrak.com or write care of ***Satellite Times***. Until next time, keep looking up!

Notes

[1] T.S. Kelso, "Astronomical Computing Benchmarks," ***Sky & Telescope,*** March 1986.

[2] Felix R. Hoots and Ronald L. Roehrich, ***Spacetrack Report No. 3: Models for Propagation of NORAD Element Sets***, December 1980.

Dr. T.S. Kelso [TS.Kelso@celestrak.com]
Follow CelesTrak on Twitter @TSKelso
Last updated: 2019 Dec 28 00:04:15 UTC