Subject: Re: vectorising versus loops
Posted by Craig Markwardt on Sun, 22 Feb 2004 18:43:19 GMT
View Forum Message <> Reply to Message

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.

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 initiiazed. 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

--

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

Subject: Re: vectorising versus loops
Posted by David Fanning on Sun, 22 Feb 2004 18:50:13 GMT
View Forum Message <> Reply to Message

Neil writes:

> Does anyone know what the speed increase factor is in IDL programmes
> when going from "do loops" to full vectorisation of arrays?

Would you believe a factor of 8100? Read it and weep:

   http://www.dfanning.com/code_tips/slowloops.html

> 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.

I think the generally accepted rule of thumb is that speed increases
exponentially with the number of JD Smith articles you have read.

> 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?

Oh, lots and lots of articles. A google search on "FOR loops" AND
"slow" should turn up at least a week of good reading. :-)

Cheers,

David
--
David Fanning, Ph.D.
Fanning Software Consulting
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: vectorising versus loops
Posted by Edd Edmondson on Mon, 23 Feb 2004 09:41:33 GMT
View Forum Message <> Reply to Message

David Fanning <david@dfanning.com> wrote:
> Neil writes:

>> Does anyone know what the speed increase factor is in IDL programmes
>> when going from "do loops" to full vectorisation of arrays?

> Would you believe a factor of 8100? Read it and weep:

> http://www.dfanning.com/code_tips/slowloops.html

Funny, that. "I'm just an average astrophysicist" too, and last week I
took my IDL program that took a *fortnight* to run and got it down to 21
seconds. That's a speed up of 55,000 times and there wasn't even a nested
loop - just one loop with a bunch of IFs in it. Through use of plenty of
WHEREs (dealing with each case previously handled with an IF) and a REFORM
plus a HISTOGRAM it's stupidly faster.

In fact it's probably more than 55,000 times because a fair few seconds of
that is needed just to read in the data file at the start.

--
Edd

Subject: Re: vectorising versus loops
Posted by David Fanning on Mon, 23 Feb 2004 13:47:15 GMT
View Forum Message <> Reply to Message

Edd Edmondson writes:

> Funny, that. "I'm just an average astrophysicist" too, and last week I
> took my IDL program that took a *fortnight* to run and got it down to 21
> seconds. That's a speed up of 55,000 times.

Well, there you go! I'll have a reporter out later
this afternoon. :-)

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting
Coyote's Guide to IDL Programming: http://www.dfanning.com/

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 initiiazed. 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

Subject: Re: vectorising versus loops
Posted by nasalmon on Sun, 07 Mar 2004 13:07:58 GMT
View Forum Message <> Reply to Message

David Fanning <david@dfanning.com> wrote in message
news:<MPG.1aa3b24b992bd018989699@news.frii.com>...
> Edd Edmondson writes:
>
>> Funny, that. "I'm just an average astrophysicist" too, and last week I
>> took my IDL program that took a *fortnight* to run and got it down to 21
>> seconds. That's a speed up of 55,000 times.
>
> Well, there you go! I'll have a reporter out later
> this afternoon. :-)
>
> Cheers,
>
> David

Many thanks all for these very useful comments. My software is well on the way to being vectorised, using all the stuff like WHERE statements and matrix multiplications, mainly on inner most loops.

However, there is one small outstanding problem in the vectorisation, and this involves vectorisation of a routine that uses the cross or vector product, the IDL routine being CROSSP, generating a vector from the cross product of two vectors.

Currently i am forced to use this in a loop, as my vectorising attempts (see below) have failed. The loop statement below gives the correct result (i being the counter of one of the outer loops):

for j = 0,jtot-1 do vec_pip[*,j] = crossp(pixpoly[*,j],normal[*,i])

Naively, putting this in the vector structure i have:

vec_pip[*,0:jtot-1] = crossp(pixpoly[*,0:jtot-1],normal[*,i])

which gives the IDL error message: "% Array subscript for VEC_PIP must have same size as source expression." - well of course i know that!

Would anybody have any ideas as to how this might be resolved, without having to resort to working out the cross product from first principles using determinants and matrix multiplication. Is there any way the IDL routine could be made to see this as a vectorising process and compute it without making a fuss.

many thanks, Neil
