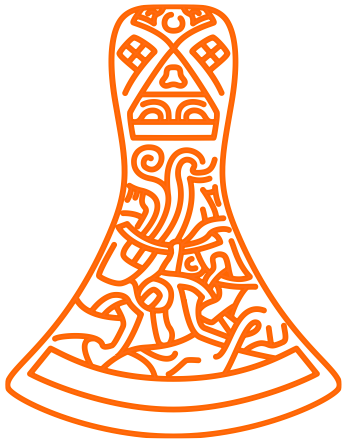
DYALOG

Glasgow 2024

Co-dfns: Roadmap and Update

Aaron W. Hsu
aaron@dyalog.com



Who am I?

Who are any of us?

For now we see but dimly, as in a mirror....

Introduction to Co-dfns

What is Co-dfns?

- A compiled implementation of APL
  - Multiple hardware platforms
  - Static analysis of APL
  - Multiple workloads
  - Flexible backend design
  - Low overhead, high productivity APL
  - Flexible integration

What is Co-dfns?

- A guide to APL programming techniques
  - Tree “wrangling” and manipulation
  - Architecture design
  - APL optimization techniques
  - Parallel techniques
  - Coding style
  - Low abstraction coding techniques

What is Co-dfns?

- A tool for scaling APL
  - Domains
  - Performance
  - Platforms
  - Integrations
  - Experiments/design

Target Use Cases

Use Case #1: GPUs

- Leverage the GPU for APL computation
- Best with primitives over large arrays
- Heavy data-crunching
- Computationally expensive algorithms
- “Hot spots” and “Kernels”
- Simulation, LLMs, Time Series, Graphs, …

Use Case #2: Scalar Hot Spots

- Inherently scalar code
- Small utility functions
- Called extremely often
- Incur excessive interpreter overhead
  - Memory churn
  - Execution churn
  - Jay Foad's Compiler

Use Case #3: Static Analysis

- Co-dfns has a static, offline parser
- Linting
- Traditional static analysis
- Migrations
  - Older versions
  - Different vendors
  - Refactoring

Use Case #4: Integration

- Low overhead packaging/deployment
- Environments lacking interpreter support
- Embedded environments
- “Hostile” deployment environments
- Different language platforms
  - JavaScript/WASM, Rust, Java, C#, etc.

Traditional APL

Traditional APL Features

- Improve integration with existing code
- Support wider range of domains
- Leverage some dynamic features
- Static analysis/migration

- Parser support vs. Executable support

Traditional APL: Parse vs. Exec

- Parser-only support:
  - :Class, OOP
  - .NET, Win32
  - HTMLRenderer
  - Interpreter Quad functions
  - Non-Dyalog dialects
- Executable support:
  - Trad-fns
  - Structured statements
  - Nested Namespaces
  - Execute
  - Generic Quad functions

Parser-only Supported Features

- Produce a viable static AST
- Aware of the semantics of these features
  - Type inference
  - Name resolution
  - Control flow
- No executability

Trad-fns

Trad-fns: Scoping concerns

- Dynamic scoping
  - Can be inherently “broken”
  - Dynamic nameclasses for variables
  - Functions that are “name builders”
  - Functions called only internally (shadowed)
  - Functions called at the root-level

Trad-fns: Scoping Approach

- Lexically “compatible”
  - Don’t shadow nameclasses of free variables
  - Don’t change “effective” nameclass
- Statically computable call graph
- Lexical/global scope
- Shadowed variables can “close” free vars

Lending a helping hand

- Parser can be parameterized with additional nameclass information
  - Resolves ambiguous situations
  - Allows for “trust me” behaviors
  - Supports a wider range of otherwise unparseable programs

Current Progress

Current Efforts

- Static Parser
- Runtime memory overheads
- GPU parallel algorithms
- Code generation
- Compiler optimizations
- Benchmarking

Progress: Parser

- Partial trad-fns support
- Tokenization:
  - Structured statements
  - OOP
  - Non-dyalog quad funcs (parameterization)
  - Dyalog Quad functions

Progress: Memory Overheads

- A major limitation on performance
- Hurts “scalar APL” and utility functions
- Creates excessive constant overheads

- Goal: Reduce or eliminate

Benchmarking

Benchmarking

- Black Scholes
- N-body
- MultiGrid (NAS Parallel)
- Volatility / Time series
- QuickHull
- FlashAttention
- Mystika (Crypto, Bignum)
- LLM Transformers
- U-net
- Graphs / Tree (Compiler)
- Small funct. Sampler
- QuAPL

Benchmarking

- Typical: 10×
- Low end: 2 – 4×
- High end: 40 – 100×

Recommendations

Recommendations/Limits

- Current versions have issues on Mac

- V4 is still the best to use for GPU:
  - Flat arrays
  - Primitives applied to large arrays

Recommendations/Limits

- V5+:
  - Most language features for dfns
  - No error guards
  - No selective assignment
  - CPU + GPU
  - Extra features and APIs

Some Unique Features

Unique Features

- Self-contained EXEs for all platforms
- High degree of integration with backends
- Platform agnostic Foreign Functions
- Extensible runtime for operators/funcs.
- Dual-licensed (Dyalog + AGPL)

Thank you.