Co-Dfns Compiler

Aaron W. Hsu
Indiana University
APL is fast. Who needs a compiler?
Compilers can enhance reliability and performance.
Interpreter Limitations

Scalar fusion leads to poor cache behavior

Procedure calls can be expensive

Common optimizations are not possible

No static analysis of code for errors or correctness
Co-Dfns is a headlong jump into APL compilation.
D-fns eliminates serious roadblocks to performance.
The interpreter can have poor cache behavior.

\[((S÷X)+(r+2÷v×2)×T)÷v\sqrt{T←v×T×0.5}\]
Co-Dfns is for predictable, safe, controllable performance.
Tuning expertise should not be a black art.
Co-Dfns is a two-primitive extension of the D-fns language.
Examples

X←5 5ρ® A 5×5 single-assignment array
X[0;0]←3 A Set a single assignment cell
3≡X[0;0] A Reference a cell
?≡X[0;1] A Blocking call waiting for data
F∥¨ A Parallel Each
A “Parallel” reduction of depth 1 vector

VECRED←{
  Z[]←(⊃ω),(−1↓iρω)αα{Z[α] αα ω}∥¨1↓ω→Z←⊙ρ·ρω
}

"Parallel" reduction of depth 1 vector
Examples

TreeVecRed ← {
}

Life ← { 1 w v . ^ 3 4 = + / , 1 0 - 1 0 . 0 0 - 1 φ . . < ω }
LifeP ← Life ∥
LifeP2 ← { 1 w v . ^ 3 4 = + / , 1 0 - 1 . 0 . ( θ ∥ ) 1 0 - 1 φ ∥ . . < ω }
Why shouldn’t we be able to reason about performance?

ZPL and others make this a critical component.
APL is math you can use. Let's use it for performance.
Examples

\[ \text{Life} \leftarrow \{ \gamma 1 \omega \nu \cdot ^3 4 = +/-, -1 0 1 \cdot \theta -1 0 1 \phi \cdots = \omega \} \]
\[ (\rho \text{Life} A) \leftrightarrow (\rho A) \]

\[ (Z \ F \ (Y \ F \ X)) \leftrightarrow (Z \ F \ Y) \ F \ X \]
\[ F / \omega \]

\[ (G \ \text{DeepMap} \leftrightarrow G) \]
\[ X + Y \ G \ Z \leftrightarrow \{ x \ y \ z \leftarrow \omega \ \diamond \ x + y \ G \ z \} \ \text{DeepMap} \ X \ Y \ Z \]
\[ \leftrightarrow \{ ((, 3) \equiv \rho \omega) \ x \ y \ z \leftarrow \omega \ \diamond \ x + y \ G \ z \} \ \text{DeepMap} \ X \ Y \ Z \]
Examples

\[
\{(\emptyset \equiv \rho S)^\land (F \ \text{DeepMap} \leftrightarrow F)\}
\]

\[
M \ F \ (\rho M)\rho S \rightarrow M \ F \ S
\]

\[
\{(\emptyset \equiv \rho X)^\land (X \ F \ Y \leftrightarrow Y \ F \ X)\}
\]

\[
X \ F \ Y \rightarrow Y \ F \ X
\]
Plans and Schemes

Usable version this year

Target multi-core, GPU, and distributed clusters

Fully integrate with Dyalog

Focus on scalable parallel performance

Leverage APL-style formalism

Create a dialog between tuning expert and compiler
Version 1 will have low-hanging fruit optimizations
Interpreter vs. Compiler

**INTERPRETER**
- Garbage collected
- Idiom-based special casing
- No static verification
- No user-guided optimization
- Supports much more of APL
- Numerous extensions

**COMPILER**
- Stack-based allocation
- Whole program optimization
- Explicit verification and proof
- Safe user-defined optimizations
- Restricted to Co-Dfns
- Limited extensions and interop
Using Co-Dfn should require less than trivial effort.
Demo time
Performance not included, some assembly required.
We're not preaching super-compilation here.
Coding Goodies

Easily integrated with other code

A complete, useful, general parser for APL

A complete, rigorous specification of the language

Supports multiple runtimes

Language integration
Acknowledgments

Dyalog has provided significant funding and support.

PL Group at IU is a great place to test and strengthen these ideas.

John Scholes, for creating D-fns.

Intel’s Concurrent Collections
Mathematics of Arrays
C++ (Oh dear!)
NanoPass
Thank you

Email: awhsu@indiana.edu
Photo Credits

Hotnstock
MoonsongStock
Ladyaway
Fall_Stock
Estruda
Animalphotos
Dark_dragon_stock
Drezdany_stocks
Stupiddeeppeople
Thiselectricheart
Vampsstock
NefaroStock
NikiljuiceStock