SPIRAL Demo:

Generating Hyper-Portable Future-Proof Computational Kernels

F. Franchetti, J. C. Hoe, T.-M. Low, J.M.F. Moura
Carnegie Mellon University

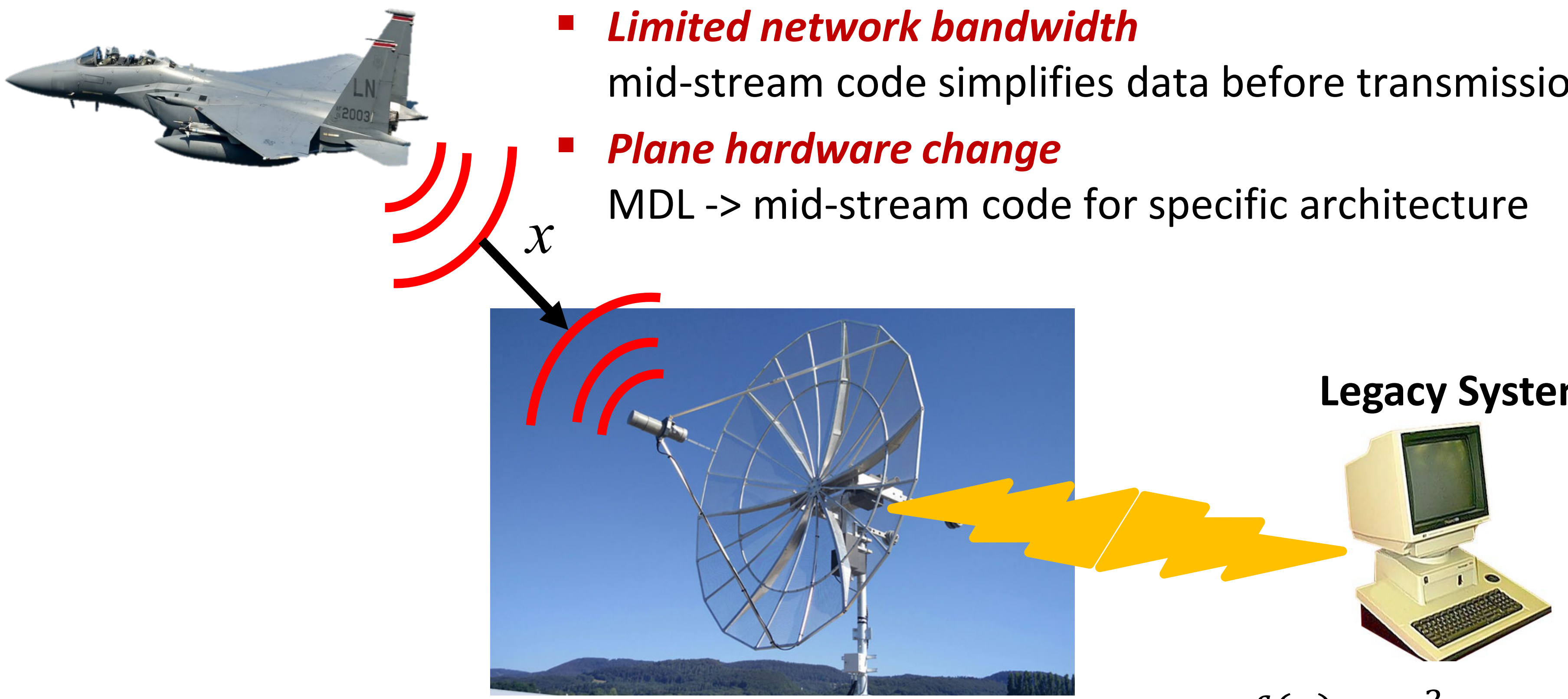
D. A. Padua
UIUC

M. Franusich
SpiralGen, Inc.

BRASS Phase III Challenge Problems

Post-deployment and in-mission adaptations

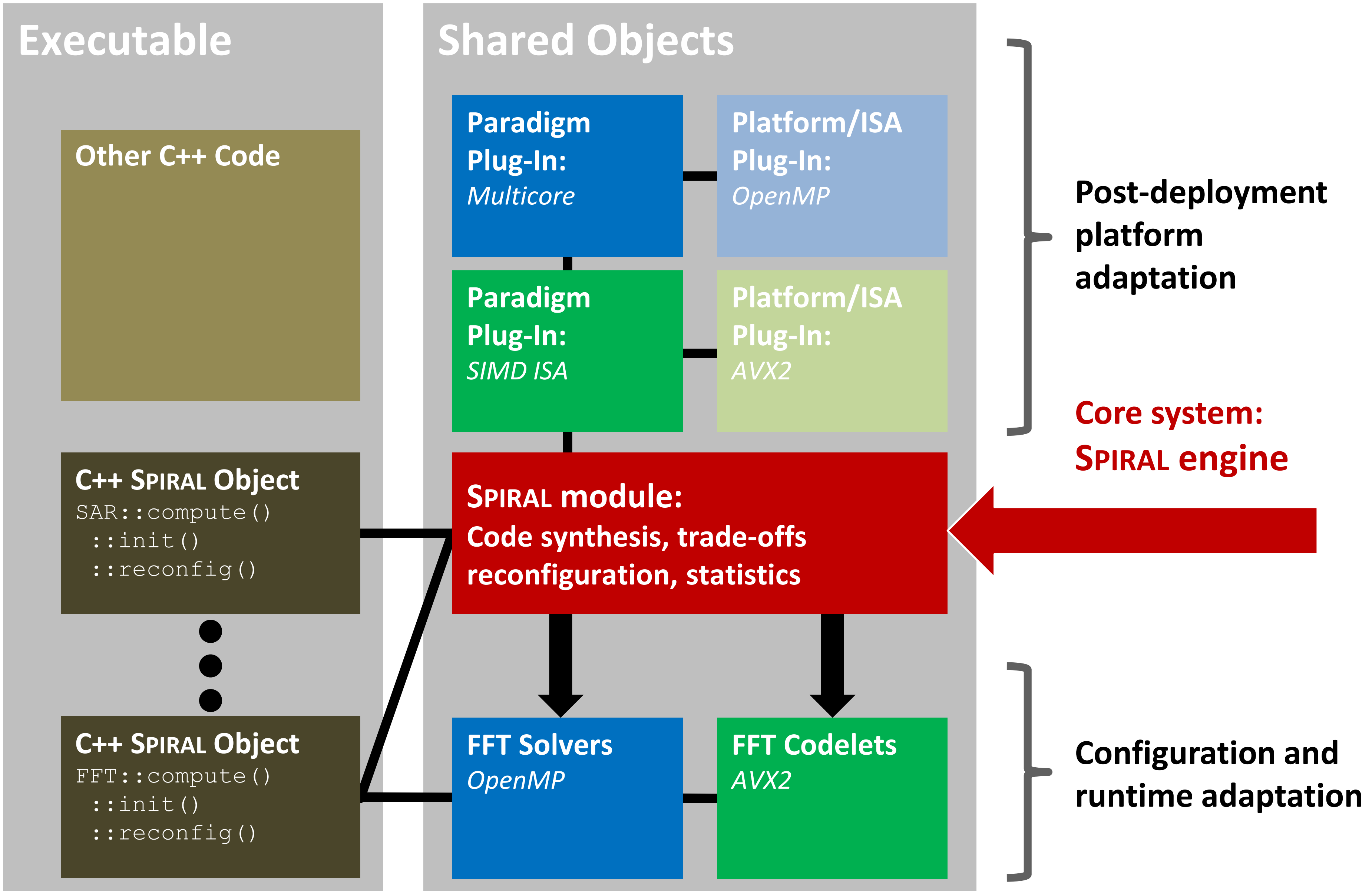
- ***Data stream too complex for processor***
  mid-stream code precomputes functions
- ***Limited network bandwidth***
  mid-stream code simplifies data before transmission
- ***Plane hardware change***
  MDL -> mid-stream code for specific architecture



$$f(x) = x^2 + x + 1$$

Numerical Processing for Computationally Limited Systems

Performance Portability: Behind The Scenes



Autotuning in Constraint Solution Space

$$\underbrace{\mathrm{DFT}_8}_{\mathrm{AVX}(2\text{-way }\mathbb{C})}$$

AVX 2-way
`_Complex double`

DFT_8

Base cases

$A^{n\times n}\vec{\otimes}\, I_2$

$\underbrace{L^4_2}_{\text{vec}(2)}$

$\underbrace{T^{mn}_n}_{\text{vec}(2)}$

Transformation rules

$$(I_m\otimes A^{n\times n})L^{mn}_m \to \left(I_{m/\nu}\otimes L^{n\nu}_\nu(A^{n\times n}\otimes I_\nu)\right)(L^{mn/\nu}_{m/\nu}\otimes I_\nu)$$

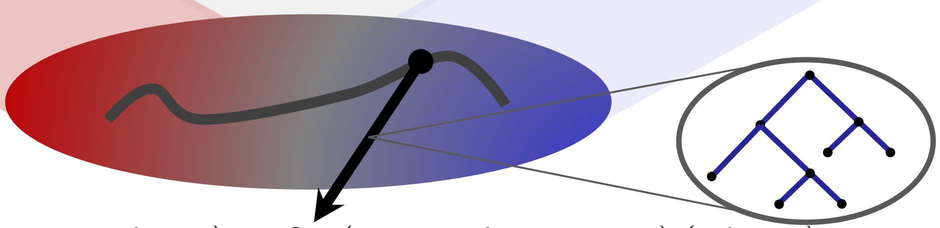
$$L^{n\nu}_\nu \to (L^n_\nu\otimes I_\nu)(I_{n/\nu}\otimes L^{\nu^2}_\nu)$$

$$A^{m\times m}\otimes I_n \to (A^{m\times m}\otimes I_{n/\nu})\otimes I_\nu$$

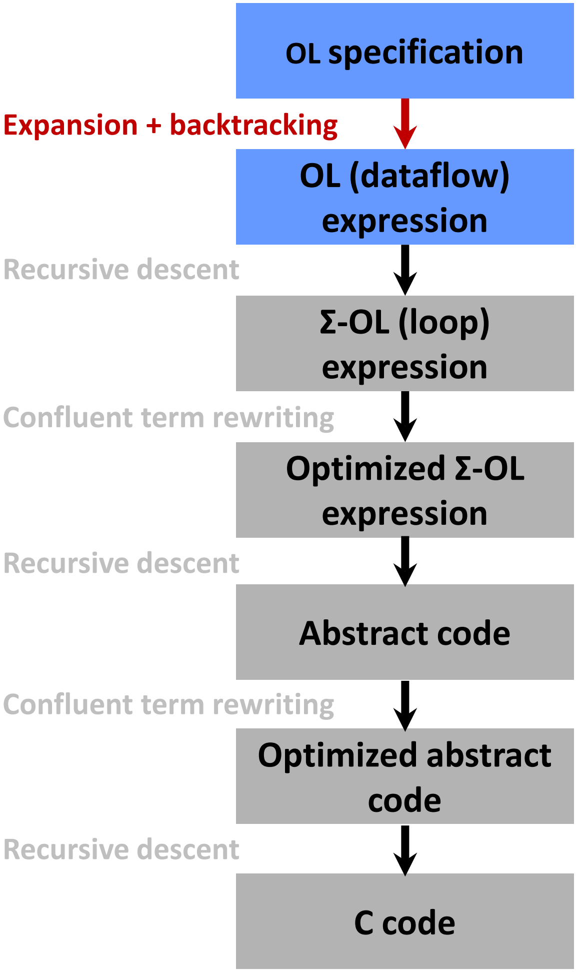
Breakdown rules

$$\mathrm{DFT}_{mn}\to(\mathrm{DFT}_m\otimes I_n)T^{mn}_n(I_m\otimes \mathrm{DFT}_n)L^{mn}_m$$

$$\mathrm{DFT}_2\to F_2$$

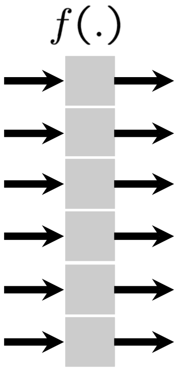


$$\left((F_2\otimes I_2)T^4_2(I_2\otimes F_2)L^4_2\vec{\otimes}\, I_2\right)\underbrace{T^8_2}_{\text{vec}(2)}\left(I_2\otimes \underbrace{L^4_2}_{\text{vec}(2)}(F_2\vec{\otimes}\, I_2)\right)\left(L^4_2\vec{\otimes}\, I_2\right)$$



Operator Language

- **Basic operators ≈ functional programming constructs**



map	$\text{Pointwise}_{n,f_i} : \mathbb{R}^n \to \mathbb{R}^n$ $(x_i)_i \mapsto f_0(x_0) \oplus \cdots \oplus f_{n-1}(x_{n-1})$
binop	$\text{Pointwise}_{n\times n,f_i} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$ $\big((x_i)_i, (y_i)_i\big) \mapsto f_0(x_0, y_0) \oplus \cdots \oplus f_{n-1}(x_{n-1}, y_{n-1})$
fold	$\text{Reduction}_{n,f_i} : \mathbb{R}^n \to \mathbb{R}$ $(x_i)_i \mapsto f_{n-1}(x_{n-1}, f_{n-2}(x_{n-2}, f_{n-3}(\ldots f_0(x_0, \text{id}())\ldots)$

- **One-time effort:** **mathematical library**

$$d_\infty^n(.,.) \to \|.\|_\infty^n \circ (-)_n$$

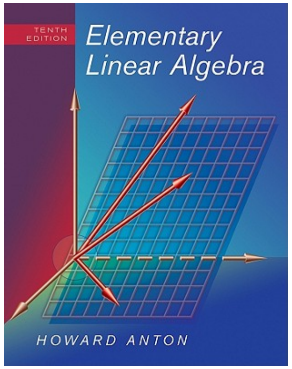
$$(\diamond)_n \to \text{Pointwise}_{n\times n,(a,b)\mapsto a\diamond b}, \quad \diamond \in \{+, -\cdot, \wedge, \vee, \ldots\}$$

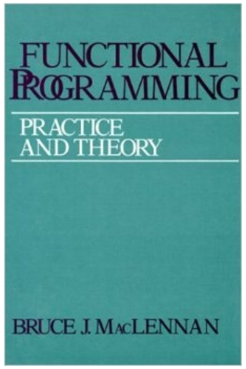
$$\|.\|_\infty^n \to \text{Reduction}_{n,(a,b)\mapsto \max(|a|,|b|)}$$

$$< .,. >_n \to \text{Reduction}_{n,(a,b)\mapsto a+b} \circ \text{Pointwise}_{n\times n,(a,b)\mapsto ab}$$

$$P[x, (a_0, \ldots, a_n)] \to < (a_0, \ldots, a_n), . > \circ (x^i)_n$$

$$(x^i)_n \to \text{Induction}_{n,(a,b)\mapsto ab,1}$$



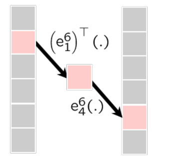


Loop and Code Level Rule System

Mathematical Loop Abstraction

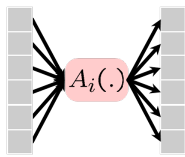
- **Selection and embedding operator:** *gather and scatter*

$$(\mathrm{e}_i^n)^\top(.) : \mathbb{R}^n \to \mathbb{R}^1$$
$$(x_i)_{i=0,\dots,n-1} \mapsto x_i$$
$$\mathrm{e}_i^n(.) : \mathbb{R}^1 \to \mathbb{R}^n$$
$$(x) \mapsto (0,\dots,0,\underbrace{x}_{i^{\text{th}}},0,\dots,0)$$



- **Iterative operations:** *loop*

$$\coprod_{i=0}^{n-1} : (D \to R)^n \to (D \to R)$$
$$A_i \mapsto (x \mapsto A_0(x) \sqcup \dots \sqcup A_{n-1}(x))$$
$$\text{with } \sqcup \in \{\textstyle\sum, \vee, \wedge, \prod, \min, \max, \dots\}$$



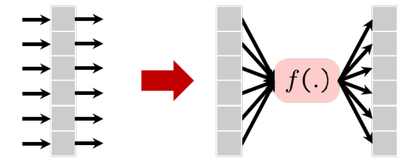
- **Atomic operators:** *nonlinear scalar functions*

$$\mathrm{Atomic}_f : \mathbb{R}^1 \to \mathbb{R}^1$$
$$(x) \mapsto (f(x))$$

Translation and Optimization

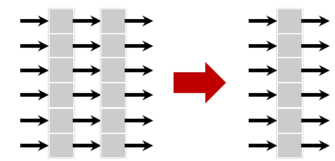
- **Translating Basic OL into Σ-OL**

$$\mathrm{Pointwise}_{n,f_i} \to \sum_{i=0}^{n-1} \left(\mathrm{e}_i^n \circ \mathrm{Atomic}_{f_i} \circ (\mathrm{e}_i^n)^\top\right)$$
$$\mathrm{Reduction}_{n,(a,b)\mapsto a+b} \to \sum_{i=0}^{n-1} (\mathrm{e}_i^n)^\top$$



- **Optimizing Basic OL/Σ-OL**

$$\mathrm{Pointwise}_{n,f_i} \circ \mathrm{Pointwise}_{n,g_i} \to \mathrm{Pointwise}_{n,f_i \circ g_i}$$
$$\mathrm{Pointwise}_{n,f_i} \circ \mathrm{e}_n^j \to \mathrm{e}_n^j \circ \mathrm{Pointwise}_{1,f_j}$$



Abstract Code

Code objects

- Values and types
- Arithmetic operations
- Logic operations
- Constants, arrays and scalar variables
- Assignments and control flow

Properties: at the same time

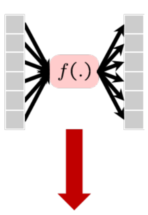
- Program = (abstract syntax) tree
- Represents program in restricted C
- OL operator over real numbers and machine numbers (floating-point)
- Pure functional interpretation
- Represents lambda expression

```
# Dynamic Window Monitor

let(
  i3 := var("i3", TInt), i5 := var("i5", TInt),
  w2 := var("w2", TBool), w1 := var("w1", T_Real(64)),
  s8 := var("s8", T_Real(64)), s7 := var("s7", T_Real(64)),
  s6 := var("s6", T_Real(64)), s5 := var("s5", T_Real(64)),
  s4 := var("s4", T_Real(64)), s1 := var("s1", T_Real(64)),
  q4 := var("q4", T_Real(64)), q3 := var("q3", T_Real(64)),
  D := var("D", TPtr(T_Real(64)).aligned([16, 0])),
  X := var("X", TPtr(T_Real(64)).aligned([16, 0])),

  func(TInt, "dwmonitor", [ X, D ],
     decl([q3, q4, s1, s4, s5, s6, s7, s8, w1, w2],
        chain(
           assign(s5, V(0.0)),
           assign(s8, nth(X, V(0))),
           assign(s7, V(1.0)),
           loop(i5, [0..2],
              chain(
                 assign(s4, mul(s7, nth(D, i5))),
                 assign(s5, add(s5, s4)),
                 assign(s7, mul(s7, s8))
              )
           ),
           assign(s1, V(0.0)),
           loop(i3, [0..1],
              chain(
                 assign(q3, nth(X, add(i3, V(1)))),
                 assign(q4, nth(X, add(V(3), i3))),
                 assign(w1, sub(q3, q4)),
                 assign(s6, cond(geq(w1, V(0)), w1, neg(w1))),
                 assign(s1, cond(geq(s1, s6), s1, s6))
              )
           ),
           assign(w2, geq(s1, s5)),
           creturn(w2)
        )
     )
  )
)
```

Rule Based Compiler

Compilation rules: recursive descent

$$\mathrm{Code}\big(y = (A \circ B)(x)\big) \to \big\{\mathtt{decl}(t), \mathrm{Code}\big(t = B(x)\big), \mathrm{Code}\big(y = A(t)\big)\big\}$$
$$\mathrm{Code}\left(y = \left(\sum_{i=0}^{n-1} A_i\right)(x)\right) \to \big\{y := \vec{0}, \mathtt{for}(i = 0..n-1)\ \mathrm{Code}\big(y {+}{=} A_i(x)\big)\big\}$$
$$\mathrm{Code}\big(y = (\mathrm{e}_i^n)^\top(x)\big) \to y[0] := x[i]$$
$$\mathrm{Code}\big(y = \mathrm{e}_i^n(x)\big) \to \big\{y = \vec{0}, y[i] := x[0]\big\}$$
$$\mathrm{Code}\big(y = \mathrm{Atomic}_f(x)\big) \to y[0] := f(x[i])$$




```
chain(
   assign(Y, V(0.0),
   loop(i1, [0..5],
      assign(nth(y, i1),
       f(nth(X, i1)))
   )
)
```

Cleanup rules: term rewriting

```
chain(a, chain(b))     → chain([a, b])
decl(D, decl(E, c))    → decl([D, E], c)
loop(i, decl(D, c))    → decl(D, loop(i, c))
chain(a, decl(D, b))   → decl(D, chain([a, b]))
```

BRASS Phase III: MDL of Complex Expression

```
<Function ID="F_003">
    <Name>Conditional Measurement</Name>
    <Description>Output 1 when input > 0, else output sin(input)</Description>
    <InputCount>1</InputCount>
    <UpdateFrequency>IfAny</UpdateFrequency>
    <ConditionalBlock>
      <If>
        <Conditional>x &lt; 0</Conditional>
        <Result>1.0</Result>
      </If>
      <Else>
        <Result>sin(x)</Result>
      </Else>
    </ConditionalBlock>
  </Function>
```

SwRI MDL file snippet

https://www.spiral.net/doc/usermanual/examples/index.html#advanced-examples-from-darpa-brass-demonstrating-hcol

Leave Behind: SPIRAL 8.1.1/BRASS

- **Open Source SPIRAL available**
  - non-viral license (BSD)
  - Initial version, effort ongoing to open source whole system
  - Open sourced under DARPA PERFECT
  - Commercial support via SpiralGen, Inc.
- **Developed over 20 years**
  Funding: DARPA (OPAL, DESA, HACMS, PERFECT, ***BRASS***), NSF, ONR, DoD HPC, JPL, DOE, CMU SEI, Intel, Nvidia, Mercury
- **SPIRAL Tutorial**
  yearly at IEEE HPEC
  www.spiral.net/tutorial-hpec2019.html

F. Franchetti, T. M. Low, D. T. Popovici, R. M. Veras, D. G. Spampinato, J. R. Johnson, M. Püschel, J. C. Hoe, J. M. F. Moura:
SPIRAL: Extreme Performance Portability, Proceedings of the IEEE, Vol. 106, No. 11, 2018.
Special Issue on *From High Level Specification to High Performance Code*

Simple FFT Examples

FFT: Scalar C code

```
opts := SpiralDefaults;
transform := DFT(4);
ruletree := RandomRuleTree(transform, opts);
icode := CodeRuleTree(ruletree, opts);
PrintCode("DFT4", icode, opts);
```

FFT: AVX 4-way double-precision C + intrinsics

```
opts := SIMDGlobals.getOpts(AVX_4x64f);
transform := TRC(DFT(16)).withTags(opts.tags);
ruletree := RandomRuleTree(transform, opts);
icode := CodeRuleTree(ruletree, opts);
PrintCode("AVX_DFT16", icode, opts);
```

Advanced FFT Examples

FFT: 2-way OpenMP Multi-Threaded SSE2 Code

```
opts := LocalConfig.getOpts(
            rec(dataType := T_Real(64), globalUnrolling := 512),
            rec(numproc := 2, api := "OpenMP"),
            rec(svct := true, splitL := false, oddSizes := false,
                stdTTensor := true, tsplPFA := false));
transform := TRC(DFT(32)).withTags(opts.tags);
ruletree := RandomRuleTree(transform, opts);
icode := CodeRuleTree(ruletree, opts);
PrintCode("SSE_OMP2_DFT32", icode, opts);
```

FFT: 4-way AVX C + intrinsics, Dynamic Programming

```
opts := SIMDGlobals.getOpts(AVX_4x64f);
transform := TRC(DFT(16)).withTags(opts.tags);
best := DP(transform, rec(), opts);
ruletree := best[1].ruletree;
icode := CodeRuleTree(ruletree, opts);
PrintCode("AVX_DFT16", icode, opts);
```

BRASS Phase III Hello World

```
Load(hcol);
Import(hcol);

opts := HCOLopts.getOpts(rec());
opts.useCReturn := true;
opts.YType := TPtr(T_Real(64));

hcol := Reduction(300, (a, b)->add(a, b), V(0), False);

icode := CoSynthesize(hcol, opts);

PrintCode("sum", icode, opts);
```

More Complex: Length of Vector

```
Load(hcol);
Import(hcol);

opts := HCOLopts.getOpts(rec());
opts.useCReturn := true;
opts.YType := TPtr(T_Real(64));

len := 128;
funcname := "l2norm"::StringInt(len);
filename := funcname::".c";
x := var("x", T_Real(64));
i := Ind(1);

hcol := OLCompose(
    PointWise(1, Lambda([x, i], sqrt(x))),
    Reduction(len, (a, b)->add(a, b), V(0), False),
    PointWise(len, Lambda([x, i], mul(x, x)))
);
icode := CoSynthesize(hcol, opts);

PrintCode(funcname, icode, opts);
```

Complex Expression: Preamble

```
Load(hcol);
Import(hcol);

opts := HCOLopts.getOpts(rec());
opts.useCReturn := false;
opts.includes := ["<math.h>"];
opts.XType := TPtr(TInt);
X.t := TPtr(TInt);
opts.globalUnrolling := 2;
opts.YType := TPtr(TDouble);
opts.arrayDataModifier := "";
opts.arrayBufModifier := "";
funcname := "F_003";
filename := "F_003.c";
tx := var("tx", TInt);
ty := var("ty", TInt);
i := var("i", TInt);
n := var("N", TInt);
runs := Ind(n);
opts.params := [n];
__NUM_VAR__ := 1;
```

Complex Expression: MDL->OL Parsed

```
hcol := IterDirectSum(runs, n,
  TCond(
    TLess(
      GathH(__NUM_VAR__, 1, 0, 1),
      OLCompose(
        PointWise(1, Lambda([tx,i], 0)),
        GathH(__NUM_VAR__, 1, 0, 1)
    )),
    OLCompose(
      PointWise(1, Lambda([tx,i], 1.0)),
      GathH(__NUM_VAR__, 1, 0, 1)),
    OLCompose(
      PointWise(1, Lambda([tx,i], sin(tx))),
      GathH(__NUM_VAR__, 1, 0, 1)
    )
  )
);

icode := CoSynthesize(hcol, opts);
PrintCode(funcname, icode, opts);
```

More Examples: Signal Processing

Definitions

```
t1 := DFT(4);           # complex DFT of size 4
t2 := MDDFT([4,4]);     # 2D DFT
t3 := DFT(5);           # non 2-power DFT
Import(dct_dst);        # load DCT/DST package
t4 := DCT3(8);          # cosine transform of type 3, size 8
Import(filtering);      # load package filtering
t5 := Filt(4, [1,2,3,4]);      # FIR filter with constant taps
Import(wht);            # load Walsh-Hadamard Transform
t6 := WHT(3);           # WHT of size 8
```

Operations on functions

```
DoForAll([t1,t2,t3,t4,t5,t6], # print them all as matrices
    t->Print(pm(t), "\n"));
t1.terminate();               # translate into matrix
t4.transpose();               # transposed transform
t1.conjTranspose();           # conjugated transposed transform
t3.inverse();                 # inverse transform transform
t2.dims();                    # transforms have a size

SpiralDefaults.breakdownRules;# all transforms known to the system
```

Walkthrough Example

From Transform to code: stepwise

```
n := 8; k := -1;                    # transform parameters
opts := SpiralDefaults;             # default options
opts.useDeref := false;             # prefer array[] over *(deref)
t := DFT(n, k);                     # transform
rt := RandomRuleTree(t, opts);             # get rule tree
spl := SPLRuleTree(rt);             # Debug: SPL formula
ss1 := spl.sums();       # Debug: SPL->Sigma-SPL w/o optimization
ss := SumsRuleTree(rt, opts); # Correct: from rt -> Sigma-SPL
c1 := CodeSums(ss, opts);           # Debug: Sigma-SPL->code
c := CodeRuleTree(rt, opts);        # Correct: rt-> code in one shot
PrintCode("dft8", c, opts);         # final code
```

Correctness checks

```
tm := MatSPL(t);                    # symbolic complex cyclotomic matrix
tmr := MatSPL(RC(t));               # symbolic real cyclotomic matrix
splm := MatSPL(spl);                # symbolic complex cyclotomic matrix
tmr := MatSPL(RC(t));               # symbolic real cyclotomic matrix
ssm := MatSPL(ss);                  # symbolic double-precision matrix
cm := CMatrix(c, opts);             # symbolic double-precision matrix
tm = splm;                          # symbolically equivalent
InfinityNormMat(tmr - ssm);         # only equivalent up to rounding error
InfinityNormMat(tmr - cm);          # only equivalent up to rounding error
```

More Examples

From Transform to code -- stepwise

```
n := 1024; k := -1;              # transform parameters
opts := SpiralDefaults;          # default options
opts.globalUnrolling := 16;      # set smaller unrolling
t := DFT(n, k);                  # transform
best := DP(t, rec(), opts);      # run search
rt := best[1].ruletree;
c := CodeRuleTree(rt, opts);     # Correct: rt-> code in one shot
PrintCode("dft"::StringInt(n), c, opts);     # final code
```

Other Examples

```
Import(dct_dst, realdft);        # load DCT/DST and Real DFT package
opts := SpiralDefaults;          # default options
t1 := DFT(31);          # a larger prime size
t2 := DCT3(32);         # a larger cosine transform of type 3
t3 := PRDFT(17);        # Real DFT in the "pack" format
t4 := PrunedDFT(128, 16, [0,1,5,6,7]);

ts := [t1, t2, t3, t4];
rts := List(ts, tt->RandomRuleTree(tt, opts));
cs := List(rts, rr->CodeRuleTree(rr,
          CopyFields(SpiralDefaults, rec(globalUnrolling := 64))));
```

Basic Block Compiler

Top-level flow

```
opts := SpiralDefaults;
s := SumsRuleTree(RandomRuleTree(DFT(8), opts), opts);
c := DefaultCodegen(s, Y, X, opts);
Compile(c, opts);
```

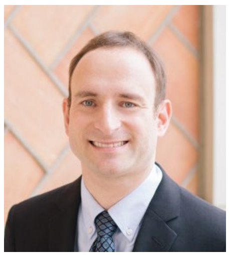
Basic Block Compilation, Stage by Stage

```
c := Compile.pullDataDeclsRefs(c);
c := Compile.fastScalarize(c);
c := UnrollCode(c);
c := FlattenCode(c);
c := UntangleChain(c);
c := CopyPropagate.initial(c, opts);
c := HashConsts(c, opts);
c := MarkDefUse(c);
c := BinSplit(c, opts);
c := MarkDefUse(c);
c := CopyPropagate(c, opts);
c := BinSplit(c, opts);
c := FixValueTypes(c);
c := Compile.declareVars(c);
PrintCode("dft8", c, opts);
```

More Information:

www.spiral.net

SPIRAL BRASS PIs


Franz Franchetti


José M.F. Moura


James C. Hoe


Tze Meng Low


David Padua


Mike Franusich

SPIRAL BRASS Research Scientist, Post-Docs and Students

Daniele Spampinato

Thom Popovici
Fazle Sadi

Paul Brouwer
Guanglin Xu
Jiyuan Zhang
Mark Blanco
Elliott Binder