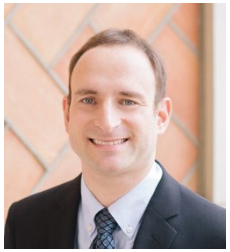
Open Source SPIRAL 8.3
System Description

Tutorial

Franz Franchetti
Carnegie Mellon University

Mike Franusich
SpiralGen, Inc.


Franz Franchetti


Mike Franusich

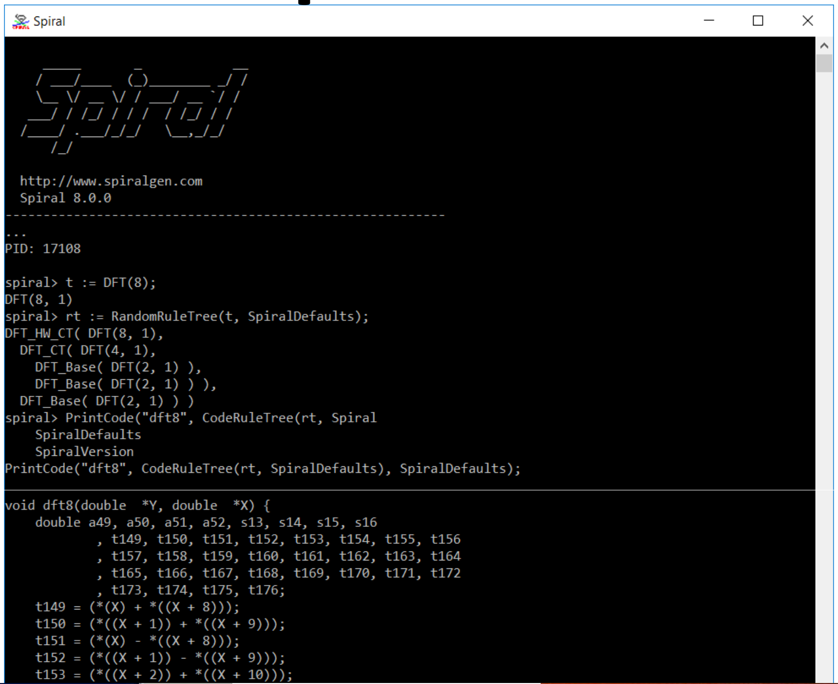
Tutorial based on joint work with the Spiral team at CMU, UIUC, and Drexel

Organization

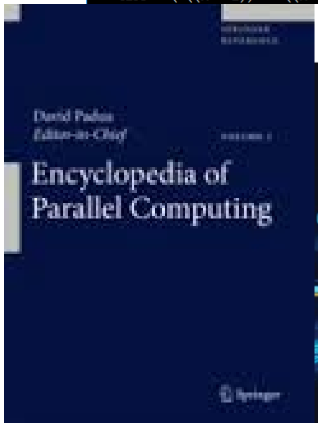
- **Overview**
- System
- Top level commands
- Abstractions
- Rewriting System I: RuleTree/backtracking search
- Rewriting System II: Visitor Patterns
- Rewriting System III: Associative/large context rules
- Basic block compiler

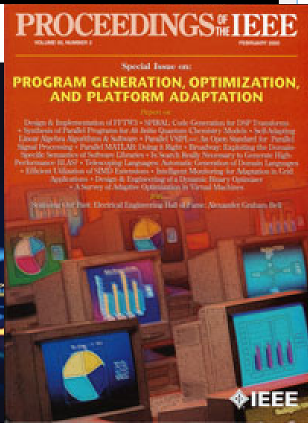
SPIRAL 8.3.0: Available Under Open Source

- **Open Source SPIRAL available**
  - non-viral license (BSD)
  - Initial version, effort ongoing to open source whole system
  - 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
  - Open sourced under DARPA PERFECT



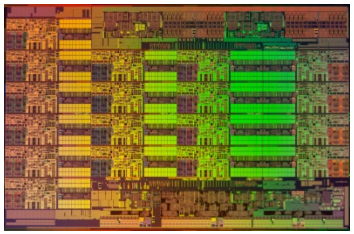


www.spiral.net

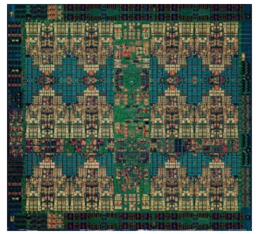
F. Franchetti, T. M. Low, D. T. Popovici, R. M. Veras, D. G. Spampinato, J. R. Johnson, M. Püschel, J. C. Hoe, J. M. F. Moura: SPIRAL: Extreme Performance Portability, **Proceedings of the IEEE, Vol. 106, No. 11, 2018.**
Special Issue on *From High Level Specification to High Performance Code*

Today’s Computing Landscape

1 Gflop/s = one billion floating-point operations (additions or multiplications) per second

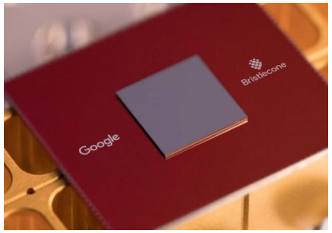


Intel Xeon 8380HL
2.5 Tflop/s, 205 W
28 cores, 2.9—4.3 GHz
2-way—16-way AVX-512



IBM POWER9
768 Gflop/s, 300 W
24 cores, 4 GHz
4-way VSX-3

Nvidia Tesla A100
9.7/19.5 Tflop/s, 400 W
6912 cores, 1.4 GHz
32-way SIMT, tensor cores



Google Bristlecone
72 qubits

Snapdragon 835
15 Gflop/s, 2 W
8 cores, 2.3 GHz
A540 GPU, 682 DSP, NEON

Intel Atom C3858
32 Gflop/s, 25 W
16 cores, 2.0 GHz
2-way/4-way SSSE3

Dell PowerEdge R940
3.2 Tflop/s, 6 TB, 850 W
4x 24 cores, 2.1 GHz
4-way/8-way AVX

Summit
187.7 Pflop/s, 13 MW
9,216 x 22 cores POWER9
+ 27,648 V100 GPUs

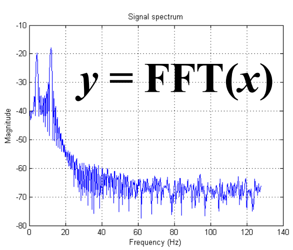
SPIRAL: AI for Performance Engineering

Given:

- **Mathematical problem specification**
  core mathematics does not change
- **Target computer platform**
  varies greatly, new platforms introduced often

Wanted:

- **Very good implementation of specification on platform**
- **Proof of correctness**



on

automatic

```
void fft64(double  *Y, double  *X) {
    ...
    s5674 = _mm256_permute2f128_pd(s5672, s5673, (0) | ((2) << 4));
    s5675 = _mm256_permute2f128_pd(s5672, s5673, (1) | ((3) << 4));
    s5676 = _mm256_unpacklo_pd(s5674, s5675);
    s5677 = _mm256_unpackhi_pd(s5674, s5675);
    s5678 = *((a3738 + 16));
    s5679 = *((a3738 + 17));
    s5680 = _mm256_permute2f128_pd(s5678, s5679, (0) | ((2) << 4));
    s5681 = _mm256_permute2f128_pd(s5678, s5679, (1) | ((3) << 4));
    s5682 = _mm256_unpacklo_pd(s5680, s5681);
    s5683 = _mm256_unpackhi_pd(s5680, s5681);
    t5735 = _mm256_add_pd(s5676, s5682);
    t5736 = _mm256_add_pd(s5677, s5683);
    t5737 = _mm256_add_pd(s5670, t5735);
    t5738 = _mm256_add_pd(s5671, t5736);
    t5739 = _mm256_sub_pd(s5670, _mm256_mul_pd(_mm_vbroadcast_sd(&(C22)), t5735));
    t5740 = _mm256_sub_pd(s5671, _mm256_mul_pd(_mm_vbroadcast_sd(&(C22)), t5736));
    t5741 = _mm256_mul_pd(_mm_vbroadcast_sd(&(C23)), _mm256_sub_pd(s5677, s5683));
    t5742 = _mm256_mul_pd(_mm_vbroadcast_sd(&(C23)), _mm256_sub_pd(s5676, s5682));
    ...
}
```

performance

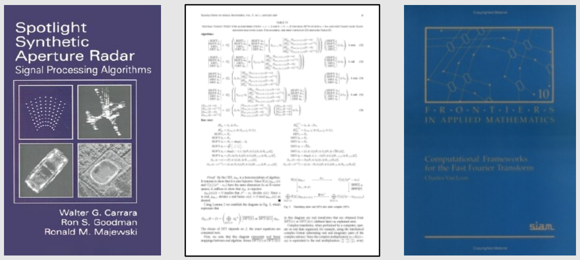
+

PROOF

QED.

SPIRAL As AI System

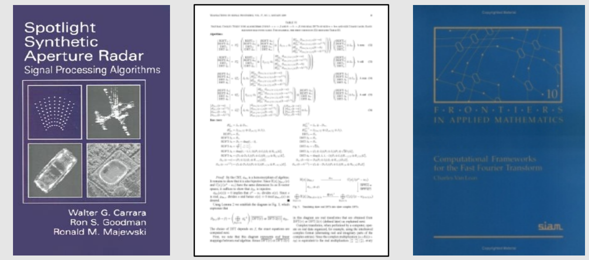
Traditionally



High performance library optimized for given platform

Comparable performance

Spiral Approach

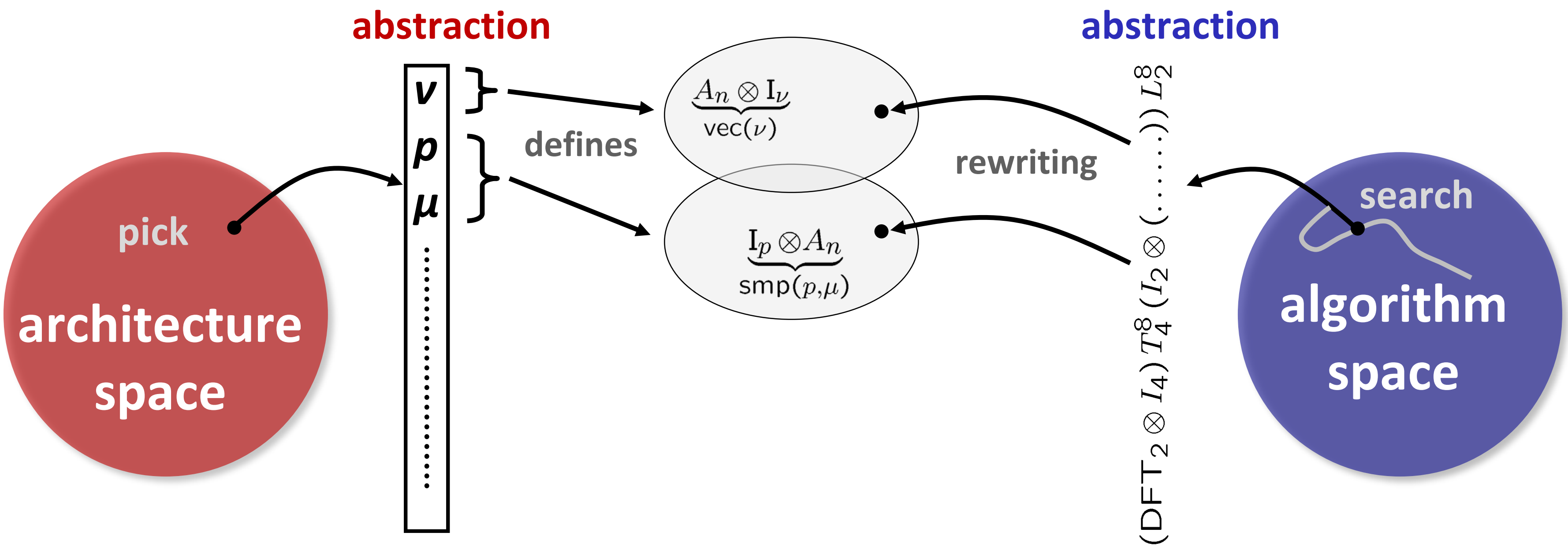


Spiral

High performance library optimized for given platform

Platform-Aware Formal Program Synthesis

Model: common abstraction
= spaces of matching formulas



Architectural parameter:
Vector length,
#processors, …

optimization

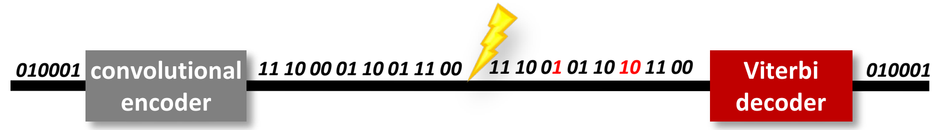
Kernel:
problem size,
algorithm choice

Some Application Domains in SPL

Linear Transforms

$$
\begin{aligned}
\mathbf{DFT}_n &\to (\mathbf{DFT}_k \otimes \mathrm{I}_m)\,\mathrm{T}^n_m(\mathrm{I}_k \otimes \mathbf{DFT}_m)\,\mathrm{L}^n_k, \quad n = km \\
\mathbf{DFT}_n &\to P_n(\mathbf{DFT}_k \otimes \mathbf{DFT}_m)Q_n, \quad n = km,\ \mathrm{gcd}(k,m) = 1 \\
\mathbf{DFT}_p &\to R_p^T(\mathrm{I}_1 \oplus \mathbf{DFT}_{p-1})D_p(\mathrm{I}_1 \oplus \mathbf{DFT}_{p-1})R_p, \quad p \text{ prime} \\
\mathbf{DCT\text{-}3}_n &\to (\mathrm{I}_m \oplus \mathrm{J}_m)\,\mathrm{L}^n_m(\mathbf{DCT\text{-}3}_m(1/4) \oplus \mathbf{DCT\text{-}3}_m(3/4)) \\
&\quad \cdot(\mathrm{F}_2 \otimes \mathrm{I}_m)\begin{bmatrix}\mathrm{I}_m & 0 \oplus -\mathrm{J}_{m-1} \\ & \frac{1}{\sqrt{2}}(\mathrm{I}_1 \oplus 2\,\mathrm{I}_m)\end{bmatrix}, \quad n = 2m \\
\mathbf{DCT\text{-}4}_n &\to S_n\mathbf{DCT\text{-}2}_n\,\mathrm{diag}_{0 \le k < n}(1/(2\cos((2k+1)\pi/4n))) \\
\mathbf{IMDCT}_{2m} &\to (\mathrm{J}_m \oplus \mathrm{I}_m \oplus \mathrm{I}_m \oplus \mathrm{J}_m)\left(\left(\begin{bmatrix}1\\-1\end{bmatrix} \otimes \mathrm{I}_m\right) \oplus \left(\begin{bmatrix}-1\\-1\end{bmatrix} \otimes \mathrm{I}_m\right)\right)\mathrm{J}_{2m}\,\mathbf{DCT\text{-}4}_{2m} \\
\mathbf{WHT}_{2^k} &\to \prod_{i=1}^{t}(\mathrm{I}_{2^{k_1+\cdots+k_{i-1}}} \otimes \mathbf{WHT}_{2^{k_i}} \otimes \mathrm{I}_{2^{k_{i+1}+\cdots+k_t}}), \quad k = k_1 + \cdots + k_t \\
\mathbf{DFT}_2 &\to \mathrm{F}_2 \\
\mathbf{DCT\text{-}2}_2 &\to \mathrm{diag}(1, 1/\sqrt{2})\,\mathrm{F}_2 \\
\mathbf{DCT\text{-}4}_2 &\to \mathrm{J}_2\,\mathrm{R}_{13\pi/8}
\end{aligned}
$$

Software Defined Radio

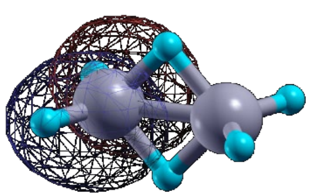


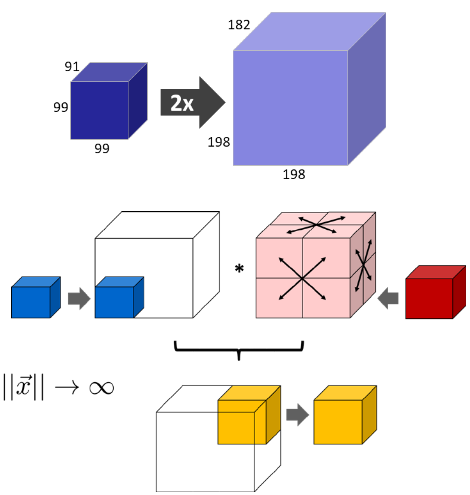
$$F_{K,F} \to \prod_{i=1}^{F}\left((I_{2^{K-2}} \otimes_j B_{F-i,j})L^{2^{K-1}}_{2^{K-2}}\right)$$

$$\underline{\mathbf{F}_{K,F}}\ \nu \to \prod_{i=1}^{F}\left(\left(\mathrm{I}_{2^{K-2}/\nu} \otimes_{j_1} \mathrm{L}^{2\nu}_{\nu}\vec{B}^{\nu}_{F-i,j_1}\right)\left(\mathrm{L}^{2^{K-1}/\nu}_{2^{K-2}/\nu} \bar{\otimes} \mathrm{I}_\nu\right)\right)$$

$$B_{i,j} : \begin{cases} \pi_U = \min_{d_U}(\pi_A + \beta_{A\to U}, \pi_B + \beta_{B\to U}) \\ \pi_V = \min_{d_V}(\pi_A + \beta_{A\to V}, \pi_B + \beta_{B\to V}) \end{cases}$$

PDEs/HPC Simulations



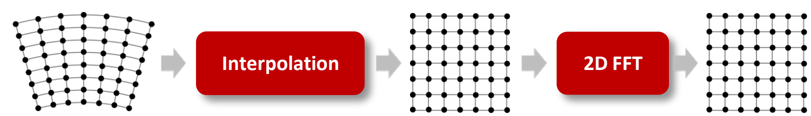


$$\Phi : \mathbb{R}^3 \to \mathbb{R}$$

$$\Phi(\vec{x}) = \frac{Q}{4\pi||\vec{x}||} + o\left(\frac{1}{||\vec{x}||}\right) \text{ as } ||\vec{x}|| \to \infty$$

$$Q = \int_D \rho d\vec{x}$$

Radar Image Formation (SAR, STAP)

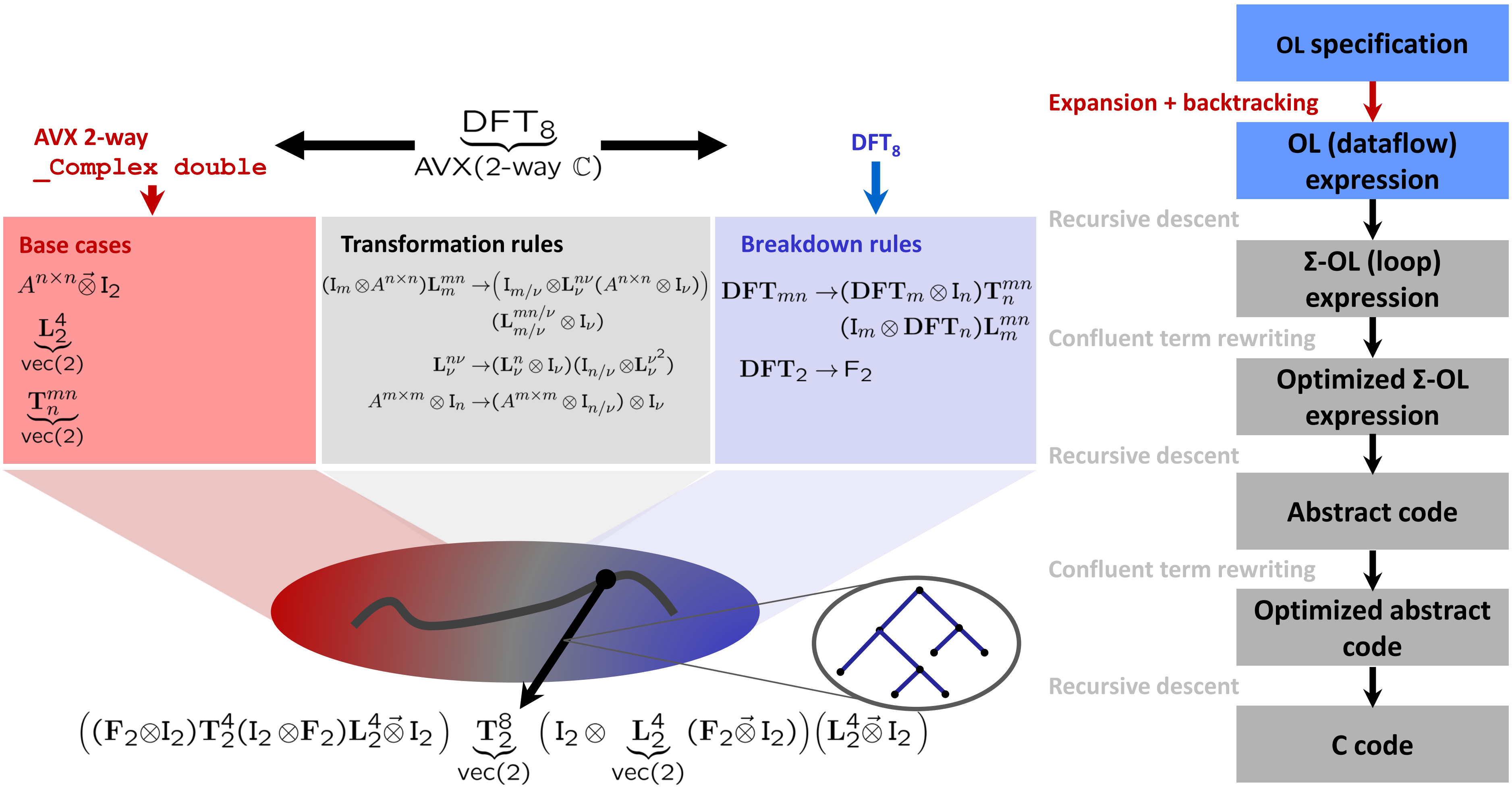


$$
\begin{aligned}
\mathrm{SAR}_{k\times m \to n\times n} &\to \mathrm{DFT}_{n\times n} \circ \mathrm{Interp}_{k\times m\to n\times n} \\
\mathrm{DFT}_{n\times n} &\to (\mathrm{DFT}_n \otimes \mathrm{I}_n) \circ (\mathrm{I}_n \otimes \mathrm{DFT}_n) \\
\mathrm{Interp}_{k\times m\to n\times n} &\to (\mathrm{Interp}_{k\to n} \otimes_i \mathrm{I}_n) \circ (\mathrm{I}_k \otimes_i \mathrm{Interp}_{m\to n}) \\
\mathrm{Interp}_{r\to s} &\to \left(\bigoplus_{i=0}^{n-2} \mathrm{InterpSeg}_k\right) \oplus \mathrm{InterpSegPruned}_{k,\ell} \\
\mathrm{InterpSeg}_k &\to \mathrm{G}_f^{u\cdot n\to k} \circ \mathrm{iPrunedDFT}_{n\to u\cdot n} \circ \left(\frac{1}{n}\right) \circ \mathrm{DFT}_n
\end{aligned}
$$

Formal Approach for all Types of Parallelism

- **Multithreading** (Multicore) — $\mathrm{I}_p \otimes_{\|} A_{\mu n}, \quad \mathrm{L}_m^{mn} \bar{\otimes} \mathrm{I}_\mu$
- **Vector SIMD** (SSE, VMX/Altivec,…) — $A \hat{\otimes} \mathrm{I}_\nu \quad \underbrace{\mathrm{L}_2^{2\nu}}_{\text{isa}}, \quad \underbrace{\mathrm{L}_\nu^{2\nu}}_{\text{isa}}, \quad \underbrace{\mathrm{L}_\nu^{\nu^2}}_{\text{isa}}$
- **Message Passing** (Clusters, MPP) — $\mathrm{I}_p \otimes_{\|} A_n, \quad \underbrace{\mathrm{L}_p^{p^2} \bar{\otimes} \mathrm{I}_{n/p^2}}_{\text{all-to-all}}$
- **Streaming/multibuffering** (Cell) — $\mathrm{I}_n \otimes_2 A_{\mu n}, \quad \mathrm{L}_m^{mn} \bar{\otimes} \mathrm{I}_\mu$
- **Graphics Processors** (GPUs) — $\prod_{i=0}^{n-1} A_i, \quad A_n \hat{\otimes} \mathrm{I}_w, \quad P_n \otimes Q_w$
- **Gate-level parallelism** (FPGA) — $\prod_{i=0}^{n-1}{}^{\text{ir}} A, \quad \mathrm{I}_s \tilde{\otimes} A, \quad \underbrace{\mathrm{L}_n^m}_{\text{bram}}$
- **HW/SW partitioning** (CPU + FPGA) — $\underbrace{A_1}_{\text{fpga}}, \quad \underbrace{A_2}_{\text{fpga}}, \quad \underbrace{A_3}_{\text{fpga}}, \quad \underbrace{A_4}_{\text{fpga}}$

Autotuning in Constraint Solution Space



Translating an SPL Expression Into Code

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

Output =

Ruletree, expanded into

SPL Expression:

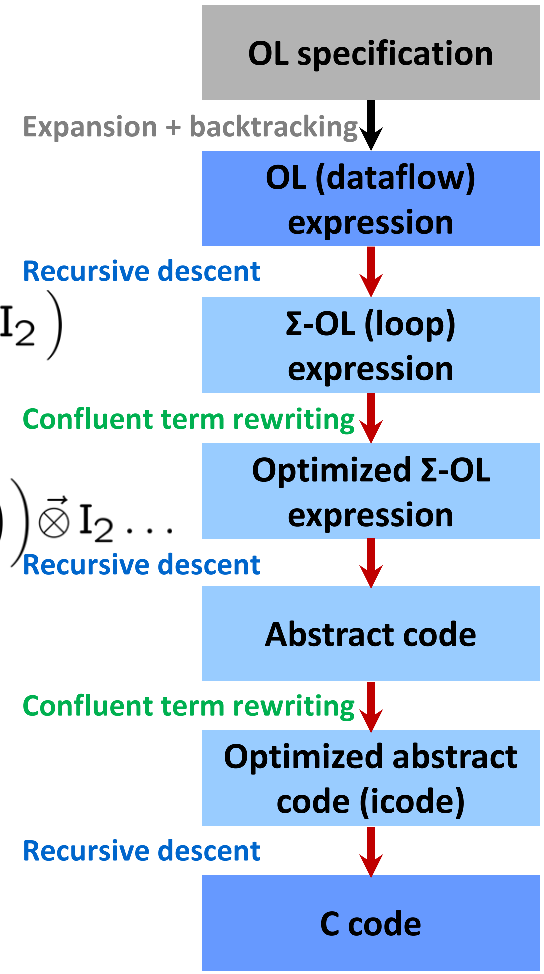
$$\Big((\mathrm{F}_2 \otimes \mathrm{I}_2)\mathrm{T}_2^4(\mathrm{I}_2 \otimes \mathrm{F}_2)\mathrm{L}_2^4 \vec{\otimes}\, \mathrm{I}_2\Big)\ \underbrace{\mathrm{T}_2^8}_{\text{vec}(2)}\ \Big(\mathrm{I}_2 \otimes \underbrace{\mathrm{L}_2^4}_{\text{vec}(2)}\ (\mathrm{F}_2 \vec{\otimes}\, \mathrm{I}_2)\Big)\Big(\mathrm{L}_2^4 \vec{\otimes}\, \mathrm{I}_2\Big)$$

Σ-SPL:

$$\Big(\sum_{j=0}^{1}\big(\mathrm{S}_{\imath_2 \otimes (j)_2}\mathrm{F}_2 \mathrm{Map}^2_{x \mapsto \omega_4^{2i+j} x}\mathrm{G}_{\imath_2 \otimes (j)_2}\big)\sum_{j=0}^{1}\big(\mathrm{S}_{(j)_2 \otimes \imath_2}\mathrm{F}_2 \mathrm{G}_{\imath_2 \otimes (j)_2}\big)\Big)\vec{\otimes}\, \mathrm{I}_2 \ldots$$

C Code:

```
void dft8(_Complex double *Y, _Complex double *X) {
    __m256d s38, s39, s40, s41,...
    __m256d  *a17, *a18;
    a17 = ((__m256d  *) X);
    s38 = *(a17);
    s39 = *((a17 + 2));
    t38 = _mm256_add_pd(s38, s39);
    t39 = _mm256_sub_pd(s38, s39);
    ...
    s52 = _mm256_sub_pd(s45, s50);
    *((a18 + 3)) = s52;
}
```



Some SPIRAL Generated C/AVX Code

```
int dwmonitor(float  *X, double  *D) {
    __m128d u1, u2, u3, u4, u5, u6, u7, u8 , x1, x10, x13, x14, x17, x18, x19, x2, x3, x4, x6, x7, x8, x9;
    int w1;
    unsigned _xm = _mm_getcsr();
    _mm_setcsr(_xm & 0xffff0000 | 0x0000dfc0);
    u5 = _mm_set1_pd(0.0);
    u2 = _mm_cvtps_pd(_mm_addsub_ps(_mm_set1_ps(FLT_MIN), _mm_set1_ps(X[0])));
    u1 = _mm_set_pd(1.0, (-1.0));
    for(int i5 = 0; i5 <= 2; i5++) {
        x6 = _mm_addsub_pd(_mm_set1_pd((DBL_MIN + DBL_MIN)), _mm_loaddup_pd(&(D[i5])));
        x1 = _mm_addsub_pd(_mm_set1_pd(0.0), u1);
        x2 = _mm_mul_pd(x1, x6);
        x3 = _mm_mul_pd(_mm_shuffle_pd(x1, x1, _MM_SHUFFLE2(0, 1)), x6);
        x4 = _mm_sub_pd(_mm_set1_pd(0.0), _mm_min_pd(x3, x2));
        u3 = _mm_add_pd(_mm_max_pd(_mm_shuffle_pd(x4, x4, _MM_SHUFFLE2(0, 1)), _mm_max_pd(x3, x2)), _mm_set1_pd(DBL_MIN));
        u5 = _mm_add_pd(u5, u3);
        x7 = _mm_addsub_pd(_mm_set1_pd(0.0), u1);
        x8 = _mm_mul_pd(x7, u2);
        x9 = _mm_mul_pd(_mm_shuffle_pd(x7, x7, _MM_SHUFFLE2(0, 1)), u2);
        x10 = _mm_sub_pd(_mm_set1_pd(0.0), _mm_min_pd(x9, x8));
        u1 = _mm_add_pd(_mm_max_pd(_mm_shuffle_pd(x10, x10, _MM_SHUFFLE2(0, 1)), _mm_max_pd(x9, x8)), _mm_set1_pd(DBL_MIN));
    }
    u6 = _mm_set1_pd(0.0);
    for(int i3 = 0; i3 <= 1; i3++) {
        u8 = _mm_cvtps_pd(_mm_addsub_ps(_mm_set1_ps(FLT_MIN), _mm_set1_ps(X[(i3 + 1)])));
        u7 = _mm_cvtps_pd(_mm_addsub_ps(_mm_set1_ps(FLT_MIN), _mm_set1_ps(X[(3 + i3)])));
        x14 = _mm_add_pd(u8, _mm_shuffle_pd(u7, u7, _MM_SHUFFLE2(0, 1)));
        x13 = _mm_shuffle_pd(x14, x14, _MM_SHUFFLE2(0, 1));
        u4 = _mm_shuffle_pd(_mm_min_pd(x14, x13), _mm_max_pd(x14, x13), _MM_SHUFFLE2(1, 0));
        u6 = _mm_shuffle_pd(_mm_min_pd(u6, u4), _mm_max_pd(u6, u4), _MM_SHUFFLE2(1, 0));
    }
    x17 = _mm_addsub_pd(_mm_set1_pd(0.0), u6);
    x18 = _mm_addsub_pd(_mm_set1_pd(0.0), u5);
    x19 = _mm_cmpge_pd(x17, _mm_shuffle_pd(x18, x18, _MM_SHUFFLE2(0, 1)));
    w1 = (_mm_testc_si128(_mm_castpd_si128(x19), _mm_set_epi32(0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff)) –
         (_mm_testnzc_si128(_mm_castpd_si128(x19), _mm_set_epi32(0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff))));
    __asm nop;
    if (_mm_getcsr() & 0x0d) {
        _mm_setcsr(_xm);
        return -1;
    }
    _mm_setcsr(_xm);
    return w1;
}
```

Dynamic Range: Single Node/Shared Memory

Embedded
Server GPU
Deep Learning GPU/TPU
FPGA SoC
ARM CPU DSP, SoC
Architecture

Dynamic range:

- <1W – 10 kW
- 1 to 500/41k cores (CPU/GPU)
- 500 MB – 6 TB RAM
- 1 Gflop/s – 200 Tflop/s
- 20 years of release dates

…one source code, one tool, always highest performance…

2000: SSE2 | 2007: SSSE3 | 2011: AVX | 2013: AVX2 | 2019: AVX512

Symbolic Verification

- **Transform = Matrix-vector multiplication**
  matrix fully defines the operation

$$\mathrm{DFT}_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix}$$

= ?

- **Algorithm = Formula**
  represents a matrix expression, can be evaluated to a matrix

$$(\mathrm{DFT}_2 \otimes \mathrm{I}_2)\,\mathrm{T}^4_2(\mathrm{I}_2 \otimes \mathrm{DFT}_2)\,\mathrm{L}^4_2 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & j \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Empirical Verification

- **Run program on all basis vectors, compare to columns of transform matrix**

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

= ?

```
DFT4([0,1,0,0])
```

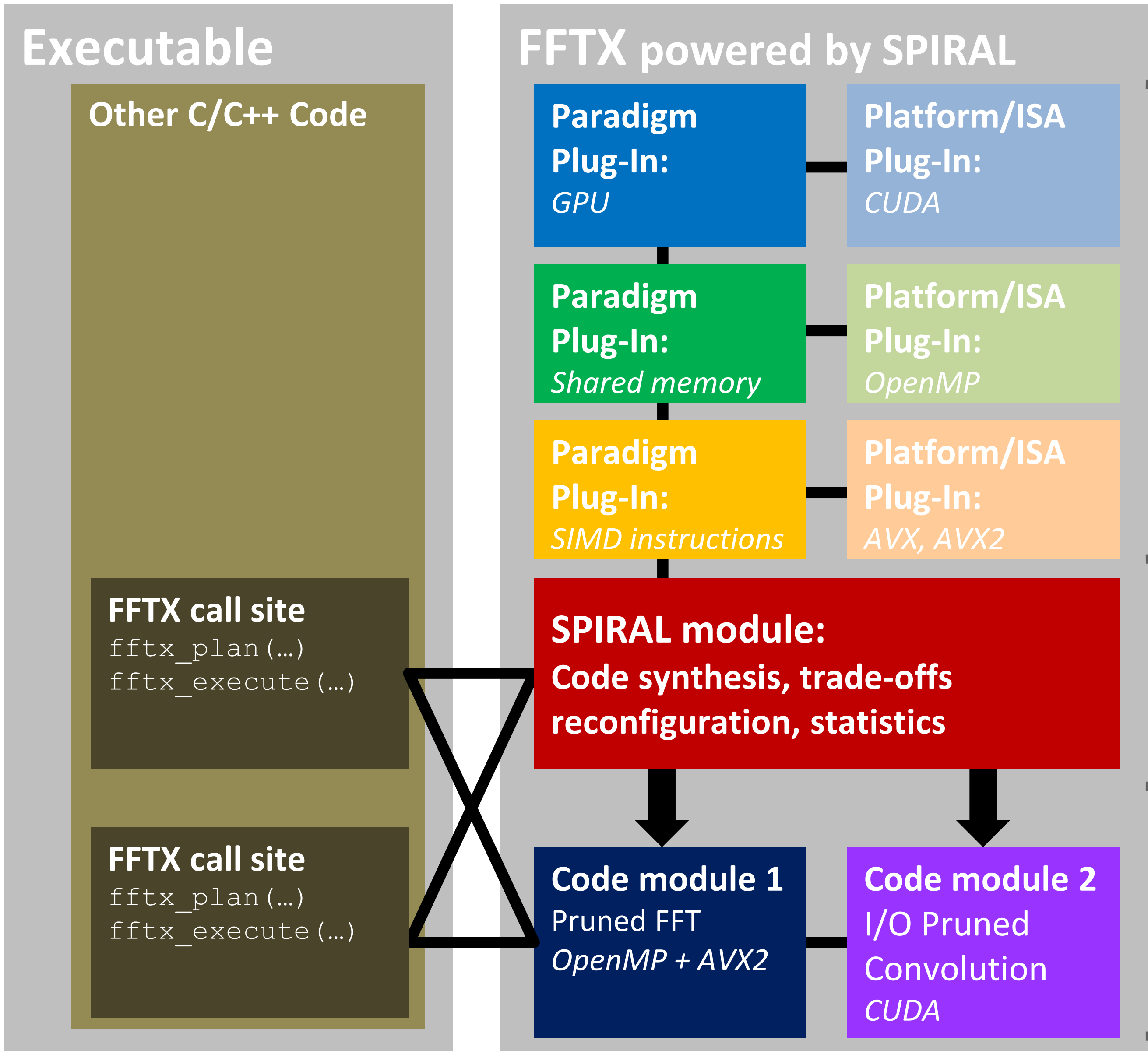
- **Compare program output on random vectors to output of a random implementation of same kernel**

```
DFT4([0.1,1.77,2.28,-55.3])
```

= ?

```
DFT4_rnd([0.1,1.77,2.28,-55.3]))
```

SPIRAL as Core Engine in Active Library



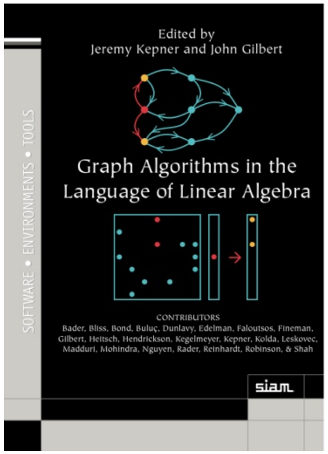
Extensible platform and programming model definitions

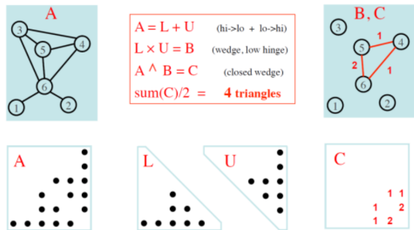
Core system: SPIRAL engine

Automatically generated tuned components and special cases

Current Research Directions

SPIRAL for Graphs

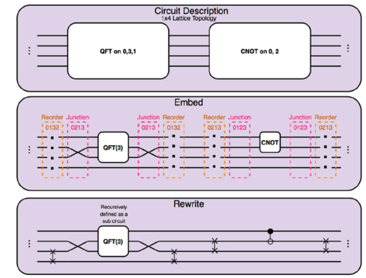


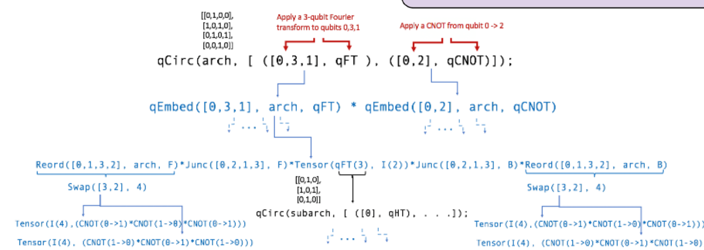


$$\Delta = \Delta + \frac{1}{2}\alpha_{10}A_{00}\alpha_{01}$$

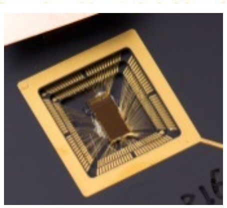
```
# OL Algorithm Specification
Accum(i4, 1, X.N-1,
   Accum_X(i6, [ i4, 0 ], i4,
      Dot([ i6, add(i4, V(1)) ],
         [ i4, add(i4, V(1)) ],
          sub(sub(X.N, i4), V(1)))
   )
)
```

Spiral for Quantum Computing

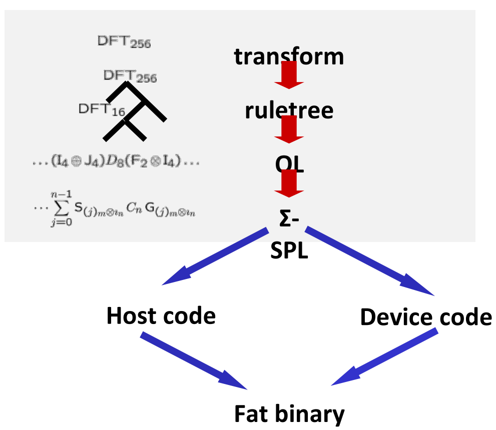




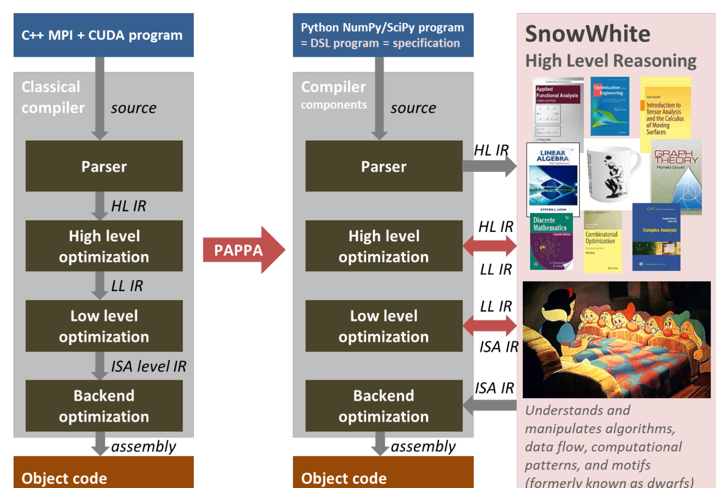
HW/SW Co-Design



RISC-V



SPIRAL as AI in Compilers



Examples — SPIRAL documentation

https://spiral-software.github.io/spiral-software/examples/index.html

SPIRAL

Search docs

- Introduction
- Getting Started
- Examples
  - Basic FFTs
  - GPU Examples (using CUDA)
  - Advanced Examples from FFTX
  - Advanced Examples from DARPA BRASS Demonstrating HCOL
- GAP3 Language
- SPIRAL Objects and Data Types
- SPIRAL Infrastructure
- Special Hardware
- Debugger
- Profiler
- Getting Help with SPIRAL
- Contributing to SPIRAL
- Further Reading

» Examples

View page source

Examples

Basic FFTs

- Generate a Simple FFT
- FFT with SIMD Instructions
- Threaded FFT with OpenMP
- Find a Good Rule Tree with Dynamic Programming
- 3D FFT with AVX Using Dynamic Programming Search

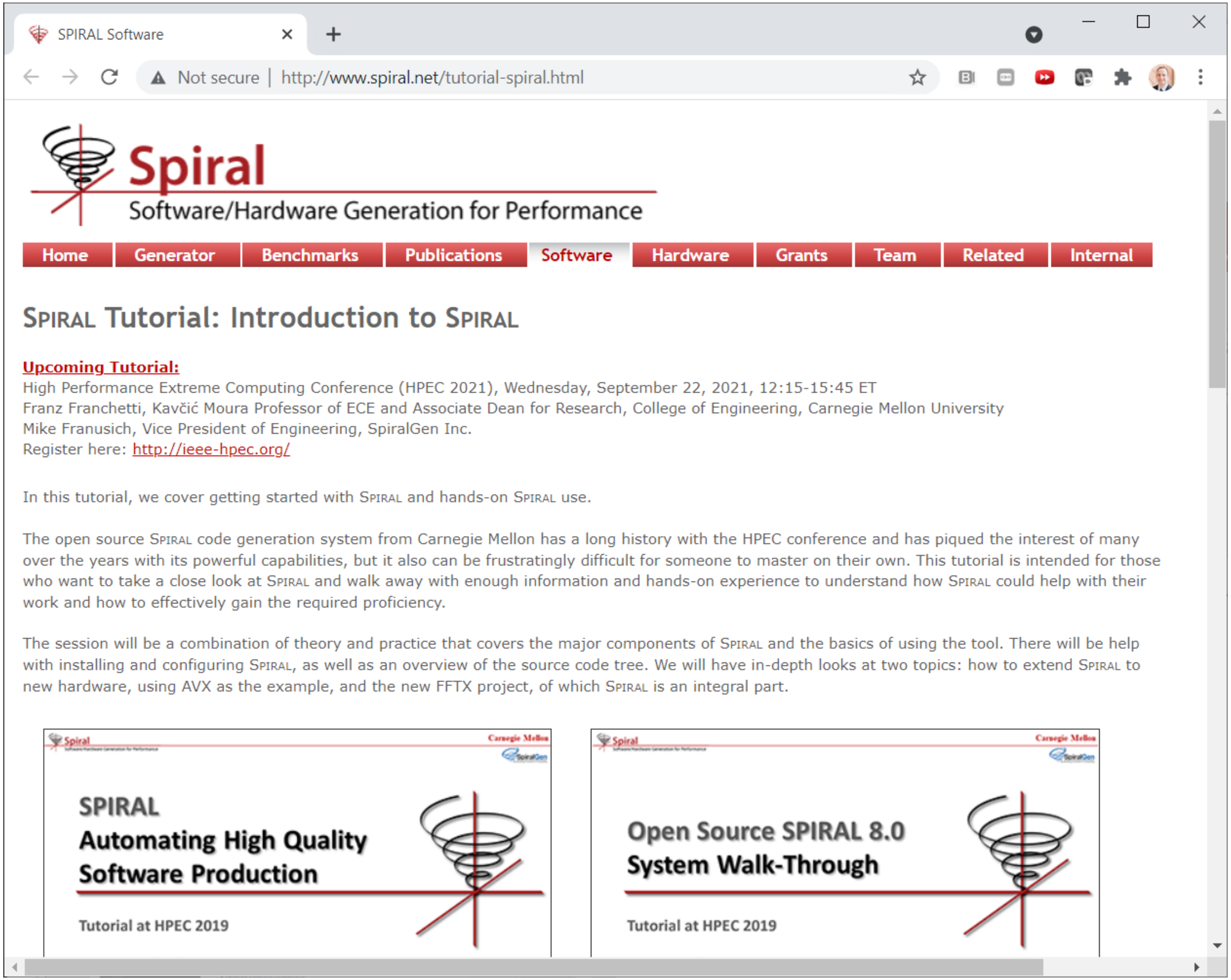
GPU Examples (using CUDA)

- Generate a 3D FFT for NVIDIA GPU
- Generate a batch of 3D FFTs
- Generate a Pruned Real 3D FFT
- Generate a Batch of 1d Real DFTs
- Real Convolution

Advanced Examples from FFTX

- Pruned 3D FFT from Planewave
- 3D Pruned Convolution from Hockney
- WarpX 1D

https://spiral-software.github.io/spiral-software/



http://www.spiral.net/tutorial-spiral.html

SPIRAL Software

Not secure | http://www.spiral.net/software.html

FFTE and SPIRAL-Generated Kernels

FFTE is a Fortran subroutine library for computing the FFT in one or more dimensions. It includes real, complex, mixed-radix, and parallel transforms. We proposed an implementation of the fast Fourier transform (FFT) targeting the ARM Scalable Vector Extension (SVE), and found that with the ARM compiler, SPIRAL-generated FFT kernels written in SVE intrinsic are up to 3.16 times faster than FFT kernels of FFTE written in Fortran and up to 5.62 times faster than SPIRAL-generated FFT kernels written in C. [From Takahashi and Franchetti, "FFTE on SVE: SPIRAL-Generated Kernels," Proceedings of HPC Asia 2020, Jan. 2020, pp. 114-122]
Go to the FFTE project page.

FFTX and SPIRAL

FFTX is the exascale follow-on to the FFTW open source discrete FFT package for executing the Fast Fourier Transform as well as higher-level operations composed of linear operations combined with DFT transforms. At the heart of FFTX is a build-time code generator, SPIRAL, that produces very high performance kernels targeted to their specific uses and platform environments.
Go to the FFTX project page.

HELIX: Formally Verified SPIRAL

HELIX is a formally verified language and rewriting engine for generating high-performance implementations for a variety of numerical algorithms. Based on the existing SPIRAL system, HELIX adds the rigor of formal verification of its correctness using the Coq proof assistant.
Go to the HELIX project page.

Quantum Circuit Optimization with SPIRAL

In the SPIRAL quantum package, we define the primitives and directives necessary to leverage SPIRAL's code generation framework for quantum circuit optimization. The input to our system is a circuit description written in terms of qubit transform objects. SPIRAL outputs both a simplified tensor expression representing the optimized circuit and QASM code that can be executed on a quantum computer.
Go to the Quantum Circuit Optimization page.

The SnowWhite System

Since the inception of compiler research the Holy Grail has been to devise a system that provides high level abstraction (programmers express their intent as concisely as in an algorithms textbook), and an automatic system that translates these programs or specifications into executables targeting an ever-evolving landscape of platforms, extracting close-to-optimal performance on all these platforms. The SnowWhite effort works towards addressing this problem by adding a novel AI approach to compiler analysis: It introduces high level reasoning to orchestrate the complex components and enables the systems to "understand" the computation much like human experts would do.
Go to the SnowWhite System page.

http://www.spiral.net/software.html

SPIRAL Publication

Not secure | http://spiral.ece.cmu.edu:8080/pub-spiral/submitfilter.jsp?order=year&order=type

4. Franz Franchetti, Daniele G. Spampinato, Anuva Kulkarni, Tze-Meng Low, M. Franusich, Thom Popovici, A. Canning, P. McCorquodale, B. Van Straalen and P. Colella
   FFT and Solvers for Exascale: FFTX and SpectralPACK
   Exascale Computing Project (ECP) Annual Meeting, 2019
5. Anuva Kulkarni, Daniele G. Spampinato and Franz Franchetti
   FFTX for Micromechanical Stress-Strain Analysis
   IEEE High Performance Extreme Computing Conference (HPEC), 2019
6. Yoko Franchetti, Thomas Nolin and Franz Franchetti
   Towards Precision Medicine: Simulation Based Parameter Estimation for Drug Metabolism
   SIAM Conference on Computational Science and Engineering (CSE), 2019

2018

Journal

1. Franz Franchetti, Tze-Meng Low, Thom Popovici, Richard Veras, Daniele G. Spampinato, Jeremy Johnson, Markus Püschel, James C. Hoe and José M. F. Moura
   SPIRAL: Extreme Performance Portability
   Proceedings of the IEEE, special issue on ``From High Level Specification to High Performance Code'', Vol. 106, No. 11, 2018

Conference (fully reviewed)

1. Anuva Kulkarni, Franz Franchetti and Jelena Kovacevic
   Algorithm Design for Large Scale Parallel FFT-Based Simulations on Heterogeneous Platforms
   Proc. High Performance Extreme Computing (HPEC), 2018
2. Vit Ruzicka and Franz Franchetti
   Fast and Accurate Object Detection in High Resolution 4K and 8K Video Using GPUs
   Proc. IEEE High Performance Extreme Computing (HPEC), 2018
3. Franz Franchetti, Daniele G. Spampinato, Anuva Kulkarni, Thom Popovici, Tze-Meng Low, M. Franusich, A. Canning, P. McCorquodale, B. Van Straalen and P. Colella
   FFTX and SpectralPack: A First Look
   Proc. IEEE International Conference on High Performance Computing, Data, and Analytics (HiPC), 2018
4. Vadim Zaliva and Franz Franchetti
   HELIX: A Case Study of a Formal Verification of High Performance Program Generation
   Proc. Workshop on Functional High Performance Computing (FHPC), 2018
5. Jiyuan Zhang, Franz Franchetti and Tze-Meng Low
   High Performance Zero-Memory Overhead Direct Convolutions
   Proc. International Conference on Machine Learning (ICML), 2018
6. Thom Popovici, Tze-Meng Low and Franz Franchetti

http://spiral.ece.cmu.edu:8080/pub-spiral

(Part of) The Team

James C. Hoe
Jeremy Johnson
Tze Meng Low
José M. F. Moura
David Padua
André Platzer
Markus Püschel
Manuela Veloso
Scott McMillan
Mike Franusich
Peter Milder
Phil Colella

Yevgen Voronenko
Srinivas Chellappa
Frédéric de Mesmay
Daniel S. McFarlin
Thom Popovici
Richard S. Veras
Daniele Spampinato
Vadim Zaliva
Sanil Rao
Paul Brouwer
Guanglin Xu

Volodymyr Arbatov
Brian Duff
Jason Larkin
Aliaksei Sandryhaila
Patrick Broderick
Christos Angelopoulos
Khalil Ghorbal
Stefan Mitsch
Brian Van Straalen
Peter McCorquodale
Andrew Canning
Gheorghe Almasi

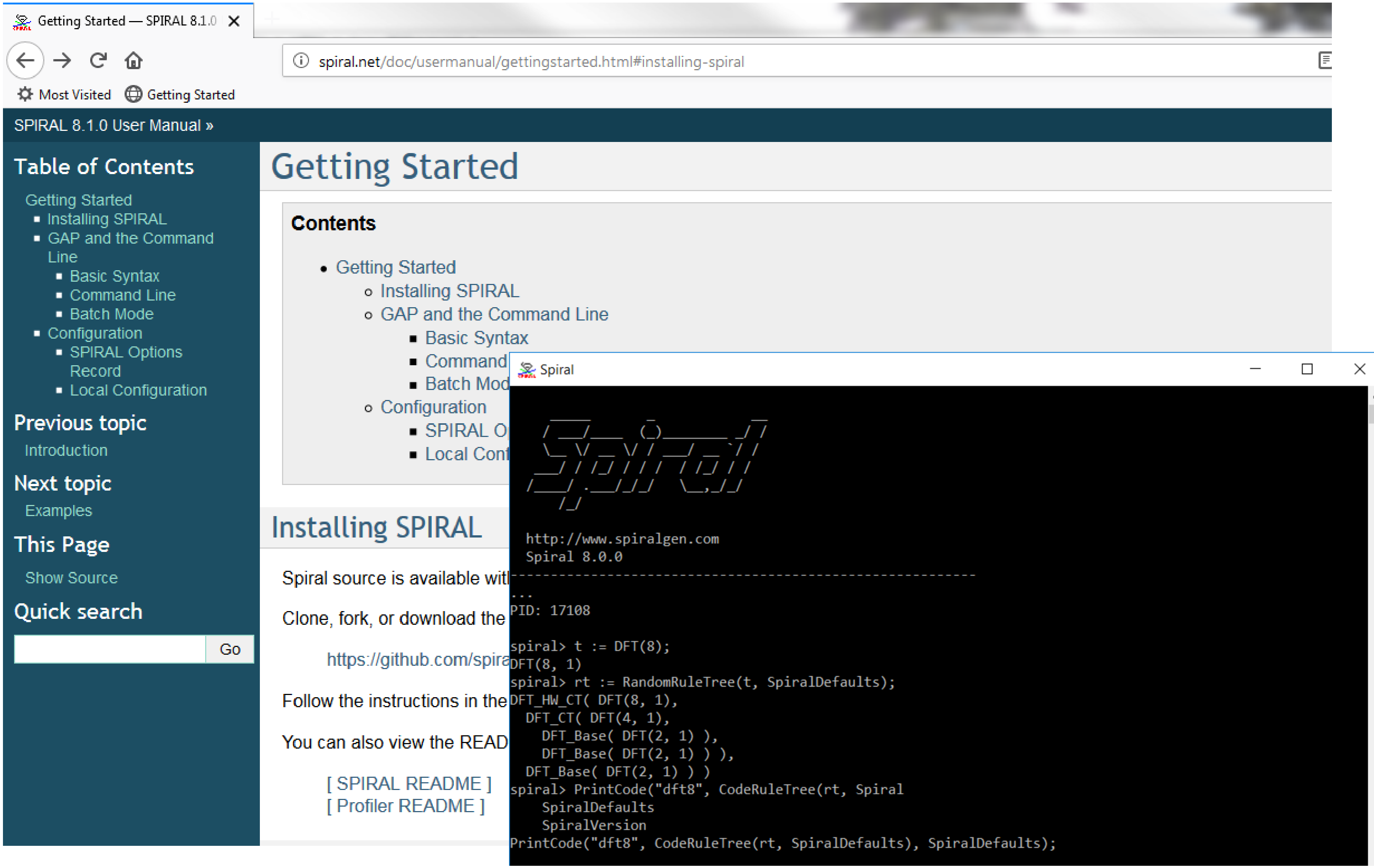
…and many more who have contributed to the broader SPIRAL *effort…*

This work was supported by DARPA, ONR, DOE, NSF, Intel, Mercury, and Nvidia

Organization

- Overview
- **System**
- Top level commands
- Abstractions
- Rewriting System I: RuleTree/backtracking search
- Rewriting System II: Visitor Patterns
- Rewriting System III: Associative/large context rules
- Basic block compiler

The Spiral System



http://www.spiral.net/doc/usermanual/

Hello, Universe!

FFT: Scalar C code

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

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);
PrintTo("SSE_OMP2_DFT32.c",
        PrintCode("SSE_OMP2_DFT32", icode, opts));
```

GAP/Spiral Packages and Name Spaces

Load tree: `init.g`

```
RequirePackage("arep");
Package(spiral);
Include(config);
Include(trace);
...
Load(spiral.rewrite);
Load(spiral.code);
ProtectNamespace(code);
Declare(CMeasure);
...
```

Packages and name spaces commands

```
avx.addsub_4x64f;
Import(avx);
addsub_4x64f;
avx.addsub_4x64f := false;
addsub_4x64f;

Dir(avx);
Info(addsub_4x64f);
Doc(DP);
```

GAP/Spiral Debugging

Stack-based debugger

```
Error("msg");
f := (n) -> When(n = 1, Error("at bottom"), n * f(n - 1));
f(10);
Top();
n;
Down();
n;
Down();
n;
Up();
n;
n + 1;
```

File I/O

```
Print("Hello World!");
LogTo("log.txt");                    # log to log.txt
PrintLine("Numbers: ", 1, ", ", 2);
PrintTo("file.txt", "a := ", 2);
AppendTo("file.txt", ";\n");
Read("file.txt");                    # read file into stdin
Exec("del file.txt");                # MS/DOS system call
a;                                   # file contents was executed
LogTo();                             # Turn off logging
```

Organization

- Overview
- System
- Top level commands
  Standard code generation
- Abstractions
- Rewriting System I: RuleTree/backtracking search
- Rewriting System II: Visitor Patterns
- Rewriting System III: Associative/large context rules
- Basic block compiler

End to End from DFT(8) -> C Code

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
```

Spiral

```
   _____       _            __
  / ___/____  (_)________ _/ /
  \__ \/ __ \/ / ___/ __ `/ /
 ___/ / /_/ / / /  / /_/ / /
/____/ .___/_/_/   \__,_/_/
    /_/

  http://www.spiralgen.com
  Spiral 8.0.0
----------------------------------------------------------
...
PID: 17108

spiral> t := DFT(8);
DFT(8, 1)
spiral> rt := RandomRuleTree(t, SpiralDefaults);
DFT_HW_CT( DFT(8, 1),
  DFT_CT( DFT(4, 1),
    DFT_Base( DFT(2, 1) ),
    DFT_Base( DFT(2, 1) ) ),
  DFT_Base( DFT(2, 1) ) )
spiral> PrintCode("dft8", CodeRuleTree(rt, Spiral
    SpiralDefaults
    SpiralVersion
PrintCode("dft8", CodeRuleTree(rt, SpiralDefaults), SpiralDefaults);

void dft8(double  *Y, double  *X) {
    double a49, a50, a51, a52, s13, s14, s15, s16
            , t149, t150, t151, t152, t153, t154, t155, t156
            , t157, t158, t159, t160, t161, t162, t163, t164
            , t165, t166, t167, t168, t169, t170, t171, t172
            , t173, t174, t175, t176;
    t149 = (*(X) + *((X + 8)));
    t150 = (*((X + 1)) + *((X + 9)));
    t151 = (*(X) - *((X + 8)));
    t152 = (*((X + 1)) - *((X + 9)));
    t153 = (*((X + 2)) + *((X + 10)));
    t154 = (*((X + 3)) + *((X + 11)));
    a49 = (0.70710678118654757*(*((X + 2)) - *((X + 10))));
    a50 = (0.70710678118654757*(*((X + 3)) - *((X + 11))));
    s13 = (a49 - a50);
    s14 = (a49 + a50);
```

Transforms, RuleTrees, and SPL Formulas

From Transform to SPL

```
# create the objects and lower them
t := DFT(4);
rt := RandomRuleTree(t, SpiralDefaults);
s := SPLRuleTree(rt);
```

Conversions Transform/SPL/GAP Matrix

```
t.terminate();                # transform -> SPL
tm := MatSPL(t);              # transform -> GAP matrix
sm := MatSPL(s);              # SPL formula -> GAP matrix
```

Symbolic Correctness

```
tm = sm;                      # same, as GAP matrices
InfinityNormMat(tm - sm);     # computes the matrix norm

t1 := DFT(13);                # for this we lose symbolic equivalency
rt1 := RandomRuleTree(t1, SpiralDefaults);
s1 := SPLRuleTree(rt1);       # size 13 triggers Rader
tm1 := MatSPL(t1);            # this is fully symbolic, but
sm1 := MatSPL(s1);            # Rader requires the FFT
tm1 = sm1;                    # thus floating-point matrix
InfinityNormMat(tm1 - sm1);   # but equivalent wrt. floating-point
```

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))));
```

Profiler

Top-level flow

```
opts := SpiralDefaults;
c := CodeRuleTree(RandomRuleTree(DFT(8), opts), opts);
PrintCode("dft8", c, opts);
CMeasure(c, opts);              # measure the runtime
CMatrix(c, opts);               # construct the transform matrix from c
```

Inspect Profiles

```
opts.profile;
default_profiles;
spd := GetEnv("SPIRAL_DIR");
Exec("dir ":: spd::"\\profiler\\targets");
Exec("dir ":: spd::"\\profiler\\targets\\win-x86-vcc");
Exec("type ":: spd::"\\profiler\\targets\\win-x86-vcc\\Makefile");
```

Look at the disk contents

```
# see in which drive we are. Usually C: or D:
Exec("cd");
# if no outdir is bound in opts this is the default temp path
IsBound(opts.outdir);
Exec("dir \\tmp\\"::StringInt(GetPid()));
Exec("type \\tmp\\"::StringInt(GetPid())::"\\testcode.c");
```

DPBench Infrastructure

DP Benchmark Object

```
Doc(DPBench);
```

Using DPBench

```
opts := SIMDGlobals.getOpts(AVX_4x64f);
t := TRC(DFT(512)).withTags(opts.tags);
dpbench := DPBench.build([t], opts, rec(), "DFT512", rec());

dpbench.runRandomAll();          # Random ruletree through DPBench
dpbench.runAll();                # now run search

dpbench.generateCode(dpbench.transforms, "DFT512");
Exec("dir");
Exec("type DFT512_TRC_DFT512.c");
```

Organization

- Overview
- System
- **Top level commands**
  Special hardware code generation
- Abstractions
- Rewriting System I: RuleTree/backtracking search
- Rewriting System II: Visitor Patterns
- Rewriting System III: Associative/large context rules
- Basic block compiler

Shared Memory Parallelization: OpenMP

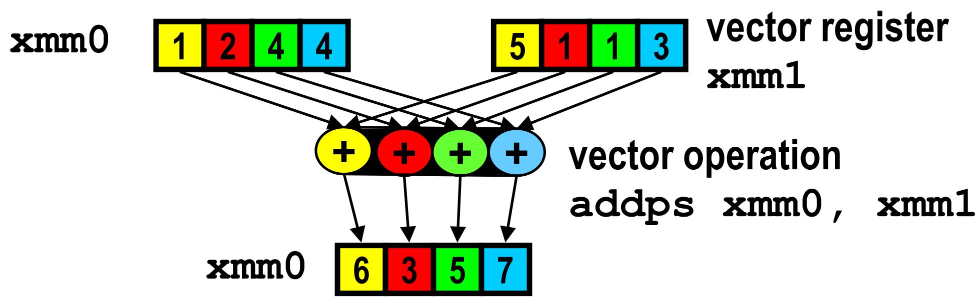
OpenMP
Enabling HPC since 1997

```
int main(int argc, char **argv)
{
    int a[100000];

    #pragma omp parallel for
    for (int i = 0; i < 100000; i++) {
        a[i] = 2 * i;
      printf("assigning i=%d\n");
    }

    return 0;
}
```

https://www.openmp.org/

SIMD (Signal Instruction Multiple Data) Vector Instructions in a Nutshell

- **What are these instructions?**
  - Extension of the ISA. Data types and instructions for parallel computation on short **(2-way–16-way) vectors** of integers and floats

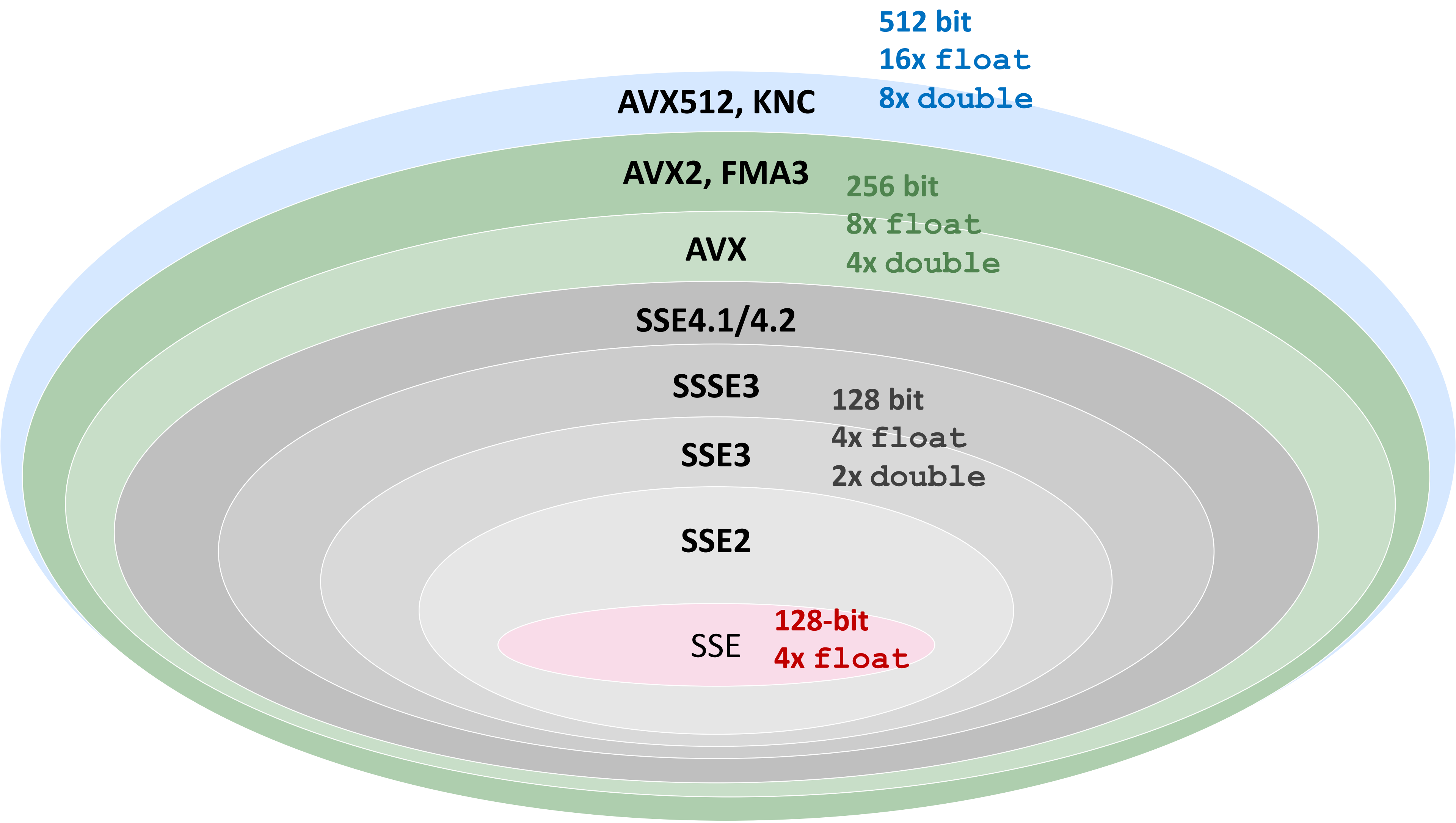


- **Intel MMX**
- **AMD 3DNow!**
- **Intel SSE**
- **AMD Enhanced 3DNow!**
- **Motorola AltiVec/VMX**
- **AMD 3DNow! Professional**
- **Intel SSE2**
- **IBM BlueGene/L PPC440FP2**
- **IBM QPX**
- **IBM VSX**
- **Intel SSE3**
- **Intel SSSE3**
- **Intel SSE4, 4.1, 4.2**
- **Intel AVX, AVX2**
- **Intel AVX512**

- **Problems:**
  - Not standardized
  - Compiler vectorization limited
  - Low-level issues (data alignment,…)
  - Reordering data kills runtime

One can easily slow down a program by vectorizing it

Complexity of Intel SSE/AVX



Intel Intrinsics Guide

_mm_search ?

Technologies

- MMX
- SSE
- SSE2
- SSE3
- SSSE3
- SSE4.1
- SSE4.2
- AVX
- AVX2
- FMA
- AVX-512
- KNC
- SVML
- Other

Categories

- Application-Targeted
- Arithmetic
- Bit Manipulation
- Cast
- Compare
- Convert
- Cryptography
- Elementary Math Functions
- General Support
- Load
- Logical
- Mask
- Miscellaneous
- Move
- OS-Targeted
- Probability/Statistics
- Random
- Set
- Shift
- Special Math Functions

```
__m512i _mm512_4dpwssd_epi32 (_m512i src, _m512i a0, _m512i a1, _m512i a2, _m512i a3, _m128i * b)    vp4dpwssd
__m512i _mm512_mask_4dpwssd_epi32 (_m512i src, _mmask16 k, _m512i a0, _m512i a1, _m512i a2, _m512i a3, _m128i * b)    vp4dpwssd
__m512i _mm512_maskz_4dpwssd_epi32 (_mmask16 k, _m512i src, _m512i a0, _m512i a1, _m512i a2, _m512i a3, _m128i * b)    vp4dpwssd
__m512i _mm512_4dpwssds_epi32 (_m512i src, _m512i a0, _m512i a1, _m512i a2, _m512i a3, _m128i * b)    vp4dpwssds
__m512i _mm512_mask_4dpwssds_epi32 (_m512i src, _mmask16 k, _m512i a0, _m512i a1, _m512i a2, _m512i a3, _m128i * b)    vp4dpwssds
__m512i _mm512_maskz_4dpwssds_epi32 (_m512i src, _mmask16 k, _m512i a0, _m512i a1, _m512i a2, _m512i a3, _m128i * b)    vp4dpwssds
__m512 _mm512_4fmadd_ps (_m512 a, _m512i b0, _m512i b1, _m512i b2, _m512i b3, _m128i * c)    v4fmaddps
__m512 _mm512_mask_4fmadd_ps (_m512 a, _mmask16 k, _m512i b0, _m512i b1, _m512i b2, _m512i b3, _m128i * c)    v4fmaddps
__m512 _mm512_maskz_4fmadd_ps (_m512 a, _mmask16 k, _m512i b0, _m512i b1, _m512i b2, _m512i b3, _m128i * c)    v4fmaddps
__m128 _mm_4fmadd_ss (__m128 a, __m128 b0, __m128 b1, __m128 b2, __m128 b3, __m128 * c)    v4fmaddss
__m128 _mm_mask_4fmadd_ss (__m128 a, __mmask8 k, __m128 b0, __m128 b1, __m128 b2, __m128 b3, __m128 * c)    v4fmaddss
__m128 _mm_maskz_4fmadd_ss (__m128 a, __mmask8 k, __m128 b0, __m128 b1, __m128 b2, __m128 b3, __m128 * c)    v4fmaddss
__m512 _mm512_4fnmadd_ps (_m512 a, _m512i b0, _m512i b1, _m512i b2, _m512i b3, _m128i * c)    v4fnmaddps
__m512 _mm512_mask_4fnmadd_ps (_m512 a, _mmask16 k, _m512i b0, _m512i b1, _m512i b2, _m512i b3, _m128i * c)    v4fnmaddps
__m512 _mm512_maskz_4fnmadd_ps (_m512 a, _mmask16 k, _m512i b0, _m512i b1, _m512i b2, _m512i b3, _m128i * c)    v4fnmaddps
__m128 _mm_4fnmadd_ss (__m128 a, __m128 b0, __m128 b1, __m128 b2, __m128 b3, __m128 * c)    v4fnmaddss
__m128 _mm_mask_4fnmadd_ss (__m128 a, __mmask8 k, __m128 b0, __m128 b1, __m128 b2, __m128 b3, __m128 * c)    v4fnmaddss
__m128 _mm_maskz_4fnmadd_ss (__m128 a, __mmask8 k, __m128 b0, __m128 b1, __m128 b2, __m128 b3, __m128 * c)    v4fnmaddss
__m128i _mm_abs_epi16 (__m128i a)    pabsw
__m128i _mm_mask_abs_epi16 (__m128i src, __mmask8 k, __m128i a)    vpabsw
__m128i _mm_maskz_abs_epi16 (__mmask8 k, __m128i a)    vpabsw
```

Targeting SIMD Vector Instructions

Simple Example: Intel AVX 4-way double precision

```
opts := SIMDGlobals.getOpts(AVX_4x64f); # default: real vectorization
t := TRC(DFT(16)).withTags(opts.tags);
rt := RandomRuleTree(t, opts);
c := CodeRuleTree(rt, opts);
PrintCode("AVX_DFT16", c, opts);
```

Stepwise code generation

```
opts.tags;                                  # what are the tags
opts.tags[1].v;                             # vector length
opts.tags[1].isa;                           # targeted ISA
opts.vector;                                # check out the options used
spl := SPLRuleTree(rt);                     # There are SIMD SPL objects
s := SumsRuleTree(rt, opts);                # and a SMP ISum
InfinityNormMat(MatSPL(s) - MatSPL(t));        # correctness check
```

Complex Vectorization Example

```
optsc := SIMDGlobals.getOpts(AVX_4x64f, # __m256d = (re,im,re,im)
    rec(realVect := false, cplxVect := true));
rtc := RandomRuleTree(t, optsc);            # complex vectorized DFT(16)
sc := SumsRuleTree(rtc, optsc);
cc := CodeRuleTree(rtc, optsc);             # far fewer shuffle operations
PrintCode("AVXcplx_DFT16", cc, optsc);
```

Spiral 8.2

```
----------------------------------------------------------
...
PID: 19620

spiral> opts := SIMDGlobals.getOpts(AVX_4x64f); # default: real vectorization
<Spiral SIMD options>
spiral> t := TRC(DFT(16)).withTags(opts.tags);
TRC(DFT(16, 1)).withTags([ AVecReg(AVX_4x64f) ])
spiral> rt := RandomRuleTree(t, opts);
TRC_vect( TRC(DFT(16, 1)).withTags([ AVecReg(AVX_4x64f) ]),
  @_Base( DPWrapper(DFT(16, 1).withTags([ AVecReg(AVX_4x64f) ]), VWrapTRC(AVX_4x64f)),
    DFT_tSPL_CT( DFT(16, 1).withTags([ AVecReg(AVX_4x64f) ]),
      TCompose_tag( TCompose([ TGrp(TCompose([ TTensorI(DFT(4, 1), 4, AVec, AVec), TTwiddle(16, 4, 1) ])), TGrp(TTensorI(DFT(4, 1), 4, APar, AVec)) ])
.withTags([ AVecReg(AVX_4x64f) ]),
        TGrp_tag( TGrp(TCompose([ TTensorI(DFT(4, 1), 4, AVec, AVec), TTwiddle(16, 4, 1) ])).withTags([ AVecReg(AVX_4x64f) ]),
          TCompose_tag( TCompose([ TTensorI(DFT(4, 1), 4, AVec, AVec), TTwiddle(16, 4, 1) ]).withTags([ AVecReg(AVX_4x64f) ]),
            AxI_vec( TTensorI(DFT(4, 1), 4, AVec, AVec).withTags([ AVecReg(AVX_4x64f) ]),
              @_Base( DPWrapper(DFT(4, 1), VWrap(AVX_4x64f)),
                DFT_CT( DFT(4, 1),
                  DFT_Base( DFT(2, 1) ),
                  DFT_Base( DFT(2, 1) ) ) ) ),
            TTwiddle_Tw1( TTwiddle(16, 4, 1).withTags([ AVecReg(AVX_4x64f) ]) ) ) ),
        TGrp_tag( TGrp(TTensorI(DFT(4, 1), 4, APar, AVec)).withTags([ AVecReg(AVX_4x64f) ]),
          IxA_L_vec( TTensorI(DFT(4, 1), 4, APar, AVec).withTags([ AVecReg(AVX_4x64f) ]),
            @_Base( DPWrapper(TL(16, 4, 1, 1).withTags([ AVecReg(AVX_4x64f) ]), VWrapId),
              IxLxI_kmn_n( TL(16, 4, 1, 1).withTags([ AVecReg(AVX_4x64f) ]),
                SIMD_ISA_Bases2( TL(8, 4, 1, 2).withTags([ AVecReg(AVX_4x64f) ]) ),
                SIMD_ISA_Bases1( TL(8, 4, 2, 1).withTags([ AVecReg(AVX_4x64f) ]) ) ) ),
            @_Base( DPWrapper(DFT(4, 1), VWrap(AVX_4x64f)),
              DFT_CT( DFT(4, 1),
                DFT_Base( DFT(2, 1) ),
                DFT_Base( DFT(2, 1) ) ) ) ) ) ) ) ) )
spiral> c := CodeRuleTree(rt, opts);;
spiral> PrintCode("AVX_DFT16", c, opts);

/*
 * This code was generated by Spiral 8.2.1a04, www.spiral.net
 */

#include <math.h>
#include <include/omega64.h>
#include <immintrin.h>

void init_AVX_DFT16() {
}

void AVX_DFT16(double  *Y, double  *X) {
    __m256d  *a45, *a46;
    __m256d s211, s212, s213, s214, s215, s216, s217, s218,
    s219, s220, s221, s222, s223, s224, s225, s226,
```

Generated AVX Code

```
void AVX_DFT16(double  *Y, double  *X) {
    __m256d  *a45, *a46;
    __m256d s211, s212, s213, s214, s215, s216, s217, s218,
        s219, s220, s221, s222, s223, s224, s225, s226,...,
        t155, t156, t157, t158, t159, t160;
    a45 = ((__m256d  *) X);
    s211 = *(a45);
    s212 = *((a45 + 1));
    s213 = _mm256_permute2f128_pd(s211, s212, (0) | ((2) << 4));
    s214 = _mm256_permute2f128_pd(s211, s212, (1) | ((3) << 4));
    s215 = _mm256_unpacklo_pd(s213, s214);
    s216 = _mm256_unpackhi_pd(s213, s214);
    s217 = *((a45 + 4));
    ...
    t147 = _mm256_sub_pd(s251, s259);
    t148 = _mm256_sub_pd(s255, s260);
    s261 = _mm256_sub_pd(_mm256_mul_pd(_mm256_set_pd(0.38268343236508978, 0.70710678118654757,
        0.92387953251128674, 1.0), s252), _mm256_mul_pd(_mm256_set_pd(0.92387953251128674, 0.70710678118654757,
        0.38268343236508978, 0.0), s256));
    s262 = _mm256_add_pd(_mm256_mul_pd(_mm256_set_pd(0.92387953251128674, 0.70710678118654757,
        0.38268343236508978, 0.0), s252), _mm256_mul_pd(_mm256_set_pd(0.38268343236508978, 0.70710678118654757,
        0.92387953251128674, 1.0), s256));
    s263 = _mm256_sub_pd(_mm256_mul_pd(_mm256_set_pd((-0.92387953251128674), (-0.70710678118654757),
        0.38268343236508978, 1.0), s254), _mm256_mul_pd(_mm256_set_pd((-0.38268343236508978), 0.70710678118654757,
        0.92387953251128674, 0.0), s258));
    s264 = _mm256_add_pd(_mm256_mul_pd(_mm256_set_pd((-0.38268343236508978), 0.70710678118654757,
        0.92387953251128674, 0.0), s254), _mm256_mul_pd(_mm256_set_pd((-0.92387953251128674),
        (-0.70710678118654757), 0.38268343236508978, 1.0), s258));
    t149 = _mm256_add_pd(s261, s263);
    t150 = _mm256_add_pd(s262, s264);
    ...
    s279 = _mm256_permute2f128_pd(s277, s278, (0) | ((2) << 4));
    *((a46 + 6)) = s279;
    s280 = _mm256_permute2f128_pd(s277, s278, (1) | ((3) << 4));
    *((a46 + 7)) = s280;
}
```

Vector Benchmarking Infrastructure

LocalConfig provides unit tests

```
LocalConfig.bench;
```

Create a test and run it

```
dpbench := LocalConfig.bench.AVX().4x64f.1d.dft_ic.medium();
dpbench.runAll();
```

Underlying infrastructure

```
# spiral-core\namespaces\spiral\platforms\avx\bench.gi
medium := _defaultSizes(s->doSimdDft(s, AVX_4x64f,
    rec(globalUnrolling := 128, tsplRader:=false,
    tsplBluestein:=false, tsplPFA:=false, oddSizes:=false,
    interleavedComplex := true, cplxVect := false, realVect := true)),
    List([4..16], i->2^i));

Print(doSimdDft);      # constructor from paradigms.vector
```

Combining SIMD Vector and OpenMP

IAGlobals = SIMDGlobals + SMPGlobals

```
opts := LocalConfig.getOpts(
    rec(cpu := LocalConfig.cpuinfo,   # some of the params and defaults
        useSIMD := true,
        useSMP := true,
        dataType := T_Real(64),
        globalUnrolling := 128),
    rec(numproc := LocalConfig.cpuinfo.cores,
        api := "OpenMP"),
    rec(svct:=true,
        splitL:=false,
        oddSizes := false,
        stdTTensor := true,
        tsplPFA := false));
opts.vector;
opts.smp;
t := TRC(DFT(1000)).withTags(opts.tags);
rt := RandomRuleTree(t, opts);
c := CodeRuleTree(rt, opts);
PrintCode("DFT1000", c, opts);
```

Organization

- Overview
- System
- Top level commands
- **Abstractions**

  icode
- Rewriting System I: RuleTree/backtracking search
- Rewriting System II: Visitor Patterns
- Rewriting System III: Associative/large context rules
- Basic block compiler

Spiral's Abstract Code Representation

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
- SPL operator over real numbers and machine numbers (floating-point)
- Pure functional interpretation
- Represents lambda expression

Spiral Abstract Code (icode)

```
program(
   chain(
      func(TVoid, "init", [  ],
         chain()
      ),
      func(TVoid, "transform", [ Y, X ],
         decl([ t57, t58, t59, t60, t61, t62, t63, t64 ],
            chain(
               assign(t57, add(deref(X), deref(add(X, V(4))))),
               assign(t58, add(deref(add(X, V(1))), deref(add(X, V(5))))),
               assign(t59, sub(deref(X), deref(add(X, V(4))))),
               assign(t60, sub(deref(add(X, V(1))), deref(add(X, V(5))))),
               assign(t61, add(deref(add(X, V(2))), deref(add(X, V(6))))),
               assign(t62, add(deref(add(X, V(3))), deref(add(X, V(7))))),
               assign(t63, sub(deref(add(X, V(2))), deref(add(X, V(6))))),
               assign(t64, sub(deref(add(X, V(3))), deref(add(X, V(7))))),
               assign(deref(Y), add(t57, t61)),
               assign(deref(add(Y, V(1))), add(t58, t62)),
               assign(deref(add(Y, V(4))), sub(t57, t61)),
               assign(deref(add(Y, V(5))), sub(t58, t62)),
               assign(deref(add(Y, V(2))), sub(t59, t64)),
               assign(deref(add(Y, V(3))), add(t60, t63)),
               assign(deref(add(Y, V(6))), add(t59, t64)),
               assign(deref(add(Y, V(7))), sub(t60, t63))
            )
         )
      )
   )
)
```

Spiral icode

Basics

```
c1 := skip();                           # NOP
a := var.fresh_t("j", TInt);            # create a "fresh" variable
c2 := assign(a, V(0));                  # assignment
c3 := assign(a, fcall("foo"));          # call to foo()
c4 := chain(c1, c2, c3);                # basic block
i := Ind(4);                            # loop index
c5 := loop(i, 4, c4);                   # loop
c6 := decl([a], c5);                    # declare a variable

PrintCode("", c6, SpiralDefaults);      # pretty print as C code
```

Internal fields

```
a.id; a.t;
c2.exp; c2.loc;
c3.cmds;
c4.var; c4.range; c4.cmd;
c5.vars; c5.cmd;
```

Spiral Variables and Expressions

Variables

```
i := Ind();                              # integer index
j := Ind(4);                             # index, 0 <= j < 4
k := var.fresh_t("j", TInt);             # create a "fresh" variable
var.table.(v.id);                        # global variable table
```

Expressions

```
a := var.fresh_t("a", TReal);            # a few real variables
b := var.fresh_t("b", TReal);
c := var.fresh_t("c", TReal);
a + b;                                   # + is overloaded
add(a, V(2.0));
a + 2;
e := add(a, b);                          # use add function
e.args;                                  # the operands
e.t;                                     # expressions carry a type
g := add(a, b, c);                       # not just binary
h := add(a, mul(b, c));                  # expressions
k := add(neg(a), mul(b, c, V(1.1)));     # expressions
mul(V(1.0), V(2.0));                     # evaluates at construction
sub(V(1), V(2.0)).t;                     # type unification
```

Special Hardware: Tagged icode Objects

Parallel Loop == smp_for

```
# spiral-core\namespaces\spiral\paradigms\smp\code.gi
Class(smp_loop, loop_base, rec(
   __call__ := meth(self, nthreads, tidvar, tidexp,
                    loopvar, range, cmd)
        local result;
        range := toRange(range);
        loopvar.setRange(range);
        loopvar.isLoopIndex := true;
        return WithBases(self, rec(
            operations := CmdOps,
            nthreads := nthreads,
            cmd := cmd,
            var := loopvar,
            tidvar := Checked(IsLoc(tidvar), tidvar),
            tidexp := toExpArg(tidexp),
            range := range));
    end,
    print := (self, i, is) >> Print(self.name,"(",self.nthreads,", ",
       self.tidvar, ", ", self.tidexp, ", ", self.var, ", ",
       self.range, ",\n", Blanks(i+is),
       self.cmd.print(i+is, is),
       Print("\n", Blanks(i), ")")),
));
```

Vector Instructions as Matrices

Intel C++ Compiler Manual

```
__m128 _mm_unpackhi_ps(__m128 a, __m128 b)
r0 := a2; r1 := b2; r2 := a3; r3 := b3
```

Instruction specification (GAP code)

```
Intel_SSE2.4_x_float._mm_unpackhi_ps := rec(
   v := 4,
   semantics := (a, b, p) -> [a[2], b[2], a[3], b[3]],
   parameters := []
);
```

SSE instruction as matrix

```
__m128 t, x0, x1;
t = _mm_unpackhi_ps(x0, x1);
```

➡

$$\vec{t} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \vec{x}_0 \\ \vec{x}_1 \end{bmatrix}$$

Automatically build matrix from `semantics()` function

Vector Instructions icode

Every ISA defines ISA specific instructions and polymorphic add,…

```
# spiral-core\namespaces\spiral\platforms\avx\code.gi
# __m256d _mm256_insertf128_pd(__m256d a, __m128d b, int offset);
Class(vinsert_2l_4x64f, VecExp_4.binary(), rec(
    ev := self >> let(
        a := _unwrap(self.args[1].ev()),
        b := _unwrap(self.args[2].ev()),
        When( self.args[3].p[1] = 0,
            b :: a{[3 .. 4]}, a{[1 .. 2]} :: b )),
    computeType := self >> self.args[1].t,
));

# __m256d _mm256_unpackhi_pd(__m256d a, __m256d b);
Class(vunpackhi_4x64f, VecExp_4.binary(), rec(
    semantic := (in1, in2, p) -> [in1[2], in2[2], in1[4], in2[4]],
    ev := _evpack
));

#__m256 _mm256_blend_ps(__m256 m1, __m256 m2, const int mask);
Class(vblend_8x32f,  VecExp_8.binary(), rec(
    semantic := (in1, in2, p) ->
    List( Zip2(TransposedMat([in1, in2]), p), e -> e[1][e[2]]),
    params := self >> Replicate(8, [1,2]), ev := _evshuf2
));
```

Vector ISA Definition

ISA Definition file ties everyting together

```
# spiral-core\namespaces\spiral\platforms\avx\isa.gi
Class(AVX_4x64f, AVX_Intel, rec(
    includes     := () -> ["<include/omega64.h>"] :: _AVXINTRIN(),
    v        := 4,
    t        := TVect(T_Real(64), 4),
    ctype    := "double",
    instr    := [ vunpacklo_4x64f, vunpackhi_4x64f, vshuffle_4x64f,
        vperm2_4x64f, vpermf128_4x64f, vperm_4x64f, vblend_4x64f ],
    mul_cx := (self, opts) >>
        ((y, x, c) -> let( u1 := self.freshU(), u2 := self.freshU(),
            u3 := self.freshU(),
            decl([u1, u2, u3], chain(
                assign(u1, mul(x, vunpacklo_4x64f(c, c))),
                assign(u2, vshuffle_4x64f(x, x, [2,1,2,1])),
                assign(u3, mul(u2, vunpackhi_4x64f(c, c))),
                assign(y, addsub_4x64f(u1, u3)))))),
    svload_init := (vt) -> [
        [ y,x,opts) -> let(u1 := var.fresh_t("U", TVect(vt.t, 2)),
            decl([u1], chain(
                assign(u1, vload1sd_2x64f(x[1].toPtr(vt.t))),
                assign(y, vinsert_2l_4x64f(vt.zero(), u1, [0]))))),
    ...
));
```

Organization

- Overview
- System
- Top level commands
- **Abstractions**

  Symbolic functions
- Rewriting System I: RuleTree/backtracking search
- Rewriting System II: Visitor Patterns
- Rewriting System III: Associative/large context rules
- Basic block compiler

Lambda Functions $f : D \to R;\ i \mapsto f(i)$

Definition

```
i := Ind(4);                            # variable with range
f := Lambda(i, i+1);                    # i -> i+1
j := Ind(4);                            # variable with range
g := Lambda([i, j], imod(i*j, 4));      # function in 2 variables
```

Operations on functions

```
f.at(0);                                # evaluate function
f.tolist();                             # create table for function
k := Ind(4);
Lambda(k, g.at(k, V(1)));               # partially evaluate g(.,1)
m := Ind(4);
h := Lambda(m, 2*m);                    # i -> 2*i
LambdaCompose(f, h);                    # function composition
```

Function properties/fields

```
f.domain();
f.range();
f.t;
f.vars;
f.expr;
```

Symbolic Index Mapping Functions

Symbolic functions: definitions

$$f : \mathbb{I}_n \to \mathbb{I}_N;\ i \mapsto f(i) \qquad \mathbb{I}_n = \{0\ldots, n-1\}$$

$$f_j : \mathbb{I}_n \to \mathbb{I}_N;\ i \mapsto f_j(i)$$

$$\imath_n : \mathbb{I}_n \to \mathbb{I}_n;\ i \mapsto i$$

$$(j)_n : \mathbb{I}_1 \to \mathbb{I}_n;\ i \mapsto j$$

$$(k)_+^{n\to N} : \mathbb{I}_n \to \mathbb{I}_N;\ i \mapsto i + k$$

$$\ell_k^{km}(i) = \left\lfloor \frac{i}{m} \right\rfloor + k(i \mod m)$$

$$v_{k,m}(i) = \left(m \left\lfloor \frac{i}{m} \right\rfloor + k(i \mod m)\right) \mod km$$

$$w_{\phi,g}^p(i) = \begin{cases} 0, & i = 0, \\ \phi g^i \mod p, & \text{else.} \end{cases}$$

Algebra of symbolic functions

$$g \circ f : \mathbb{I}_m \to \mathbb{I}_N;\ i \mapsto g(f(i))$$

$$f \otimes g : \mathbb{I}_{mn} \to \mathbb{I}_{MN};\ i \mapsto Nf\left(\left\lfloor \tfrac{i}{n} \right\rfloor\right) + g(i \mod n)$$

$$\imath_n \otimes (j)_m : \mathbb{I}_n \to \mathbb{I}_{mn};\ i \mapsto im + j$$

$$(j)_m \otimes \imath_n : \mathbb{I}_n \to \mathbb{I}_{mn};\ i \mapsto i + jn$$

$$f : \mathbb{I}_m \to \mathbb{I}_M;\ i \mapsto f(i)$$

$$g : \mathbb{I}_n \to \mathbb{I}_N;\ i \mapsto g(i)$$

Index Mapping Functions

Definition

```
f := fId(4);                 # I4->I4: i->i
j := Ind(4);                 # variable with range
g := fBase(j);               # I1->I4: i->j
h := fAdd(4,2,1);            # I2->I4: i->i+1
u := L(16, 4);               # permutation i -> \ell(16,4)(i)
```

Operations on functions

```
f.at(0);                     # evaluate function
f.tolist();                  # create table for function
f.lambda();                  # convert to Lambda function
r := fTensor(f, g);          # tensor product of functions
s := fCompose(r, u);         # function composition
```

Function properties/fields

```
f.domain();
f.range();
```

Diagonal Functions $f^{n \to \mathbb{C}} : \mathbb{I}_n \to \mathbb{C}$

Definition

```
f := fConst(4, 1.1);                  # I4->R: i->1.1
g := dOmega(8, 2);                    # f_N,k : N -> C : i -> omega(N, k*i)
h := FList(TReal, [1.1, 1.2, 1.4, 1.4]);       # table lookup
u := FData([V(1.1), V(1.2), V(1.3), V(1.4)]); # table lookup w/var
```

Operations on functions

```
g.at(3);                              # evaluate function
g.at(3).ev();                         # simplify the result
f.tolist();                           # create table for function
g.lambda();                           # convert to Lambda function
r := diagTensor(f, u);                # tensor product of functions
r.tolist();                           # what does diagTensor do?
s := fCompose(r, L(16,4));            # composition of permutation and
s.tolist();                           #   tensor product of functions
```

Function properties/fields

```
f.domain();
f.range();
u.var;
u.var.t;
u.var.value;
```

XChains: Symbolic Functions for GT

XChain Definition

```
# spiral-core\namespaces\spiral\spl\gtfuncs.gi
Class(XChain, GTIndexFunction, rec(
    def := perm -> Checked(IsList(perm), ForAll(perm, IsPosInt0),
        Set(Copy(perm))=[0..Length(perm)-1], rec()),
    range := self >> 0,
    domain := self >> 0,
    equals := (self, o) >> ObjId(self)=ObjId(o) and
        self.params[1]=o.params[1],
    toSpl := (self, inds, kernel_size) >> let(fbases := List(inds,
        fBase), ApplyFunc(fTensor, List(self.params[1],
        i -> When(i=0, fId(kernel_size), fbases[i])))),
    ...
));
```

Convert GT and XChain to SPL and Symbolic Functions

```
x := XChain([0,2,1]);
x.toSpl([Ind(8), Ind(4)], 4);
gt1 := GT(DFT(2), XChain([ 0, 1 ]), XChain([ 0, 1 ]), [ 4 ]);
gt2 := GT(DFT(2), XChain([ 1, 0 ]), XChain([ 1, 0 ]), [ 4 ]);
gt3 := GT(DFT(2), XChain([ 0, 1 ]), XChain([ 1, 0 ]), [ 4 ]);
gt4 := GT(DFT(2), XChain([ 1, 0 ]), XChain([ 0, 1 ]), [ 4 ]);
gt5 := GT(DFT(2), XChain([ 2, 1, 0 ]), XChain([ 0, 2, 1 ]), [ 2, 4 ]);
```

Organization

- Overview
- System
- Top level commands
- **Abstractions**

  Nonterminals, tags, and tSPL
- Rewriting System I: RuleTree/backtracking search
- Rewriting System II: Visitor Patterns
- Rewriting System III: Associative/large context rules
- Basic block compiler

Nonterminals: Abstract Linear Transforms

- **Mathematically: Matrix-vector multiplication**

$$x \mapsto y = T \cdot x$$

input vector (signal) → x

output vector (signal) → y

transform = matrix → T

- **Example: Discrete Fourier transform (DFT)**

$$\mathrm{DFT}_n = [e^{-2k\ell\pi i/n}]_{0 \le k,\ell < n}$$

Point free: we drop the input/output vectors and only represent the transform

Examples: Transforms

$$\mathbf{DCT\text{-}2}_n = \left[\cos(k(2\ell+1)\pi/2n)\right]_{0\le k,\ell<n},$$

$$\mathbf{DCT\text{-}3}_n = \mathbf{DCT\text{-}2}_n^T \quad \text{(transpose)},$$

$$\mathbf{DCT\text{-}4}_n = \left[\cos((2k+1)(2\ell+1)\pi/4n)\right]_{0\le k,\ell<n},$$

$$\mathbf{IMDCT}_n = \left[\cos((2k+1)(2\ell+1+n)\pi/4n)\right]_{0\le k<2n, 0\le \ell<n},$$

$$\mathbf{RDFT}_n = [r_{k\ell}]_{0\le k,\ell<n}, \quad r_{k\ell} = \begin{cases} \cos\frac{2\pi k\ell}{n}, & k \le \lfloor \frac{n}{2} \rfloor \\ -\sin\frac{2\pi k\ell}{n}, & k > \lfloor \frac{n}{2} \rfloor \end{cases},$$

$$\mathbf{WHT}_n = \begin{bmatrix} \mathbf{WHT}_{n/2} & \mathbf{WHT}_{n/2} \\ \mathbf{WHT}_{n/2} & -\mathbf{WHT}_{n/2} \end{bmatrix}, \quad \mathbf{WHT}_2 = \mathbf{DFT}_2,$$

$$\mathbf{DHT} = \left[\cos(2k\ell\pi/n) + \sin(2k\ell\pi/n)\right]_{0\le k,\ell<n}.$$

Spiral Non-Terminals (Transforms)

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
```

Tags Encode Hardware for Nonterminals

- **Identify crucial hardware parameters**
  - Number of processors: *p*
  - Cache line size: *μ*

- **Introduce them as tags in SPL:**

$$\underbrace{A}_{\mathrm{smp}(p,\mu)}$$

This means: **formula *A* is to be optimized for *p* processors and cache line size *μ***

- **Tags express hardware constraints within the ruletree system**

Example: SMP Tag

Definition

```
# spiral-core\namespaces\spiral\paradigms\smp\sigmaspl.gi
Class(AParSMP, AGenericTag, rec(
    isSMP := true,
    updateParams := meth(self)
        if Length(self.params)=1 then self.params :=
            [self.params[1], threadId()];
        elif Length(self.params)=2 then ;
        else Error("Usage: AParSMP(<num_threads>, [<tid>])");
        fi;
    end
));
```

Use case

```
Import(paradigms.smp);                # not imported by default
tag := AParSMP(2);                    # define a SMP/OpenMP tag
t := TRC(DFT(4)).withTags([tag]);     # tag a transform
pm(t);                                # tagged transforms need to
                                      # be in real arithmetic
MatSPL(DFT(2));                       # TRC(.) lifts RC(.) to the
MatSPL(TRC(DFT(2)));                  # transform level
```

SIMD Vector Tag

Real Vectorization

```
# spiral-core\namespaces\spiral\paradigms\vector\tags.gi
Class(AVecReg, AGenericTag, rec(
    isReg := true, isRegCx := false, isVec := true,
    updateParams := meth(self)
        Checked(IsSIMD_ISA(self.params[1]));
        Checked(Length(self.params)=1);
        self.v := self.params[1].v; self.isa := self.params[1];
    end,
    container := (self, spl) >>
        paradigms.vector.sigmaspl.VContainer(spl, self.isa)
));
```

Complex Vectorization

```
Class(AVecRegCx, AVecReg, rec(
    updateParams := meth(self)
        Checked(IsSIMD_ISA(self.params[1]));
        Checked(Length(self.params)=1);
        self.v := self.params[1].v/2; self.isa := self.params[1];
    end,
    container := (self, spl) >>
        paradigms.vector.sigmaspl.VContainer(spl, self.isa.cplx()),
    isRegCx := true
));
```

Tagged SPL Non-Terminal Expressions

Lift RC to Transform Level

```
# spiral-core\namespaces\spiral\paradigms\common\nonterms.gi
Class(TRC, Tagged_tSPL_Container, rec(
    abbrevs :=  [ (A) -> Checked(IsNonTerminal(A) or IsSPL(A), [A]) ],
    dims := self >> 2*self.params[1].dims(),
    terminate := self >> Mat(MatSPL(RC(self.params[1]))),
    transpose := self >> ObjId(self)(
        self.params[1].conjTranspose()).withTags(self.getTags()),
    conjTranspose := self >> self.transpose(),
   ...
));
```

Lift Compose to Transform Level

```
Class(TCompose, Tagged_tSPL_Container, rec(
    abbrevs :=  [ (l) -> Checked(IsList(l), [l]) ],
    dims := self >> [self.params[1][1].dims()[1],
        self.params[1][Length(self.params[1])].dims()[2]],
    terminate := self >> Compose(
        List(self.params[1], i->i.terminate())),
    transpose := self >> TCompose(Reversed(List(self.params[1],
        i->i.transpose()))).withTags(self.getTags()),
    ...
));
```

Generalized Tensors: GT

GT Non-Terminal

```
# spiral-core\namespaces\spiral\paradigms\common\gt.gi
Class(GT, GTBase, Tagged_tSPL, rec(
    abbrevs := [
    (spl, gath, scat, v) -> Checked(IsSPL(spl),
        IsIndexMapping(gath), IsIndexMapping(scat), IsList(v),
        ForAll(v, IsPosInt0Sym), [spl, gath, scat, v] ) ],
    ...
    _scat := Scat,
    _gath := Gath,
    toSpl := self >> self.toSplCx([]),
    toSplCx := (self, outer_inds) >> let(p := self.params,
        spl := Copy(p[1]), g := Copy(p[2]), s := Copy(p[3]),
        dims := p[4], inds := List(dims, Ind),
        allinds := Concatenation(inds, outer_inds),
        kernel := self._scat(s.toSpl(allinds, Rows(spl))) * spl *
            self._gath(g.toSpl(allinds, Cols(spl))),
        kerneld := Cond(inds=[], kernel,
            SubstTopDownNR(kernel, @.cond(
                e->IsFunction(e) or IsFuncExp(e)),
                e->e.downRankFull(allinds))),
        FoldL(inds, (ker, idx) ->
            ISum(idx.setAttr("GT"), ker), kerneld))
);
```

Organization

- Overview
- System
- Top level commands
- **Abstractions**

  Symbolic matrices and operators: SPL
- Rewriting System I: RuleTree/backtracking search
- Rewriting System II: Visitor Patterns
- Rewriting System III: Associative/large context rules
- Basic block compiler

SPL Formulas as Data Flows

Example: Cooley/Tukey fast Fourier transform (FFT)

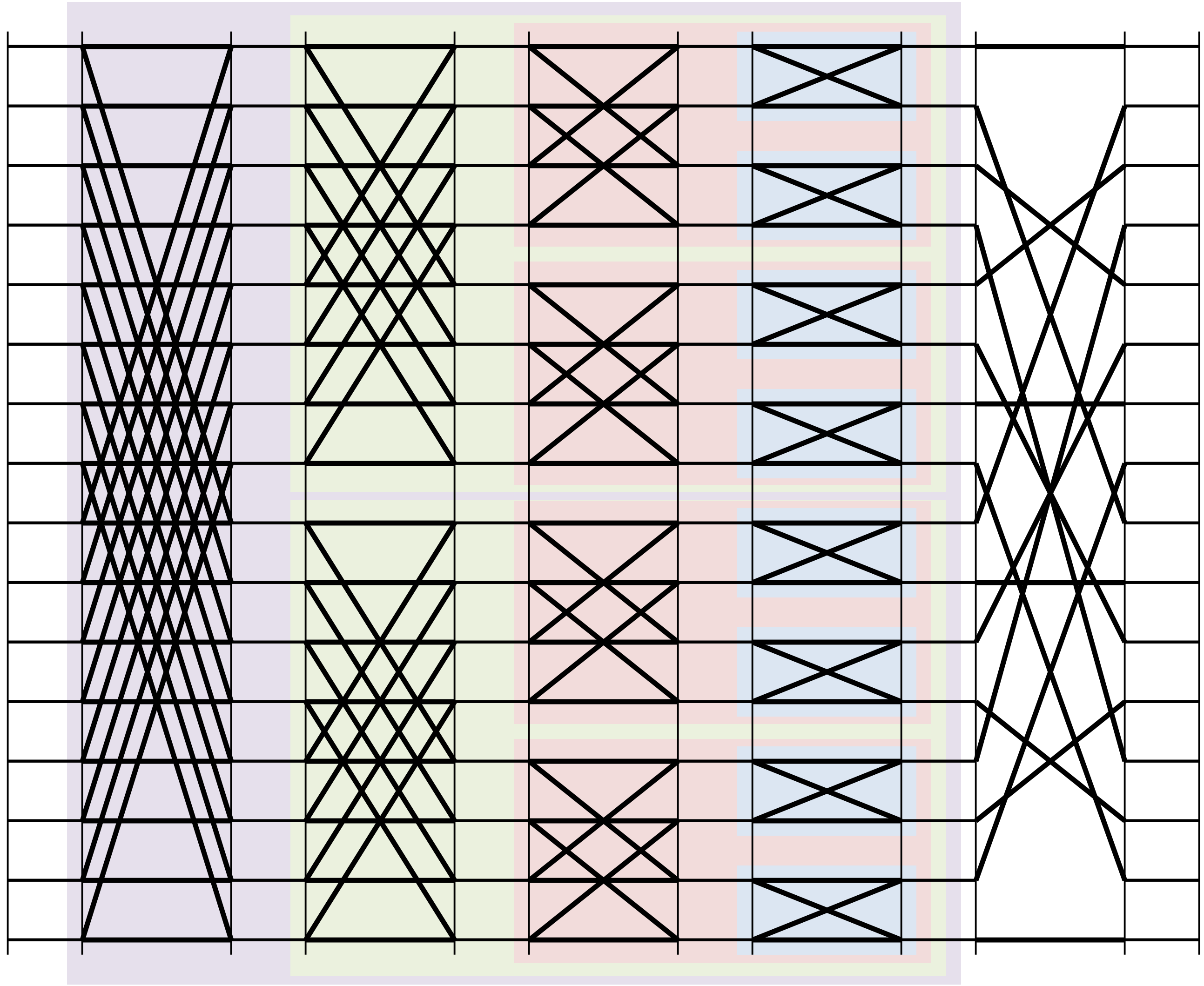
$$\begin{bmatrix}1 & 1 & 1 & 1\\ 1 & j & -1 & -j\\ 1 & -1 & 1 & -1\\ 1 & -j & -1 & j\end{bmatrix} = \begin{bmatrix}1 & \cdot & 1 & \cdot\\ \cdot & 1 & \cdot & 1\\ 1 & \cdot & -1 & \cdot\\ \cdot & 1 & \cdot & -1\end{bmatrix}\begin{bmatrix}1 & \cdot & \cdot & \cdot\\ \cdot & 1 & \cdot & \cdot\\ \cdot & \cdot & 1 & \cdot\\ \cdot & \cdot & \cdot & j\end{bmatrix}\begin{bmatrix}1 & 1 & \cdot & \cdot\\ 1 & -1 & \cdot & \cdot\\ \cdot & \cdot & 1 & 1\\ \cdot & \cdot & 1 & -1\end{bmatrix}\begin{bmatrix}1 & \cdot & \cdot & \cdot\\ \cdot & \cdot & 1 & \cdot\\ \cdot & 1 & \cdot & \cdot\\ \cdot & \cdot & \cdot & 1\end{bmatrix}$$

Fourier transform | Diagonal matrix (twiddles)

$$\mathrm{DFT}_4 \rightarrow (\mathrm{DFT}_2 \otimes \mathrm{I}_2)\,\mathrm{T}^4_2\,(\mathrm{I}_2 \otimes \mathrm{DFT}_2)\,\mathrm{L}^4_2$$

Kronecker product | Identity | Permutation

Example FFT Dataflow and SPL Formula



$$(\mathrm{DFT}_2 \otimes I_8)T_8^{16}\Big(I_2\otimes\Big((\mathrm{DFT}_2 \otimes I_4)T_4^8\big(I_2\otimes\big((\mathrm{DFT}_2 \otimes I_2)T_2^4(I_2\otimes \mathrm{DFT}_2)L_2^4\big)\big)L_2^8\Big)\Big)L_2^{16}$$

SPL: Matrices and Symbolic Matrices

Important examples: SPL objects

```
s1 := I(4);                   # Identity matrix
s2 := F(2);                   # Butterfly matrix
s3 := L(8, 2);                # Stride permutation matrix
s4 := Mat([[1,2],[3,4]]);     # GAP Matrix as SPL object
RowVec(4); ColVec(4);         # row and column vectors
O(4); O(3, 4);                # zero matrix, square and rectangular
```

Important examples: SPL Operations

```
s2 * s4; Compose(s2, s4);     # product of SPL
DirectSum(s1, s2);            # matrix direct sum
Tensor(s1, s2);               # Kronecker product of matrices
HStack(s2, s4);               # [[s2,s4]]
VStack(s2, s4);               # [[s2],[s4]]
```

Operations for SPL objects

```
pm(s1);                       # print as matrix
MatSPL(s2);                   # convert to GAP matrix
s3.transpose();               # symbolic transposition
# List all SPL objects
List(Filtered(Dir(spiral.spl), o->IsSPL(spiral.spl.(o))),
     e->spiral.spl.(e));
```

Still SPL, But Towards Σ-SPL

Permutations

```
s1 := L(8,4);              # The stride permutation...
MatSPL(s1);                # ...is a matrix...
s1.lambda();               # ...and a symbolic function
L(8, 4) * L(8, 2);         # == I(8)
Tensor(I(2), L(4,2)) * L(8,2);         # digit perm needs expression

# other permutation matrices/functions
J(4); Z(4,1); CRT(4,5); RR(13,3,2);
Tensor(J(4), Z(4, 1));
```

Diagonals

```
d:= Diag(fConst(4,1.1));               # Diagonal matrix
d.element;
e:= RCDiag(FList(TReal, [1..16]));     # RC(Diag(...))
e.element;
f := DiagCpxSplit(FList(TReal, [1..16]));
f.element;

# diagonal matrix with dependency on free variable
i := Ind(2);
Diag(fCompose(FList(TReal, [1..4]), fTensor(fId(2), fBase(i))));
```

More SPL Operators

More Operators

```
s1 := DFT(4).terminate();      # Get the complex DFT(4) matrix
s2 := RC(s1);                  # convert it to a real 8x8 matrix
s3 := COND(Ind(), I(2), F(2));         # conditional matrix
s4:= Tensor(DFT(4), I(4));             # transforms are SPL objects
MatSPL(DFT(4)) * [1..4];       # and support SPL and Matrix operations
ConjLR(Tensor(I(2), F(2)), L(4,2), L(4,2));  # classical identity

RowDirectSum(1, F(2), J(2));   # overlapped direct sum
RowTensor(5, 1, F(2));         # overlapped tensor
ColDirectSum(1, F(2), J(2));   # overlapped direct sum
ColTensor(5, 1, F(2));         # overlapped tensor
```

Iterative Operators

```
i := Ind(4);
s5 := IterDirectSum(i, F(2));          # iterative direct sum
s6 := IterDirectSum(i, Mat([[i+1, 2*i],[-3*i, 4*i+5]]));
MatSPL(s6);

s7 := IterHStack(i, F(2));             # iterative HStack
s8 := IterVStack(i, F(2));             # iterative VStack
```

Vector SPL Objects

Vector Tensor SPL Object

```
# spiral-core\namespaces\spiral\paradigms\vector\sigmaspl\vtensor.gi
Class(VTensor, Tensor, rec(
    new := (self, L) >> SPL(WithBases(self, rec(
        _children := [L[1]],
        dimensions := When(IsBound(L[1].dims), L[1].dims(),
        L[1].dimensions) * L[2], vlen := L[2]))),
    from_rChildren := (self, rch) >> ObjId(self)(rch[1], self.vlen),
    print := (self,i,is) >> Print(self.name, "(",
        self.child(1).print(i+is,is), ", ", self.vlen, ")"),
    toAMat := self >> Tensor(self.child(1), I(self.vlen)).toAMat(),
    sums := self >> Inherit(self, rec(_children :=
        [self.child(1).sums()])),
    isPermutation := False,
    dims := self >> self.child(1).dims() * self.vlen,
    needInterleavedLeft := False,
    needInterleavedRight := False,
    transpose := self >> VTensor(self.child(1).transpose(), self.vlen),
    isBlockTransitive := true,
    cannotChangeDataFormat := False,
));
```

Vector RC Objects: Manage Data Layout

SPL Complex-to-real translation for interleaved complex

$$z = z + ib \cong \begin{bmatrix} a & -b \\ b & a \end{bmatrix} \qquad u = c + id \cong \begin{bmatrix} c \\ d \end{bmatrix} \qquad z \cdot u = \begin{bmatrix} a & -b \\ b & a \end{bmatrix} \cdot \begin{bmatrix} c \\ d \end{bmatrix}$$

$$\overline{(.)} : \mathbb{C} \to \mathbb{R}^{2\times 2};\ a + ib \mapsto \begin{bmatrix} a & -b \\ b & a \end{bmatrix}$$

$$\overline{(.)} : \mathbb{C}^{m\times n} \to \mathbb{R}^{2m\times 2n};\ [c_{j,k}]_{j,k} \mapsto \left[\overline{(c_{j,k})}\right]_{j,k}, \quad 0 \le j < m,\ 0 \le k < n$$

Complex block vector interleaved data format

$$\overleftrightarrow{A}^{\nu} = \overline{A},$$

$$\overrightarrow{A}^{\nu} = (\mathrm{I}_{n/\nu} \otimes \mathrm{L}_2^{2\nu})\overline{A}$$

$$\overleftarrow{A}^{\nu} = \overline{A}(\mathrm{I}_{n/\nu} \otimes \mathrm{L}_{\nu}^{2\nu})$$

$$\overline{A}^{\nu} = (\mathrm{I}_{n/\nu} \otimes \mathrm{L}_2^{2\nu})\overline{A}(\mathrm{I}_{n/\nu} \otimes \mathrm{L}_{\nu}^{2\nu})$$

Vector RC Objects: Manage Data Layout

```
# spiral-core\namespaces\spiral\paradigms\vector\sigmaspl\vrc.gi
Class(VRC, RC, rec(
    toAMat := (self) >> AMatMat(RCMatCyc(MatSPL(self.child(1)))),
    new := meth(self, spl, v)
        local res;
        res := SPL(WithBases(self, rec(_children:=[spl], v:=v,
                                  dimensions := spl.dimensions)));
        res.dimensions := res.dims();
        return res;
    end,
    print := (self, i, is) >> Print(self.__name__,
        "(\n", Blanks(i+is), self.child(1).print(i+is,is), ", ",
        #"\n", Blanks(i+is),
        self.v,
        #"\n", Blanks(i),
        ")", self.printA()),
    unroll := self >> self,
    transpose := self >> VRC(self.child(1).conjTranspose(), self.v),
    vcost := self >> self.child(1).vcost(),
    from_rChildren := (self, rch) >> ObjId(self)(rch[1], self.v)
));
```

Vector RC Objects: Manage Data Layout

Vector RC

```
v1  := VRC(Tensor(I(2), F(2)), 4);     # Implicit data reorganization
MatSPL(v1);
v2  := VRCL(Tensor(I(2), F(2)), 4);    # The L and R encodes
MatSPL(v2);
v3  := VRCR(Tensor(I(2), F(2)), 4);    # which side goes from
MatSPL(v3);
v4  := VRCLR(Tensor(I(2), F(2)), 4);   # interleaved complex format to
MatSPL(v4);                            # block split complex format
```

Terminate VRC, VRCL, VRCR, VRCLR

```
Import(paradigms.vector.rewrite);
opts := SIMDGlobals.getOpts(AVX_4x64f);
# see how the format gets propagated down the tree
v5 := VRC(Tensor(I(2), F(2)) * Tensor(F(2), I(2)), 4);
RulesVRC(v5);
# when propagated to the leftmost/rightmost tree leaves, terminate
v6 := VRCL(VTensor(F(2), 4), 4);
v7 := Rewrite(v6, [RulesVRCTerm], opts);
# termnination inserts VPerms to implement local data format change
v8 := VRCR(VTensor(F(2), 4), 4);
v9 := Rewrite(v9, [RulesVRCTerm], opts);
```

Tagged SPL Objects

- **Load balanced, avoiding false sharing**

$$y = \left(\mathrm{I}_p \otimes A\right) x \quad \text{with} \quad A \in \mathbb{C}^{m\mu \times m\mu}$$

$$y = \left(\bigoplus_{i=0}^{p-1} A_i\right) x \quad \text{with} \quad A_i \in \mathbb{C}^{m\mu \times m\mu}$$

$$y = \left(P \otimes \mathrm{I}_\mu\right) x \quad \text{with } P \text{ a permutation matrix}$$

- **Tagged operators (no further rewriting necessary)**

$$\mathrm{I}_p \otimes_{\parallel} A, \qquad \bigoplus_{i=0}^{p-1}{}_{\parallel} A_i, \qquad P \overline{\otimes} \mathrm{I}_\mu$$

Organization

- Overview
- System
- Top level commands
- **Abstractions**
  Σ-SPL: Parameterized matrices, loops
- Rewriting System I: RuleTree/backtracking search
- Rewriting System II: Visitor Patterns
- Rewriting System III: Associative/large context rules
- Basic block compiler

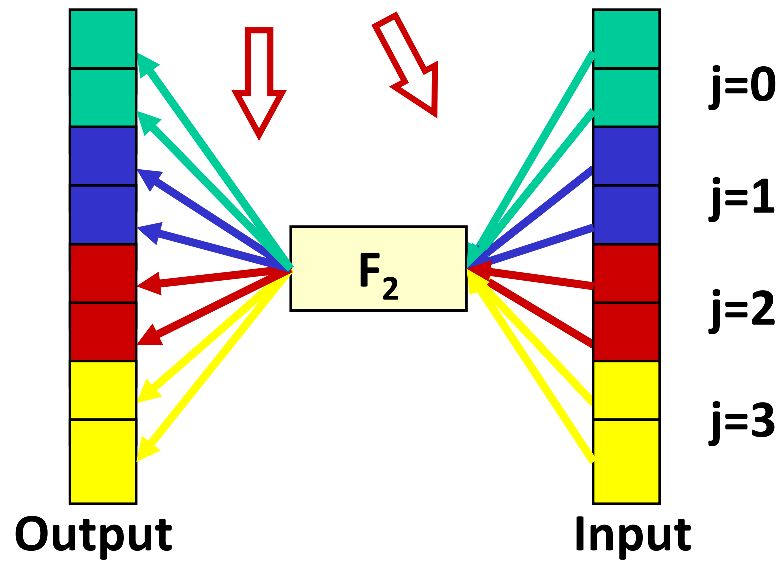
Σ-SPL

- **Four central constructs: S, G, S, Perm**
  - Σ (sum) – makes loops explicit
  - G_f (gather) – reads data using the index mapping f
  - S_f (scatter) – writes data using the index mapping f
  - Perm_f – permutes data using the index mapping f

- **Every Σ-SPL formula still represents a matrix factorization**

Example: $$(I_4 \otimes F_2) \rightarrow \sum_{j=0}^{3} S_{f_j} F_2 G_{f_j}$$

$$\begin{bmatrix} F_2 & & & \\ & F_2 & & \\ & & F_2 & \\ & & & F_2 \end{bmatrix}$$



Σ-SPL: Gather and Scatter

Gather and Scatter matrices

```
g1 := Gath(fId(2));                        # Gath(fId(.)) = I(.)
g2 := Gath(fBase(4, 0));                   # Gath(fBase(.,.)) = base vec
g3 := Gath(fTensor(fBase(4, 0), fId(2)));     # standard pattern

s1 := Scat(fId(2));                        # Scat(fId(.)) = I(.)
s2 := Scat(fBase(4, 2));                   # Scat(fBase(.,.)) = base vec
s3 := Scat(fTensor(fId(2), fBase(4, 3)));     # standard pattern
```

Scatter/Kernel/Gather Pattern

```
A := F(2); j := 0;
# iteration j of Tensor(I(4), F(2))
sag1 := Scat(fTensor(fBase(4, j), fId(2))) * A *
        Gath(fTensor(fBase(4, j), fId(2)));

# iteration j of Tensor(F(2), I(4))
sag2 := Scat(fTensor(fId(2), fBase(4, j))) * A *
        Gath(fTensor(fId(2), fBase(4, j)));

# iteration j of Tensor(I(4), F(2)) * L(8, 4)
sag3 := Scat(fTensor(fBase(4, j), fId(2))) * A *
        Gath(fTensor(fId(2), fBase(4, j)));
pm(sag1); pm(sag2); pm(sag3);
```

Σ-SPL: Gather, Scatter, and ISum

Tensor Product and Gath/Scat/ISum

```
A := F(2);
j := Ind(4);
# Sigma-SPL for Tensor(I(4), F(2))
sag1 := Scat(fTensor(fBase(j), fId(2))) * A *
        Gath(fTensor(fBase(j), fId(2)));
s1 := ISum(j, sag1);
MatSPL(s1) = MatSPL(Tensor(I(4), F(2)));
```

Other Tensor Patterns

```
# Sigma-SPL for Tensor(F(2), I(4))
s2 := ISum(j, Scat(fTensor(fId(2), fBase(j))) * A *
        Gath(fTensor(fId(2), fBase(j))));
MatSPL(s2) = MatSPL(Tensor(F(2), I(4)));

# Sigma-SPL for Tensor(I(4), F(2)) * L(8, 4)
s3 := ISum(j, Scat(fTensor(fBase(j), fId(2))) * A *
        Gath(fTensor(fId(2), fBase(j))));
MatSPL(s3) = MatSPL(Tensor(I(4), F(2)) * L(8, 4));

# Direct sum
ISum(j, Scat(fTensor(fBase(j), fId(2))) * Mat([[j,-j],[j, j]]) *
        Gath(fTensor(fBase(j), fId(2))));
```

Σ-SPL: Advanced Loops

Tensor Product and Gath/Scat/ISum

```
A := F(2);
j := Ind(4);
k := Ind(2);

# Sigma-SPL for Tensor(I(4), F(2), I(2))
sag := Scat(fTensor(fBase(j), fId(2), fBase(k))) * A *
        Gath(fTensor(fBase(j), fId(2), fBase(k)));
s := ISum(k, ISum(j, sag));
MatSPL(s) = MatSPL(Tensor(I(4), F(2), I(2)));
```

More complex example

```
i := Ind(8);
j := Ind(4);
k := Ind(2);
A := Mat([[j+1,-2*j],[j*k, j+1]]); B := F(2);

s := ISum(k, ISum(j, Scat(fTensor(fBase(j), fBase(k), fId(2))) * A *
                     Gath(fTensor(fId(2), fBase(k), fBase(j))))) *
     ISum(i, Scat(fTensor(fBase(i), fId(2))) * B
           * Gath(fTensor(J(2), fCompose(Z(8,3), fBase(i)))));
MatSPL(s);
```

Σ-SPL: Gath/Scat, Diag and Perms

Gather Functions

```
# use Lambda functions directly in Gath/Scat
i := Ind(8);
f1 := Lambda(i, imod(5*i+7, 32)).setRange(32);
g := Gath(f1);

# Use indirection tables in Gath/Scat. By default not supported
f2 := CopyFields(FData(List([0..7], j->V(Mod(5*j+7, 32)))),
                 rec(range := self >> 32));
s := Scat(f2);
```

Diagonals

```
# the true Twiddle diagonal in DFT(8)
d := Diag(fPrecompute(fCompose(dOmega(8, 1),
          diagTensor(dLin(V(4), 1, 0, TInt), dLin(2, 1, 0, TInt)))));
MatSPL(d);
e:= RCDiag(fCompose(FData(List([1..32], u->Value(TReal, u))),
           fTensor(fId(4), fBase(i))));
# print out the function values
e.element.tolist();
# access the indirection table
e.element.children()[1].var.value;
```

SMP Tagged Σ-SPL Objects

Parallel Loop == SMPSum

```
# spiral-core\namespaces\spiral\paradigms\smp\sigmaspl.gi
Class(SMPSum, ISum, rec(
    doNotMarkBB := true,
    abbrevs := [ (p, var, domain, spl) -> Checked(IsInt(p) or
        IsScalar(p), IsVar(var), IsInt(p) or IsScalar(domain),
        IsSPL(spl), [p, threadId(), var, domain, spl]) ],
    new := meth(self, nthreads, tid, var, domain, spl)
        local res;
        Constraint(IsSPL(spl));
        Constraint(IsPosIntSym(domain));
        var.isLoopIndex := true;
        var.range := domain;
        res := SPL(WithBases(self, rec(nthreads:=nthreads, tid:=tid,
            _children := [spl], var := var, domain := domain)));
        res.dimensions := res.dims();
        return res;
    end,
    print := (self, i, is) >> Print(self.__name__, "(",
        self.nthreads, ", ", self.tid, ", ", self.var, ", ",
        self.domain, ",\n",
        Blanks(i+is), self.child(1).print(i+is, is), "\n",
        Blanks(i), ")", self.printA())
));
```

SMP Tagged Σ-SPL Objects

Barrier Object

```
# spiral-core\namespaces\spiral\paradigms\smp\sigmaspl.gi
Class(SMPBarrier, Buf, BaseContainer, rec(
    doNotMarkBB := true,
    new := (self, nthreads, tid, spl) >> Checked(IsPosInt0Sym(tid),
       IsPosIntSym(nthreads), IsSPL(spl),
       SPL(WithBases(self, rec(_children:=[spl],
       tid := tid, nthreads := nthreads, dimensions := spl.dims())))),
    dims := self >> self._children[1].dims(),
));
```

Context Object

```
Class(SMP, BaseContainer, rec(
    doNotMarkBB := true,
    abbrevs := [ (nthreads, spl) -> Checked(IsInt(nthreads) or
        IsScalar(nthreads), IsSPL(spl), [nthreads, threadId(), spl]) ],
    new := (self, nthreads, tid, spl) >> SPL(WithBases(self,
        rec(nthreads:=nthreads, tid:=tid,
        dimensions := spl.dimensions, _children := [spl]))),
    sums := self >> ObjId(self)(
        self.nthreads, self.tid, self.child(1).sums()),
));
```

Vector Σ-SPL Objects

Vector Gather

```
# spiral-core\namespaces\spiral\paradigms\vector\sigmaspl\gather.gi
Class(VGath, BaseVGath, SumsBase, rec(
    rChildren := self >> [self.func],
    rSetChild := rSetChildFields("func"),
    from_rChildren := (self, rch) >> ObjId(self)(rch[1], self.v),
    new := (self, func, v) >> SPL(WithBases(self,
        rec(func := func, v := v))).setDims(),
    dims := self >> [self.v*self.func.domain(),
        self.v*self.func.range()],
    transpose := self >> VScat(self.func, self.v),
    print := (self,i,is) >> Print(self.name, "(", self.func, ", ",
        self.v,")", self.printA()),
    toAMat := self >> let(v:=self.v, n :=
        EvalScalar(v*self.func.domain()),
            N := EvalScalar(v*self.func.range()),
            func := fTensor(self.func, fId(v)).lambda(),
            AMatMat(List([0..n-1], row -> BasisVec(N,
                EvalScalar(func.at(row).ev()))))),
));
```

SPL And Σ-SPL Vector Objects

Generate The Example

```
Import(paradigms.vector.sigmaspl);
opts := SIMDGlobals.getOpts(AVX_4x64f);
rt := RandomRuleTree(TRC(DFT(16)).withTags(opts.tags), opts);
s := SPLRuleTree(rt);                    # SPL Objects
ss := SumsRuleTree(rt, opts);            # Sigma-SPL Objects
```

Inspecting the Vector Objects

```
Collect(s, VTensor)[1];         # Vectorized Tensor(., I(v))
Collect(ss, VTensor)[1];        # Vectorized Tensor(., I(v))
Collect(s, VPerm)[1];           # Vectorized Prm(.)
Collect(s, BlockVPerm)[1];      # Vectorized Tensor(I(.), Prm(.))
Collect(s, VContainer)[1];      # Provides context for rewriting
Collect(s, VRC)[1];             # Carries interleaved complex format
Collect(ss, VGath)[1];          # Tensor(Gath(.), I(v))
Collect(ss, VScat)[1];          # Tensor(Scat(.), I(v))
Collect(ss, VRCDiag)[1];        # Vectorized Diag(.)
```

Organization

- Overview
- System
- Top level commands
- Abstractions
- **Rewriting System I: RuleTree/backtracking search**
  General breakdown and search
- Rewriting System II: Visitor Patterns
- Rewriting System III: Associative/large context rules
- Basic block compiler

Simple Example Breakdown Rules

$$\begin{array}{rcl}
\mathbf{DFT}_n & \rightarrow & (\mathbf{DFT}_k \otimes \mathrm{I}_m)\, \mathsf{T}^n_m (\mathrm{I}_k \otimes \mathbf{DFT}_m)\, \mathsf{L}^n_k, \quad n = km \\
\mathbf{DFT}_n & \rightarrow & P_n(\mathbf{DFT}_k \otimes \mathbf{DFT}_m) Q_n, \quad n = km,\ \gcd(k,m) = 1 \\
\mathbf{DFT}_p & \rightarrow & R_p^T(\mathrm{I}_1 \oplus \mathbf{DFT}_{p-1}) D_p (\mathrm{I}_1 \oplus \mathbf{DFT}_{p-1}) R_p, \quad p \text{ prime} \\
\mathbf{DCT\text{-}3}_n & \rightarrow & (\mathrm{I}_m \oplus \mathsf{J}_m)\, \mathsf{L}^n_m (\mathbf{DCT\text{-}3}_m(1/4) \oplus \mathbf{DCT\text{-}3}_m(3/4)) \\
 & & \cdot(\mathsf{F}_2 \otimes \mathrm{I}_m) \begin{bmatrix} \mathrm{I}_m & 0 \oplus -\mathsf{J}_{m-1} \\ & \frac{1}{\sqrt{2}}(\mathrm{I}_1 \oplus 2\,\mathrm{I}_m) \end{bmatrix}, \quad n = 2m \\
\mathbf{DCT\text{-}4}_n & \rightarrow & S_n \mathbf{DCT\text{-}2}_n\, \mathrm{diag}_{0 \le k < n}(1/(2\cos((2k+1)\pi/4n))) \\
\mathbf{IMDCT}_{2m} & \rightarrow & (\mathsf{J}_m \oplus \mathrm{I}_m \oplus \mathrm{I}_m \oplus \mathsf{J}_m) \left(\left(\begin{bmatrix} 1 \\ -1 \end{bmatrix} \otimes \mathrm{I}_m \right) \oplus \left(\begin{bmatrix} -1 \\ -1 \end{bmatrix} \otimes \mathrm{I}_m \right) \right) \mathsf{J}_{2m}\, \mathbf{DCT\text{-}4}_{2m} \\
\mathbf{WHT}_{2^k} & \rightarrow & \prod_{i=1}^{t} (\mathrm{I}_{2^{k_1+\cdots+k_{i-1}}} \otimes \mathbf{WHT}_{2^{k_i}} \otimes \mathrm{I}_{2^{k_{i+1}+\cdots+k_t}}), \quad k = k_1 + \cdots + k_t \\
\mathbf{DFT}_2 & \rightarrow & \mathsf{F}_2 \\
\mathbf{DCT\text{-}2}_2 & \rightarrow & \mathrm{diag}(1, 1/\sqrt{2})\, \mathsf{F}_2 \\
\mathbf{DCT\text{-}4}_2 & \rightarrow & \mathsf{J}_2\, \mathsf{R}_{13\pi/8}
\end{array}$$

Combining these rules yields many algorithms for every given transform

More Complicated Breakdown Rules

$$\mathbf{DFT}_n \to P^\top_{k/2,2m}\left(\mathbf{DFT}_{2m} \oplus \left(I_{k/2-1} \otimes_i C_{2m}\,\mathbf{rDFT}_{2m}(i/k)\right)\right)\left(\mathbf{RDFT}'_k \otimes I_m\right), \quad k \text{ even},$$

$$\begin{vmatrix}\mathbf{RDFT}_n\\ \mathbf{RDFT}'_n\\ \mathbf{DHT}_n\\ \mathbf{DHT}'_n\end{vmatrix} \to (P^\top_{k/2,m} \otimes I_2)\left(\begin{vmatrix}\mathbf{RDFT}_{2m}\\ \mathbf{RDFT}'_{2m}\\ \mathbf{DHT}_{2m}\\ \mathbf{DHT}'_{2m}\end{vmatrix} \oplus \left(I_{k/2-1} \otimes_i D_{2m}\begin{vmatrix}\mathbf{rDFT}_{2m}(i/k)\\ \mathbf{rDFT}_{2m}(i/k)\\ \mathbf{rDHT}_{2m}(i/k)\\ \mathbf{rDHT}_{2m}(i/k)\end{vmatrix}\right)\right)\left(\begin{vmatrix}\mathbf{RDFT}'_k\\ \mathbf{RDFT}'_k\\ \mathbf{DHT}'_k\\ \mathbf{DHT}'_k\end{vmatrix} \otimes I_m\right), \quad k \text{ even},$$

$$\begin{vmatrix}\mathbf{rDFT}_{2n}(u)\\ \mathbf{rDHT}_{2n}(u)\end{vmatrix} \to L^{2n}_m\left(I_k \otimes_i \begin{vmatrix}\mathbf{rDFT}_{2m}((i+u)/k)\\ \mathbf{rDHT}_{2m}((i+u)/k)\end{vmatrix}\right)\left(\begin{vmatrix}\mathbf{rDFT}_{2k}(u)\\ \mathbf{rDHT}_{2k}(u)\end{vmatrix} \otimes I_m\right),$$

$$\mathbf{RDFT\text{-}3}_n \to (Q^\top_{k/2,m} \otimes I_2)\,(I_k \otimes_i \mathbf{rDFT}_{2m})(i+1/2)/k))\,(\mathbf{RDFT\text{-}3}_k \otimes I_m), \quad k \text{ even},$$

$$\mathbf{DCT\text{-}2}_n \to P^\top_{k/2,2m}\left(\mathbf{DCT\text{-}2}_{2m}\,K^{2m}_2 \oplus \left(I_{k/2-1} \otimes N_{2m}\,\mathbf{RDFT\text{-}3}^\top_{2m}\right)\right)B_n(L^{n/2}_{k/2} \otimes I_2)(I_m \otimes \mathbf{RDFT}'_k)Q_{m/2,k},$$

$$\mathbf{DCT\text{-}3}_n \to \mathbf{DCT\text{-}2}^\top_n,$$

$$\mathbf{DCT\text{-}4}_n \to Q^\top_{k/2,2m}\left(I_{k/2} \otimes N_{2m}\,\mathbf{RDFT\text{-}3}^\top_{2m}\right)B'_n(L^{n/2}_{k/2} \otimes I_2)(I_m \otimes \mathbf{RDFT\text{-}3}_k)Q_{m/2,k}.$$

$$\mathbf{DFT}_n \;\to\; (\mathbf{DFT}_k \otimes \mathrm{I}_m)\,\mathrm{T}^n_m(\mathrm{I}_k \otimes \mathbf{DFT}_m)\,\mathrm{L}^n_k, \quad n = km$$

$$\mathbf{DFT}_n \;\to\; P_n(\mathbf{DFT}_k \otimes \mathbf{DFT}_m)Q_n, \quad n = km,\ \gcd(k,m) = 1$$

$$\mathbf{DFT}_p \;\to\; R^T_p(\mathrm{I}_1 \oplus \mathbf{DFT}_{p-1})D_p(\mathrm{I}_1 \oplus \mathbf{DFT}_{p-1})R_p, \quad p \text{ prime}$$

$$\mathbf{DCT\text{-}3}_n \;\to\; (\mathrm{I}_m \oplus \mathrm{J}_m)\,\mathrm{L}^n_m(\mathbf{DCT\text{-}3}_m(1/4) \oplus \mathbf{DCT\text{-}3}_m(3/4)) \cdot(\mathrm{F}_2 \otimes \mathrm{I}_m)\begin{bmatrix}\mathrm{I}_m & 0 \oplus -\mathrm{J}_{m-1}\\ & \frac{1}{\sqrt{2}}(\mathrm{I}_1 \oplus 2\,\mathrm{I}_m)\end{bmatrix}, \quad n = 2m$$

$$\mathbf{DCT\text{-}4}_n \;\to\; S_n\mathbf{DCT\text{-}2}_n\,\mathrm{diag}_{0\le k<n}(1/(2\cos((2k+1)\pi/4n)))$$

$$\mathbf{IMDCT}_{2m} \;\to\; (\mathrm{J}_m \oplus \mathrm{I}_m \oplus \mathrm{I}_m \oplus \mathrm{J}_m)\left(\left(\begin{bmatrix}1\\-1\end{bmatrix} \otimes \mathrm{I}_m\right) \oplus \left(\begin{bmatrix}-1\\-1\end{bmatrix} \otimes \mathrm{I}_m\right)\right)\mathrm{J}_{2m}\,\mathbf{DCT\text{-}4}_{2m}$$

$$\mathbf{WHT}_{2^k} \;\to\; \prod_{i=1}^{t}(\mathrm{I}_{2^{k_1+\cdots+k_{i-1}}} \otimes \mathbf{WHT}_{2^{k_i}} \otimes \mathrm{I}_{2^{k_{i+1}+\cdots+k_t}}), \quad k = k_1 + \cdots + k_t$$

$$\mathbf{DFT}_2 \;\to\; \mathrm{F}_2$$

$$\mathbf{DCT\text{-}2}_2 \;\to\; \mathrm{diag}(1, 1/\sqrt{2})\,\mathrm{F}_2$$

$$\mathbf{DCT\text{-}4}_2 \;\to\; \mathrm{J}_2\,\mathrm{R}_{13\pi/8}$$

- *"Teaches" Spiral algorithm knowledge*
- *Combining these rules yields many algorithms for every given transform*

Example Expansion (Rule Tree)

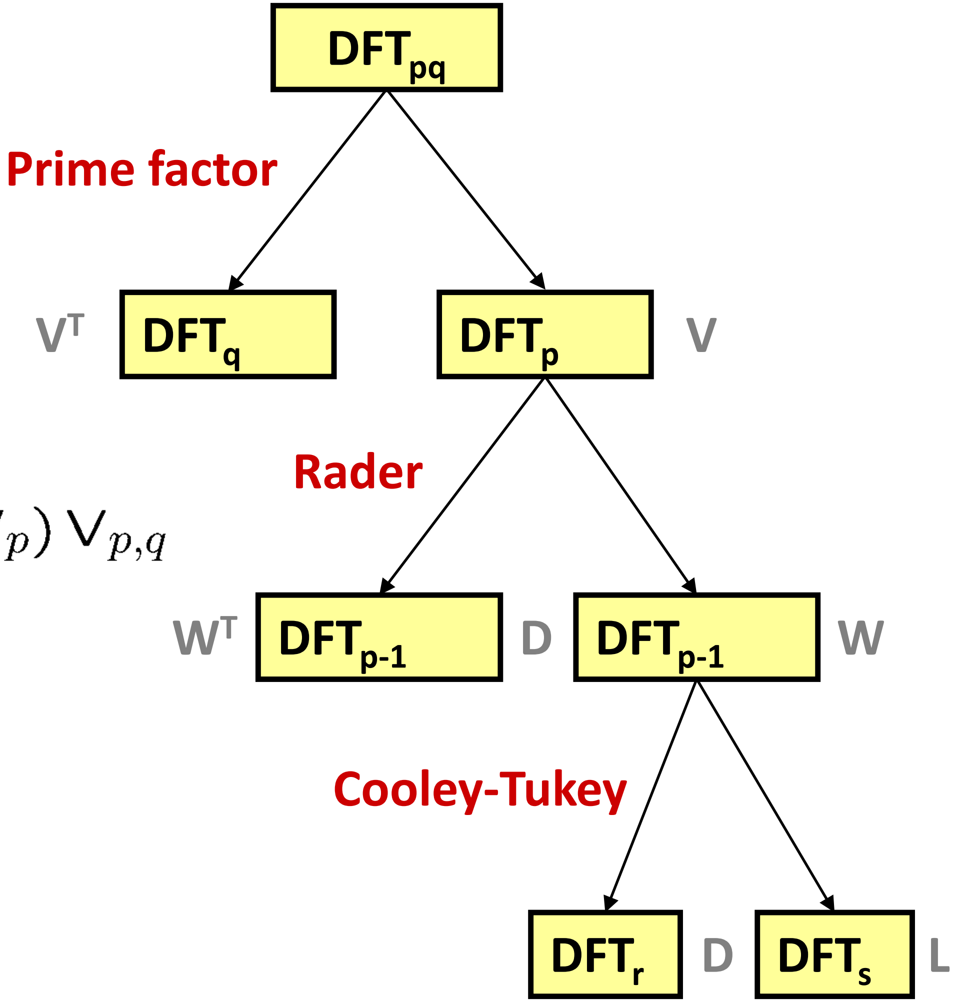
Given DFT_{pq}

p – prime

p-1 = rs

Formula fragment

$$(I_p \otimes (I_1 \oplus (I_r \otimes DFT_s)\, L_r^{rs})\, W_p)\, V_{p,q}$$



Rule Trees

Expand Non-Terminals

```
opts := SpiralDefaults;
t1 := DFT(4);                        # complex DFT of size 4
rt1 := RandomRuleTree(t1, opts);     # create a random rule tree
t2 := DFT(80);                       # complex DFT of size 80
rt2 := RandomRuleTree(t2, opts);     # create a random rule tree
```

Exploring a Rule Tree

```
rt1.node;                            # this node
rt1.rule;                            # rule applied at node
rt1.transposed;                      # rule applied transposed ?
rt1.children;                        # a level down in the tree
rt1.children[1];                     # first child node
rt1.children[1].node;                # same as root node
rt1.children[1].rule;
rt1.children[1].children;
rt1.children[1].transposed;
rt1.children[2];                     # second child node
rt1.children[2].node;                # again tree node structure
rt1.children[2].rule;
rt1.children[2].children;
rt1.children[2].transposed;
```

Breakdown Rules: Base Rule

Definition

```
# In spiral-core\namespaces\spiral\transforms\dft\dft_rules.gi
 DFT_Base := rec(
     forTransposition := false,
     applicable       := nt -> nt.params[1] = 2 and not nt.hasTags(),
     apply            := (nt, C, cnt) -> F(2)
)
```

Twiddle function for DFT

```
Tw1 := (n,d,k) -> Checked(
    IsPosIntSym(n), IsPosIntSym(d), IsIntSym(k),
    fCompose(dOmega(n,k),
        diagTensor(dLin(div(n,d), 1, 0, TInt),
            dLin(d, 1, 0, TInt))));
```

Rule methods

```
PrintActiveRules(DFT);          # rules for DFT currently active
DFT_Base.switch;                # filed in rule to determine active
t := DFT(2);
DFT_Base.applicable(t);         # is the rule applicable
DFT_Base.children(t);           # all possible Algorithmic choices
DFT_Base.apply(t, [], []);      # t->spl for a particular choice
```

Breakdown Rules: Cooley-Tukey Rule

Definition

```
# In spiral-core\namespaces\spiral\transforms\dft\dft_rules.gi
DFT_CT := rec(
    maxSize          := false,
    forcePrimeFactor := false,
    applicable := (self, nt) >> nt.params[1] > 2
        and not nt.hasTags()
        and (self.maxSize=false or nt.params[1] <= self.maxSize)
        and not IsPrime(nt.params[1])
        and When(self.forcePrimeFactor, not
                 DFT_GoodThomas.applicable(nt), true),
    children   := nt -> Map2(DivisorPairs(nt.params[1]),
            (m,n) -> [ DFT(m, nt.params[2] mod m),
                        DFT(n, nt.params[2] mod n) ]),
    apply := (nt, C, cnt) -> let(mn := nt.params[1],
                   m := Rows(C[1]), n := Rows(C[2]),
            Tensor(C[1], I(n)) *
            Diag(fPrecompute(Tw1(mn, n, nt.params[2]))) *
            Tensor(I(m), C[2]) *
            L(mn, m)
        )
    )
```

Breakdown Rules: Cooley-Tukey Rule

Applicability: Cooley Tukey requires non-prime size

```
t  := DFT(2);
t1  := DFT(4);
t2  := DFT(8);
t3  := DFT(20);

DFT_CT.applicable(t);          # see for which sized DFT_CT
DFT_CT.applicable(DFT(5));     # is applicable
DFT_CT.applicable(t1);
DFT_CT.applicable(t2);
DFT_CT.applicable(t3);
```

Children: algorithmic choices

```
c1  := DFT_CT.children(t2);    # enumerate all algorithmic choices
c2  := DFT_CT.children(t2);
c3  := DFT_CT.children(t3);
```

Expand DFT(8) by hand

```
s  := DFT_Base.apply(t, [], []);          # expand DFT(2) -> F(2)
s1  := DFT_CT.apply(t1, [s, s], [t, t]);        # DFT(4) -> SPL
s2  := DFT_CT.apply(t2, [s1, s], [t1, t]);     # DFT(8) -> SPL
MatSPL(t2) = MatSPL(s2);
```

Ruletrees and SPL

From Transform to SPL Formula

```
n := 8; k := -1;                        # transform parameters
opts := CopyFields(SpiralDefaults,      # local configuration
    rec(breakdownRules := rec(
        DFT := [DFT_Base,
        CopyFields(DFT_CT, rec(maxSize := 20))]))));
t := DFT(n, k);                         # transform
rt := RandomRuleTree(t, opts);          # get rule tree
spl := SPLRuleTree(rt);                 # SPL formula
```

Impact of configuration

```
PrintActiveRules(DFT);
opts.breakdownRules.DFT;
DFT_CT.maxSize;                 # global configuration unchanged
ct := Filtered(opts.breakdownRules.DFT, i->i.name =  DFT_CT.name)[1];
ct.maxSize;                     # access local configuration
t2 := DFT(21);                  # works with global but not local opts
rt := RandomRuleTree(t2, SpiralDefaults);
rt2 := RandomRuleTree(t2, opts);
FindUnexpandableNonterminal(t2, opts);        # Where do we fail?
ct.maxSize := false;                          # remove guard in DFT_CT
rt2 := RandomRuleTree(t2, opts);              # try again
FindUnexpandableNonterminal(t2, opts);        # Where do we fail now?
```

Backtracking Search: Dynamic Programming

Standard dynamic programming

```
n := 15; 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;          # get best rule tree
```

Hashing and custom measure functions

```
dpopts := rec(verbosity := 5, hashTable := HashTableDP());
dpopts.measureFunction := (rt, opts) ->
    let(c:= CodeRuleTree(rt, opts),   # generate code
        Length(Filtered(              # count flops in code
            Collect(c, @(1,[add, sub, mul, neg])), i->i.t=TReal)));
best := DP(t, dpopts, opts);  # run search with flop minimization
#look what's in the hash table
HashLookup(dpopts.hashTable, DFT(5, 1))[1].ruletree;
HashLookup(dpopts.hashTable, DFT(5,-1));

# now time the tree found through flop minimization
rt1 := HashLookup(dpopts.hashTable, DFT(15, 1))[1].ruletree;
DPMeasureRuleTree(rt1, opts);
```

Organization

- Overview
- System
- Top level commands
- Abstractions
- **Rewriting System I: RuleTree/backtracking search**
  Constrained search, hardware specific rules
- Rewriting System II: Visitor Patterns
- Rewriting System III: Associative/large context rules
- Basic block compiler

Architecture Specific Breakdown Rules

- **Goal: Transform formulas into fully optimized formulas**
  - Formulas rewritten, tags propagated
  - There may be choices

$$\underbrace{AB}_{\mathsf{smp}(p,\mu)} \rightarrow \underbrace{A}_{\mathsf{smp}(p,\mu)}\ \underbrace{B}_{\mathsf{smp}(p,\mu)}$$

$$\underbrace{A_m \otimes I_n}_{\mathsf{smp}(p,\mu)} \rightarrow \underbrace{\left(L^{mp}_m \otimes I_{n/p}\right)\left(I_p \otimes (A_m \otimes I_{n/p})\right)\left(L^{mp}_p \otimes I_{n/p}\right)}_{\mathsf{smp}(p,\mu)}$$

$$\underbrace{L^{mn}_m}_{\mathsf{smp}(p,\mu)} \rightarrow \begin{cases} \underbrace{\left(I_p \otimes L^{mn/p}_{m/p}\right)}_{\mathsf{smp}(p,\mu)} \underbrace{\left(L^{pn}_p \otimes I_{m/p}\right)}_{\mathsf{smp}(p,\mu)} \\ \underbrace{\left(L^{pm}_m \otimes I_{n/p}\right)}_{\mathsf{smp}(p,\mu)} \underbrace{\left(I_p \otimes L^{mn/p}_m\right)}_{\mathsf{smp}(p,\mu)} \end{cases}$$

$$\underbrace{I_m \otimes A_n}_{\mathsf{smp}(p,\mu)} \rightarrow I_p \otimes_{\parallel} \left(I_{m/p} \otimes A_n\right)$$

$$\underbrace{(P \otimes I_n)}_{\mathsf{smp}(p,\mu)} \rightarrow \left(P \otimes I_{n/\mu}\right) \bar{\otimes} I_\mu$$

Ruletrees are Result of Backtracking

Tagged Nonterminal

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

AVX 2-way `_Complex double`

Base cases

$A^{n\times n}\vec{\otimes}\mathrm{I}_2$

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

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

Transformation rules

$$(\mathrm{I}_m\otimes A^{n\times n})\mathrm{L}^{mn}_m \rightarrow \big(\mathrm{I}_{m/\nu}\otimes \mathrm{L}^{n\nu}_\nu(A^{n\times n}\otimes \mathrm{I}_\nu)\big)(\mathrm{L}^{mn/\nu}_{m/\nu}\otimes \mathrm{I}_\nu)$$

$$\mathrm{L}^{n\nu}_\nu \rightarrow (\mathrm{L}^n_\nu\otimes \mathrm{I}_\nu)(\mathrm{I}_{n/\nu}\otimes \mathrm{L}^{\nu^2}_\nu)$$

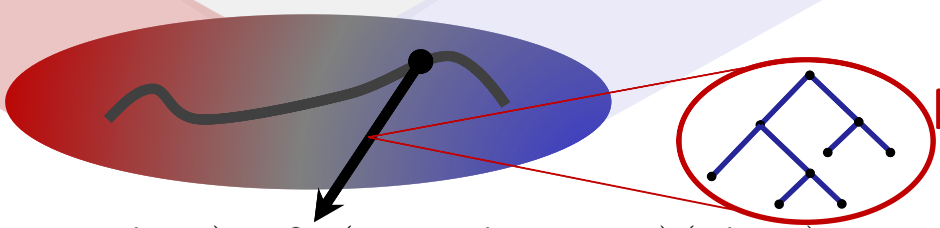
$$A^{m\times m}\otimes \mathrm{I}_n \rightarrow (A^{m\times m}\otimes \mathrm{I}_{n/\nu})\otimes \mathrm{I}_\nu$$

DFT_8

Breakdown rules

$$\mathrm{DFT}_{mn} \rightarrow (\mathrm{DFT}_m\otimes \mathrm{I}_n)\mathrm{T}^{mn}_n(\mathrm{I}_m\otimes \mathrm{DFT}_n)\mathrm{L}^{mn}_m$$

$$\mathrm{DFT}_2 \rightarrow \mathrm{F}_2$$



Ruletree

$$\big((\mathrm{F}_2\otimes \mathrm{I}_2)\mathrm{T}^4_2(\mathrm{I}_2\otimes \mathrm{F}_2)\mathrm{L}^4_2\vec{\otimes}\mathrm{I}_2\big)\underbrace{\mathrm{T}^8_2}_{\mathrm{vec}(2)}\big(\mathrm{I}_2\otimes \underbrace{\mathrm{L}^4_2}_{\mathrm{vec}(2)}(\mathrm{F}_2\vec{\otimes}\mathrm{I}_2)\big)\big(\mathrm{L}^4_2\vec{\otimes}\mathrm{I}_2\big)$$

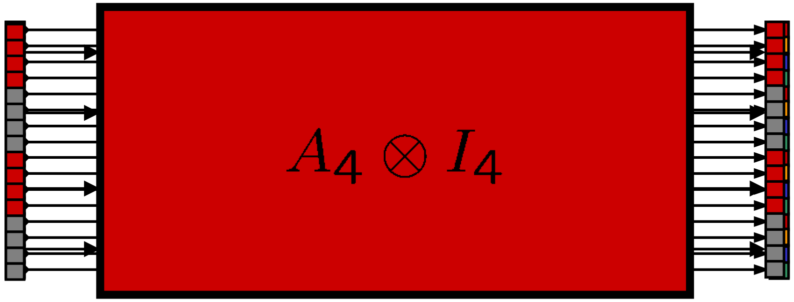
Tagged SPL Formula

SPL specification
↓ Expansion + backtracking
SPL (dataflow) expression
↓ Recursive descent
Σ-SPL (loop) expression
↓ Confluent term rewriting
Optimized Σ-SPL expression
↓ Recursive descent
icode
↓ Confluent term rewriting
Optimized icode
↓ Recursive descent
C code

Vectorization: Basic Idea

- **Good construct: tensor product**

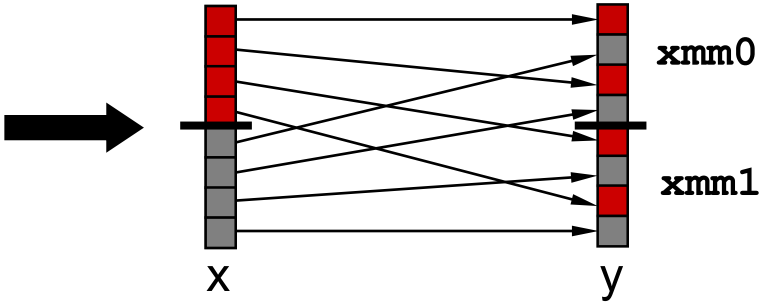
$$y = \left(A \otimes \mathrm{I}_\nu\right) x$$



Characteristics: block operation and alignment preserving

- **Problematic construct: permutations must be done in register**

$$y = \mathrm{L}^8_4 x$$



Task: Rewrite formulas to extract tensor product + minimize in-register shuffles

Some Vectorization Rules

$$\underbrace{\left(\overline{A}\right)}_{\mathrm{vec}(\nu)} \rightarrow \overleftrightarrow{\left(\underbrace{A}_{\mathrm{vec}(\nu)}\right)}^{\nu}$$

$$\overleftrightarrow{AB}^{\nu} \rightarrow \overleftarrow{A}^{\nu}\,\overrightarrow{B}^{\nu}$$

$$\overleftarrow{AB}^{\nu} \rightarrow \overleftarrow{A}^{\nu}\,\overline{B}^{\nu}$$

$$\underbrace{A \otimes \mathrm{I}_m}_{\mathrm{vec}(\nu)} \rightarrow \left(A \otimes \mathrm{I}_{m/\nu}\right) \otimes_{\nu} \mathrm{I}_{\nu}$$

$$\underbrace{\left(\mathrm{I}_m \otimes A\right) \mathrm{L}^{mn}_{m}}_{\mathrm{vec}(\nu)} \rightarrow \left(\mathrm{I}_{m/\nu} \otimes \underbrace{\mathrm{L}^{n\nu}_{\nu}}_{\mathrm{vec}(\nu)} \left(A \otimes_{\nu} \mathrm{I}_{\nu}\right)\right)\left(\mathrm{L}^{mn/\nu}_{m/\nu} \otimes_{\nu} \mathrm{I}_{\nu}\right)$$

$$\underbrace{\mathrm{L}^{n\nu}_{\nu}}_{\mathrm{vec}(\nu)} \rightarrow \left(\mathrm{L}^{n}_{\nu} \otimes_{\nu} \mathrm{I}_{\nu}\right)\left(\mathrm{I}_{n/\nu} \otimes \underbrace{\mathrm{L}^{\nu^2}_{\nu}}_{\mathrm{vec}(\nu)}\right)$$

$$\overleftarrow{A \otimes_{\nu} \mathrm{I}_{\nu}}^{\nu} \rightarrow (\mathrm{I}_{n/\nu} \otimes \underbrace{\mathrm{L}^{2\nu}_{\nu}}_{\mathrm{vec}(\nu)})(\overline{A} \otimes_{\nu} \mathrm{I}_{\nu})$$

$$\overline{\left(\underbrace{\mathrm{L}^{\nu^2}_{\nu}}_{\mathrm{vec}(\nu)}\right)}^{\nu} \rightarrow (\mathrm{L}^{2\nu}_{\nu} \otimes_{\nu} \mathrm{I}_{\nu})(\mathrm{I}_2 \otimes \underbrace{\mathrm{L}^{\nu^2}_{\nu}}_{\mathrm{vec}(\nu)})(\mathrm{L}^{2\nu}_{2} \otimes_{\nu} \mathrm{I}_{\nu})$$

$$\overleftarrow{\mathrm{I}_m \otimes A}^{\nu} \rightarrow \mathrm{I}_m \otimes \overleftarrow{A}^{\nu}$$

Ruletree Encodes the Search Solution

```
spiral> rt := RandomRuleTree(t, opts);
TRC_vect( TRC(DFT(16, 1)).withTags([ AVecReg(AVX_4x64f) ]),
  @_Base( DPWrapper(DFT(16, 1).withTags([ AVecReg(AVX_4x64f) ]), VWrapTRC(AVX_4x64f)),
    DFT_tSPL_CT( DFT(16, 1).withTags([ AVecReg(AVX_4x64f) ]),
      TCompose_tag( TCompose([ TGrp(TCompose([ TTensorI(DFT(4, 1), 4, AVec, AVec),
          TTwiddle(16, 4, 1) ])),
          TGrp(TTensorI(DFT(4, 1), 4, APar, AVec)) ]).withTags([ AVecReg(AVX_4x64f) ]),
        TGrp_tag( TGrp(TCompose([ TTensorI(DFT(4, 1), 4, AVec, AVec),
                  TTwiddle(16, 4, 1) ])).withTags([ AVecReg(AVX_4x64f) ]),
          TCompose_tag( TCompose([ TTensorI(DFT(4, 1), 4, AVec, AVec),
                  TTwiddle(16, 4, 1) ]).withTags([ AVecReg(AVX_4x64f) ]),
            AxI_vec( TTensorI(DFT(4, 1), 4, AVec, AVec).withTags([ AVecReg(AVX_4x64f) ]),
              @_Base( DPWrapper(DFT(4, 1), VWrap(AVX_4x64f)),
                DFT_CT( DFT(4, 1),
                  DFT_Base( DFT(2, 1) ),
                  DFT_Base( DFT(2, 1) ) ) ) ),
            TTwiddle_Tw1( TTwiddle(16, 4, 1).withTags([ AVecReg(AVX_4x64f) ]) ) ) ),
        TGrp_tag( TGrp(TTensorI(DFT(4, 1), 4, APar, AVec)).withTags([ AVecReg(AVX_4x64f) ]),
          IxA_L_vec( TTensorI(DFT(4, 1), 4, APar, AVec).withTags([ AVecReg(AVX_4x64f) ]),
            @_Base( DPWrapper(TL(16, 4, 1, 1).withTags([ AVecReg(AVX_4x64f) ]), VWrapId),
              IxLxI_kmn_n( TL(16, 4, 1, 1).withTags([ AVecReg(AVX_4x64f) ]),
                SIMD_ISA_Bases2( TL(8, 4, 1, 2).withTags([ AVecReg(AVX_4x64f) ]) ),
                SIMD_ISA_Bases1( TL(8, 4, 2, 1).withTags([ AVecReg(AVX_4x64f) ]) ) ) ),
            @_Base( DPWrapper(DFT(4, 1), VWrap(AVX_4x64f)),
              DFT_CT( DFT(4, 1),
                DFT_Base( DFT(2, 1) ),
                DFT_Base( DFT(2, 1) ) ) ) ) ) ) ) ) )
```

SIMD Vectorization by Ruletree Expansion

$$\underbrace{\left(\overline{\mathbf{DFT}_{mn}}\right)}_{\mathrm{vec}(\nu)} \rightarrow \underbrace{\left(\overline{(\mathbf{DFT}_m \otimes \mathrm{I}_n)\mathrm{T}_n^{mn}(\mathrm{I}_m \otimes \mathbf{DFT}_n)\,\mathrm{L}_m^{mn}}\right)}_{\mathrm{vec}(\nu)}$$

$$\dots$$

$$\rightarrow \overleftarrow{\underbrace{(\mathrm{DFT}_m \otimes \mathrm{I}_n)}_{\mathrm{vec}(\nu)}}^{\nu} \overline{\underbrace{(\mathrm{T}_n^{mn})}_{\mathrm{vec}(\nu)}}^{\nu} \overrightarrow{\underbrace{(\mathrm{I}_m \otimes \mathrm{DFT}_n)\,\mathrm{L}_m^{mn}}_{\mathrm{vec}(\nu)}}^{\nu}$$

$$\dots$$

$$\rightarrow (\mathrm{I}_{mn/\nu} \otimes \underbrace{\mathrm{L}_\nu^{2\nu}}_{\mathrm{sse}})(\overline{\mathrm{DFT}_m \otimes \mathrm{I}_{n/\nu} \vec{\otimes} \mathrm{I}_\nu})\overline{(\underbrace{\mathrm{T}_n^{mn}}_{\mathrm{sse}})}^{\nu}$$
$$\left(\mathrm{I}_{m/\nu} \otimes (\overline{\mathrm{L}_\nu^n} \vec{\otimes} \mathrm{I}_\nu)(\mathrm{I}_{n/\nu} \otimes (\mathrm{L}_\nu^{2\nu} \vec{\otimes} \mathrm{I}_\nu)(\mathrm{I}_2 \otimes \underbrace{\mathrm{L}_\nu^{\nu^2}}_{\mathrm{sse}})(\mathrm{L}_2^{2\nu} \vec{\otimes} \mathrm{I}_\nu))(\overline{\mathrm{DFT}_n} \vec{\otimes} \mathrm{I}_\nu)\right)$$
$$\left((\mathrm{L}_m^{mn} \otimes \mathrm{I}_2) \vec{\otimes} \mathrm{I}_\nu\right)(\mathrm{I}_{mn/\nu} \otimes \underbrace{\mathrm{L}_2^{2\nu}}_{\mathrm{sse}})$$

tensor products

base cases

Formula is vectorized w.r.t. Definition

SMP Parallelization by Ruletree Expansion

$$\underbrace{\mathbf{DFT}_{mn}}_{\mathsf{smp}(p,\mu)} \rightarrow \underbrace{\left((\mathbf{DFT}_m \otimes \mathrm{I}_n)\mathsf{T}_n^{mn}(\mathrm{I}_m \otimes \mathbf{DFT}_n)\,\mathsf{L}_m^{mn}\right)}_{\mathsf{smp}(p,\mu)}$$

$$\dots$$

$$\rightarrow \underbrace{\left(\mathbf{DFT}_m \otimes \mathrm{I}_n\right)}_{\mathsf{smp}(p,\mu)} \underbrace{\mathsf{T}_n^{mn}}_{\mathsf{smp}(p,\mu)} \underbrace{\left(\mathrm{I}_m \otimes \mathbf{DFT}_n\right)}_{\mathsf{smp}(p,\mu)} \underbrace{\mathsf{L}_m^{nm}}_{\mathsf{smp}(p,\mu)}$$

$$\dots$$

$$\rightarrow \left((\mathsf{L}_m^{mp} \otimes \mathrm{I}_{n/p\mu})\bar{\otimes}\,\mathrm{I}_\mu\right)\left(\mathrm{I}_p \otimes_{\|} (\mathbf{DFT}_m \otimes \mathrm{I}_{n/p})\right)\left((\mathsf{L}_p^{mp} \otimes \mathrm{I}_{n/p\mu})\bar{\otimes}\,\mathrm{I}_\mu\right)$$

$$\left(\bigoplus_{i=0}^{p-1}{}_{\|}\, \mathsf{T}_n^{mn,i}\right)\left(\mathrm{I}_p \otimes_{\|} (\mathrm{I}_{m/p} \otimes \mathbf{DFT}_n)\right)\left(\mathrm{I}_p \otimes_{\|} \mathsf{L}_{m/p}^{mn/p}\right)\left((\mathsf{L}_p^{pn} \otimes \mathrm{I}_{m/p\mu})\bar{\otimes}\,\mathrm{I}_\mu\right)$$

Fully optimized (load-balanced, no false sharing)
in the sense of our definition

Targeting Advanced Hardware

Simple Example: Multicore/OpenMP

```
opts := IAGlobals.getOpts(rec(dataType := T_Real(64)),
    rec(numproc := 2, api := "OpenMP", OmpMode := "for"));
Add(opts.breakdownRules.TRC, TRC_tag);        # needs some cleanup
opts.tags := opts.tags{[1]};                  # needs some cleanup

t := TRC(DFT(4)).withTags(opts.tags);         # need to add TRC(.)
rt := RandomRuleTree(t, opts);
c := CodeRuleTree(rt, opts);
PrintCode("DFT4_OMP", c, opts);
```

Stepwise code generation

```
opts.tags;                          # what are the tags
spl := SPLRuleTree(rt);             # There are SMP SPL objects
s := SumsRuleTree(rt, opts);        # and a SMP ISum

opts.unparser := OpenMP_Unparser;   # now we use parallel region
PrintCode("DFT4_OMP", c, opts);
```

Vectorization Rules: Simple Vectorization

Example Vectorization Rule

```
# spiral-core\namespaces\spiral\paradigms\vector\breakdown.gi
NewRulesFor(TTensorI, rec(
AxI_vec := rec(
    forTransposition := false,
    applicable := nt -> nt.hasTags() and IsVecVec(nt.params) and
        (nt.isTag(1,AVecReg) or nt.isTag(1,AVecRegCx)) and
        IsInt(nt.params[2]/nt.firstTag().v),
    children := nt -> let(r := nt.params[2] / nt.firstTag().v,
            isa := nt.firstTag().isa,
            [[ When(r = 1,
                When(nt.numTags() = 1,
                    nt.params[1].setWrap(VWrap(isa)),
                    nt.params[1].setWrap(
                        Drop(nt.getTags(), 1)).setWrap(VWrap(isa))
                ),
                TTensorI(nt.params[1].setWrap(VWrap(isa)), r,
                    AVec, AVec).withTags(Drop(nt.getTags(), 1))
            )]]
        ),
        apply := (nt, c, cnt) -> VTensor(c[1], nt.firstTag().v)
    )
));
```

Vectorization Rules: Formula Rewrite

Kronecker Commute

```
# spiral-core\namespaces\spiral\paradigms\vector\breakdown.gi
NewRulesFor(TTensorI, rec(
    IxA_vec := rec(forTransposition := false,
    applicable := nt -> IsParPar(nt.params) and nt.hasTags() and
        (nt.isTag(1,AVecReg) or nt.isTag(1,AVecRegCx)) and
        IsInt(nt.params[2]/nt.firstTag().v),
    children := nt -> let(pv := nt.getTags(), v := pv[1].v,
        isa := pv[1].isa, d := nt.params[1].dims(),
        [[
            TL(d[1]*v, v, 1, 1).withTags(pv).setWrap(VWrapId),
            When(Length(pv)=1, nt.params[1].setWrap(VWrap(isa)),
                nt.params[1].setpv(Drop(pv, 1)).setWrap(VWrap(isa))),
            TL(d[2]*v, d[2], 1, 1).withTags(pv).setWrap(VWrapId)
        ]]),
    apply := (nt, c, cnt) -> let(
        l := nt.params[2] / nt.firstTag().v,
        A := c[1] * VTensor(c[2], nt.firstTag().v) * c[3],
        NoDiagPullin(When(l=1, A , Tensor(I(l), A))))
    )
));
```

Search with Timing Context: Wrapping

Needed in Context of Search

```
# spiral-core\namespaces\spiral\paradigms\vector\vwrap.gi
Class(VWrap, VWrapBase, rec(
    __call__ := (self,isa) >> Checked(IsSIMD_ISA(isa),
        WithBases(self, rec(operations:=PrintOps, isa:=isa))),
    wrap := (self,r,t,opts) >> let(
        isa := self.isa, v := isa.v,
        tt := When(t.isReal(),
            @_Base(paradigms.vector.sigmaspl.VTensor(r.node, v), r),
                paradigms.vector.breakdown.AxI_vec(
                    TTensorI(TRC(t).withTags([AVecReg(isa)]), v,
                        AVec, AVec).withTags([AVecReg(isa)]),
                paradigms.vector.breakdown.TRC_VRCLR(
                    TRC(t).withTags([AVecReg(isa)]), r))),
    print := self >> Print(self.name, "(", self.isa, ")"),
));
```

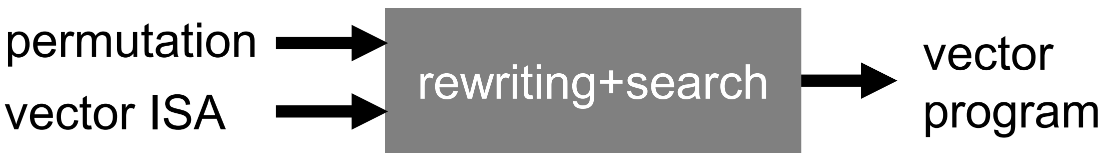
VWrap Transformation, used in DP to time sub-trees

```
opts := SIMDGlobals.getOpts(AVX_4x64f);
t := DFT(2).setWrap(VWrap(AVX_4x64f));
rt := RandomRuleTree(t, opts);
wrt := t.wrap.wrap(rt, t, opts);
SPLRuleTree(wrt);
```

Organization

- Overview
- System
- Top level commands
- Abstractions
- **Rewriting System I: RuleTree/backtracking search**
  Stride permutation vectorization
- Rewriting System II: Visitor Patterns
- Rewriting System III: Associative/large context rules
- Basic block compiler

Automatically Deriving Vector Base Cases



- **Translate SIMD vector ISA into matrix representation**
- **Design rule system to generate *vector matrix formulas***
- **Define cost measure on matrix formulas**
- **Use dynamic programming with backtracking to find vector program with minimal cost**

Vector matrix formula in BNF

$$\begin{aligned}\langle\text{vmf}\rangle \quad ::= \quad & \langle\text{vmf}\rangle\,\langle\text{vmf}\rangle \mid \mathrm{I}_m \otimes \langle\text{vmf}\rangle \mid \begin{pmatrix}\langle\text{vmf}\rangle\\ \langle\text{vmf}\rangle\end{pmatrix} \mid \langle\text{perm}\rangle \otimes \mathrm{I}_\nu \mid \\ & \langle\text{perm}\rangle \otimes \mathrm{I}_{\nu/2} \;\text{ if }\; \mathbf{L}^4_2 \otimes \mathrm{I}_{\nu/2} \;\text{ possible} \mid M_{\texttt{instr}} \text{ with } \texttt{instr} \text{ in ISA} \\ \langle\text{perm}\rangle \quad ::= \quad & \mathbf{L}^{mn}_m \mid \mathrm{I}_m \otimes \langle\text{perm}\rangle \mid \langle\text{perm}\rangle \otimes \mathrm{I}_m \mid \langle\text{perm}\rangle\langle\text{perm}\rangle\end{aligned}$$

Translating Instructions into Matrices

Intel C++ Compiler Manual

```
__m128 _mm_unpackhi_ps(__m128 a, __m128 b)
r0 := a2; r1 := b2; r2 := a3; r3 := b3
```

Instruction specification (GAP code)

```
Intel_SSE2.4_x_float._mm_unpackhi_ps := rec(
   v := 4,
   semantics := (a, b, p) -> [a[2], b[2], a[3], b[3]],
   parameters := []
);
```

SSE instruction as matrix

```
__m128 t, x0, x1;
t = _mm_unpackhi_ps(x0, x1);
```

➡

$$\vec{t} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \vec{x}_0 \\ \vec{x}_1 \end{bmatrix}$$

Automatically build matrix from `semantics()` function

Example: Sequence of Two Instructions

Instruction set: Intel SSE 4-way float

```
y = _mm_unpacklo_ps(x0, x1);
```

$$\begin{bmatrix} 1&0&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&0&0&0&1&0&0 \end{bmatrix}$$

```
y = _mm_shuffle_ps(x0, x1,
        _MM_SHUFFLE(1,2,1,2));
```

$$\begin{bmatrix} 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0 \end{bmatrix}$$

```
y = _mm_shuffle_ps(x0, x1,
        _MM_SHUFFLE(3,4,3,4));
```

$$\begin{bmatrix} 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&1 \end{bmatrix}$$

Translating a vector matrix into a instruction sequence

$$L_2^4 \otimes I_2 = \left[\begin{array}{cccccccc} 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0\\ \hline 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&1 \end{array}\right]$$

```
// __m128 *y, *x
Y[0] = _mm_shuffle_ps(x0, x1,
         _MM_SHUFFLE(1,2,1,2));
Y[1] = _mm_shuffle_ps(x0, x1,
         _MM_SHUFFLE(3,4,3,4));
```

Rule System: Recursive Matrix Factorization

- Recursively factorizes stride permutations
- "Blocking of matrix transposition" in linear memory
- Choices -> Dynamic programming with backtracking
- Trigger ISA-specific termination rules

Start: stride permutation

$$\mathrm{L}_m^{mn} \rightarrow \mathrm{I}_1 \otimes \mathrm{L}_m^{mn} \otimes \mathrm{I}_1$$

$$\mathrm{I}_\ell \otimes \mathrm{L}_n^{kmn} \otimes \mathrm{I}_r \rightarrow \left(\mathrm{I}_\ell \otimes \mathrm{L}_n^{kn} \otimes \mathrm{I}_{mr}\right)\left(\mathrm{I}_{\ell k} \otimes \mathrm{L}_n^{mn} \otimes \mathrm{I}_r\right)$$

$$\mathrm{I}_\ell \otimes \mathrm{L}_n^{kmn} \otimes \mathrm{I}_r \rightarrow \left(\mathrm{I}_\ell \otimes \mathrm{L}_{kn}^{kmn} \otimes \mathrm{I}_r\right)\left(\mathrm{I}_\ell \otimes \mathrm{L}_{mn}^{kmn} \otimes \mathrm{I}_r\right)$$

$$\mathrm{I}_\ell \otimes \mathrm{L}_{km}^{kmn} \otimes \mathrm{I}_r \rightarrow \left(\mathrm{I}_{k\ell} \otimes \mathrm{L}_m^{mn} \otimes \mathrm{I}_r\right)\left(\mathrm{I}_\ell \otimes \mathrm{L}_k^{kn} \otimes \mathrm{I}_m\right)$$

$$\mathrm{I}_\ell \otimes \mathrm{L}_{km}^{kmn} \otimes \mathrm{I}_r \rightarrow \left(\mathrm{I}_\ell \otimes \mathrm{L}_k^{kmn} \otimes \mathrm{I}_r\right)\left(\mathrm{I}_\ell \otimes \mathrm{L}_m^{kmn} \otimes \mathrm{I}_r\right)$$

$$\mathrm{I}_{k\ell} \otimes \mathrm{L}_m^{mn} \otimes \mathrm{I}_r \rightarrow \mathrm{I}_k \otimes \left(\mathrm{I}_\ell \otimes \mathrm{L}_m^{mn} \otimes \mathrm{I}_r\right) \quad \text{if } \ell mnr \in \{\nu, 2\nu\}$$

Choice: factorize kmn

Triggers termination rules

Cost Function: Weighted Instruction Count

- **Defines recursive cost function for matrix formulas**
- **Each instruction has an associated cost**
- **Vector assignments are "for free"**

$$
\begin{aligned}
\mathrm{Cost}_{\mathrm{ISA},\nu}(P) &= \infty, \quad P \text{ not a } \langle \mathrm{vmf} \rangle \\
\mathrm{Cost}_{\mathrm{ISA},\nu}(M^{\nu}_{\mathtt{instr}}) &= c_{\mathtt{instr}} \\
\mathrm{Cost}_{\mathrm{ISA},\nu}(P \otimes \mathrm{I}_\nu) &= 0, \quad P \text{ permutation} \\
\mathrm{Cost}_{\mathrm{ISA},\nu}\left(P \otimes \mathrm{I}_{\nu/2}\right) &= \lfloor n/2 \rfloor c_{\mathtt{i1}} + \lceil n/2 \rceil c_{\mathtt{i2}}, \quad P\ 2n \times 2n \text{ permutation} \\
\mathrm{Cost}_{\mathrm{ISA},\nu}(A\,B) &= \mathrm{Cost}_{\mathrm{ISA},\nu}(A) + \mathrm{Cost}_{\mathrm{ISA},\nu}(B) \\
\mathrm{Cost}_{\mathrm{ISA},\nu}\left(\binom{A}{B}\right) &= \mathrm{Cost}_{\mathrm{ISA},\nu}(A) + \mathrm{Cost}_{\mathrm{ISA},\nu}(B) \\
\mathrm{Cost}_{\mathrm{ISA},\nu}(\mathrm{I}_m \otimes A) &= m\,\mathrm{Cost}_{\mathrm{ISA},\nu}(A)
\end{aligned}
$$

Vector Program: 8-way Vectorized L_8^{64}

$$L_8^{64} = \left(I_4 \otimes (L_2^4 \otimes I_4)\right)\left(L_4^8 \otimes I_8\right)\left(I_4 \otimes (L_4^8 \otimes I_2)\right)\left((I_2 \otimes L_2^4) \otimes I_8\right)\left(I_4 \otimes L_8^{16}\right)$$

```
__m128 X[8], Y[8], t3, t4, t7, t8, t11, t12, t15, t16,
       t17, t18, t19, t20, t21, t22, t23, t24;
t3 = _mm_unpacklo_epi16(X[0], X[1]); t4 = _mm_unpackhi_epi16(X[0], X[1]);
t7 = _mm_unpacklo_epi16(X[2], X[3]); t8 = _mm_unpackhi_epi16(X[2], X[3]);
t11 = _mm_unpacklo_epi16(X[4], X[5]);t12 = _mm_unpackhi_epi16(X[4], X[5]);
t15 = _mm_unpacklo_epi16(X[6], X[7]);t16 = _mm_unpackhi_epi16(X[6], X[7]);
t17 = _mm_unpacklo_epi32(t3, t7);     t18 = _mm_unpackhi_epi32(t3, t7);
t19 = _mm_unpacklo_epi32(t4, t8);     t20 = _mm_unpackhi_epi32(t4, t8);
t21 = _mm_unpacklo_epi32(t11, t15);   t22 = _mm_unpackhi_epi32(t11, t15);
t23 = _mm_unpacklo_epi32(t12, t16);   t24 = _mm_unpackhi_epi32(t12, t16);
Y[0] = _mm_unpacklo_epi64(t17, t21); Y[1] = _mm_unpackhi_epi64(t17, t21);
Y[2] = _mm_unpacklo_epi64(t18, t22); Y[3] = _mm_unpackhi_epi64(t18, t22);
Y[4] = _mm_unpacklo_epi64(t19, t23); Y[5] = _mm_unpackhi_epi64(t19, t23);
Y[6] = _mm_unpacklo_epi64(t20, t24); Y[7] = _mm_unpackhi_epi64(t20, t24);
```

8-way vectorized transposition of 8x8 matrix

Special Role of Stride Permutations

TL: Lift Stride Permutation to Non-Terminal Level

```
# spiral-core\namespaces\spiral\paradigms\common\nonterms.gi
Class(TL, Tagged_tSPL_Container, rec(
    abbrevs :=  [ (size, stride) -> Checked(ForAll([size, stride],
        IsPosIntSym), [size, stride, 1, 1]),
            (size, stride, left, right) ->
        Checked(ForAll([size, stride, left, right], IsPosIntSym),
            [size, stride, left, right]) ],
    __call__ := arg >> let(self := arg[1], args := Drop(arg, 1),
        Cond(args=[1,1,1,1], I(1), ApplyFunc(Inherited,args))),
    dims := self >>
        Replicate(2, self.params[1]*self.params[3]*self.params[4]),
    terminate := self >> Tensor(I(self.params[3]),
        L(self.params[1], self.params[2]), I(self.params[4])),
    transpose := self >> TL(self.params[1],
        self.params[1]/self.params[2], self.params[3],
        self.params[4]).withTags(self.getTags()),
));
```

VWrap Transformation, used in DP to time sub-trees

```
t := TL(8,2,2,4);
t.terminate();
```

Architecture Specific Permutations

Looking up vectorized Implementations for TL

```
Import(paradigms.vector.sigmaspl);
opts := SIMDGlobals.getOpts(AVX_4x64f);
opts.breakdownRules.TL;
t := TL(16,4,1,1).withTags(opts.tags);       # a in-register perm
rt := RandomRuleTree(t, opts);
HashLookup(opts.baseHashes[1], t);    # the implementation is cached

s := SPLRuleTree(rt);
vp := Collect(s, VPerm)[1];           # SPL object carries its code
vp.code;                              # code generator refers to ISA
AVX_4x64f.rules;                      # ISA carries implementations
PrintCode("", vp.code(Y, X), opts);   # for in-register perms
```

SIMD ISA Database and bootstrapping a vector architecture

```
SIMD_ISA_DB;                   # central SIMD data base
SIMD_ISA_DB.installed();       # all the ISAs supported
Doc(AVX_4x64f);                # The ISA carries all the relevant info
Print(SIMD_ISA_DB.buildBases);        # How the base cases are built
AVX_4x64f.buildRules;          # bootstrapping function
SIMD_ISA_DB.getBases(AVX_4x64f);      # all the base cases needed
SIMD_ISA_DB.lookupBases(AVX_4x64f);   # and how they are implemented
```

ISA Database and Hashed Breakdowns

Generic Breakdown Rules to look up architecture specific code

```
# spiral-core\namespaces\spiral\paradigms\vector\bases\tl_bases.gi
NewRulesFor(TL, rec(
    SIMD_ISA_Bases1 := rec(
        forTransposition := false,
        applicable := (self, t) >> t.isTag(1, AVecReg) and
            let(isa := t.firstTag().isa, P:=t.params,
                isa.active and ForAny(isa.supportedTL(),
                e -> _TL_applicable(e, P[1], P[2], P[3], P[4]))),
        apply := function(nt,C,cnt)
            local isa, tl, ll, vprm, P;
            P:=nt.params;
            isa := nt.firstTag().isa;
            tl := isa.getTL(P);
            ll := P[3] / tl.perm.l;
            vprm := tl.vperm;
            return When(ll = 1, vprm,
                BlockVPerm(ll, isa.v, vprm, tl.perm.spl));
        end,
    )
));
```

Stride Permutatiopn Identities as Rules

Generic Breakdown Rules to look up architecture specific code

```
# spiral-core\namespaces\spiral\paradigms\common\breakdown.gi
NewRulesFor(TL, rec(
    IxLxI_kmn_n := rec (forTransposition := false,
        applicable := nt ->
            Length(DivisorsIntDrop(nt.params[1]/nt.params[2])) > 0,
        children := nt -> let(
            N := nt.params[1], n := nt.params[2],
            km := N/n, ml := DivisorsIntDrop(km),
            l := nt.params[3], r := nt.params[4],
            List(ml, m -> let( k := km/m, [
                    TL(k*n, n, l, r*m).withTags(nt.getTags()),
                    TL(m*n, n, k*l, r).withTags(nt.getTags())
                ]))
            ),
        apply := (nt, c, cnt) -> let(
            spl := c[1] * c[2],
            When(nt.params[1] = nt.params[2]^2,
                SymSPL(spl),
                spl
            )
        )
));
```

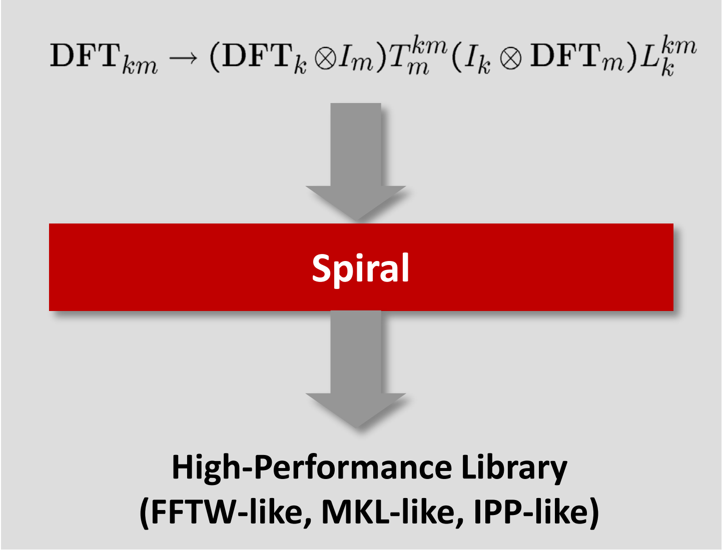
Organization

- Overview
- System
- Top level commands
- Abstractions
- **Rewriting System I: RuleTree/backtracking search**
  GT+XChain loop parallelization/vectorization
- Rewriting System II: Visitor Patterns
- Rewriting System III: Associative/large context rules
- Basic block compiler

General Size Library Generation

Input:

- **Transform**: DFT_n
- **Algorithms**: $\mathrm{DFT}_{km} \rightarrow (\mathrm{DFT}_k \otimes I_m) T^{km}_m (I_k \otimes \mathrm{DFT}_m) L^{km}_k$
  $\mathrm{DFT}_2 \rightarrow \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
- **Vectorization**: 2-way SSE
- **Threading**: Yes



Output:

- Optimized library (10,000 lines of C++)
- For general input size
  (**not** collection of fixed sizes)
- Vectorized
- Multithreaded
- With runtime adaptation mechanism
- Performance competitive with hand-written code

Yevgen Voronenko, Frédéric de Mesmay and Markus Püschel
Computer Generation of General Size Linear Transform Libraries
Proc. International Symposium on Code Generation and Optimization (CGO), pp. 102-113, 2009

GT Breakdown Rules

Convert TTensorI to GT

```
TTensorI_toGT := rec(
    applicable := t -> true,
    freedoms := t -> [],
    child := (t, fr) -> [ GT_TTensorI(t) ],
    apply := (t, C, Nonterms) -> C[1]
);
```

Helper Functions

```
# spiral-core\namespaces\spiral\paradigms\common\gt.gi
GTVec := XChain([0,1]);
GTPar := XChain([1,0]);

GT_TTensorI := function(tt)
    local spl, g, s, v, tags;
    [spl,v,s,g] := tt.params;
    tags := tt.getTags();
    g := When(g=AVec, GTVec, GTPar);
    s := When(s=AVec, GTVec, GTPar);
    return GT(spl, g, s, [v]).withTags(tags);
end;
```

GT Parallelization Rule

Very dense—handles many cases and options

```
# spiral-core\namespaces\spiral\paradigms\common\breakdown.gi
GT_Par := rec(requiredFirstTag:=AParSMP,
applicable := (self, t) >> let(rank := Length(t.params[4]),
    nthreads:=t.firstTag().params[1],rank=1 and let(its:=t.params[4][1],
    PatternMatch(t, [GT,@(1),@(2,XChain),@(3,XChain),...],empty_cx())
    and IsPosInt(its/nthreads))),
children := (self, t) >> let(spl := t.params[1], g := t.params[2],
    s := t.params[3], its := t.params[4][1],
    nthreads := t.firstTag().params[1], tags := Drop(t.getTags(), 1),
    [[ GT(spl,g,s,[its / nthreads]).withTags(tags), InfoNt(nthreads) ],
     [ GT(spl,XChain([0]),XChain([0]),[]).withTags(tags),InfoNt(its) ]],
apply := (self, t, C, Nonterms) >> let(
    spl := t.params[1], N := Minimum(spl.dimensions),
    g := t.params[2], s := t.params[3], its := t.params[4][1],
    gg := When(g.params[1]=[0,1], XChain([0,1,2]), XChain([1,2,0])),
    ss := When(s.params[1]=[0,1], XChain([0,1,2]), XChain([1,2,0])),
    nthreads := t.firstTag().params[1],tid:=t.firstTag().params[2],
    par_its:=When(IsBound(Nonterms[2]),Nonterms[2].params[1],nthreads),
    i := Ind(par_its), SMPBarrier(nthreads, tid,
    SMPSum(nthreads, tid, i, par_its,
        Scat(ss.part(1, i, Rows(spl), [par_its, its/par_its])) *
        C[1]*Gath(gg.part(1, i, Cols(spl), [par_its, its/par_its])))))
)
```

Organization

- Overview
- System
- Top level commands
- Abstractions
- Rewriting System I: RuleTree/backtracking search
- **Rewriting System II: Visitor Patterns**
  Standard code generation
- Rewriting System III: Associative/large context rules
- Basic block compiler

Translating an SPL Expression Into Code

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

Output =

Ruletree, expanded into

See Figure 5

SPL Expression:

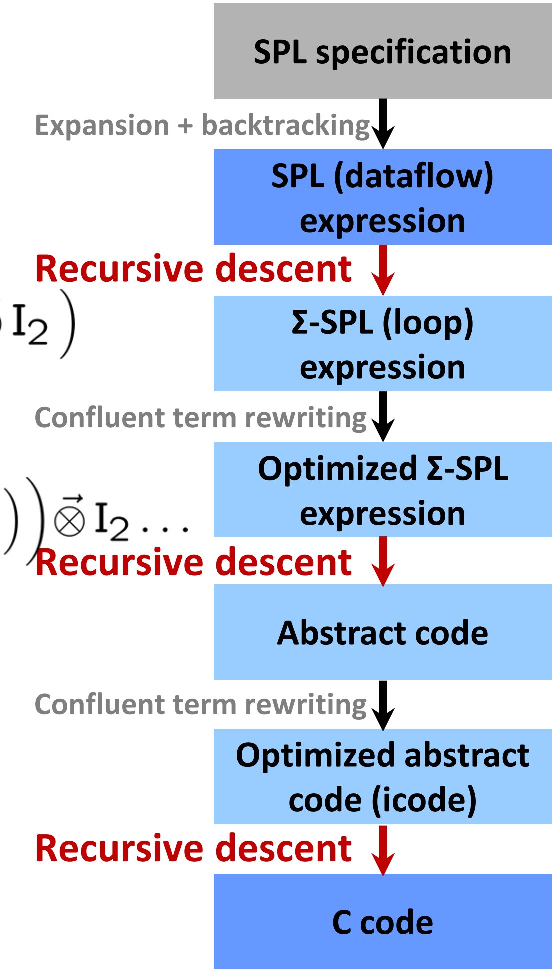
$$\left(\left((F_2\otimes I_2)T^4_2(I_2\otimes F_2)L^4_2\vec{\otimes} I_2\right)\underbrace{T^8_2}_{\mathrm{vec}(2)}\left(I_2\otimes\underbrace{L^4_2}_{\mathrm{vec}(2)}(F_2\vec{\otimes} I_2)\right)\left(L^4_2\vec{\otimes} I_2\right)\right.$$

Σ-SPL:

$$\left(\sum_{j=0}^{1}\left(S_{\imath_2\otimes(j)_2}F_2\mathrm{Diag}^2_{x\mapsto\omega_4^{2i+j}x}G_{\imath_2\otimes(j)_2}\right)\sum_{j=0}^{1}\left(S_{(j)_2\otimes\imath_2}F_2G_{\imath_2\otimes(j)_2}\right)\right)\vec{\otimes} I_2\dots$$

C Code:

```
void dft8(_Complex double *Y, _Complex double *X) {
    __m256d s38, s39, s40, s41,...
    __m256d  *a17, *a18;
    a17 = ((__m256d  *) X);
    s38 = *(a17);
    s39 = *((a17 + 2));
    t38 = _mm256_add_pd(s38, s39);
    t39 = _mm256_sub_pd(s38, s39);
    ...
    s52 = _mm256_sub_pd(s45, s50);
    *((a18 + 3)) = s52;
}
```



Implementing Recursive Descent: Visitors

Simple example

```
# spiral-core\namespaces\spiral\rewrite\visitor.gi
Class(LispGen, Visitor, rec(
    add := (self, o) >> Print("(+ ", self(o.args[1]), " ",
        self(o.args[2]), ")"),
    mul := (self, o) >> Print("(* ", self(o.args[1]), " ",
        self(o.args[2]), ")"),
    sub := (self, o) >> Print("(- ", self(o.args[1]), " ",
        self(o.args[2]), ")"),
    var := (self, o) >> Print("(var ", o.id, ")"),
    Value := (self, o) >> Print("(value ", o.v, ")")
));
LispGen(4*X+2);
```

Visitors used in standard translation flow

```
opts := SpiralDefaults;
opts.sumsgen;
DefaultSumsGen;
opts.codegen;
DefaultCodegen;
opts.unparser;
CUnparser;
```

SumsGen

The DefaultSumsGen Visitor

```
# spiral-core\namespaces\spiral\sigma\sumsgen.gi
# all the fields
Filtered(RecFields(DefaultSumsGen), i->not IsSystemRecField(i));
Print(DefaultSumsGen.__call__);

# the recursive definitions needed for DFT
DefaultSumsGen.Compose;
Print(DefaultSumsGen.Tensor);
DefaultSumsGen.I;
DefaultSumsGen.F;
DefaultSumsGen.Diag;
DefaultSumsGen.L;
```

Visitors used in standard translation flow

```
opts := SpiralDefaults;
s := SPLRuleTree(RandomRuleTree(DFT(8), opts));
SumsSPL(s, opts);
opts.sumsgen(s, opts);

# legacy and backwards compatibility framework
F(2).sums();
Tensor(F(2), I(2)).sums();
```

CodeGen

The DefaultCodegen Visitor

```
# spiral-core\namespaces\spiral\compiler\codegen.gi
# all the fields
Filtered(RecFields(DefaultCodegen), i->not IsSystemRecField(i));
Print(DefaultCodegen.__call__);

# Some of the fields
Print(DefaultCodegen.Formula);
Print(DefaultCodegen.Compose);
DefaultCodegen.ISum;
Print(DefaultCodegen.Gath);
Print(DefaultCodegen.Scat);
DefaultCodegen.Diag;
```

Visitors used in standard translation flow

```
opts := SpiralDefaults;
s := SumsRuleTree(RandomRuleTree(DFT(8), opts), opts);
# only translate Sigma-SPL to icode
opts.codegen(s, Y, X, opts);
# also invoke the basic block compiler
opts.codegen(Formula(s), Y, X, opts);
```

C Pretty Printer

The CUnparser Visitor

```
# spiral-core\namespaces\spiral\compiler\unparse.gi
# all the fields
Filtered(RecFields(CUnparser), i->not IsSystemRecField(i));
Filtered(RecFields(CUnparserBase), i->not IsSystemRecField(i));
Print(CUnparser.gen);

# Some of the fields
Print(CUnparser.loop);
CUnparser.deref;
CUnparser.add;
CUnparser.Value;
Print(CUnparser.decl);
CUnparser.chain;
```

Visitors used in standard translation flow

```
opts := SpiralDefaults;
c := CodeRuleTree(RandomRuleTree(DFT(8), opts), opts);
# Print full header etc.
PrintCode("dft8", c, opts);
# unparser needs opts as context
Unparse(c.cmds[1].cmds[2].cmd,
    CopyFields(CUnparser, rec(opts:=opts)), 0, 1);
```

Organization

- Overview
- System
- Top level commands
- Abstractions
- Rewriting System I: RuleTree/backtracking search
- **Rewriting System II: Visitor Patterns**
  Targeting special hardware
- Rewriting System III: Associative/large context rules
- Basic block compiler

SMP Code Objects and Code Generator

Thread ID

```
# spiral-core\namespaces\spiral\paradigms\smp\sigmaspl.gi
Class(threadId, Exp, rec(computeType := self >> TInt));
```

Barrier

```
Class(barrier, call, rec(visitAs := call));
```

SMP Codegenerator

```
Class(SMPCodegenMixin, Codegen, rec(
    SMPBarrier := (self, o, y, x, opts) >> chain(
        self(o.child(1), y, x, opts),
        barrier(o.nthreads, o.tid, "&GLOBAL_BARRIER")),
    SMPSum := (self, o, y, x, opts) >> let(
        outer_tid     := When(IsBound(opts._tid), opts._tid, 0),
        outer_num_thr := When(IsBound(opts._tid), opts._tid.range, 1),
        tid := var.fresh("tid", TInt, o.nthreads * outer_num_thr),
        smp_loop(o.nthreads, tid, (outer_tid * outer_num_thr) + o.tid,
            o.var, o.domain,
            self(o.child(1), y, x,
                CopyFields(opts, rec(_tid := tid))))
        )
));
```

OpenMP Unparser

Unparser Definition

```
# Unparser for #pragma omp parallel for
Class(OpenMP_Unparser, OpenMP_UnparseMixin_ParFor, CUnparserProg);
```

Unparser Parallel For Mixin

```
# spiral-core\namespaces\spiral\paradigms\smp\unparsed.gi
Class(OpenMP_UnparseMixin_ParFor, SMP_UnparseMixin, rec(
    includes := ["<omp.h>"],
    threadId := (self,o,i,is) >> Print("omp_get_thread_num()"),
    barrier := (self,o,i, is) >>
        Print(Blanks(i), "/* SMP barrier */\n"),
    smp_loop := (self,o,i,is) >> let(v := o.var,
        lo := 0, hi := o.range,
        Print(Blanks(i),
        "#pragma omp parallel for schedule(static, ",
        Int((hi+1)/2),")\n",
        Blanks(i), "for(int ", v, " = ", lo, "; ", v,
        " < ", hi, "; ", v, "++) {\n",
        Blanks(i + is), "int ", o.tidvar, " = ", v, ";\n",
        self.opts.unparser(o.cmd,i+is,is),
        Blanks(i), "}\n")),
));
```

OpenMP Unparser

Unparser Definition

```
# Unparser for #pragma omp parallel regions
# spiral-core\namespaces\spiral\libgen\recgt.gi
Class(OpenMP_Unparser, OpenMP_UnparseMixin, CUnparserProg);
```

Unparser Parallel Region Mixin

```
# spiral-core\namespaces\spiral\paradigms\smp\unparsed.gi
Class(OpenMP_UnparseMixin, SMP_UnparseMixin, rec(
    includes := ["<omp.h>"],
    smp_fork := (self, o, i, is) >> Print(
        Blanks(i), "#pragma omp parallel num_threads(",
        o.nthreads, ")\n",
        Blanks(i), "{\n",
        self(o.cmd, i+is, is),
        Blanks(i), "}\n"
    ),
    threadId := (self,o,i,is) >> Print("omp_get_thread_num()"),
    barrier := (self,o,i, is) >> Print("#pragma omp barrier\n")
));
```

Vector SumsGen and Rewriting

Default Sumsgen Handles Vector SPL to Σ-SPL

```
opts := SIMDGlobals.getOpts(AVX_4x64f);
opts.sumsgen;
opts.sumsgen.VTensor;
opts.sumsgen.VPerm;
```

Rewrite Rule Strategies

```
opts.formulaStrategies;
opts.formulaStrategies.sigmaSpl;
opts.formulaStrategies.preRC;
opts.formulaStrategies.postProcess;
```

Compile Strategies

```
opts.compileStrategy;
Print(opts.vector.isa.fixProblems);
```

Vector CodeGen

Default Sumsgen Handles SPL to Σ-SPL

```
# spiral-core\namespaces\spiral\paradigms\vector\sigmaspl\codegen.gi
Class(VectorCodegen, DefaultCodegen, rec(
    VContainer := (self, o, y, x, opts) >>
        self(o.child(1), y, x, CopyFields(opts, rec(
            vector := rec(
                isa  := o.isa,
                SIMD := LocalConfig.cpuinfo.SIMDname )))),
    VPrm_x_I := (self, o, y, x, opts) >>
        self(VTensor(Prm(o.func), o.v), y, x, opts),
    VPerm := (self, o, y, x, opts) >> o.code(y, x),
    VTensor := (self, o, y, x, opts) >> let(
        CastToVect := p -> StripList(List(Flat([p]), e ->
        tcast(TPtr(TVect(opts.vector.isa.t.t, o.vlen)), e))),
        self(o.child(1), CastToVect(y), CastToVect(x), opts)),
    VGath := (self, o, y, x, opts) >> Cond(IsUnalignedPtrT(x.t),
        self(VGath_u(fTensor(o.func, fBase(o.v, 0)), o.v), y, x, opts),
        self(VTensor(Gath(o.func), o.v), y, x, opts)),
    VScat := (self, o, y, x, opts) >> Cond(IsUnalignedPtrT(y.t),
        self(VScat_u(fTensor(o.func, fBase(o.v, 0)), o.v), y, x, opts),
        self(VTensor(Scat(o.func), o.v), y, x, opts))
));
```

Vector Unparser

Polymorphic Unparsing for standard icode, adds special instructions

```
# spiral-core\namespaces\spiral\platforms\avx\unparse.gi
Class(AVXUnparser, SSEUnparser, rec(
    TVect := (self, t, vars, i, is) >> let(
        ctype := self.ctype(t, _isa(self)),
        Print(ctype, " ", self.infix(vars, ", "))),
    vpack := (self, o, i, is) >> Cond( _avxT(o.t, self.opts),
            Print("_mm256_set_", self.ctype_suffix(o.t, _isa(self)),
        "(", self.infix(Reversed(o.args), ", "), ")"),
        Inherited(o, i, is)),
    sub := (self, o, i, is) >> Cond( _avxT(o.t, self.opts), let(
        isa := _isa(self),
        sfx := self.ctype_suffix(o.t, isa),
        saturated := When(isa.isFixedPoint and
            isa.saturatedArithmetic, "s", ""),
            self.printf("_mm256_sub$1_$2($3, $4)", [saturated, sfx,
                o.args[1], o.args[2]])),
        Inherited(o, i, is)),
    vextract_2l_4x64f := (self, o, i, is) >>
        self.prefix("_mm256_extractf128_pd", o.args),
    vstore_2l_4x64f := (self, o, i, is) >>
        self.prefix("_mm256_extractf128_pd", o.args),
));
```

Organization

- Overview
- System
- Top level commands
- Abstractions
- Rewriting System I: RuleTree/backtracking search
- Rewriting System II: Visitor Patterns
- **Rewriting System III: Associative/large context rules**
- Basic block compiler

Translating an SPL Expression Into Code

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

Output =

Ruletree, expanded into

SPL Expression:

See Figure 5

$$\big((F_2\otimes I_2)T^4_2(I_2\otimes F_2)L^4_2\vec{\otimes} I_2\big)\ \underbrace{T^8_2}_{\mathrm{vec}(2)}\ \big(I_2\otimes \underbrace{L^4_2}_{\mathrm{vec}(2)}(F_2\vec{\otimes} I_2)\big)\big(L^4_2\vec{\otimes} I_2\big)$$

∑-SPL:

$$\Big(\sum_{j=0}^{1}\big(S_{\imath_2\otimes(j)_2}F_2\mathrm{Diag}^2_{x\mapsto\omega_4^{2i+j}x}G_{\imath_2\otimes(j)_2}\big)\sum_{j=0}^{1}\big(S_{(j)_2\otimes\imath_2}F_2G_{\imath_2\otimes(j)_2}\big)\Big)\vec{\otimes} I_2\ldots$$

C Code:

```
void dft8(_Complex double *Y, _Complex double *X) {
    __m256d s38, s39, s40, s41,...
    __m256d  *a17, *a18;
    a17 = ((__m256d  *) X);
    s38 = *(a17);
    s39 = *((a17 + 2));
    t38 = _mm256_add_pd(s38, s39);
    t39 = _mm256_sub_pd(s38, s39);
    ...
    s52 = _mm256_sub_pd(s45, s50);
    *((a18 + 3)) = s52;
}
```

SPL specification
↓ Expansion + backtracking
SPL (dataflow) expression
↓ Recursive descent
Σ-SPL (loop) expression
↓ Confluent term rewriting
Optimized Σ-SPL expression
↓ Recursive descent
Abstract code
↓ Confluent term rewriting
Optimized abstract code (icode)
↓ Recursive descent
C code

Problem: Fusing Permutations and Loops

$$y = (I_4 \otimes F_2) L^8_4 \, x$$

direct mapping

Two passes over the working set
Complex index computation

```
void sub(double *y, double *x) {
   double t[8];
   for (int i=0; i<=7; i++)
      t[(i/4)+2*(i%4)] = x[i];
   for (int i=0; i<4; i++){
      y[2*i] = t[2*i] + t[2*i+1];
      y[2*i+1] = t[2*i] - t[2*i+1];
   }
}
```

C compiler cannot do this

hardcoded special case

One pass over the working set
Simple index computation

```
void sub(double *y, double *x) {
   for (int j=0; j<=3; j++){
      y[2*j] = x[j] + x[j+4];
      y[2*j+1] = x[j] - x[j+4];
   }
}
```

State-of-the-art
SPIRAL: Hardcoded with templates
FFTW: Hardcoded in the infrastructure

How does hardcoding scale?

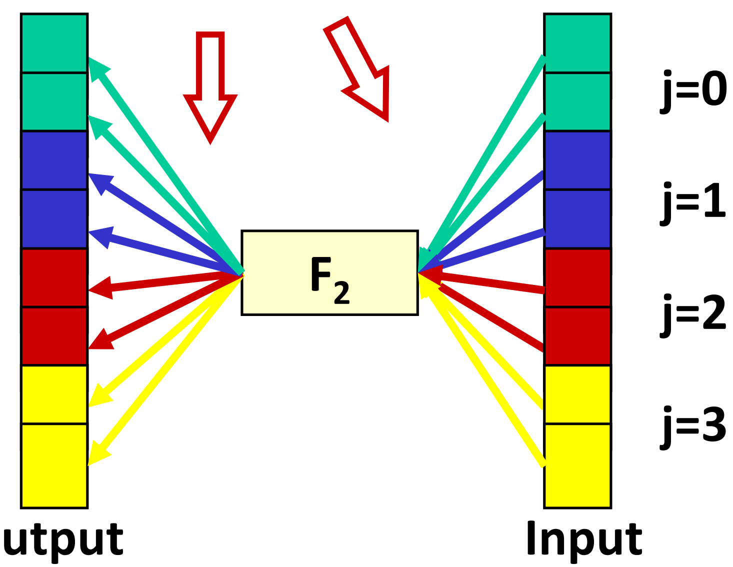
Σ-SPL

- **Four central constructs: S, G, S, Perm**
  - Σ (sum) – makes loops explicit
  - G_f (gather) – reads data using the index mapping f
  - S_f (scatter) – writes data using the index mapping f
  - $Perm_f$ – permutes data using the index mapping f

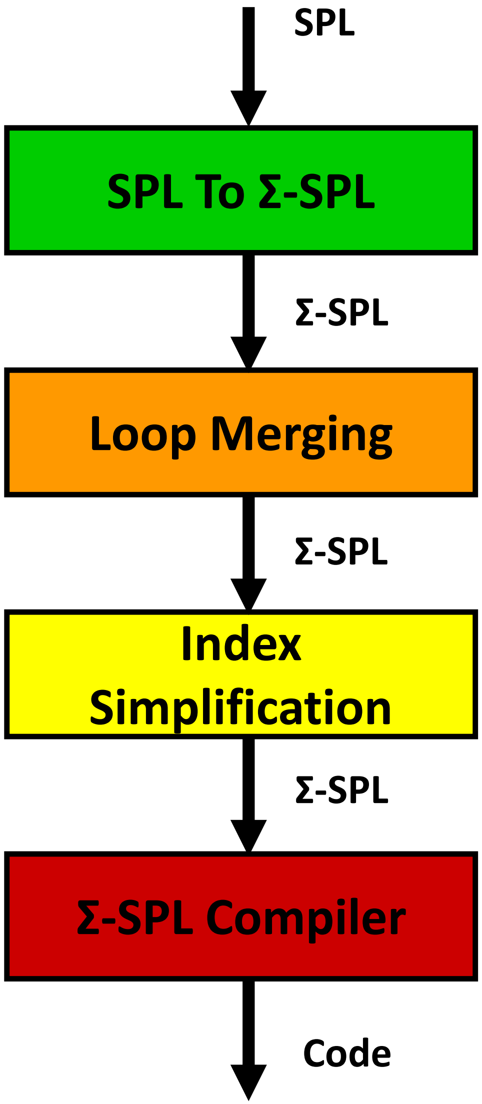
- **Every Σ-SPL formula still represents a matrix factorization**

Example: $$(I_4 \otimes F_2) \rightarrow \sum_{j=0}^{3} S_{f_j} F_2 G_{f_j}$$

$$\begin{bmatrix} F_2 & & & \\ & F_2 & & \\ & & F_2 & \\ & & & F_2 \end{bmatrix}$$



Loop Merging With Rewriting Rules



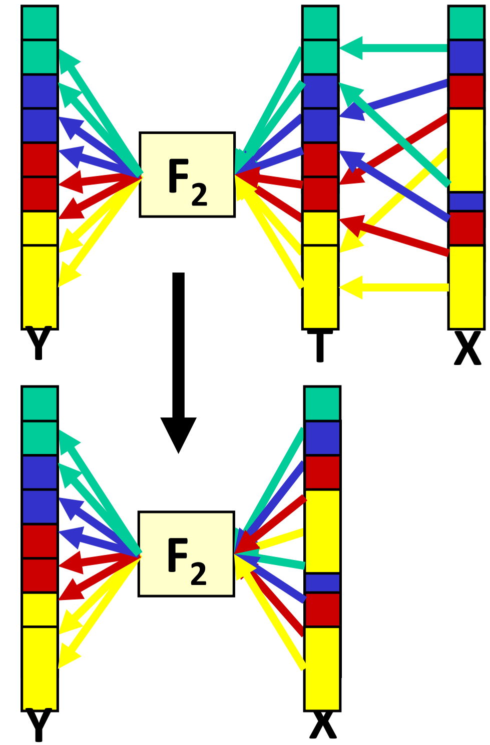
$$y = (I_4 \otimes F_2)\, L^8_4\, x$$

$$\left(\sum_{j=0}^{3} S_{(j)_4 \otimes \imath_2}\, F_2\, G_{(j)_4 \otimes \imath_2}\right) \mathrm{Perm}_{\ell^8_4}$$

$$\left(\sum_{j=0}^{3} S_{(j)_4 \otimes \imath_2}\, F_2\, G_{\ell^8_4 \circ ((j)_4 \otimes \imath_2)}\right)$$

$$\left(\sum_{j=0}^{3} S_{(j)_4 \otimes \imath_2}\, F_2\, G_{\imath_2 \otimes (j)_4}\right)$$

```
for (int j=0; j<=3; j++) {
      y[2*j]   = x[j] + x[j+4];
      y[2*j+1] = x[j] - x[j+4];
}
```



Rules:

$$G_r\, \mathrm{Perm}_p \rightarrow G_{p \circ r}$$

$$\ell^{mn}_m \circ \left((j)_m \otimes \imath_n\right) \rightarrow \imath_n \otimes (j)_m$$

Index Simplification: Basic Idea

Example: Identity necessary for fusing successive Rader and prime-factor step

$$\left(\varphi g^{(b+si) \bmod N'}\right) \bmod N = \left((\varphi g^b)(g^s)^i\right) \bmod N$$

$$s | N', \; N' | N, \; 0 \leq i < n$$

Performed at the Σ-SPL level through rewrite rules on function objects:

$$\overline{w}_{\phi,g}^{N' \to N} \circ \overline{h}_{b,s}^{n \to N'} \quad \longrightarrow \quad \overline{w}_{\phi g^b, g^s}^{n \to N}$$

Advantages:

- no analysis necessary
- efficient (or doable at all)

```
// Input: _Complex double x[28], output: y[28]
double t1[28];
for(int i5 = 0; i5 <= 27; i5++)
    t1[i5] = x[(7*3*(i5/7) + 4*2*(i5%7))%28];
for(int i1 = 0; i1 <= 3; i1++) {
    double t3[7], t4[7], t5[7];
    for(int i6 = 0; i6 <= 6; i6++)
        t5[i6] = t1[7*i1 + i6];
    for(int i8 = 0; i8 <= 6; i8++)
        t4[i8] = t5[i8 ? (5*pow(3, i8))%7 : 0];
    {
        double t7[1], t8[1];
        t8[0] = t4[0];
        t7[0] = t8[0];
        t3[0] = t7[0];
    }
    {
        double t10[6], t11[6], t12[6];
        for(int i13 = 0; i13 <= 5; i13++)
            t12[i13] = t4[i13 + 1];
        for(int i14 = 0; i14 <= 5; i14++)
            t11[i14] = t12[(i14/2) + 3*(i14%2)];
        for(int i3 = 0; i3 <= 2; i3++) {
            double t14[2], t15[2];
            for(int i15 = 0; i15 <= 1; i15++)
                t15[i15] = t11[2*i3 + i15];
            t14[0] = (t15[0] + t15[1]);
            t14[1] = (t15[0] - t15[1]);
            for(int i17 = 0; i17 <= 1; i17++)
                 t10[2*i3 + i17] = t14[i17];
        }
        for(int i19 = 0; i19 <= 5; i19++)
            t3[i19 + 1] = t10[i19];
    }
    for(int i20 = 0; i20 <= 6; i20++)
        y[7*i1 + i20] = t3[i20];
}
```

```
// Input: _Complex double x[28], output: y[28]
  int p1, b1;
  for(int j1 = 0; j1 <= 3; j1++) {
     y[7*j1] = x[(7*j1%28)];
     p1 = 1; b1 = 7*j1;
     for(int j0 = 0; j0 <= 2; j0++) {
        y[b1 + 2*j0 + 1] = x[(b1 + 4*p1)%28] +
                           x[(b1 + 24*p1)%28];
        y[b1 + 2*j0 + 2] = x[(b1 + 4*p1)%28] -
                           x[(b1 + 24*p1)%28];
        p1 = (p1*3%7);
     }
  }
```

After, 2 Loops

Before, 11 Loops

Pattern Matching

Collect

```
opts := SpiralDefaults;
s := SumsRuleTree(RandomRuleTree(DFT(8), opts), opts);
c := CodeSums(s, opts);

Collect(s, Scat);              # get list of scatter operations
Set(Collect(s, Value));        # get all unique values
```

Simple Patterns

```
Collect(c, @(1, [add, sub, neg, mul]));       # get all arith ops...
Collect(c, @(1, [add, sub, neg, mul], e->e.t=TReal)); #...on reals

List(Collect(s, @(1, ISum)), e->e.var);       # all loop variables
Set(Collect(s, @@(1, Value,    # all values inside Blk objects
    (e, cx)->IsBound(cx.Blk) and Length(cx.Blk) > 0)));
```

Subtree patterns

```
Collect(c, [deref, add, sub]);
Collect(c, [mul, @(1), sub]);
Collect(c, [mul, Value, ...]);
Collect(c, [mul, @(1), [sub, deref, @(2)]]);
Collect(c, [mul, @(1), [sub, @(2, deref, e->X in e.free()), @(3)]]);
```

Substitutions

SubstTopDown/SubstBottomUp

```
opts := SpiralDefaults;
c := CodeSums(SumsRuleTree(RandomRuleTree(DFT(8), opts), opts), opts);

# Ordered substitution: traversal order can matter greatly
SubstTopDown(Copy(c), @(1, Value, e->e.v=1), e->V(25));
SubstBottomUp(Copy(c), @(1, Value, e->e.v=1), e->V(-25));
```

Variable substitutions

```
vars := Collect(c, @(1, var, e->e.t=TReal)); # all the real variables
SubstVars(Copy(c), rec((vars[1].id) := V(1.1)));     # substitute one

# record of assignment of consecutive numbers to all real variables
substrec := FoldR(Zip2(vars, [1..Length(vars)]),
    (a,b) -> CopyFields(a, rec((b[1].id) := V(b[2]))), rec());
SubstVars(Copy(c), substrec); # substitute them

# loop unrolling example
i := Ind(4);
c2 := loop(i, 4, assign(nth(X, i), i));      # loop to be unrolled
chain(List(c2.range,   # chain of partially evaluated loop iterations
    i->SubstVars(Copy(c2.cmd), rec((c2.var.id) := V(i)))));
```

Rules

Simple Rules

```
Rule([neg, [neg, @1]], e -> @1.val);
Rule([add, Value, Value],
    e->Value.new(e.args[1].t, e.args[1].v + e.args[2].v));
Rule([im, [conj, @(1)]], x->-im(@(1).val));
Rule([IF, @(1), skip, skip], e -> skip());

Rule([RC, @(1, Compose)], e -> Compose(List(@(1).val.children(), RC)));
Rule([RC, @(1, Gath)], e -> Gath(fTensor(@(1).val.func, fId(2))));

Rule([Tensor, ..., @(1,O), ...], e -> O(Rows(e), Cols(e)));
```

Complex Rules

```
_v0none := @(0).target([ Value, noneExp ]).cond(
    (e) -> Cond(ObjId(e) = noneExp, true, isValueZero(e)));
_0noneOrZero :=(t) -> When(
    ObjId(@(0).val) = noneExp, noneExp(t), t.zero());
Rule([mul, ..., _v0none, ...], e -> _0noneOrZero(e.t));
Rule([@@(0,mul,(e,cx)->IsBound(cx.nth) and cx.nth<>[]), @(1), @(2,add)],
     e -> ApplyFunc(add, List(@(2).val.args, a->@(1).val * a)));
Rule( [im, [mul, [cxpack, @(1), @(2)], [conj, [cxpack,
    @(3).cond(x->x=@(1).val), @(4).cond(x->x=@(2).val)]]]],
    e -> e.t.zero() );
```

Associative Rules

Simple Rules

```
ARule(add, [ @(1,add) ], e -> @1.val.args);
ARule(fTensor,  [@(1, fTensor) ], e -> @(1).val.children());
ARule(fCompose, [@(1), fId ], e -> [@(1).val]);

ARule(Compose, [ @(1, Prm), @(2, Prm) ],
     e -> [ Prm(fCompose(@(2).val.func, @(1).val.func)) ])
ARule(Compose, [ @(1, Gath), @(2, [Gath, Prm]) ],
     e -> [ Gath(fCompose(@(2).val.func, @(1).val.func)) ]);
```

Complex Rules

```
ARule(leq, [@(1, Value, x->x.v<=0), [@(0,mul), @(2, Value, x->x.v>0),
    @(3,var,IsLoopIndex)]], e -> [@(0).val]);

ARule( Compose, [ @(1, [Prm, Scat, ScatAcc, Conj, ConjL, ConjR, ConjLR]),
    @(2, [RecursStep, Grp, BB, SUM, ISum, Data, COND]) ],
    e -> [ CopyFields(@(2).val, rec(
             _children :=  List(@(2).val._children, c -> @(1).val * c),
             dimensions := [Rows(@(1).val), Cols(@(2).val)] )) ]);

ARule(fCompose, [ @(1, L), [ @(3, fTensor),
    @(2).cond(e->range(e) = @(1).val.params[2] and domain(e)=1), ... ] ],
    e->[ fTensor(Copy(Drop(@(3).val.children(), 1)), Copy(@(2).val)) ] );
```

Rule Sets

Define a Rule Set

```
# spiral-core\namespaces\spiral\code\sreduce.gi
Class(RulesStrengthReduce, RuleSet);
RewriteRules(RulesStrengthReduce, rec(
     leq_single := Rule([leq, @(1)], e-> V_true),
     add_assoc  := ARule(add, [ @(1,add) ], e -> @1.val.args),
# hundreds of rules
));
```

Add Rules to Existing Rule Set

```
# somewhere else in the source code
RewriteRules(RulesStrengthReduce, rec(       # add more rules
    logic_single := Rule([@(1, [logic_and, logic_or]), @1], e->@1.val)
));
```

Using Rule Sets

```
RulesStrengthReduce.rules.leq_single;
opts := SpiralDefaults;
s := SPLRuleTree(RandomRuleTree(DFT(8), opts)).sums();
s := Rewrite(s, RulesSums, opts);
s := Rewrite(s, RulesDiag, opts);
s := RulesDiagStandalone(s);
```

Rule Strategies

Define a Rule Strategy

```
# spiral-core\namespaces\spiral\code\sreduce.gi
LibStrategy := [ StandardSumsRules, HfuncSumsRules ];
```

Combining Rule Sets

```
StandardSumsRules := MergedRuleSet(
    RulesSums, RulesFuncSimp, RulesDiag, RulesDiagStandalone,
    RulesStrengthReduce, RulesRC, RulesII, OLRules
);
```

Use of Rule Strategies

```
SpiralDefaults.formulaStrategies.sigmaSpl;
SpiralDefaults.formulaStrategies.rc;
SReduce := (c,opts) ->          # handy shortcut
    ApplyStrategy(c, [RulesStrengthReduce], BUA, opts);

opts := SpiralDefaults;
s := SumsRuleTree(RandomRuleTree(DFT(4), opts), opts);
c := DefaultCodegen(s, Y, X, opts);
c := SubstTopDown(c, @(1, loop), e->e.unroll());
Collect(c, mul);
c := SReduce(c, opts);
Collect(c, mul);
```

Rewriting: General Mechanics

Interface for rewriting

```
# spiral-core\namespaces\spiral\code\ir.gi
Class(deref, nth, rec(
    __call__ := (self, loc) >> Inherited(loc, TInt.value(0)),
    rChildren := self >> [self.loc],
    rSetChild := rSetChildFields("loc"),
));
deref.from_rChildren;
```

Unifying interface across all rewritable objects

```
c := deref(X);              # code objects
c.rChildren();              # get rewritable fields
c.rSetChild(1, Y);          # change a rewritable field

DFT.rChildren;              # Transform level
RandomRuleTree(DFT(4), SpiralDefaults).rChildren;    # ruletree level
F.rChildren;                # SPL level
L.rChildren;                # Permutations
Gath.rChildren;             # Sigma-SPL
Lambda.rChildren;           # Lambda function
fId.rChildren;              # symbolic functions
add.rChildren;              # expressions
T_Real.rChildren;           # data types
```

Hardware Specific Strength Reduction

Strength Reduction and Fixup Rules

```
# spiral-core\namespaces\spiral\platforms\avx\sreduce.gi
RewriteRules(FixCodeAVX, rec(
    fix_noneExp := Rule( noneExp, e -> e.t.zero()),
    vpermf128_8x32f_to_vextract := Rule( [assign, [deref, @(1)],
        [vpermf128_8x32f, @(2), @(3), @(4)]],
        e -> let( p := @(4).val.p,
            a := [[@(2).val, [0]], [@(2).val, [1]], [@(3).val,
            [0]], [@(3).val, [1]]],
            dst := tcast(TPtr(TVect(T_Real(32), 4)), @(1).val),
            chain(
                assign(deref(dst),
                    ApplyFunc(vextract_4l_8x32f, a[p[1]])),
                assign( deref(dst+1),
                    ApplyFunc(vextract_4l_8x32f, a[p[2]])))
        )),
    addsub_4x64f_to_mul := Rule( [addsub_4x64f, _0, @(1)],
        e -> mul(e.t.value([-1,1,-1,1]), @(1).val)),
    addsub_8x32f_to_mul := Rule( [addsub_8x32f, _0, @(1)],
        e -> mul(e.t.value([-1,1,-1,1,-1,1,-1,1]), @(1).val)),
    avx_add_addsub_vzero := Rule([add, @(1), [@(2, [addsub_4x64f,
        addsub_8x32f]), _0, @(3)]],
        e -> ObjId(@(2).val)(@(1).val, @(3).val)),
));
```

Organization

- Overview
- System
- Top level commands
- Abstractions
- Rewriting System I: RuleTree/backtracking search
- Rewriting System II: Visitor Patterns
- Rewriting System III: Associative/large context rules
- **Basic block compiler**

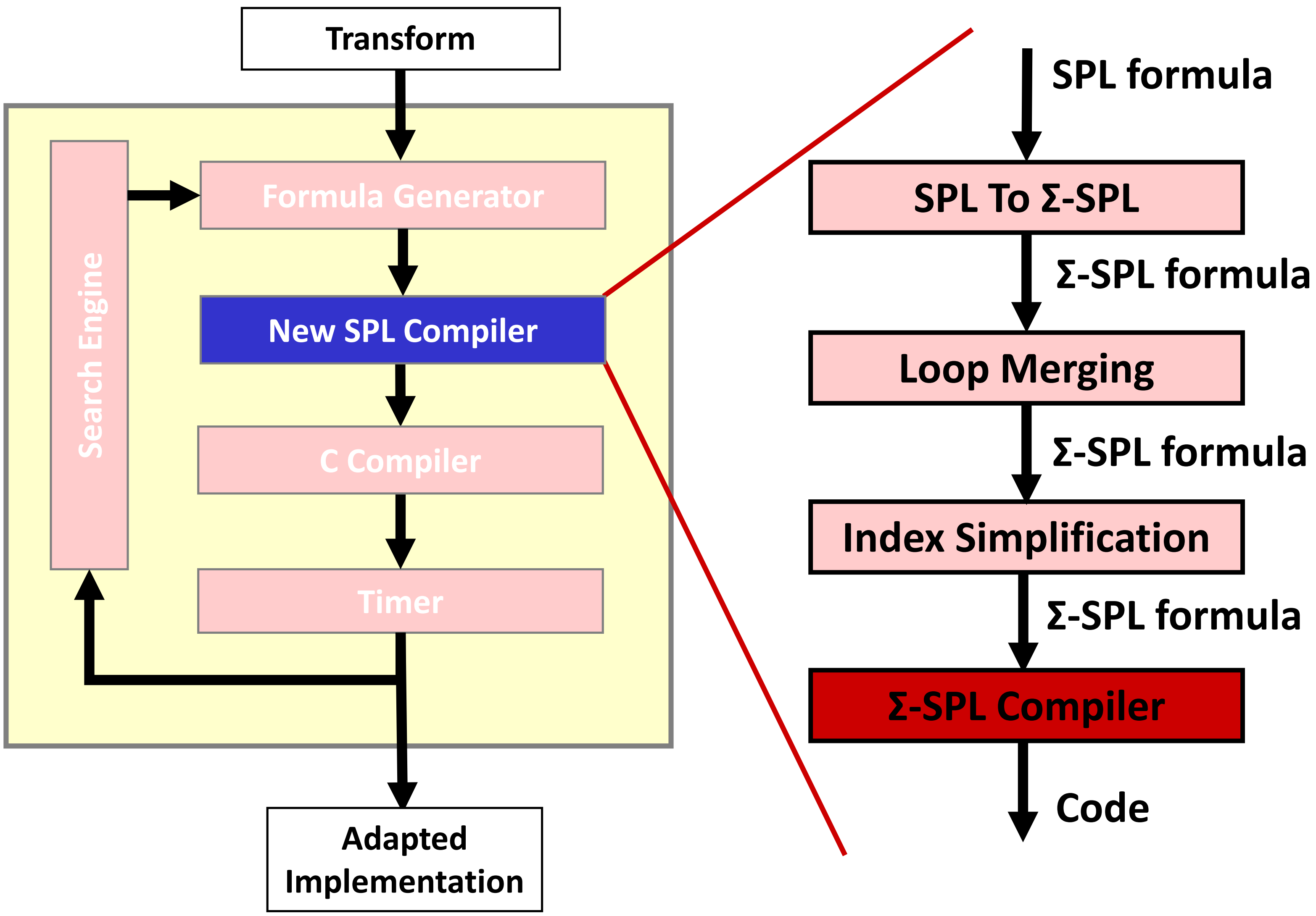
SPL to Sequential Code: SPL Compiler

SPL construct	code
$y = (A_n B_n)x$	`t[0:1:n-1] = B(x[0:1:n-1]);` `y[0:1:n-1] = A(t[0:1:n-1];)`
$y = (I_m \otimes A_n)x$	`for (i=0;i<m;i++)` `y[i*n:1:i*n+n-1] =` `A(x[i*n:1:i*n+n-1])`
$y = (A_m \otimes I_n)x$	`for (i=0;i<m;i++)` `y[i:n:i+m-1] =` `A(x[i:n:i+m-1]);`
$y = \left(\bigoplus_{i=0}^{m-1} A_n^i\right) x$	`for (i=0;i<m;i++)` `y[i*n:1:i*n+n-1] =` `A(i, x[i*n:1:i*n+n-1]);`
$y = D_{m,n} x$	`for (i=0;i<m*n;i++)` `y[i] = Dmn[i]*x[i];`
$y = L_m^{mn} x$	`for (i=0;i<m;i++)` `for (j=0;j<n;j++)` `y[i+m*j]=x[n*i+j];`

Example: tensor product

$$I_m \otimes A_n = \begin{bmatrix} A_n & & \\ & \ddots & \\ & & A_n \end{bmatrix}$$

New Approach for Loop Merging



Translating Σ–SPL to Abstract Code

Compilation rules: recursive descent

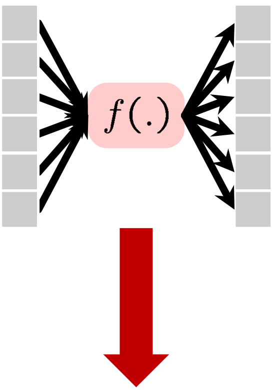
$$\texttt{Code}\left(y = (A \circ B)(x)\right) \;\rightarrow\; \left\{\texttt{decl}(t), \texttt{Code}\left(t = B(x)\right), \texttt{Code}\left(y = A(t)\right)\right\}$$

$$\texttt{Code}\left(y = \left(\sum_{i=0}^{n-1} A_i\right)(x)\right) \;\rightarrow\; \left\{y := \vec{0}, \texttt{for}(i = 0..n-1)\ \texttt{Code}\left(y{+}= A_i(x)\right)\right\}$$

$$\texttt{Code}\left(y = (\mathrm{e}_i^n)^\top(x)\right) \;\rightarrow\; y[0] := x[i]$$

$$\texttt{Code}\left(y = \mathrm{e}_i^n(x)\right) \;\rightarrow\; \left\{y = \vec{0}, y[i] := x[0]\right\}$$

$$\texttt{Code}\left(y = \texttt{Atomic}_f(x)\right) \;\rightarrow\; y[0] := f(x[i])$$




```
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]))
```

Rule-based code generation and backend compilation

Compiling from Σ-SPL to icode

Top-level flow

```
opts := SpiralDefaults;
s := SumsRuleTree(RandomRuleTree(DFT(4), opts), opts);
c := CodeSums(s, opts);
```

Basic Block Compilation

```
# the actual code generator is a configuration option
opts.codegen;
# What happens in CodeSums
DefaultCodegen(Formula(s), Y, X, opts);

# without Formula() the basic block compiler is not run
c := DefaultCodegen(s, Y, X, opts);
# invoke the basic block compiler
Compile(c, opts);

# Compile calls a number of compile strategies
opts.compileStrategy;
for i  in [ 1 .. Length(opts.compileStrategy) ] do
    c := let(stage := opts.compileStrategy[i],
        When(IsCallableN(stage, 2), stage(c, opts), stage(c)));
od;
c;
```

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);
```

A Closer Look at Compiler Stages

Implementation of important stages

```
Print(Compile.pullDataDeclsRefs);
Print(Compile.fastScalarize);
UnrollCode;
FlattenCode;
UntangleChain;
Print(CopyPropagate.copyProp);
Print(CSE.__call__);
Print(HashConsts);
Print(_MarkDefUse);
Print(BinSplit.__call__);
FixValueTypes;
Compile.declareVars;
```

More Information:

www.spiral.net

www.spiralgen.com

References

Overview Papers

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*

M. Püschel, F. Franchetti, Y. Voronenko: **Spiral.** Encyclopedia of Parallel Computing, D. A. Padua (Editor).

M. Püschel, J.M.F. Moura, J. Johnson, D. Padua, M. Veloso, B. Singer, J. Xiong, F. Franchetti, A. Gacic, Y. Voronenko, K. Chen, R.W. Johnson, and N. Rizzolo: **SPIRAL: Code Generation for DSP Transforms.** Special issue, Proceedings of the IEEE 93(2), 2005.

F. Franchetti, Y. Voronenko, S. Chellappa, J. M. F. Moura, and M. Püschel: **Discrete Fourier Transform on Multicores: Algorithms and Automatic Implementation.** IEEE Signal Processing Magazine, special issue on "Signal Processing on Platforms with Multiple Cores", 2009.

Core Technology Papers

F. Franchetti, F. de Mesmay, Daniel McFarlin, and M. Püschel: **Operator Language: A Program Generation Framework for Fast Kernels.** Proceedings of IFIP Working Conference on Domain Specific Languages (DSL WC), 2009.

Y. Voronenko, F. de Mesmay and M. Püschel: **Computer Generation of General Size Linear Transform Libraries.**

Proc. International Symposium on Code Generation and Optimization (CGO), pp. 102-113, 2009.

F. Franchetti, Y. Voronenko, M. Püschel: **Loop Merging for Signal Transforms.** Proceedings Programming Language Design and Implementation (PLDI) 2005, pages 315-326.

F. Franchetti, T. M. Low, S. Mitsch, J. P. Mendoza, L. Gui, A. Phaosawasdi, D. Padua, S. Kar, J. M. F. Moura, M. Franusich, J. Johnson, A. Platzer, and M. Veloso: **High-Assurance SPIRAL: End-to-End Guarantees for Robot and Car Control.** IEEE Control Systems Magazine, 2017, pages 82-103.

References

SIMD Vectorization

F. Franchetti, M. Püschel: **Short Vector Code Generation for the Discrete Fourier Transform.** Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS '03), pages 58-67.

F. Franchetti, Y. Voronenko, M. Püschel: **A Rewriting System for the Vectorization of Signal Transforms.** Proceedings High Performance Computing for Computational Science (VECPAR) 2006, LNCS 4395, pages 363-377.

F. Franchetti and M. Püschel: **SIMD Vectorization of Non-Two-Power Sized FFTs.** Proceedings of International Conference on Acoustics, Speech, and Signal Processing (ICASSP) 07.

F. Franchetti and M. Püschel: **Generating SIMD Vectorized Permutations.** Proceedings of International Conference on Compiler Construction (CC) 2008.

D. S. McFarlin, V. Arbatov, F. Franchetti, M. Püschel: **Automatic SIMD Vectorization of Fast Fourier Transforms for the Larrabee and AVX Instruction Sets.** Proceedings of International Conference on Supercomputing (ICS), 2011.

Multicore and Distributed Memory

F. Franchetti, Y. Voronenko, and M. Püschel: **FFT Program Generation for Shared Memory: SMP and Multicore.** Proceedings Supercomputing 2006.

A. Bonelli, F. Franchetti, J. Lorenz, M. Püschel, and C. W. Ueberhuber: **Automatic Performance Optimization of the Discrete Fourier Transform on Distributed Memory Computers.** Proceedings of ISPA 06. Lecture Notes in Computer Science, Volume 4330, 2006, Pages 818 – 832.

S. Chellappa, F. Franchetti and M. Püschel: **Computer Generation of Fast FFTs for the Cell Broadband Engine.** Proceedings of International Conference on Supercomputing (ICS), 2009.

F. Franchetti, Y. Voronenko, and G. Almasi: **Automatic Generation of the HPC Challenges Global FFT Benchmark for BlueGene/P.** In Proceedings of High Performance Computing for Computational Science (VECPAR) 2012.

References

FPGA and Energy

P. A. Milder, F. Franchetti, J. C. Hoe, and M. Püschel: **Computer Generation of Hardware for Linear Digital Signal Processing Transforms.** ACM Transactions on Design Automation of Electronic Systems,17(2), Article 15, 2012.

P. A. Milder, F. Franchetti, J. C. Hoe, and M. Püschel: **Formal Datapath Representation and Manipulation for Implementing DSP Transforms.** Proceedings of Design Automation Conference (DAC), 2008.

P. A. Milder, F. Franchetti, J. C. Hoe, and M. Püschel: **Hardware Implementation of the Discrete Fourier Transform With Non-Power-Of-Two Problem Size.** Proceedings of International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 2010.

P. D'Alberto, F. Franchetti, P. A. Milder, A. Sandryhaila, J. C. Hoe, J. M. F. Moura, and M. Püschel: **Generating FPGA Accelerated DFT Libraries.** Proceedings of Field-Programmable Custom Computing Machines (FCCM) 2007.

B. Akin, P.A. Milder, F. Franchetti, and J. Hoe: **Memory Bandwidth Efficient Two-Dimensional Fast Fourier Transform Algorithm and Implementation for Large Problem Sizes.** IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM), 188-191, 2012.

B. Akin, F. Franchetti, J. Hoe: **FFTs with Near-Optimal Memory Access Through Block Data Layouts.** ICASSP 2014.

P. D'Alberto, M. Püschel, and F. Franchetti: **Performance/Energy Optimization of DSP Transforms on the XScale Processor.** Proceedings of International Conference on High Performance Embedded Architectures & Compilers (HiPEAC) 2007.

References

Applications (SAR and Software-Defined Radio)

D. McFarlin, F. Franchetti, M. Püschel, and J. M. F. Moura: **High Performance Synthetic Aperture Radar Image Formation On Commodity Multicore Architectures.** in Proceedings SPIE, 2009.

F. de Mesmay, S. Chellappa, F. Franchetti and M. Püschel: **Computer Generation of Efficient Software Viterbi Decoders.** Proceedings of International Conference on High-Performance Embedded Architectures and Compilers (HIPEAC), 2010.

Y. Voronenko, V. Arbatov, C. Berger, R. Peng, M. Püschel, and F. Franchetti: **Computer Generation of Platform-Adapted Physical Layer Software.** Proceedings of Software Defined Radio (SDR), 2010.

C. R. Berger, V. Arbatov, Y. Voronenko, F. Franchetti, M. Püschel: **Real-Time Software Implementation of an IEEE 802.11a Baseband Receiver on Intel Multicore.** Proceedings of International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 2011.

FFT and FIR Algorithms

F. Franchetti and M. Püschel: **Generating High-Performance Pruned FFT Implementations.** Proceedings of International Conference on Acoustics, Speech, and Signal Processing (ICASSP) 09.

LC Meng, J. Johnson, F. Franchetti, Y. Voronenko, M.M. Maza and Y. Xie: **Spiral-Generated Modular FFT Algorithms.** Proc. Parallel Symbolic Computation (PASCO), pp. 169-170, 2010.

Yevgen Voronenko and Markus Püschel: **Algebraic Derivation of General Radix Cooley-Tukey Algorithms for the Real Discrete Fourier Transform.** Proc. International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Vol. 3, 2006

Aca Gacic, Markus Püschel and José M. F. Moura: **Fast Automatic Implementations of FIR Filters.** Proc. International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Vol. 2, pp. 541-544, 2003

References

FFTX and SpectralPACK

F. Franchetti, D. G. Spampinato, A. Kulkarni, D. T. Popovici, T. M. Low, M. Franusich, A. Canning, P. McCorquodale, B. Van Straalen, P. Colella: **FFTX and SpectralPack: A First Look**, IEEE International Conference on High Performance Computing, Data, and Analytics (HiPC), 2018