Subject: Re: Vectorization question
Posted by Liam E. Gumley on Thu, 14 Sep 2000 07:00:00 GMT
View Forum Message <> Reply to Message

"Liam E. Gumley" wrote:
> Given the following arrays
>
> a = intarr(10)
> x = [2, 2, 2, 3, 3, 4]
> b = [1, 3, 4, 2, 1, 8]
>
> How would I vectorize the following operation
>
> for i = 0, n_elements(x) - 1 do a[x[i]] = a[x[i]] + b[i]
>
> To achieve this result
>
> print, a, format='(10i4)'
> 0 0 8 3 8 0 0 0 0 0
>
> In the real-world case where this occurs, I need to repeat this kind of
> operation several hundred times, where the size of 'a' is around
> 1,000,000 and the size of 'x' is around 100,000 ('a' and 'b' are float
> type in the real-world case).

It dawned on me that this is a perfect case for an external routine.
Following the example in the 'External Development Guide' for calling a
FORTRAN routine with a FORTRAN wrapper, I created the following source
file named vecadd.f

```
c----------
c ... This is the interface routine called by IDL
      subroutine vecadd(argc, argv)
      integer*4 argc, argv(*), j
      j = loc(argc)
      call vecadd1(%val(argv(1)), %val(argv(2)), %val(argv(3)),
     &   %val(argv(4)), %val(argv(5)))
      end

c ... This is the routine which does the work.
c ... The arguments are defined exactly the same as in the
c ... call_external procedure call in IDL.
      subroutine vecadd1(a, na, x, nx, b)
      integer*4 na, nx
      real*4 a(na), b(nx)
      integer*4 x(nx), i
      do i = 1, nx
        a(x(i)) = a(x(i)) + b(i)
```

```
      end do
      end
c----------
```

I run IDL 5.3 on SGI IRIX 6.4, so the compile went as follows:

```
% f77 -n32 -KPIC -u -fullwarn -c vecadd.f
% ld -n32 -o vecadd.so vecadd.o
```

The IDL wrapper for this routine is named vecadd.pro:

```
;----------
FUNCTION VECADD, ARRAY, INDEX, VALUE

;- Check arguments
if (n_elements(array) eq 0) then $
  message, 'Argument A is undefined'
if (n_elements(index) eq 0) then $
  message, 'Argument X is undefined'
if (n_elements(value) eq 0) then $
  message, 'Argument B is undefined'
if (n_elements(index) ne n_elements(value)) then $
  message, 'Arguments X abd B must have the same number of elements'

;- Create copies of the arguments with correct type
a = float(array)
x = (long(index) > 0L) < (n_elements(a) - 1L)
b = float(value)

;- Call the external routine
result = call_external('vecadd.so', 'vecadd_', $
  a, n_elements(a), x, n_elements(x), b)

;- Return result
return, a

END
;----------
```

So the operation I described is now quite simple:

```
a = fltarr(10)
x = [2, 2, 2, 3, 3, 4]
b = [1, 3, 4, 2, 1, 8]
result = vecadd(a, x, b)
help, result
RESULT          FLOAT     = Array[10]
print, result, format='(10i4)'
```

0 8 3 8 0 0 0 0 0 0

The result is always returned as FLOAT, which is what I really wanted anyway. For the large arrays I described, VECADD is at least 10 times faster than a loop.

Thanks IDL!

Cheers,
Liam.
http://cimss.ssec.wisc.edu/~gumley
PS: Pavel, thanks for your suggestion as well.
