Subject: WHERE Command Bug with Floating underflow
Posted by jargoogle on Mon, 26 Apr 2004 21:08:09 GMT
View Forum Message <> Reply to Message

Hi,

Actually, I haven't found an error in the WHERE command, but I did encounter some fairly odd behavior that had me stumped for a bit. I thought I'd post a note in case anyone else runs into this issue. I've been with IDL since 1.0 and was bewildered how I could not have run into this "bug" before now. The reason I hadn't was simple enough, as I'll explain.

Here's what happened. I have some floating point data stored in a binary file as big endian floats. Those numbers and that file were created by another IDL program. In a second IDL program, I make use of the IDL WHERE command to get the nonzero values of the array:

```
indexNonZero = where(data)  ; data is fltarr(nx,ny,nz) read from file
```

In the analysis, I'm interested in the minimum nonzero value.

```
minNonZero = min(data[indexNonZero])
```

Imagine my surprise when I suddenly started getting:

```
IDL> print, minNonZero
0.000000
```

Zero! Huh?!? But I asked for the nonzero values! Hmmm. Very strange. It seemed to be some sort of roundoff issue, but I could't pin it down. I tried things like:

```
IDL> print, minNonZero, format = '(e20.5)'
        0.00000e+00
```

Hmmm. Ok, I tried:

```
IDL> indexNonZero = where(data * 1.0)
IDL> minNonZero = min(data[indexNonZero])
IDL> print, minNonZero
(some very small non-zero number here)
```

Well now.

Now, for the missing information. Our lab runs IDL on several different platforms with chips including Sparc, AMD, Intel, Power PC, etc. Traditionally, the Sparc machines were our workhorses, but we've

recently started doing computation on all the platforms. In developing the current codes run in IDL, we decided to go with a big endian binary file format. The proper byte swapping is handled during I/O on the little endian machines.

Well, in the particular case of the data I was analyzing above, it was created on a system with the AMD Opteron chip and analyzed on an archaic Sparc Ultra 5 system. A quick inspection of the machine architecture using IDL's machar command reveals the problem:

On the Opteron:

```
IDL> help, machar(), /stru
** Structure MACHAR, 13 tags, length=52, data length=52:
   IBETA           LONG                 2
   IT              LONG                24
   IRND            LONG                 5
   NGRD            LONG                 0
   MACHEP          LONG               -23
   NEGEP           LONG               -24
   IEXP            LONG                 8
   MINEXP          LONG              -126
   MAXEXP          LONG               128
   EPS             FLOAT       1.19209e-07
   EPSNEG          FLOAT       5.96046e-08
   XMIN            FLOAT       1.17549e-38
   XMAX            FLOAT       3.40282e+38
```

On the Sparc:

```
IDL> help, machar(), /stru
** Structure MACHAR, 13 tags, length=52:
   IBETA           LONG                 2
   IT              LONG                24
   IRND            LONG                 2
   NGRD            LONG                 0
   MACHEP          LONG               -23
   NEGEP           LONG               -24
   IEXP            LONG                 8
   MINEXP          LONG              -126
   MAXEXP          LONG               128
   EPS             FLOAT       1.19209e-07
   EPSNEG          FLOAT       5.96046e-08
   XMIN            FLOAT       1.17549e-38
   XMAX            FLOAT       3.40282e+38
```

The only difference is the IRND field which indicates, among other things, how underflow is handled by the machine. The Sparc sets any number below the minimum floating point number to zero. The Opteron

follows the IEEE standard, a "kinder gentler" approach.

In any case, the opteron machine was writing to file floating point numbers which were smaller than would be commonly allowed by IDL on the Sparc architecture. The IDL procedure on Sparc would identify the values as non-zero, yet would zero them if any operation was performed (such as a print command). Hence WHERE picked them up as non-zero, but any subsequent operation on them forced them to zero. At least, that's my interpretation.

I hope this post gives a heads up to the next person who might encounter this cross platform issue.

Cheers,
JG.

P.S. Yeah, yeah. I know. Underflows. At first glance, the code which generates the underflows is not separable from the rest of the code, though I'm looking at this to see what can be done. I'd rather not slow things down by putting in checks. At the moment, I'm changing my test condition to avoid the small numbers in the second program.
