Subject: Re: Faster approach for total(data,dimension) possible?
Posted by JDS on Thu, 25 Jun 2009 15:22:11 GMT
View Forum Message <> Reply to Message

I agree with all the assessments thus far. These methods are within a factor of 2 or 3 of the best IDL-native vectorized result (and very likely a factor of 10-30 off the compiled C result). As for this calculation taking "minutes", this sounds suspiciously like running out of memory and hitting the disk. That would be unusual given the ~180MB data size here, but perhaps other processes or parts of the routine are taxing memory, or it's a very old machine with <<1GB of RAM. I'd look to this issue first. Here, no matter the algorithm, it runs in a fraction of a second.

That concern aside, there is another approach -- one you will rarely find me recommending. If you happen to know that null bands are going to be found very rarely, a thinned WHERE loop can actually outperform the native vector operation:

```
s=size(data,/DIMENSIONS)
chnk=s[0]*s[1]
zeroes=lindgen(chnk)
for i=0L,s[2]-1 do begin
  z=where(data[zeroes+i*chnk] eq 0.,cnt)
  if cnt eq 0 then break
  zeroes=zeroes[z]
endfor
```

Here I'm operating on an array of the size mentioned above:

```
data=randomu(sd,1536,231,126)
data[where(data lt .9)]=0.
```

By tuning the ".9" factor, you can arrange for as many null bands as you want.

When only a few bands are null in a given data cube, this is roughly 2.5x faster for me. When it's very rare to have *any* null bands, this method can be *much* faster: 20-30x. The reason is clear: it takes only a few iterations to prove the absence of nulls in that case, and index thinning proceeds rapidly

But here's the catch (isn't there always a catch?). If null bands are present at a frequency of even 1 in 200 or more, this loop method becomes slower than TOTAL. In the worst case (all bands null), it's about 6x slower (all on my dual-core machine, YMMV). So, as is usual with these things, the answer to "which method is faster for my data" is: "it depends on your data."

You might also notice this is a reasonable study case for the recently debated issue of "when are for loops *not* evil". Since in each iteration, a large number of elements are being compared, the looping overhead is not severe. You'll also notice this illustrates the method of "compute your own index vector and re-use." Had we used IDL's native array range operator [x:y] or [*,*,z], this most certainly would have spoiled the time savings.

One other point worth mentioning: if your data cubes are "skinny and tall", with the third dimension long compared to the others, this loop method will perform even better. For instance, using a similarly sized data cube, but much taller:

```
data=randomu(sd,153,231,1260)
data[where(data lt .998)]=0.
```

I find speed parity between TOTAL and the thinned WHERE loop occurs when 8% of the bands are null.

JD
