Subject: Re: LIST performance
Posted by chris_torrence@NOSPAM on Fri, 12 Nov 2010 17:46:01 GMT
View Forum Message <> Reply to Message

Hi JD,

Just to put in my 2 cents, LIST (like much of IDL) is a general purpose set of functionality. It was never designed to replace the use of arrays, where the data is laid out in memory in the most efficient manner. The IDL LIST is implemented as a doubly-linked list, which is very efficient for adding & removing elements from arbitrary positions, especially elements of different or complicated types. If you're only using integers or floats, you probably don't want to use a list.

Here's an example for adding & removing random elements from an array of structures:

```
print,'          N        Array(s)' + $
  '       List(s)        Ratio'
for nn=2,4 do begin
  n = 10L^nn
  iter = 10L^(6 - nn)
  a = REPLICATE(!map, n)
  t = systime(1)
  for i=0,iter-1 do begin
    index = RANDOMU(seed,1)*(n-2) + 1
    a = [a[0:index-1], !map, a[index:*]]
    a = [a[0:index-1], a[index+1:*]]
  endfor
  timearray = systime(1)-t

  a = LIST(REPLICATE(!map, n), /EXTRACT)
  t = systime(1)
  for i=0,iter-1 do begin
    index = RANDOMU(seed,1)*(n-2) + 1
    a.Add, !map, index
    a.Remove, index
  endfor
  timelist = systime(1)-t

  print, n, timearray, timelist, timearray/timelist
endfor
end
```

When I run this on my Win32 XP laptop, I get the following results:

```
          N        Array(s)        List(s)        Ratio
```

100	3.3750000	0.38999987	8.6538491
1000	5.5160000	0.10899997	50.605520
10000	7.2500000	0.078000069	92.948636

Part of the reason the LIST is slower in your example is the overhead with both error checking and the operator overloading. The List::Add and List::overloadForeach are both method calls, so there is some additional overhead for making a call instead of just doing it in-place like in your pointer example.

Now, all that being said, we'll continue to make performance improvements in future versions. For example, I just rewrote the List::ToArray to be about 10 times faster. It's still "brutally slow" for your particular example, but for other scenarios it's much faster.

Finally, I like your idea about making the ::ToArray method work properly for list elements that are arrays. I'll see what I can do.

Keep the feedback coming.
Cheers,

Chris
ITTVIS
