SPIRAL: A Generator for Platform-Adapted Libraries of Signal Processing Algorithms

Markus Püschel

Faculty

- José Moura (CMU)
- Jeremy Johnson (Drexel)
- Robert Johnson (MathStar Inc.)
- David Padua (UIUC)
- Viktor Prasanna (USC)
- Markus Püschel (CMU)
- Manuela Veloso (CMU)

Students

- Franz Franchetti (TU Vienna)
- Gavin Haentjens (CMU)
- Pinit Kumhom (Drexel)
- Neungsoo Park (USC)
- David Sepiashvili (CMU)
- Bryan Singer (CMU)
- Yevgen Voronenko (Drexel)
- Jianxin Xiong (UIUC)

`http://www.ece.cmu.edu/~spiral`

Sponsor

Work supported by DARPA (DSO), Applied & Computational Mathematics Program, OPAL, through grant managed by research grant DABT63-98-1-0004 administered by the Army Directorate of Contracting.

SPIRAL

Moore's Law and High(est) Performance Scientific Computing

(single processor, off-the-shelf)

Moore's Law:

- processor-memory bottleneck
- short life cycles of computers
- very complex architectures
  - vendor specific
  - special instructions (MMX, SSE, FMA, …)
  - undocumented features

Effects on software/algorithms:

- arithmetic cost model not accurate for predicting runtime
  (one cache miss = 10 floating point ops)
- better performance models hard to get
- best code is machine dependent (registers/caches size, structure)
- hand-tuned code becomes obsolete as fast as it is written
- compiler limitations
- full performance requires (in part) assembly coding

➡ **Portable performance requires automation**

SPIRAL

SPIRAL

Automates

Implementation
- cuts development costs
- code less error-prone

Optimization
- systematic exploration of alternatives both at algorithmic and code level

Platform-Adaptation
- takes advantage of architecture specific features
- porting without loss of performance

of DSP algorithms
- are performance critical

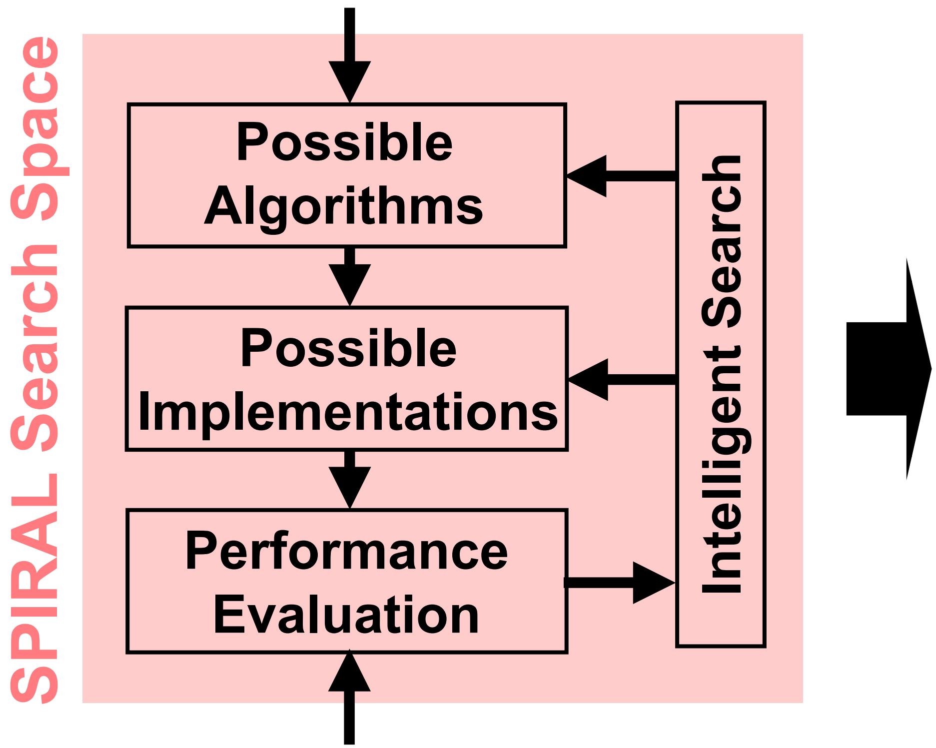
A library generator for highly optimized signal processing algorithms

SPIRAL

SPIRAL Approach

given → **DSP Transform**

(DFT, DCT, Wavelets etc.)



adapted implementation

given → **Computing Platform**

(Pentium III, Pentium 4, Athlon, SUN, PowerPC, Alpha, …)

SPIRAL

DSP Transform

⬇

Algorithm

SPIRAL

DSP Algorithms: Example 4-point DFT

Cooley/Tukey FFT (size 4):

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & i \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Fourier transform

Diagonal matrix (twiddles)

$$DFT_4 = (DFT_2 \otimes I_2) \cdot T_2^4 \cdot (I_2 \otimes DFT_2) \cdot L_2^4$$

Kronecker product

Identity

Permutation

- **product of structured sparse matrices**
- **mathematical notation**

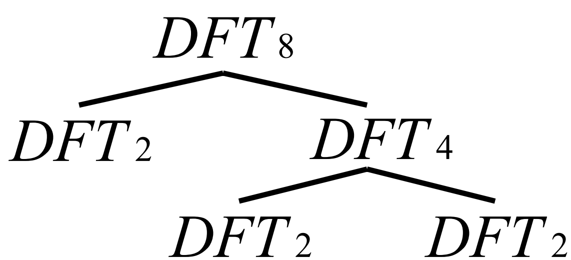
DSP Algorithms: Terminology

Transform DFT_n parameterized matrix

Rule $$DFT_{nm} \rightarrow \left(DFT_n \otimes I_m\right) \cdot D \cdot \left(I_n \otimes DFT_m\right) \cdot P$$

- a breakdown strategy
- product of sparse matrices

Ruletree



- recursive application of rules
- uniquely defines an algorithm
- efficient representation
- easy manipulation

Formula $$DFT_8 = \left(F_2 \otimes I_4\right) \cdot D \cdot \left(I_2 \otimes \left(I_2 \otimes F_2 \cdots\right)\right) \cdot P$$

- few constructs and primitives
- uniquely defines an algorithm
- can be translated into code

DSP Transforms

discrete Fourier transform	$DFT_n = \left[\exp\left(2kli\pi / n\right)\right]$
Walsh-Hadamard transform	$WHT_{2^k} = DFT_2 \otimes \cdots \otimes DFT_2$
discrete cosine and sine Transforms (16 types)	$DCT^{(II)}_n = \left[\cos\left(k\left(l+1/2\right)\pi / n\right)\right]$
	$DCT^{(IV)}_n = \left[\cos\left(\left(k+1/2\right)\left(l+1/2\right)\pi / n\right)\right]$
	$DST^{(I)}_n = \left[\sin\left(kl\pi / n\right)\right]$
modified discrete cosine transform	$MDCT_{n\times 2n} = \left[\cos\left(\left(k+(n+1)/2\right)\left(l+1/2\right)\pi / n\right)\right]$
two-dimensional transform	$T \otimes T$

Others: filters, discrete wavelet transforms, Haar, Hartley, …

Rules = Breakdown Strategies

$DCT_2^{(II)} \rightarrow diag\left(1, 1/\sqrt{2}\right) \cdot F_2$	base case
$DCT_n^{(II)} \rightarrow P \cdot \left(DCT_{n/2}^{(II)} \oplus DCT_{n/2}^{(IV)}\right) \cdot \left(I_{n/2} \otimes F_2\right)^Q$	recursive
$DCT_n^{(IV)} \rightarrow S \cdot DCT_n^{(II)} \cdot D$	translation
$DCT_n^{(IV)} \rightarrow M_1 \cdots M_r$	iterative
$DFT_n \rightarrow B \cdot (DCT_{n/2}^{(I)} \oplus DST_{n/2}^{(I)}) \cdot C$	recursive
$DFT_{nm} \rightarrow \left(DFT_n \otimes I_m\right) \cdot D \cdot \left(I_n \otimes DFT_m\right) \cdot P$	recursive
$F_n(h) \rightarrow (I_{n/d} \otimes^k I_{d+k}) \cdot (I_{n/d} \otimes F_d(h))$	recursive
$F_n(h) \rightarrow Circ(\overline{h}) \cdot E$	recursive
$DWT_n(W) \rightarrow (DWT_{n/2}(W) \oplus I_{n/2}) \cdot P \cdot (I_{n/2} \otimes_k W) \cdot E$	recursive
$WHT_{2^n} \rightarrow \prod_{i=1}^{n} (I_{2^{n_1+\ldots+n_{i-1}}} \otimes WHT_{2^{n_i}} \otimes I_{2^{n_{i+1}+\ldots+n_t}})$	iterative/ recursive
$MDCT_{n \times 2n} \rightarrow S \cdot DCT_n^{(IV)} \cdot P$	translation

SPIRAL

Formula for a DCT, size 16

$$
\begin{aligned}
&[(2,16,9,5,3)(4,15,8,13,7)(6,14,10,12,11),16]\cdot\\
&(([(2,8,5,3)(4,7),8]\cdot(([(2,4,3),4]\cdot((\mathrm{diag}(1,\sqrt{\tfrac{1}{2}})\cdot \mathrm{DFT}_2)\oplus([(1,2),2]\cdot \mathrm{R}_{\frac{13}{8}\pi})^{[(1,2),2]})\cdot\\
&(\mathbf{1}_2\otimes \mathrm{DFT}_2)^{[(2,4,3),4]})\oplus(\mathrm{diag}(\tfrac{1}{2\cos(\frac{1}{16}\pi)},\tfrac{1}{2\cos(\frac{3}{16}\pi)},\tfrac{1}{2\cos(\frac{5}{16}\pi)},\tfrac{1}{2\cos(\frac{7}{16}\pi)})\cdot(\mathbf{1}_2\otimes \mathrm{DFT}_2)^{[(2,4,3),4]}\cdot\\
&((\mathrm{DFT}_2\cdot \mathrm{diag}(1,\sqrt{\tfrac{1}{2}}))\oplus([(1,2),2]\cdot \mathrm{R}_{\frac{13}{8}\pi})^{[(1,2),2]})\cdot[(2,3,4),4]\cdot\begin{bmatrix}1&0&0&0\\1&1&0&0\\0&1&1&0\\0&0&1&1\end{bmatrix})^{[(1,4)(2,3),4]})\cdot\\
&(\mathbf{1}_4\otimes \mathrm{DFT}_2)^{[(2,8,5,3)(4,7),8]})\oplus([(2,5,4,3,7,6,8),8]\cdot(\mathbf{1}_2\otimes(\mathbf{1}_2\oplus([(1,2),2]\cdot \mathrm{R}_{\frac{7}{4}\pi})))\cdot\\
&(\mathbf{1}_2\otimes \mathrm{DFT}_2\otimes \mathbf{1}_2)\cdot(\mathbf{1}_4\oplus([(1,2),2]\cdot \mathrm{R}_{\frac{13}{8}\pi})\oplus([(1,2),2]\cdot \mathrm{R}_{\frac{1}{8}\pi}))\cdot(\mathbf{1}_1\otimes \mathrm{DFT}_2\otimes\\
&\mathbf{1}_4)\cdot(([(1,2),2]\cdot \mathrm{R}_{\frac{49}{32}\pi})\oplus([(1,2),2]\cdot \mathrm{R}_{\frac{53}{32}\pi})\oplus([(1,2),2]\cdot \mathrm{R}_{\frac{57}{32}\pi})\oplus([(1,2),2]\cdot\\
&\mathrm{R}_{\frac{61}{32}\pi}))\cdot[(2,8)(4,6),8])^{[(1,8)(2,7)(3,6)(4,5),8]})\cdot\\
&(\mathbf{1}_8\otimes \mathrm{DFT}_2)^{[(2,16,9,5,3)(4,15,8,13,7)(6,14,10,12,11),16]}
\end{aligned}
$$

DSP Transform

Algorithm (Formula)

Implementation

SPIRAL

Formulas in SPL

....

```
( compose
     ( diagonal ( 2*cos(1/16*pi) 2*cos(3/16*pi) 2*cos(5/16*pi) 2*cos(7/16*pi) ) )
     ( permutation ( 1 3 4 2 ) )
     ( tensor
       ( I 2 )
       ( F 2 )
     )
     ( permutation ( 1 4 2 3 ) )
     ( direct_sum
       ( compose
         ( F 2 )
         ( diagonal ( 1 sqrt(1/2) ) )
       )
       ( compose
         ( matrix
           ( 1 1 0 )
           ( 0 (-1) 1 )
         )
         ( diagonal ( cos(13/8*pi)-sin(13/8*pi) sin(13/8*pi) cos(13/8*pi)+sin(13/8*pi) ) )
         ( matrix
           ( 1 0 )
           ( 1 1 )
           ( 0 1 )
         )
         ( permutation ( 2 1 ) )
```

....

SPL Syntax (Subset)

- matrix operations:
```
(compose formula formula ...)
(tensor formula formula ...)
(direct_sum formula formula ...)
```
- direct matrix description:
```
(matrix (a11 a12 ...) (a21 a22 ...) ...)
(diagonal (d1 d2 ...))
(permutation (p1 p2 ...))
```
- parameterized matrices:
```
(I n)
(F n)
```
- scalars:
```
1.5, 2/7, cos(..), w(3), pi, 1.2e-04
```
- definition of new symbols: ← allows extension of SPL
```
(define name formula)
(template formula (i-code-list)
```
- directives for code generation
```
#codetype real/complex
#unroll on/off
```
← controls loop unrolling

SPIRAL

SPL Compiler, 4-point FFT

fast algorithm
as
formula
as
SPL program

```
(compose (tensor (F 2) (I 2)) (T 4 2)
  (tensor (I 2) (F 2)) (L 4 2))
```

#codetype

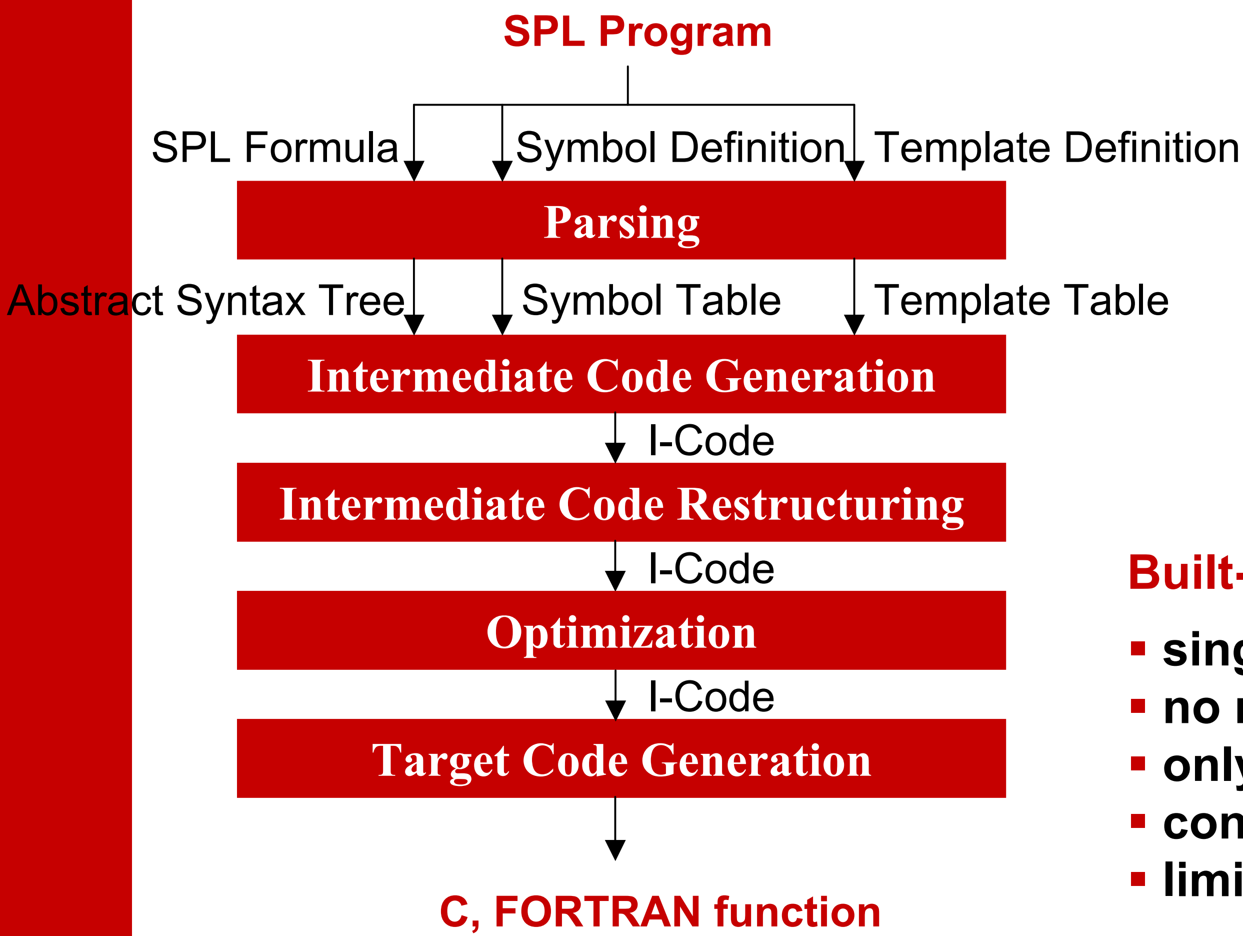
complex | real

```
f0   = x(1) + x(3)
f1   = x(1) - x(3)
f2   = x(2) + x(4)
f3   = x(2) - x(4)
f4   = (0.00d0,-1.00d0)*f(3)
y(1) = f0 + f2
y(2) = f0 - f2
y(3) = f1 + f4
y(4) = f1 - f4
```

```
r0   = x(1) + x(5)
r1   = x(1) - x(5)
r2   = x(2) + x(6)
r3   = x(2) - x(6)
r4   = x(3) + x(7)
r5   = x(3) - x(7)
r6   = x(4) + x(8)
r7   = x(4) - x(8)
y(1) = r0 + r4
y(2) = r1 + r5
y(3) = r0 - r4
y(4) = r1 - r5
y(5) = r2 + r7
y(6) = r3 - r6
y(7) = r2 - r7
y(8) = r3 + r6
```

SPIRAL

SPL Compiler: Summary



Built-in optimizations:

- single static assignment code
- no reuse of temporary vars
- only scalar temporary vars
- constants precomputed
- limited CSE

Extensible through templates

SIMD Short Vector Extensions

- Extension to instruction set architecture
- Available on most current architectures
  (SSE on Pentium, AltiVec on Motorola G4)
- Originally for multimedia (like MMX for integers)
- Requires fine grain parallelism
- Large potential speed-up

Problems:

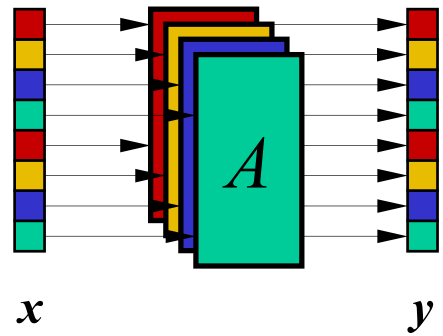
- SIMD instructions are architecture specific
- No common API (usually assembly hand coding)
- Performance very sensitive to memory access
- Automatic vectorization very limited

Vector code generation from SPL formulas

Naturally vectorizable construct

$$A \otimes I_4$$

vector length



(Current) generic construct completely vectorizable:

$$\prod_{i=1}^{k} P_i D_i (A_i \otimes I_\nu) E_i Q_i$$

P_i, Q_i	permutations
D_i, E_i	diagonals
A_i	arbitrary formulas
ν	SIMD vector length

Vectorization in two steps:

1. Formula manipulation using manipulation rules
2. Code generation (vector code + C code)

DSP Transform

↓

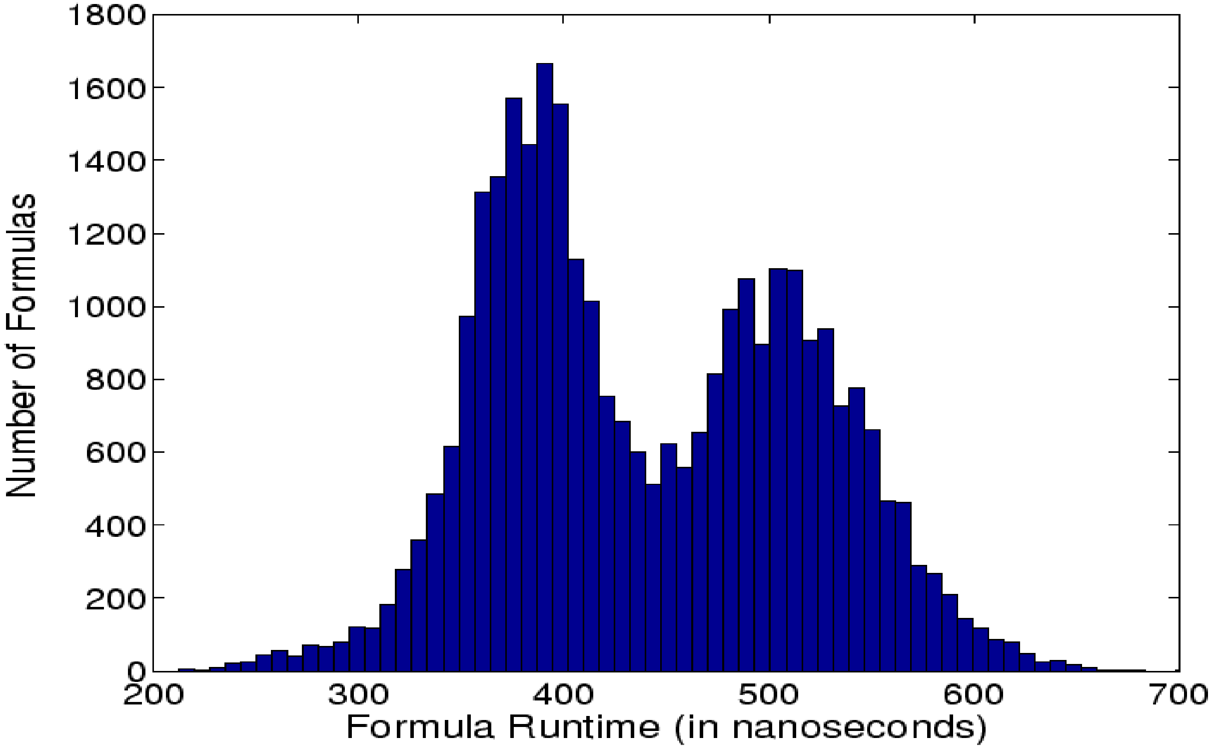
Algorithm (Formula) ← **Search**

↓

Implementation

SPIRAL

Why Search?



DCT, type IV, size 16

~31000 formulas

- **maaaany different formulas**
- **large spread in runtimes, even for modest size**
- **not due to arithmetic cost**

Search Methods available in SPIRAL

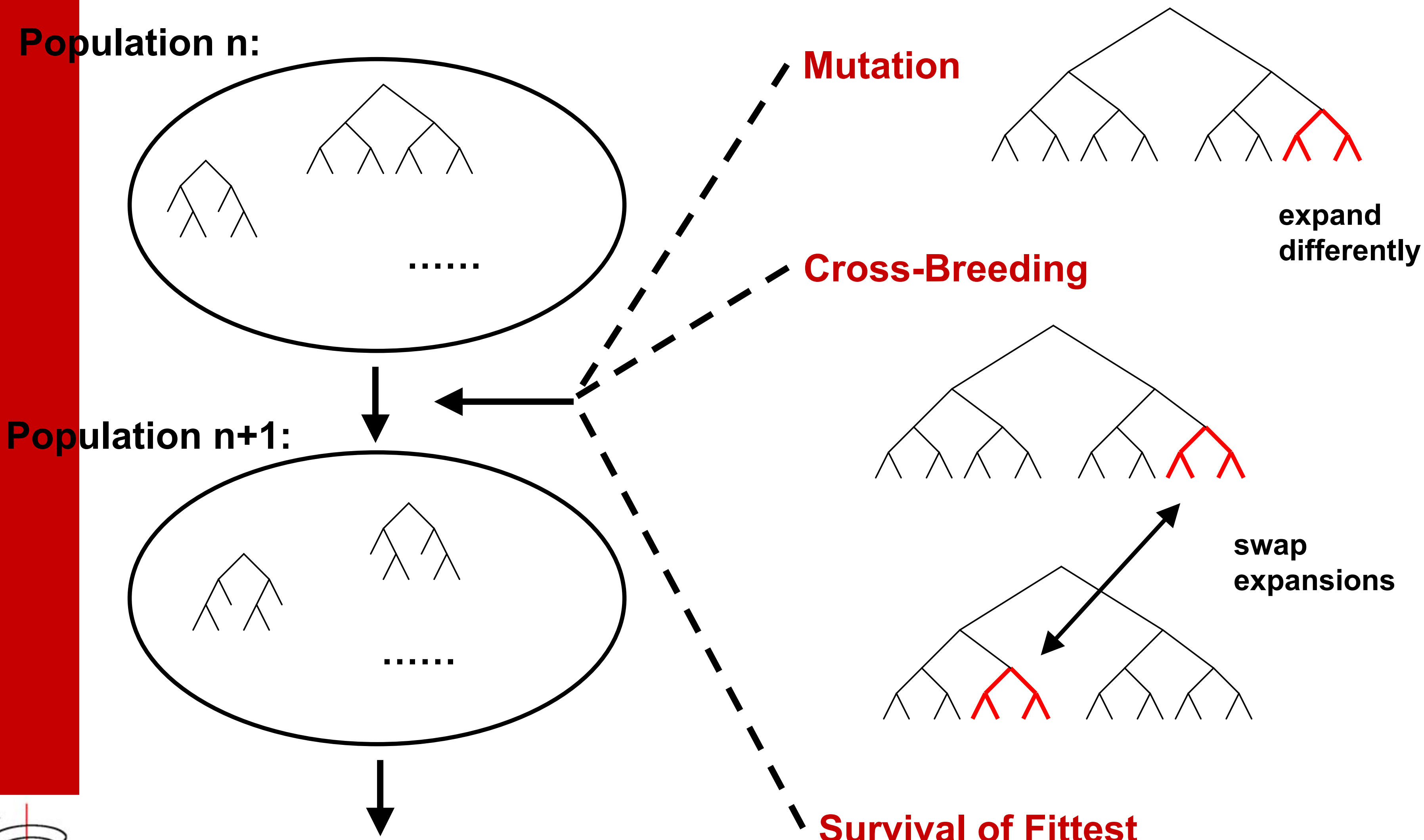
- Exhaustive Search
- Dynamic Programming (DP)
- Random Search
- Hill Climbing
- STEER (similar to a genetic algorithm)

	Possible Sizes	Formulas Timed	Results
Exhaust	Very small	All	Best
DP	All	10s-100s	(very) good
Random	All	User decided	fair/good
Hill Climbing	All	100s-1000s	Good
STEER	All	100s-1000s	(very) good

Search over
• algorithm space and
• implementation options (degree of unrolling)

SPIRAL

STEER



Population n:
......
Population n+1:
......
Mutation
expand
differently
Cross-Breeding
swap
expansions
Survival of Fittest

Learning to Generate Fast Algorithms

- Learns from given dataset (formulas+runtimes) how to design a fast algorithm (breakdown strategy)
- Learns from a transform of one size, generates the best algorithm for many sizes
- Tested for DFT and WHT

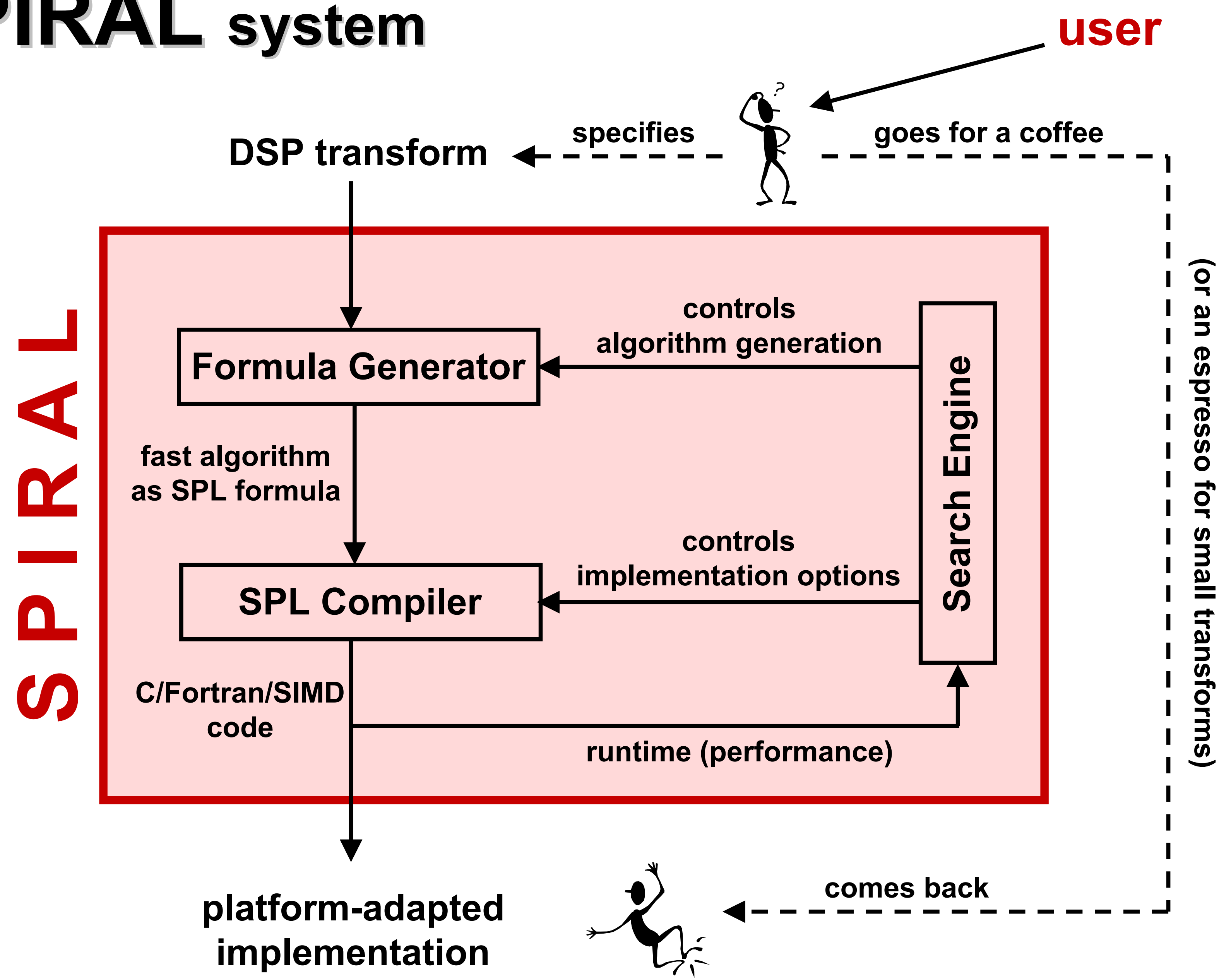
Fast Formula Generation Results

Size	Number of Formulas Generated	Generated Included the Fastest Known	Top N Fastest Known Formulas in Generated
2^{12}	101	yes	77
2^{13}	86	yes	4
2^{14}	101	yes	70
2^{15}	86	yes	11
2^{16}	101	yes	68
2^{17}	86	yes	15
2^{18}	101	yes	25
2^{19}	86	yes	16
2^{20}	101	yes	16

SPIRAL

SPIRAL

SPIRAL system



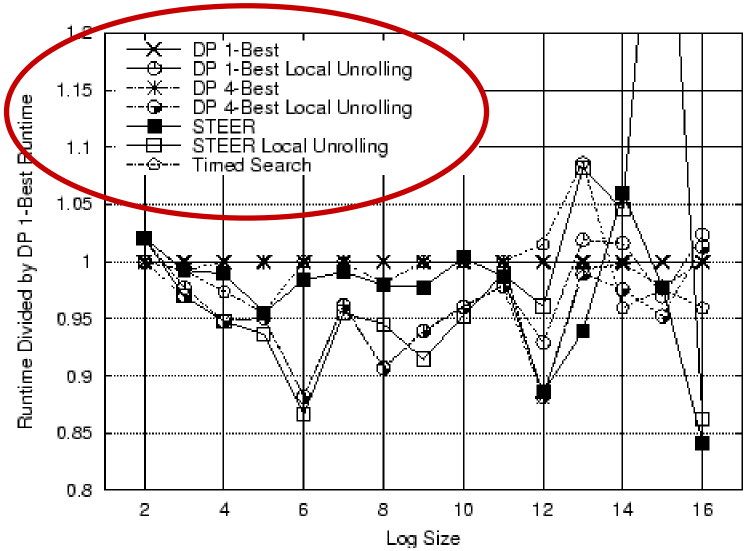
SPIRAL System: Summary

- Available for download: `www.ece.cmu.edu/~spiral`
- Easy installation (Unix: configure/make; Windows: install shield)
- Unix/Linux and Windows 98/ME/NT/2000/XP
- Current transforms: DFT, DHT, WHT, RHT, DCT/DST type I – IV, MDCT, Filters, Wavelets, Toeplitz, Circulants
- Extensible

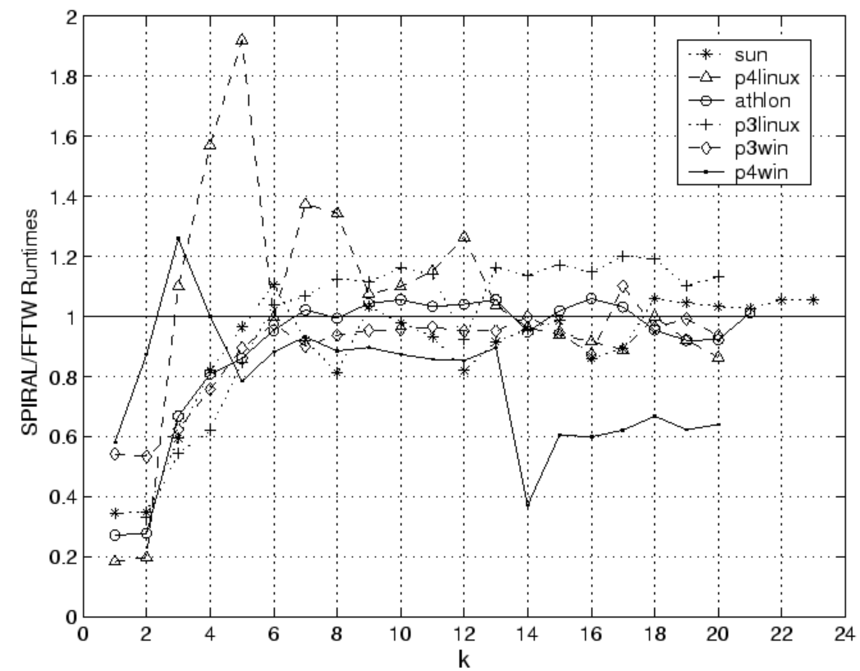
SPIRAL

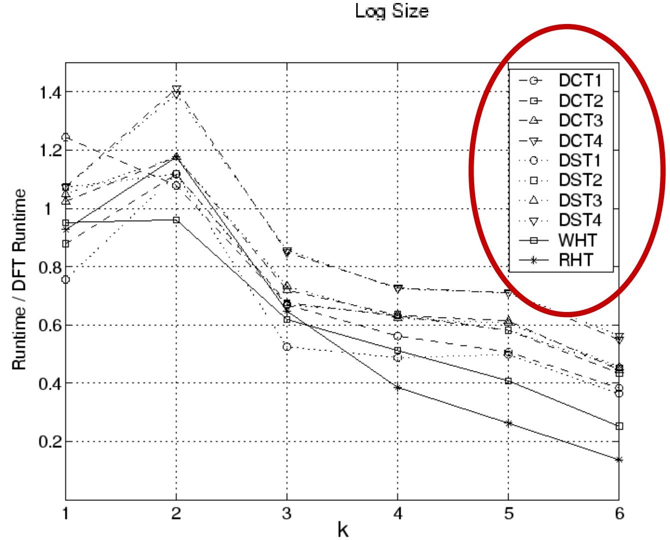
Experimental Results

search methods (applicable to all transforms)



high performance code (compared with FFTW)

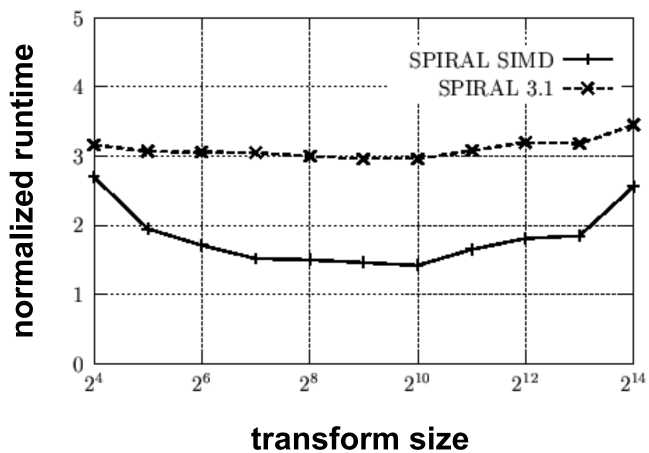




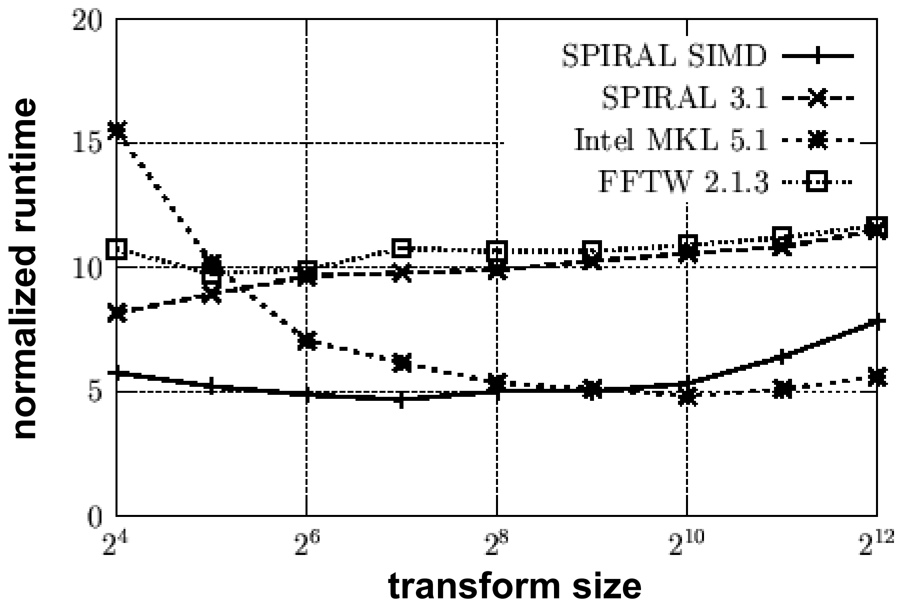
different transforms

Vectorized code

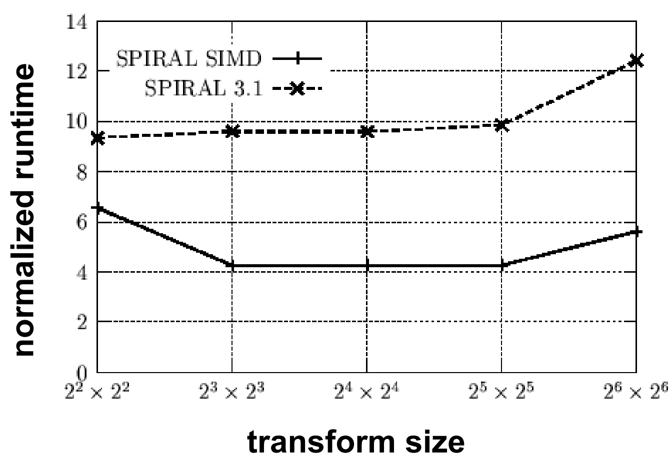
WHT



DFT



2-dim DCT



- speed-ups up to a factor of 2.5
- beats hand-tuned Intel MKL (< 1024)

(Pentium III, SSE)

Conclusions

Closing the gap between math domain (algorithms) and implementation domain (programs)

- Mathematical computer representation of algorithms
- Automatic translation of algorithms into code
- Implement constructs not transforms

Optimization as intelligent search/learning in the space of alternatives

- High level: Mathematical manipulation of algorithms
- Low level: Coding degrees of freedom

http://www.ece.cmu.edu/~spiral

SPIRAL