Subject: Re: Chunk Array Decimation
Posted by JD Smith on Thu, 10 Oct 2002 00:18:45 GMT
View Forum Message <> Reply to Message

On Mon, 07 Oct 2002 08:51:21 -0700, Jaco van Gorkom wrote:

> "JD Smith" <jdsmith@as.arizona.edu> wrote in message
> news:pan.2002.10.04.22.07.45.664757.24772@as.arizona.edu...
>
> JD, we mortals need a tutorial or two in order to even begin to
> understand that Sparse Array trick you pulled there. My humble
> contribution to this thread is to propose an additional and hopefully
> simpler (or, more intuitive) algorithm: let us first sort the data array
> based on the index array, and then use a cumulative total to do the
> chunking.

Well, think of an array multiplication of a very large array on a very long data vector. Consider only the first row. If almost everything in the first column is 0., but a few choice 1.'s, that is the same as selecting those elements of the data vector and totaling them. If you really had to do all the null operations like:

...+0.*data[12]+0.*data[13]+...

then it would do you no good at all: all those useless multiply-by-zeroes would waste far too much time to be efficient. Fortunately, "sparse" arrays were invented for just this problem: you specify only the non-zero elements, and, when multiplying using sprsax, they alone are computed. If you adjust the array values, you could obviously use this to do much more than totaling selected data elements.

> The above algorithm is by far not fast enough because of the very slow
> SORT() operation. If this is replaced by a sorting routine which is more
> optimised for the problem at hand, such as, well, I'll let you guess;) ,
> then things get much better:
> h = HISTOGRAM(inds, REVERSE_INDICES=ri) nh = N_ELEMENTS(h) sortData =
> data[inds[ri[nh+1:*]]]
> totSortData = [0., TOTAL(sortData, /CUMULATIVE)] vec8 =
> totSortData[ri[1:nh]-nh-1] - $
> totSortData[ri[0:nh-1]-nh-1]

Aha, a very interesting and compact submission. I think there's one error there. Should it not be data[ri[nh+1:*]] without the inds? I translated as:

```
h = histogram(inds, REVERSE_INDICES=ri)
nh = n_elements(h)
```

```
tdata = [0.,total(data[ri[nh+1:*]],/CUMULATIVE)]
vec9 = tdata[ri[1:nh]-nh-1]-tdata[ri[0:nh-1]-nh-1]
```

The biggest problem with this method, though, is roundoff error, which accumulates like mad in a cumulative total of any length. If you do it in floating point, as you've written, I find unacceptably large roundoff errors for as few as 500 individual indices. If I convert the addition to DOUBLE, this mitigates (but does not eliminate) the roundoff errors, but then it erases the slight time advantage this method has over the prior champ (the dual histogram). Neither, of course, come close to the compiled DLM :(.

Thanks for the example.

JD
