Subject: Re: Random-access of List() progressively slower for static list
Posted by Helder Marchetto on Mon, 05 May 2014 12:25:19 GMT
View Forum Message <> Reply to Message

On Monday, May 5, 2014 1:34:52 PM UTC+2, Tom Grydeland wrote:
> Hi all,
>
>
>
> The following snippet demonstrates very peculiar complexity in IDL 8.2.2
>
>
>
> In the first part, exchanging active and commented-out equivalent code gives equally unsatisfactory results in the list creation phase.
>
>
>
> List(), for all its nice properties, is not fit for (my) purpose in this version of IDL.
>
>
>
> Regards,
>
>
>
> Tom Grydeland, Norut
>
>
>
> ;;;;;;;;;;;;;;;;;;; Cut here ;;;;;;;;;;;;;;;;;;;;;;
>
> compile_opt idl2
>
>
>
> N = 200000L
>
> M = 5000L
>
>
>
> print, format='(%"Creating list of %d elements")', N
>
>
>
> values = List()
>

```
>  ; values = List(length=N)
>
>
>
>  t0 = systime(/seconds)
>
>  for jj=0L, N-1 do begin
>
>
>
>    ;; Nice and snappy
>
>    values.add, List(jj, indgen(jj mod 5 + 1), findgen(jj mod 4 + 1))
>
>
>
>    ;; Excruciatingly slow
>
>    ; values[jj] = List(jj, indgen(jj mod 5 + 1), findgen(jj mod 4 + 1))
>
>
>
>    if ~(jj mod M) then begin
>
>      t1 = systime(/seconds)
>
>      print, format='(%"Inserted %d elements, the last %d in %g seconds")', jj, M, t1-t0
>
>      t0 = t1
>
>    endif
>
>  endfor
>
>
>
>  itot = 0L
>
>  ftot = 0.0
>
>
>
>  ;; Now see how List() deals with random access
>
>  ii = fix(randomu(seed, N) * N)
>
>  for jj = 0L, N-1 do begin
>
```

```
>    itot += values[jj, 1, 0]
>
>    ftot += values[jj, 2, 0]
>
>    if ~(jj mod M) then begin
>
>      t1 = systime(/seconds)
>
>      print, format='(%"Processed %d elements, the last %d in %g seconds")', jj, M, t1-t0
>
>      t0 = t1
>
>    endif
>
>  endfor
>
>
>
>  end
>
>
>
>  ;;;;;;;;;;;;;;;;;;;; Cut here ;;;;;;;;;;;;;;;;;;;;;;;;
>
>
>
>  Typical output (limited by my impatience):
>
>
>
>  Creating list of 200000 elements
>
>  Inserted 0 elements, the last 5000 in 0.00215101 seconds
>
>  Inserted 5000 elements, the last 5000 in 0.348531 seconds
>
>  Inserted 10000 elements, the last 5000 in 0.346365 seconds
>
>  Inserted 15000 elements, the last 5000 in 0.346398 seconds
>
>  Inserted 20000 elements, the last 5000 in 0.343561 seconds
>
>  Inserted 25000 elements, the last 5000 in 0.345163 seconds
>
>  Inserted 30000 elements, the last 5000 in 0.344774 seconds
>
>  Inserted 35000 elements, the last 5000 in 0.345832 seconds
>
```

```
> Inserted 40000 elements, the last 5000 in 0.346793 seconds
>
> Inserted 45000 elements, the last 5000 in 0.345885 seconds
>
> Inserted 50000 elements, the last 5000 in 0.347492 seconds
>
> Inserted 55000 elements, the last 5000 in 0.34761 seconds
>
> Inserted 60000 elements, the last 5000 in 0.347977 seconds
>
> Inserted 65000 elements, the last 5000 in 0.349764 seconds
>
> Inserted 70000 elements, the last 5000 in 0.354488 seconds
>
> Inserted 75000 elements, the last 5000 in 0.354447 seconds
>
> Inserted 80000 elements, the last 5000 in 0.354993 seconds
>
> Inserted 85000 elements, the last 5000 in 0.355061 seconds
>
> Inserted 90000 elements, the last 5000 in 0.355482 seconds
>
> Inserted 95000 elements, the last 5000 in 0.355554 seconds
>
> Inserted 100000 elements, the last 5000 in 0.354784 seconds
>
> Inserted 105000 elements, the last 5000 in 0.355656 seconds
>
> Inserted 110000 elements, the last 5000 in 0.35539 seconds
>
> Inserted 115000 elements, the last 5000 in 0.356607 seconds
>
> Inserted 120000 elements, the last 5000 in 0.356207 seconds
>
> Inserted 125000 elements, the last 5000 in 0.356762 seconds
>
> Inserted 130000 elements, the last 5000 in 0.42747 seconds
>
> Inserted 135000 elements, the last 5000 in 0.356562 seconds
>
> Inserted 140000 elements, the last 5000 in 0.356965 seconds
>
> Inserted 145000 elements, the last 5000 in 0.357409 seconds
>
> Inserted 150000 elements, the last 5000 in 0.356946 seconds
>
> Inserted 155000 elements, the last 5000 in 0.356669 seconds
>
```

```
>  Inserted 160000 elements, the last 5000 in 0.356693 seconds
>
>  Inserted 165000 elements, the last 5000 in 0.356064 seconds
>
>  Inserted 170000 elements, the last 5000 in 0.357145 seconds
>
>  Inserted 175000 elements, the last 5000 in 0.356812 seconds
>
>  Inserted 180000 elements, the last 5000 in 0.35684 seconds
>
>  Inserted 185000 elements, the last 5000 in 0.358616 seconds
>
>  Inserted 190000 elements, the last 5000 in 0.357635 seconds
>
>  Inserted 195000 elements, the last 5000 in 0.358785 seconds
>
>  Processed 0 elements, the last 5000 in 0.365067 seconds
>
>  Processed 5000 elements, the last 5000 in 2.29138 seconds
>
>  Processed 10000 elements, the last 5000 in 5.68659 seconds
>
>  Processed 15000 elements, the last 5000 in 10.7704 seconds
>
>  Processed 20000 elements, the last 5000 in 24.9039 seconds
>
>  Processed 25000 elements, the last 5000 in 37.5853 seconds
>
>  Processed 30000 elements, the last 5000 in 46.8019 seconds
>
>  Processed 35000 elements, the last 5000 in 55.4 seconds
>
>  Processed 40000 elements, the last 5000 in 63.8799 seconds
>
>  Processed 45000 elements, the last 5000 in 72.3194 seconds
```

Hi Tom,
I'm not in a position to answer your question, but I discussed lists with a friend who understands what he's doing (at least that was the impression) and it all boils down to how lists are implemented. In IDL you're looking at a "singly-linked list of pointers" (from http://www.exelisvis.com/docs/LIST.html). According to Wiki, lists of this type have advantages and of course lots of disadvantages if compared to dynamic arrays:
http://en.wikipedia.org/wiki/Linked_list#Linked_lists_vs._dy namic_arrays

The performance difference between using values.add and values[jj] might be a direct reference to the last element of the list, making therefore add faster. But that is just speculation. However, I tested this using the following three different "add":

```
values.add, List(jj, indgen(jj mod 5 + 1), findgen(jj mod 4 + 1))
values.add, List(jj, indgen(jj mod 5 + 1), findgen(jj mod 4 + 1)), 0
values.add, List(jj, indgen(jj mod 5 + 1), findgen(jj mod 4 + 1)), jj-1
```

The computation times are, hmm..., quite different:

```
Creating list of 20000 elements
Append processing time = 0.998
Insert at beginning processing time = 0.979
Insert at end processing time = 29.334
```

I used "only" 20000 elements because I waited too long for the last operation.
This just confirms that adding an element at the end of the list without specifying the position is about as fast as at the beginning (within small error). But adding element at the end of the list is sloooow.

For testing the randomness, I used ii[jj] instead of jj and made M=500 instead of 5000. This way I get about 20 sec/500 operations.

I hope you'll find your answers in wiki or from the tests above. What I've learned (the hard way) is that lists in IDL should not be used for doing what arrays can do. They should only takeover when arrays don't work well (insert elements, changes of type or array extension).

Cheers,
Helder