

---

Subject: Re: vectorising versus loops

Posted by [nasalmon](#) on Mon, 23 Feb 2004 23:42:30 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Craig Markwardt <craigmnet@REMOVEcow.physics.wisc.edu> wrote in message news:<onfzd3vttk.fsf@cow.physics.wisc.edu>...

> nasalmon@onetel.net.uk (Neil) writes:

>> Does anyone know what the speed increase factor is in IDL programmes  
>> when going from "do loops" to full vectorisation of arrays? I know all  
>> programmes are different and not every process lends itself to  
>> vectorisation. However, there must be some rule of thumb, ie speed  
>> going as a linear function of the number of array elements times some  
>> factor.

>

> A very simple rule of thumb is to vectorize when the overhead of doing  
> a FOR loop passes your "pain" threshold. Example: a one million  
> iterations of an "empty" loop like this:

>

> for i = 0L, 1000000L do begin & x = 0 & end

>

> takes 0.25 sec on a reasonably modern machine I use. On an older  
> machine it takes 1.5 sec. You can do the same, and decide when the  
> loop overhead time per iteration passes your personal threshold. Bear  
> in mind that if you do multiple executions of the loop, you should  
> multiply that in.

Craig,

yes many thanks for this valuable information. One of the problems i  
have is that the condition in the WHERE statement has to contain the  
counter in the "do loop", that is basically why i put it in the "do  
loop" to start with. Is there any way i can make the condition depend  
on the counter, ie the vector index.

best regards,

Neil

>

> Whether or not I go over my personal pain threshold, I tend to be  
> picky and try to vectorize anyway. My personal approach is to remove  
> the innermost loop and vectorize where possible.

>

>

>> Also, are there any tricks to play if you want to vectorise loops that  
>> have IF statement decision in them, or any general rules for neat  
>> vectorisation of looped programmes?

>

> Yes, there are several. You can use WHERE:

>

```

> ; Example, square root of DATA
> result = data*0      ;; Initialize result
> wh = where(data GE 0, ct)  ;; Find non-negative data values
> if ct GT 0 then result(wh) = sqrt(data(wh)) ;; compute sqrt
>
> This can get cumbersome sometimes, especially because the RESULT needs
> to be initialized. In the square root example above, we don't need to
> use WHERE(), since we can use other features of IDL like the threshold
> operator.
>
> result = sqrt(data > 0)    ;; Make all negative values of DATA zero
>
> This is clearly more simple, faster, and easier to understand.
>
> Another technique is to use a "mask" variable to set offending numbers
> to zero. For example, when computing the gaussian function, one often
> wants to limit the argument of the exponential to prevent overflows.
>
> ;; Example: compute gaussian function of X (mean=xmean, sigma=sigma)
> arg = -0.5*((x-xmean)/sigma)^2
> mask = abs(arg) LT 50      ;; Arbitrary limit of < sqrt(50) sigma
>
> result = exp(arg*mask) / (2*!dpi*sqrt(sigma)) ;; WRONG
>
> Ah, but if you look carefully, multiplying by MASK will make an
> argument of 0, so RESULT will be 1 in those positions. We have
> avoided the under/overflow, but now the result is incorrect. This is
> easily remedied however, since we can multiply by MASK again to set
> these values to zero:
>
> result = mask*exp(arg*mask) / (2*!dpi*sqrt(sigma)) ;; CORRECT
>
> Hope those examples give you some ideas. Good luck!
> Craig

```

---