Subject: Re: Chunk Array Decimation
Posted by Jaco van Gorkom on Mon, 07 Oct 2002 15:51:21 GMT
View Forum Message <> Reply to Message

"JD Smith" <jdsmith@as.arizona.edu> wrote in message
news:pan.2002.10.04.22.07.45.664757.24772@as.arizona.edu...
> ... Here's a test run with 1,000,000
> elements, each index repeated 20 times on average:
>
> Literal Accumulate Loop: 1.2411
> Reverse Indices Loop: 0.7217
> Loop-Free with Sparse Arrays: 1.1401
> FDDRIZZLE Loop: 0.7815
> Dual Histogram Loop: 0.5490
> Thinned WHERE Histogram Loop: 0.8422
> Literal Accumulate: Compiled DLM : 0.0288

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.
In pseudo-code:

```
  sortInds = SORT(inds)
  totData = TOTAL(data[sortInds], /CUMULATIVE)
  uniqInds = UNIQ(inds[sortInds])
  subTotData = [0., totData[uniqInds]]
  vec7 = subTotData[1:*] - subTotData[0:*-1]
```

leaving some minor issues like non-occurring indices
unresolved.

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]
```

On my machine this seems to be always slightly faster than the
double histogram loop.

Cheers,
Jaco

P.S. Some timings:

For 1,000,000 elements, each index repeated 20 times on
average, on a single PIII 1GHz, W2K:

Literal Accumulate Loop: 1.2778

Reverse Indices Loop: 0.9794

Loop-Free with Sparse Arrays: 1.2589

FDDRIZZLE Loop: 0.9543

Dual Histogram Loop: 0.6329

Thinned WHERE Histogram Loop: 1.0506

Simple sorting plus cumulative total: 2.6347

Histogram plus cumulative total: 0.6119

And for each index repeated only once on average:

Literal Accumulate Loop: 1.3920

Reverse Indices Loop: 8.1998 (?)

Loop-Free with Sparse Arrays: 1.8647

FDDRIZZLE Loop: 1.7135

Dual Histogram Loop: 1.3129

Thinned WHERE Histogram Loop: 1.7996

Simple sorting plus cumulative total: 2.8701

Histogram plus cumulative total: 1.2488