Subject: Re: Chunk Array Decimation
Posted by Wayne Landsman on Tue, 01 Oct 2002 21:34:21 GMT
View Forum Message <> Reply to Message

> Of course, anyone familiar at all with histogram() would realize
> there's a better route when many indices are repeated:
>
> mx=max(inds)
> vec3=fltarr(mx+1)
> h=histogram(inds,reverse_indices=ri,OMIN=om)
> for j=0L,n_elements(h)-1 do if ri[j+1] gt ri[j] then $
> vec3[j+om]=total(data[ri[ri[j]:ri[j+1]-1]])
>
> This taps into the ever-so useful reverse indices vector to pick out
> those elements of data which fall in each "bin" of the index
> histogram. Notice I'm using OMIN to save time in case the minimum
> index is greater than 0. This is much faster than the where() method,
> and can be a factor of 2 or 3 faster than the literal loop approach,
> if indices are repeated at least a few times on average (a few drops
> in each histogram bin). If indices are never repeated, or especially
> if many indices are skipped (a *sparse* set), the literal loop method
> can be much faster than histogram.

The problem that discussed by JD is actually a very practical one, that can be used in "drizzling" algorithms (e.g. http://www-int.stsci.edu/~fruchter/dither/drizzle.html) This a method of combining or warping images that preserves flux -- every pixel in the input image is equally represented in the output image. Instead of starting with an input pixel and mapping to an output image (e.g. as with POLY_2D) , one starts with an output pixel and determines which input pixels get mapped into it. The flux conservation property is one very dear to astronomers, and for which there are no existing IDL tools.

My solution to the problem combined the REVERSE_INDICiES aproach of JD, with the "accumlate based on the index" approach. For the drizzle problem, one is probably only going to sum at most 3-4 pixels together, so it makes sense to loop over the number of distinct histogram values (i.e. loop only 3-4 times).

My solution is below, but I have to admit that I haven't looked at it for a while.

--Wayne

P.S. I never finished the drizzle algorithm, because I couldn't figure out a quick way to compute partial pixel overlaps in IDL...

```
pro fdrizzle, vector, index, values
;+
; NAME:
;    FDRIZZLE
; PURPOSE:
;    Add values to an array at specified indicies.     The basic usage is

;       FDRIZZLE, vector, index, values
;    where INDEX and VALUES should have same number of elements.   If
there are
;       no duplicates in INDEX then FDRIZZLE simply performs the
assignment
;    VECTOR[INDEX] = VECTOR[INDEX] + VALUES
;       But if INDEX contains repeated elements then the corresponding
VALUES
;       will be summed together.
;
; METHOD:
;    Use the REVERSE_ELEMENTS keyword of histogram to determine the
repeated
;    values in INDEX and vector sums these together.
;-

 h = histogram(index,reverse = ri,min=0,max=N_elements(vector)-1)

;Add locations with at least one pixel
 gmax = max(h)         ;Highest number of duplicate indicies

 for i=1,gmax do begin
       g = where(h GE i, Ng)
    if Ng GT 0 then  vector[g] = vector[g] + values[ri[ ri[g]+i-1]]
 endfor

 end
```

Subject: Re: Chunk Array Decimation
Posted by JD Smith on Thu, 03 Oct 2002 00:03:55 GMT

View Forum Message <> Reply to Message

On Tue, 01 Oct 2002 14:34:21 -0700, Wayne Landsman wrote:

>> Of course, anyone familiar at all with histogram() would realize
>> there's a better route when many indices are repeated:
>>
>>   mx=max(inds)
>>   vec3=fltarr(mx+1)
>>   h=histogram(inds,reverse_indices=ri,OMIN=om) for

>>   j=0L,n_elements(h)-1 do if ri[j+1] gt ri[j] then $
>>   vec3[j+om]=total(data[ri[ri[j]:ri[j+1]-1]])
>>
>> This taps into the ever-so useful reverse indices vector to pick out
>> those elements of data which fall in each "bin" of the index histogram.
>>   Notice I'm using OMIN to save time in case the minimum index is
>> greater than 0. This is much faster than the where() method, and can
>> be a factor of 2 or 3 faster than the literal loop approach, if indices
>> are repeated at least a few times on average (a few drops in each
>> histogram bin). If indices are never repeated, or especially if many
>> indices are skipped (a *sparse* set), the literal loop method can be
>> much faster than histogram.
>
> The problem that discussed by JD is actually a very practical one, that
> can be used in "drizzling" algorithms (e.g.
> http://www-int.stsci.edu/~fruchter/dither/drizzle.html)   This a
> method of combining or warping images that preserves flux -- every pixel
> in the input image is equally represented in the output image. Instead
> of starting with an input pixel and mapping to an output image (e.g. as
> with POLY_2D) , one starts with an output pixel and determines which
> input pixels get mapped into it.   The flux conservation property is
> one very dear to astronomers, and for which there are no existing IDL
> tools.
>
> My solution to the problem combined the REVERSE_INDICiES aproach of JD,
> with the "accumlate based on the index" approach.   For the drizzle
> problem, one is probably only going to sum at most 3-4 pixels together,
> so it makes sense to loop over the number of distinct histogram values
> (i.e. loop only 3-4 times).
>
> My solution is below, but I have to admit that I haven't looked at it
> for a while.
>

>   h = histogram(index,reverse = ri,min=0,max=N_elements(vector)-1)
>
> ;Add locations with at least one pixel
>   gmax = max(h)   ;Highest number of duplicate indicies
>
>   for i=1,gmax do begin
>   g = where(h GE i, Ng)
>   if Ng GT 0 then vector[g] = vector[g] + values[ri[ri[g]+i-1]]
>   endfor
>
>   end

That's a very interesting approach, Wayne. People who need to understand
the reverse indices vector would do well to study this one. I put it

into the same terms as my problem for testing:

```
  mx=max(inds)
  vec5=fltarr(mx+1)
  h=histogram(inds,REVERSE_INDICES=ri,omin=om)
  gmax = max(h)               ;Highest number of duplicate indicies
  for j=1,gmax do begin
    g = where(h GE j, Ng)
    if Ng GT 0 then  vec5[om+g] = vec5[om+g] + data[ri[ ri[g]+j-1]]
  endfor
```

I was interested to see that your method beat mine for normal densities by about a factor of 2! This should provide some cannon fodder for Craig in his loop-anti-defamation campaign: keep loops small, and they're not bad. The only change I added was using OMIN as opposed to fixing MIN=0, but that shouldn't account for much if any improvement.

However, one thing still bothered me about the your method: even though the loop through the bin depth is small (e.g. maybe up to 5-10 for DRIZZLE-type cases), you're using WHERE to search a potentially very large histogram array linearly each time. What's the solution? Why, just use another histogram to sort the histogram into bins of repeat count, of course. Now this is a true histogram of a histogram.

```
  mx=max(inds)
  vec6=fltarr(mx+1)
  h1=histogram(inds,reverse_indices=ri1,OMIN=om)
  h2=histogram(h1,reverse_indices=ri2,MIN=1)
  ;; easy case - single values w/o duplication
  if ri2[1] gt ri2[0] then begin
    vec_inds=ri2[ri2[0]:ri2[1]-1]
    vec6[om+vec_inds]=data[ri1[ri1[vec_inds]]]
  endif
  for j=1,n_elements(h2)-1 do begin
    if ri2[j+1] eq ri2[j] then continue ;none with that many duplicates
    vec_inds=ri2[ri2[j]:ri2[j+1]-1] ;indices into h1
    vinds=om+vec_inds
    vec_inds=rebin(ri1[vec_inds],h2[j],j+1,/SAMPLE)+ $
          rebin(transpose(lindgen(j+1)),h2[j],j+1,/SAMPLE)
    vec6[vinds]=vec6[vinds]+total(data[ri1[vec_inds]],2)
  endfor
```

This is absolutely the fastest I've seen... faster by a factor of ~2 than DRIZZLE. Here are some timings again, for the curious:

20,000 Indices

Indices repeated once, on average:

WHERE loop:	3.8967
Literal Accumulate Loop:	0.0250
Reverse Indices Loop:	0.0725
Loop-Free with Sparse Arrays:	0.0136
FDDRIZZLE Loop:	0.0107
Dual Histogram Loop:	0.0077

Repeated 5 times, on average:

WHERE loop:	0.9433
Literal Accumulate Loop:	0.0241
Reverse Indices Loop:	0.0214
Loop-Free with Sparse Arrays:	0.0102
FDDRIZZLE Loop:	0.0069
Dual Histogram Loop:	0.0041

Repeated 20 times, on average:

WHERE loop:	0.2510
Literal Accumulate Loop:	0.0246
Reverse Indices Loop:	0.0063
Loop-Free with Sparse Arrays:	0.0095
FDDRIZZLE Loop:	0.0075
Dual Histogram Loop:	0.0033

Repeated 50 times, on average:

WHERE loop:	0.1016
Literal Accumulate Loop:	0.0246
Reverse Indices Loop:	0.0032
Loop-Free with Sparse Arrays:	0.0094
FDDRIZZLE Loop:	0.0079
Dual Histogram Loop:	0.0033

Only 1 in 5 indices present (WHERE loop omitted -- too slow):

Literal Accumulate Loop:	0.0275
Reverse Indices Loop:	0.1754
Loop-Free with Sparse Arrays:	0.0453
FDDRIZZLE Loop:	0.0264
Dual Histogram Loop:	0.0196

Only 1 in 20 indices present:

Literal Accumulate Loop:	0.0334
Reverse Indices Loop:	0.4785

Loop-Free with Sparse Arrays: 0.1471
FDDRIZZLE Loop: 0.0623
Dual Histogram Loop: 0.0530

Only 1 in 50 indices present:

Literal Accumulate Loop: 0.0419
Reverse Indices Loop: 1.0674
Loop-Free with Sparse Arrays: 0.3461
FDDRIZZLE Loop: 0.1289
Dual Histogram Loop: 0.1127

Thanks for the pointer.

JD

Subject: Re: Chunk Array Decimation
Posted by Craig Markwardt on Thu, 03 Oct 2002 08:58:13 GMT
View Forum Message <> Reply to Message

JD Smith <jdsmith@as.arizona.edu> writes:

> On Tue, 01 Oct 2002 14:34:21 -0700, Wayne Landsman wrote:
[...]
>>
>> My solution to the problem combined the REVERSE_INDICiES aproach of JD,
>> with the "accumlate based on the index" approach. For the drizzle
>> problem, one is probably only going to sum at most 3-4 pixels together,
>> so it makes sense to loop over the number of distinct histogram values
>> (i.e. loop only 3-4 times).
>>
>> My solution is below, but I have to admit that I haven't looked at it
>> for a while.
>>
>
>> h = histogram(index,reverse = ri,min=0,max=N_elements(vector)-1)
>>
>> ;Add locations with at least one pixel
>> gmax = max(h) ;Highest number of duplicate indicies
>>
>> for i=1,gmax do begin
>> g = where(h GE i, Ng)
>> if Ng GT 0 then vector[g] = vector[g] + values[ri[ri[g]+i-1]]
>> endfor
>>
>> end
>

> That's a very interesting approach, Wayne. People who need to understand
> the reverse indices vector would do well to study this one. I put it
> into the same terms as my problem for testing:
>
> mx=max(inds)
> vec5=fltarr(mx+1)
> h=histogram(inds,REVERSE_INDICES=ri,omin=om)
> gmax = max(h) ;Highest number of duplicate indicies
> for j=1,gmax do begin
> g = where(h GE j, Ng)
> if Ng GT 0 then vec5[om+g] = vec5[om+g] + data[ri[ri[g]+j-1]]
> endfor
>
> I was interested to see that your method beat mine for normal
> densities by about a factor of 2! This should provide some cannon
> fodder for Craig in his loop-anti-defamation campaign: keep loops
> small, and they're not bad. The only change I added was using OMIN as
> opposed to fixing MIN=0, but that shouldn't account for much if any
> improvement.
>
> However, one thing still bothered me about the your method: even
> though the loop through the bin depth is small (e.g. maybe up to 5-10
> for DRIZZLE-type cases), you're using WHERE to search a potentially
> very large histogram array linearly each time. What's the solution?
> Why, just use another histogram to sort the histogram into bins of
> repeat count, of course. Now this is a true histogram of a histogram.
[...]

Here I come late to the game again. This topic actually came up
before by Liam Gumley in September 2000.

My solution then was the following loop (expressed in today's variable
names):

```
n = n_elements(vec)
hh = histogram(inds, min=0, max=n-1, reverse=rr)
wh = where(hh GT 0) & mx = max(hh(wh), min=mn)
for i = mn, mx do begin
  wh = wh(where(hh(wh) GE i, ct))        ;; Get IND cells with GE i entries
  vec(wh) = vec(wh) + data(rr(rr(wh)+i-1)) ;; Add into the total
endfor
```

This is essentially the same as Wayne's FDRIZZLE routine, with the
difference that the WHERE-generated index array is slowly whittled
away by repeated thinning. Thus, the WHERE() function gets faster and
faster as the loop proceeds. At the time, I was crowned the victor by
Pavel :-), but I don't know how I will do against this round of
competitors.

However, all of these optimized techniques that Wayne and JD have proposed in the end game here, including mine, suffer if the dynamic range of the histogram is very large. For example, if the input array contains a million 1s, then any of the proposed loops will still take 1 million iterations. There are even ways around that, which reminds me to finish an old routine named CMHISTOGRAM...

Craig

PS. In my case, NO, I do not have to check the count returned by WHERE, because by the construction of the loop, there is always guaranteed to be at least one element.

--

Craig B. Markwardt, Ph.D. EMAIL: craigmnet@cow.physics.wisc.edu
Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response

---

Subject: Re: Chunk Array Decimation
Posted by JD Smith on Thu, 03 Oct 2002 20:32:27 GMT
View Forum Message <> Reply to Message

On Thu, 03 Oct 2002 01:58:13 -0700, Craig Markwardt wrote:
> JD Smith <jdsmith@as.arizona.edu> writes:
>
>> On Tue, 01 Oct 2002 14:34:21 -0700, Wayne Landsman wrote:
> [...]
>>>
>>> My solution to the problem combined the REVERSE_INDICiES aproach of
>>> JD, with the "accumlate based on the index" approach. For the
>>> drizzle problem, one is probably only going to sum at most 3-4 pixels
>>> together, so it makes sense to loop over the number of distinct
>>> histogram values (i.e. loop only 3-4 times).
>>>
>>> My solution is below, but I have to admit that I haven't looked at it
>>> for a while.
>>>
>>>
>>> h = histogram(index,reverse = ri,min=0,max=N_elements(vector)-1)
>>>
>>> ;Add locations with at least one pixel
>>> gmax = max(h) ;Highest number of duplicate indicies
>>>
>>> for i=1,gmax do begin
>>> g = where(h GE i, Ng)

```
>>>      if Ng GT 0 then  vector[g] = vector[g] + values[ri[ ri[g]+i-1]]
>>>   endfor
>>>
>>>   end
>>
>>  That's a very interesting approach, Wayne.  People who need to
>>  understand the reverse indices vector would do well to study this one.
>>  I put it into the same terms as my problem for testing:
>>
>>    mx=max(inds)
>>    vec5=fltarr(mx+1)
>>    h=histogram(inds,REVERSE_INDICES=ri,omin=om) gmax = max(h)
>>         ;Highest number of duplicate indicies for j=1,gmax do begin
>>       g = where(h GE j, Ng)
>>       if Ng GT 0 then  vec5[om+g] = vec5[om+g] + data[ri[ ri[g]+j-1]]
>>    endfor
>>
>>  I was interested to see that your method beat mine for normal densities
>>  by about a factor of 2!  This should provide some cannon fodder for
>>  Craig in his loop-anti-defamation campaign: keep loops small, and
>>  they're not bad.  The only change I added was using OMIN as opposed to
>>  fixing MIN=0, but that shouldn't account for much if any improvement.
>>
>>  However, one thing still bothered me about the your method: even though
>>  the loop through the bin depth is small (e.g. maybe up to 5-10 for
>>  DRIZZLE-type cases), you're using WHERE to search a potentially very
>>  large histogram array linearly each time.  What's the solution? Why,
>>  just use another histogram to sort the histogram into bins of repeat
>>  count, of course.  Now this is a true histogram of a histogram.
>  [ ... ]
>
>  Here I come late to the game again.  This topic actually came up before
>  by Liam Gumley in September 2000.
>
>  My solution then was the following loop (expressed in today's variable
>  names):
>
>    n = n_elements(vec)
>    hh = histogram(inds, min=0, max=n-1, reverse=rr) wh = where(hh GT 0) &
>    mx = max(hh(wh), min=mn) for i = mn, mx do begin
>      wh = wh(where(hh(wh) GE i, ct))          ;; Get IND cells with GE i
>      entries vec(wh) = vec(wh) + data(rr(rr(wh)+i-1)) ;; Add into the
>      total
>    endfor
>
>  This is essentially the same as Wayne's FDRIZZLE routine, with the
>  difference that the WHERE-generated index array is slowly whittled away
>  by repeated thinning.  Thus, the WHERE() function gets faster and faster
```

> as the loop proceeds. At the time, I was crowned the victor by Pavel
> :-), but I don't know how I will do against this round of competitors.

Too much fun. I translated your thinned WHERE() method into my terms:

```
mx=max(inds)
vec7=fltarr(mx+1)
h = histogram(inds,OMIN=om,REVERSE_INDICES=ri)
wh = where(h GT 0)
mx = max(h[wh], min=mn)
for j=mn,mx do begin
  wh=wh[where(h[wh] GE j)]  ; Get IND cells with GE i entries
  vec7[om+wh]=vec7[om+wh] + data[ri[ri[wh]+j-1]] ; Add into the total
endfor
```

> However, all of these optimized techniques that Wayne and JD have
> proposed in the end game here, including mine, suffer if the dynamic
> range of the histogram is very large. For example, if the input array
> contains a million 1s, then any of the proposed loops will still take 1
> million iterations. There are even ways around that, which reminds me
> to finish an old routine named CMHISTOGRAM...

With a million 1's, you have only one iteration in your loop, since there's just one bin in the histogram. This example illustrates an error in your formulation: it only works if mn is 1 (which it almost always will be in a large enough vector of random indices)! Why? Because you need the loop to accumulate all of the values from ri[wh]...ri[wh]+n_bin. If you have only one bin of 1000000, you just pick out the value at ri[ri[wh]+1000000]! It's fast, but wrong. FDRIZZLE works correctly because it starts its loop explicitly at 1. Yours works if I modify it to start at 1 also:

```
mx=max(inds)
vec7=fltarr(mx+1)
h = histogram(inds,OMIN=om,REVERSE_INDICES=ri)
wh = where(h GT 0)
mx = max(h[wh],min=mn)
for j=1,mx do begin
  wh=wh[where(h[wh] GE j)]  ; Get IND cells with GE i entries
  vec7[om+wh]=vec7[om+wh] + data[ri[ri[wh]+j-1]] ; Add into the total
endfor
```

In the pathological case of 20,000 1's, I get:

WHERE loop:	0.0014
Literal Accumulate Loop:	0.0246

Reverse Indices Loop: 0.0014
FDDRIZZLE Loop: 0.2256
Dual Histogram Loop: 0.0030
Thinned WHERE Histogram Loop: 0.2623

The WHERE loop and reverse indices are essentially equivalent to one call to total with a vector of all indices, and so are quite fast. My method also uses total, but just has to skip all the empty bins. I changed it to do this by starting at min(h1) (rather than just loop through and CONTINUE all those times), and it's fairly fast.

In a more reasonable case of an index density of 5 (indices repeated 5 times on average), I get:

WHERE loop: 0.9506
Literal Accumulate Loop: 0.0245
Reverse Indices Loop: 0.0213
Loop-Free with Sparse Arrays: 0.0102
FDDRIZZLE Loop: 0.0064
Dual Histogram Loop: 0.0040
Thinned WHERE Histogram Loop: 0.0069

Strangely, yours always performs slightly worse than Wayne's, despite the thinning. This is a dual processor machine, so your mileage may vary, but in any case it's not faster. Just for fun, here's a run with 1,000,000 random indices with a density of 20:

Literal Accumulate Loop: 1.2437
Reverse Indices Loop: 0.7192
Loop-Free with Sparse Arrays: 1.1367
FDDRIZZLE Loop: 0.7882
Dual Histogram Loop: 0.5489
Thinned WHERE Histogram Loop: 0.8438

If you'd like to try this test code yourself, it's available at:

turtle.as.arizona.edu/idl/

I'd be interested to hear how others find the algorithms stack up.

JD
