Subject: Re: vectorization challenge! (help!)
Posted by mattf on Thu, 19 Jul 2007 11:53:49 GMT
View Forum Message <> Reply to Message

On Jul 17, 12:52 pm, Conor <cmanc...@gmail.com> wrote:
```
> I'm 'vectorizing' a piece of code to speed it up.  It's part of a
> larger program.  One of the sections is turning out to be very
> difficult to vectorize (am I using that word right?  What I mean is
> I'm trying to get rid of for loops).  Anyway, maybe someone has some
> thoughts on how to vectorize it.  Maybe it's just not worth it for
> this section.  Here's the basic idea, filled in with dummy data:
>
> n = 24
> npeeps = 50
> gn1 = findgen(n,npeeps)
> gn2 = findgen(n,npeeps)
> cutoff = .01
>
> for i=0,npeeps-1 do begin
>
>     ; make a random value to determine if we do anything with this
> row
>     if randomu(seed,1) lt cutoff then begin
>
>         ; this row has been selected.  Swap the last (random number)
> of digits in gn1[*,i] with gn2[*,i]
>         randindex = long(randomu(seed,1)*n*nd)
>         temp = gn1[randindex:*,i]
>         gn1[randindex:*,i] = gn2[randindex:*,i]
>         gn2[randindex:*,i] = temp
>
>     endif
>
> endfor
>
> That's it.  For randomly selected rows, swap a random number of
> elements at the end of the row with another array.  It is surpisingly
> difficult to get rid of that for loop.  Maybe I'm just a bit out of it
> today though.  I thought of generating a list of indexes to be
> swapped, but I can't quite figure it out.  Oh, if only IDL allowed the
> syntax: arr[st:ed] where st and ed are arrays themselves!  Then this
> would be really easy (something like this would do it):
>
> st = long(randomu(seed,npeeps)*n*nd)
> ed = make_array(npeeps,/integer,value=n)
> indlist = indgen(n)
> inds = indlist[st:ed] + indgen(npeeps)*n
> temp = gn1[inds]
>  gn1[inds] = gn2[inds]
>  gn2[inds] = temp
>
>  Alas, indlist[st:ed] isn't allowed!  (Also, indgen(npeeps)*n has the
>  wrong dimensions anyway...)
```

There are a number of vectorization strategies-- one is heavy use of the WHERE() function. Given a condition C for an action on elements some vector A, you can set A[WHERE(C)] to whatever you want.

Another trick is to treat a matrix as a (long) one dimensional vector and then make use of modular arithmetic. But be careful when you go back to matrix-land with the REFORM() function-- chances are you will get rows and columns backwards.

A third vectorization trick is to exchange a FOR loop for N copies of your variables all concatenated together, and then doing the N calculations with matrix arithmetic. This is a last-ditch vectorization strategy that depends on exchanging the processing time in a FOR loop for a no-branching calculation that wastes a lot of memory