Performance Workshop

Dyalog ‘22

Aaron Hsu - aaron@dyalog.com

Syllabus

Theory
Physical
Implementation

The Comp. Sci. of APL Performance

Big-O

Simplified Reasoning

Big-O “Families”

```
Constant:    O(k)
Log-Linear:  O(n×⊛n)
Linear:      O(n)
Polynomial:  O(n*k)
Exponential: O(k*n)
```

Product [edit]

$$f_1 = O(g_1) \text{ and } f_2 = O(g_2) \Rightarrow f_1 f_2 = O(g_1 g_2)$$
$$f \cdot O(g) = O(fg)$$

Sum [edit]

If $f_1 = O(g_1)$ and $f_2 = O(g_2)$ then $f_1 + f_2 = O(\max(g_1, g_2))$. It follows that if $f_1 = O(g)$ and $f_2 = O(g)$ then $f_1 + f_2 \in O(g)$. In other words, this second statement says that $O(g)$ is a convex cone.

Multiplication by a constant [edit]

Let k be a nonzero constant. Then $O(|k| \cdot g) = O(g)$. In other words, if $f = O(g)$, then $k \cdot f = O(g)$.

Theoretical Factors of Performance

Run time
Memory Usage
Memory Access
Locality/Caching
Parallelism/Critical Path
Throughput/Bandwidth
Overheads

Physical Factors of Performance

How many elements?
How many bits?
How many times?
How many?

How much data do you have?

Implementation Factors for Performance

Platform Runtime Execution Model

Data First!

The Array Data Structure

<Array Structure Diagram>

Code Data Structure

Token Stream

Variable Lookup

Auxiliary Structures

Caching/ Hashtables

Primitives

Functions

Scalar

Mixed

Operators

Idiomatic APL and “Automatic” Performance Analysis

]RunTime
]SpaceNeeded

Subtleties

```
f0 ← {+⌿⍤⍨α⊂ω}
f1 ← {(1≠⊃α)↓⍨2-⌿0,(1↓α,1)⌿+⍀ω}
f2 ← {(1≠⊃α)↓(+⍀α){+⌿ω}⌸ω}
f3 ← {1↓x⊣x[+⍀α]+←ω⊣x←0⍴⍨1++⌿α}
```

Memory Management

Let’s look at some tricks

Symbols

Vs. Hashed Primitives

Inverted Tables

When not to use inverted tables?

Flat/Nested Recursive ADTs

PEG vs. LDF

Calculate space of a tree

More Code to Play with

D2P

P2D

Free Vars

Brian's Code

Basic Rules of Thumb

Flat

Small

Primitive

Parallel

Where can APL outperform traditional languages? How?

Thank you.