Performance:
The Neverending Story

Jay Foad
Agenda

- Version 15.0
- Version 16.0
- ... and beyond!

Performance: The Neverending Story
Version 15.0

- PQA graphs look better than ever (best increase we have ever measured)
- Due to a combination of:
  - C compiler upgrades
  - Lots of individual optimisations
- Also occasional new performance features
  - E.g. 8⍷ (Inverted table index of) in version 14.1
Version 15.0 Hashed arrays

- I-beam to mark an array as a potential and likely left argument to dyadic \( \tau \) (and the other set functions)
- Better than the old \( A \circ \tau \) system
- Hash table is updated by:
  - Append idiom, ←
  - Chop idiom ↓⍨←

Performance: The Neverending Story
Version 15.0 Hashed arrays

Old way:

\[ f \leftarrow A \circ i \]
\[ f \ x \diamond f \ y \diamond f \ z \]

New way:

\[ B \leftarrow 1500 \oplus A \]
\[ B \ i x \diamond y \in B \diamond \cup B \]
\[ B, \leftarrow i 10 \diamond B \downarrow \sim \leftarrow -5 \]
Version 15.0 Chop idiom

- Fastest way of trimming a vector
- Works in place (like the append idiom)
- Also works on leading axis of any array

vec ↓⍨← -2 ⍝ chop last 2 items
mat ↓⍨← -3 ⍝ chop last 3 rows
Version 16.0

- Random bits
- Namespace refs
- Selective assignment
- Boolean algorithms
- DECF representation

Performance: The Neverending Story
Version 16.0 Random bits

Previously:

```dyalog
⎕IO←0
cmpx'?[1E6ρ2'
?1E6ρ2 → 4.5E¯3 | 0%
```

New default and optimisations in version 16.0:

```dyalog
⎕RL←θ
cmpx'?[1E6ρ2' '1E6(ρ)2'
?1E6ρ2 → 2.1E¯4 | 0%
* 1E6(ρ)2 → 7.0E¯5 | -68%
```

Performance: The Neverending Story
Version 16.0 Namespace refs

Calling a function in a namespace

ns.foo 99

has an 82% penalty

Parse dots 43%

Switch ns 39%

Call empty tradfn 100%

Parse dots 12%

Switch ns 27%

Penalty reduced to 39%
Version 16.0 Selective assignment

Selective assignment is not an efficient way to modify a few items in a large array A:

\[(2 \uparrow A) \leftarrow 99\]
\[((<2 4) \&\& A) \leftarrow 99\]

... because we generate an index array for the whole of A.
(Factor of 2 when A has 1000 items.
Factor of 1000 when A has 1E6 items.)

This has been fixed for Squad \&\& indexing
We hope to fix it for Take/Drop \&\& and Compress Bool/
Maybe others, as time permits
Version 16.0 Boolean algorithms

Coming next...

(U08) A Compendium of SIMD Boolean Array Algorithms in APL

Robert Bernecky (Snake Island Research)

Word-at-a-time algorithms for =\ and ≠\n
{ω/⍨q≠\q←ω=''''} 'Bob "SIMD" Bernecky'

"SIMD"
Version 16.0 DECF representation

- 128-bit Decimal floating point
  - Current representation is DPD: good for formatting
  - Alternative is BID: good for calculations (2x faster)

Or we could do 128-bit Binary floating point (another 2x faster for calculations)
The future

#dyalog16

Performance: The Neverending Story

<table>
<thead>
<tr>
<th>ID</th>
<th>P</th>
<th>Severity</th>
<th>Assigned To</th>
<th>Reported by</th>
<th>Updated</th>
<th>Respond by</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0013874</td>
<td>normal</td>
<td>minor</td>
<td>John Scholes</td>
<td>2016-10-07</td>
<td>speed up outer product with scalar operand</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0013737</td>
<td>normal</td>
<td>minor</td>
<td>jay</td>
<td>2016-10-06</td>
<td>compare performance of DECF DPD and BID libraries again</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0013744</td>
<td>normal</td>
<td>minor</td>
<td>jay</td>
<td>2016-10-04</td>
<td>compare performance of DECF DPD and 128-bit binary floating point libraries</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0013660</td>
<td>normal</td>
<td>minor</td>
<td>roger</td>
<td>2016-09-30</td>
<td>RFE: grade/sort of 16- and 32-column boolean matrices can be faster</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0013655</td>
<td>normal</td>
<td>feature</td>
<td>jay</td>
<td>2016-09-23</td>
<td>speed up most selective assignments by not generating the whole index array</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0013635</td>
<td>normal</td>
<td>minor</td>
<td>jay</td>
<td>2016-09-19</td>
<td>A\ can be faster</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0013224</td>
<td>normal</td>
<td>minor</td>
<td>jay</td>
<td>2016-08-24</td>
<td>don't create unnecessary &quot;apply&quot; dervs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0013736</td>
<td>normal</td>
<td>minor</td>
<td>jay</td>
<td>2016-08-05</td>
<td>don't unbias 64-bit workspaces before saving</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0013735</td>
<td>normal</td>
<td>minor</td>
<td>jay</td>
<td>2016-08-05</td>
<td>don't check for destructors and triggers before every token</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0007871</td>
<td>normal</td>
<td>minor</td>
<td>roger</td>
<td>2016-05-07</td>
<td>xxy for DECFs can be faster</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0013463</td>
<td>normal</td>
<td>minor</td>
<td>roger</td>
<td>2016-05-05</td>
<td>intolerant dyadic iota on doubles should be at least as fast as tolerant</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0012307</td>
<td>normal</td>
<td>minor</td>
<td>roger</td>
<td>2016-05-05</td>
<td>RFE: doubleint6 can be faster</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0012349</td>
<td>normal</td>
<td>minor</td>
<td>roger</td>
<td>2016-05-05</td>
<td>RFE:</td>
<td>15i-</td>
<td>x can be faster</td>
</tr>
<tr>
<td>0012306</td>
<td>normal</td>
<td>minor</td>
<td>roger</td>
<td>2016-05-05</td>
<td>RFE:</td>
<td>125boolean can be faster</td>
<td></td>
</tr>
<tr>
<td>0010150</td>
<td>normal</td>
<td>minor</td>
<td>roger</td>
<td>2016-05-05</td>
<td>RFE: Eugene Ying special: -m should be as fast as 1-m</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0013294</td>
<td>normal</td>
<td>minor</td>
<td>jay</td>
<td>2016-04-05</td>
<td>compiler: recognise some idioms with swapped arguments</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0013203</td>
<td>normal</td>
<td>minor</td>
<td>jay</td>
<td>2016-04-04</td>
<td>compiler: take advantage of optimised indexed assignment</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0013263</td>
<td>normal</td>
<td>minor</td>
<td>jay</td>
<td>2016-04-01</td>
<td>RFE: b+.xx can be sped up</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0013184</td>
<td>normal</td>
<td>minor</td>
<td>jay</td>
<td>2016-03-31</td>
<td>enable whole-program optimisation on Linux</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0012042</td>
<td>normal</td>
<td>minor</td>
<td>jay</td>
<td>2016-02-03</td>
<td>speed up comparisons</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0012031</td>
<td>normal</td>
<td>minor</td>
<td>jay</td>
<td>2015-11-26</td>
<td>speed up DECF tolerant comparison</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The future

- No shortage of work for Roger
- Squeeze more out of the C compilers
- More use of modern SIMD instructions (AVX2, POWER8)
- More to be done on namespace refs and similar targeted speed-ups