nb

February 21, 2020

1 Fast APL

An overview of topics related to writing performant code and optimising existing code.

1.1 Audience

APLers

1.2 Goals for code

In APL, the ability to express similar ideas (or even the exact same idea) in multiple ways is quite pronounced.

This double-edged sword of language is both one of the most enjoyable parts of writing (choosing an expression which suits oneself), but it is also a source of frustration ("How can I express that better?" "What is a better way to put that?" "What is the best way to express this idea?").

1.2.1 "*Better code*"

- **Aaron Hsu:** *How much (money) are you willing to bet on this code?*
- **Roger Hui:** *Monument quality code*

- **Accurate**
- **Reliable**

1.2.2 Variables

Reference: Dyalog Webinars: APL CodeGolf Autumn Tournament

- **Accurate**
-

1.3 Reliable

- **Readable:** Can a stranger understand it?
- **Fast:** Does it perform in reasonable time using reasonable resources?
- **Short:** APLers need not be convinced
- **Balanced**

Here we advocate for balanced code, as this is desirable in production.

1.4 Fast APL

- Analysis and profiling
- Mitigating hotspots through
- Implementing mechanical sympathy
- Using special cased code (The Interpretive Advantage)
- Compiling chunks
- Outsourcing jobs
- Algorithms and primitive complexity

1.5 Analysis and profiling

1.5.1 Rule `IO`

Do **not** optimise code which has **not** been measured as **slow** in realistic situations.
Optimised code is often longer and much less readable.
`dfns.life`

```
In [2]: R⍤2⍤15 7(3 39)2 3 4 5 8
        lifeNested{1 ⊖.3 4=+/,⍵1 0 1.⍵1 0 1.}
        lifeFlat{(3=⊢)4=⊢+,⌺[1 2]⍵1 0 10 9999 2⍵1 0 10 99}

In [3]: ]runtime -c "lifeFlat R" "lifeNested R"
```

1.5.2 Code analysis tools

```
PROFILE   ]Profile
dfns.cmpx  ]Runtime
```

$$(\sum_{n=1}^{N} A_n) \div N$$

$$\sum_{n=1}^{N} (A_n \div N)$$

$$(\sum_{n=1}^{N} A_n) \div N$$

```
In [4]: ]dinput
        avg1{
          N      Count elements
          s+     Sum elements
          s⌾N       Sum divided by count
        }
```

$$\sum_{n=1}^{N} (A_n \div N)$$

```
In [5]: ]dinput
        avg2{
          N      Count elements
          n⌾N     Array divided by count
          +n       Sum
        }

In [6]: )copy dfns cmpx
        n?1000
        cmpx 'avg1 n' 'avg2 n'

In [9]: PROFILE 'clear'
        PROFILE 'start'
        avg1 n
        PROFILE 'stop'
        ]Profile -lines

In [10]: repeat1e4
         _Profile{1  _PROFILE'clear'  _PROFILE 'start'  rí  _PROFILE 'stop'}

In [11]: repeat avg1 _Profile n
         VR'avg1'
         ]profile -lines

In [12]: repeat avg2 _Profile n
         VR'avg2'
         ]profile -lines
```

1.6 Mechanical Sympathy

Dyalog '18: Rectangles All The Way Down

Relatively easy gains

Avoid nested arrays or mixed-type arrays

```
In [13]: 3 4⍴ ⍳3 4    'A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I' 'J' 'K' 'L'

In [16]:  ⍳2 3    p0 p1 p2 p3 p4 p5
          p0  ⍳2 2    1 2 3 4
```

```
 p1  5    'a' 'b' 'c' 'd' 'e'
 p2  2 3    p6 p6 p6 p6
 p3      1
 p4  2    p7 p8
 p5      3
 p6  4    'w' 'o' 'r' 'd'
 p7      2
 p8      'b'
2 3(2 24)'abcde'(2 3'word')1 (2'b') 3
```

```
In [17]: 5posNest?500310
         5posFlatposNest
```

```
In [18]: ]runtime -c "0.5*+/2*-11 99posFlat" "0.5*+/Í2*.-posNest"
```

Use inverted tables: Dyalog '18: Inverted Tables

```
8
```

Do work on large arrays where possible

```
In [19]: b1=?100 1002
         ]runtime -c "+/,3<{+/,}3 3b" "+/,{3<+/,}3 3b"
```

1.7 Using special cased code

Dyalog '18: The Interpretive Advantage

```
In [20]: A?1e41e2
         ]runtime -c "{}A" "{}A"
```

Dyalog idioms
Search: *dyalog help idiom list*

```
In [21]:  Sorting idioms
         ]runtime -c "{()}A" "{}A"
```

Use `CT0` if possible

1.8 Algorithms and Primitive Complexity

Hsu, A.W., 2019. A data parallel compiler hosted on the GPU.
APL makes it easy to reason about algorithms.
https://en.wikipedia.org/wiki/Computational_complexity_of_mathematical_operations

---

Primitive complexity:

```
+      O(n)
|      O(n)
.f    O(n*2)
```

```
In [22]: PT0PrimesTil{2=+0=.|}     Primes from 1 to  using Modulo and Reduction
         PT1(~.E)1                   Without Products
         )copy dfns sieve pco
         PT2sieve 1                     Sieve of Eratosthenes
         PT310 pco 1,                   dfns.pco (lookup table)
```

```
In [23]: VR'sieve'
```

```
In [24]: _Time{0 0 0 0.2 cmpx ,' ',}
```

```
In [25]: )copy sharpplot
```

```
In [26]:  {key}Plot data;d;n;s
          :If 0=NC'key'
              key''
          :EndIf
          sNEW SharpPlot
          ndata
          :For d :In data
              s.DrawLineGraph d n
          :EndFor
          s.SetKeyText key
          View s
```

```
In [27]: n510*0.1E20
         'Modulo reduction' 'Without products' 'Sieve' 'dfns.pco' Plot n{}('PT0'_TimeÍn)('PT1'
```

```
In [28]: n510*0.1E5+18
          'Without products' 'Sieve' Plot n{}('PT1'_TimeÍn)('PT2'_TimeÍn)
         'Modulo reduction' 'Without products' 'Sieve' 'dfns.pco' Plot n{}('PT0'_TimeÍn)('PT1'
```

1.9 Compilation

- Co-dfns
- Jay's Dyalog Compiler
- APEX: The APL Parallel Executor

1.10 Fast APL

- Analysis and profiling
- Mitigating hotspots through
- Implementing mechanical sympathy
- Using special cased code (The Interpretive Advantage)
- Compiling chunks
- Outsourcing jobs
- Algorithms and primitive complexity