Jedi SAS Tricks: Warp Speed DATA Steps with DS2

Mark Jordan | APRIL 25, 2015 | Edit

I remember the first time I was faced with the challenge of parallelizing a DATA step process. It was 2001 and SAS V8.1 was shiny and new. We were processing very large data sets, and the computations performed on each record were quite complex. The processing was crawling along on impulse power and I felt the need - the need for warp speed!

From the SAS log we could see that elapsed time was almost exactly equal to CPU time, so we surmised that the process was CPU bound. So with SAS/CONNECT licensed on our well-provisioned UNIX SAS server and an amazing SUGI paper extolling the virtues of parallel processing with MPCONNECT in hand, we set out chart a course in this brave, new world. The concept behind MPCONNECT is to write a SAS control program that breaks your data up into smaller pieces, spawns several identical DATA step jobs to process the pieces in parallel, monitors progress until they all finish, then reassembles the individual outputs to obtain the final results. Labor intensive, for sure, but it definitely accelerated processing of CPU-bound jobs.

But now I have SAS9.4 with the new DS2 programming language. This was built from the ground up with threading in mind - and suddenly parallel processing with the DATA step just became a whole lot easier! For example, here is a (senseless, I'll admit) CPU intensive base SAS DATA step program:

```
data t1;
   array score[0:100];
   set t END=LAST;
   do i=LBOUND(SCORE) to hbound(score);
      Score[i]= (SQRT(((id * ru * rn) / (id + rn + ru))*ID))*
                (SQRT(((id * ru * rn) / (id + rn + ru))*ID));
   end;
   count+1;
   if last then put 'Data step processed ' count 'observations.';
   drop i count;
run;
```

When executed, this process consumes about the same amount of CPU time as elapsed time:

```
NOTE: DATA statement used (Total process time):
      real time           5.20 seconds
      cpu time            5.11 seconds
```

I suspect the process is CPU bound and could benefit from threading. First, I'll try this as a straight DS2 DATA step:

```
proc ds2;
data t2/overwrite=yes;
   dcl bigint count;
   drop count;
   vararray double score[0:100] score0-score100;
   method run();
      dcl int i;
      set t;
      do i=LBOUND(SCORE) to hbound(score);
         Score[i]= (SQRT(((id * ru * rn) / (id + rn + ru))*ID))*
                   (SQRT(((id * ru * rn) / (id + rn + ru))*ID));
      end;
```

```
      count+1;
   end;
   method term();
      put 'DS2 Data step processed' count 'observations.';
   end;
enddata;
run;
quit;
```

This process is still running single-threaded, and uses about the same resources and elapsed time as the original, with a little extra (as expected) for the PROC overhead:

```
NOTE: PROCEDURE DS2 used (Total process time):
      real time           5.98 seconds
      cpu time            5.86 seconds
```

Now, let's convert the process to a thread. First we create the THREAD program, which will be stored in a SAS library. I'm going to store it in WORK in this case. To convert the DS2 DATA step to a THREAD step, I'll simply change the DATA statement to a THREAD statement and the ENDDATA statement to ENDTHREAD:

```
proc ds2;
thread th2/overwrite=yes;
   dcl bigint count;
   drop count;
   vararray double score[0:100] score0-score100;
   method run();
      dcl int i;
      set t;
      do i=LBOUND(SCORE) to hbound(score);
         Score[i]= (SQRT(((id * ru * rn) / (id + rn + ru))*ID))*
                   (SQRT(((id * ru * rn) / (id + rn + ru))*ID));
      end;
      count+1;
   end;
   method term();
      /*Make each thread report how many obs processed*/
      put 'Thread' _threadid_ ' processed' count 'observations.';
   end;
endthread;
run;
quit;
```

Executing that program creates the thread and stores it in the WORK library in a dataset named th2. Now to write a short DARA step program to execute 4 of the threads in parallel:

```
proc ds2;
/*Multi-threaded*/
data th4/overwrite=yes;
   dcl thread th2 t;
   method run();
   set from t threads=4;
   end;
enddata;
run;
quit;
```

And the clock time is significantly reduced, at the expense of extra CPU time. Note that the CPU time is longer than the elapsed time indicating operations were conducted in parallel. The routine in the thread's TERM method reports how many observations each thread processed.

```
Thread 3  processed 281152 observations.
Thread 2  processed 219648 observations.
Thread 1  processed 294528 observations.
Thread 0  processed 204672 observations.
NOTE: PROCEDURE DS2 used (Total process time):
      real time           3.20 seconds
      cpu time            9.20 seconds
```

Our threaded process cut the elapsed time almost in half!

That's all I have for this time. As usual, you can download a ZIP file containing a copy of this blog entry and the code use to create it from this link.

Now I'm off to participate in SAS Global Forum 2015 in Dallas. There are tons of presentations that talk about DS2, SAS in-database processing and using SAS with Hadoop. Look me up! I can be found at the #SASGF15 #TweetUp Saturday night, attending various presentations (especially about DS2 and Hadoop), or hanging out in the Quad on Tuesday afternoon from 2 to 2:30 pm to answer you questions about SAS Foundation programming or DS2. I'm also teaching the post-conference DS2 Programming Essentials class at the conference center. So, I hope to see you there.

Until next time, may the SAS be with you!
Mark