Computer Generation of Efficient Software Viterbi Decoders

Frédéric de Mesmay, Srinivas Chellappa,
Franz Franchetti, Markus Püschel

Electrical and Computer Engineering
Carnegie Mellon University

Co-Founder
SpiralGen, Inc.

Sponsors: DARPA DESA program, ONR, NSF-NGS/ITR, NSF-ACR, Mercury, and Intel

Viterbi Decoder

- **Error correction**
  - Forward Error Correction
  - Digital cellular (CDMA, GSM), modems, satellite/deep space communications, 802.11 wireless LANs
  - Software defined radio (SDR)
- **Pattern Recognition**
  - Speech recognition
  - text recognition
  - computational linguistics
  - bioinformatics

GSM (TCH/FS)
K=5 rate=1/2

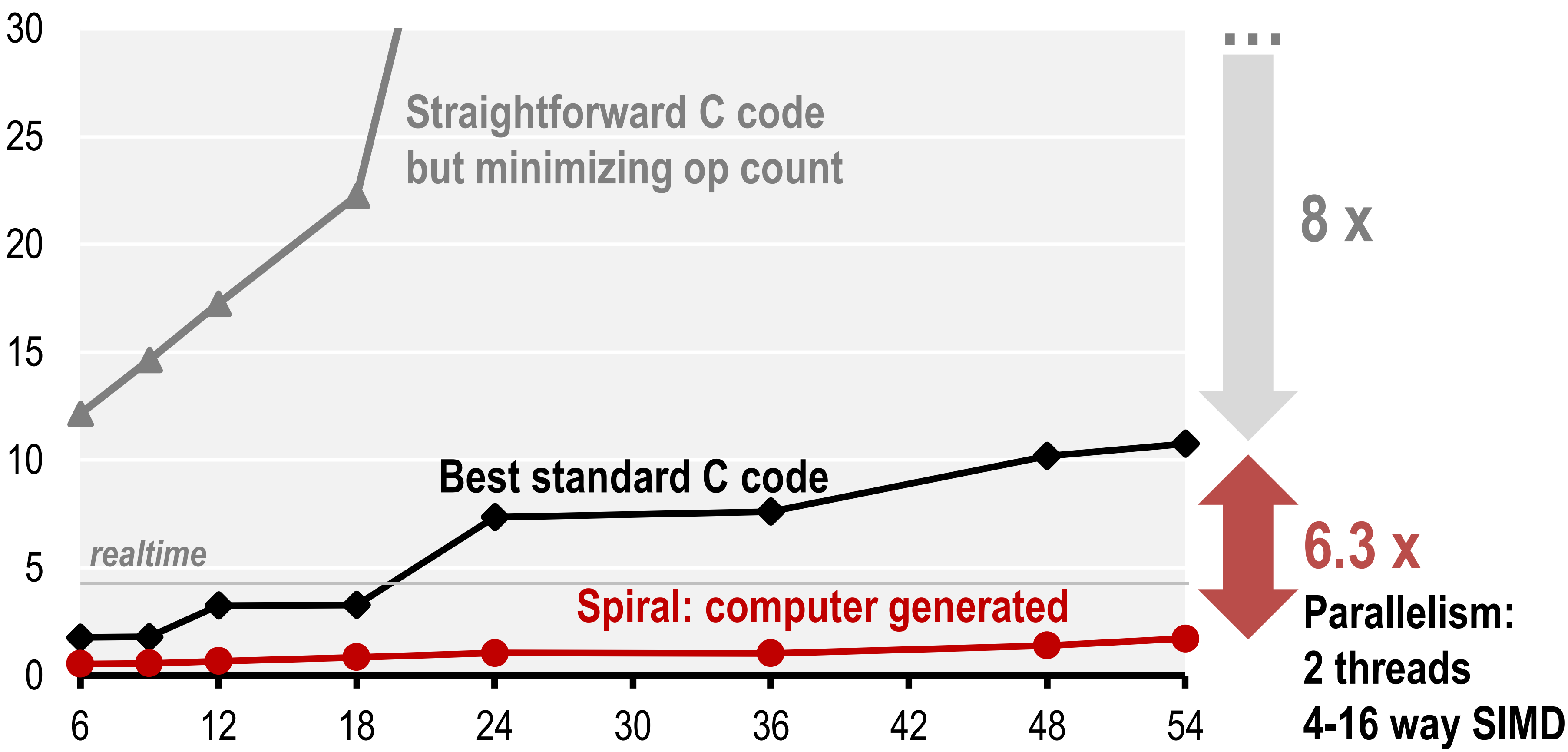
CDMA2000/UMTS/IS-95
K=9 rate=1/3

NASA Cassini Orbiter:
K=15 rate=1/6

SDR requires efficient Viterbi decoder software implementations

Software Defined Radio

WiFi transmitter on Intel Atom Dualcore
Run time per OFDM symbol [µs] vs. data rate [Mbit/s]



Compilers fail to optimize: 50x

Spiral: Viterbi Software Generation

Select convolutional code

Select a preset code or customize parameters

- ○ *custom*
- ○ Voyager
- ○ NASA-DSN
- ◉ CCSDS/NASA-GSFC
- ○ WiMax
- ○ CDMA IS-95A
- ○ LTE (3GPP - Long Term Evolution)
- ○ UWB (802.15)
- ○ CDMA 2000
- ○ Cassini
- ○ Mars Pathfinder & Stereo

rate	1 / 2	code rate (?)
K	7	constraint length (?)
polynomials	79 91	polynomials for the code in decimal notation (?)

Select implementation options

frame length	2048	unpadded frame length in bits (?)
Vectorization level	SSE 16-way	type of code (?)

[Generate Code] [Reset]

"Click": Push-button code generation

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

Spiral: Generated SSE Viterbi Code

```
void viterbi_ccsds(unsigned char *Y, unsigned char *X, unsigned char *syms,
    unsigned char *dec, unsigned char *Branchtab) {
    for(int i9 = 0; i9 <= 1026; i9++) {
        unsigned char a75, a81; int a73, a92;
        ...
        a71 = ((__m128i *) X);          s18 = *(a71); a72 = (a71 + 2);
        s19 = *(a72); a73 = (4 * i9);   a74 = (syms + a73); a75 = *(a74);
        a76 = _mm_set1_epi8(a75);       a77 = ((__m128i *) Branchtab);
        a78 = *(a77);                   a79 = _mm_xor_si128(a76, a78);
        b6 = (a73 + syms);              a80 = (b6 + 1);
        a81 = *(a80);                   a82 = _mm_set1_epi8(a81);
        a83 = (a77 + 2);                a84 = *(a83);
        a85 = _mm_xor_si128(a82, a84);  t13 = _mm_avg_epu8(a79,a85);
        a86 = ((__m128i ) t13);         a87 = _mm_srli_epi16(a86, 2);
        a88 = ((__m128i ) a87);
        t14 = _mm_and_si128(a88, _mm_set_epi8(63, 63, 63, 63, 63, 63, 63 , 63, 63, 63,
            63, 63, 63, 63, 63 , 63));
        t15 = _mm_subs_epu8(_mm_set_epi8(63, 63, 63, 63, 63, 63, 63 , 63, 63, 63, 63,
            63, 63, 63, 63 , 63), t14); m23 = _mm_adds_epu8(s18, t14);
        m24 = _mm_adds_epu8(s19, t15);  m25 = _mm_adds_epu8(s18, t15);
        m26 = _mm_adds_epu8(s19, t14);  a89 = _mm_min_epu8(m24, m23);
        ...
    }
    ...
}
```

Generate Code | Reset

"Click": Push-button code generation

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

Organization

- Spiral
- Generating software Viterbi decoders
- Performance results
- Summary

Organization

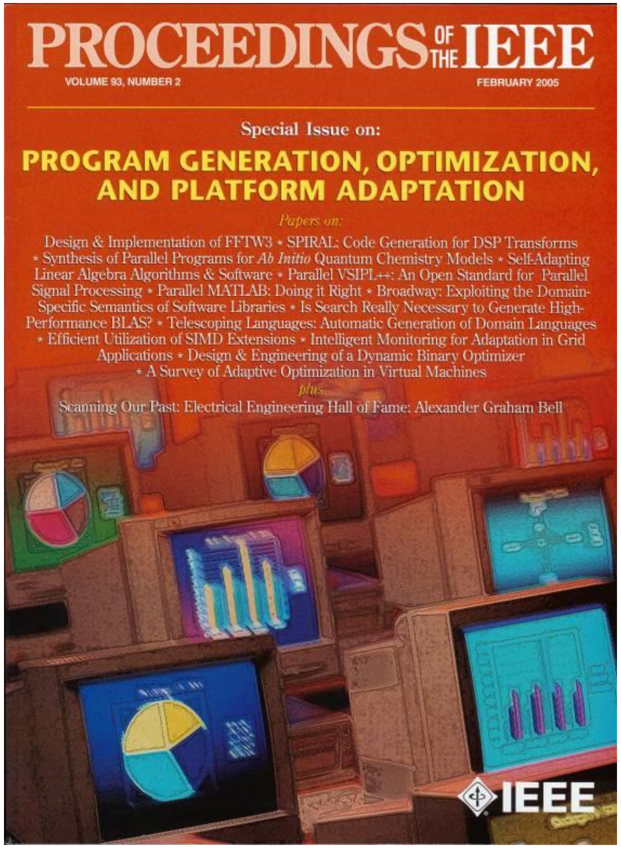
- **Spiral**
- Generating software Viterbi decoders
- Performance results
- Summary

Automatic Performance Tuning

- **Current vicious circle: Whenever a new platform comes out, the same functionality needs to be rewritten and reoptimized**

- **Automatic Performance Tuning**
  - BLAS: ATLAS, PHiPAC
  - Linear algebra: Sparsity/OSKI, Flame
  - Sorting
  - Fourier transform: FFTW
  - Linear transforms (and Viterbi): Spiral
  - …others

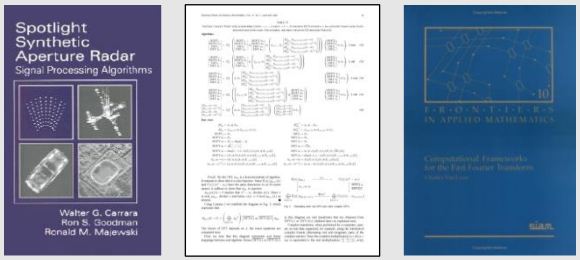
New problem class: software Viterbi decoders



Proceedings of the IEEE special issue, Feb. 2005

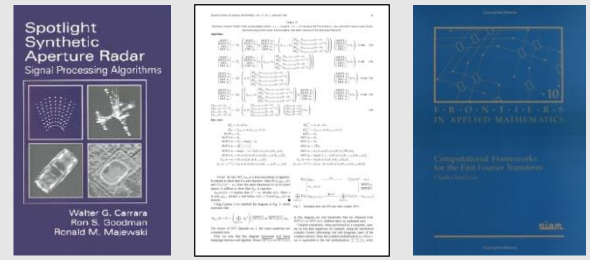
What is Spiral?

Traditionally



High performance library optimized for given platform

Spiral Approach



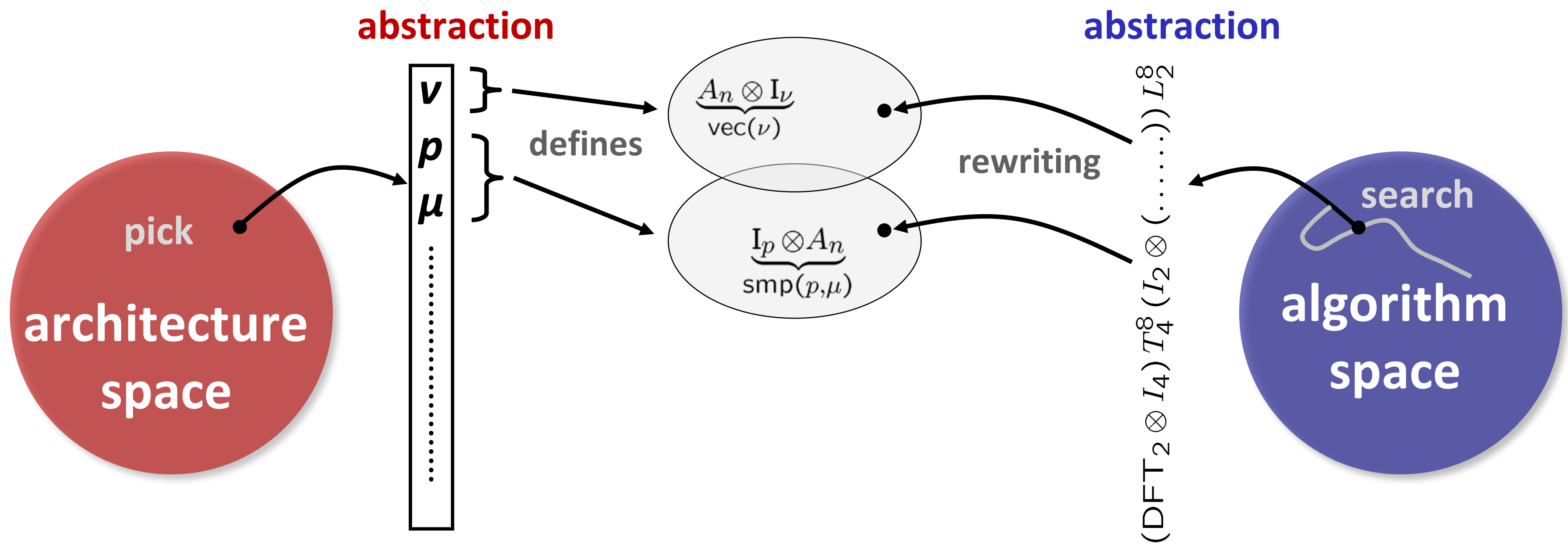
Spiral

High performance library optimized for given platform

Comparable performance

Idea: Common Abstraction and Rewriting

Model: common abstraction
= spaces of matching formulas
= domain-specific language



Architectural parameter:
Vector length,
#processors, …

optimization

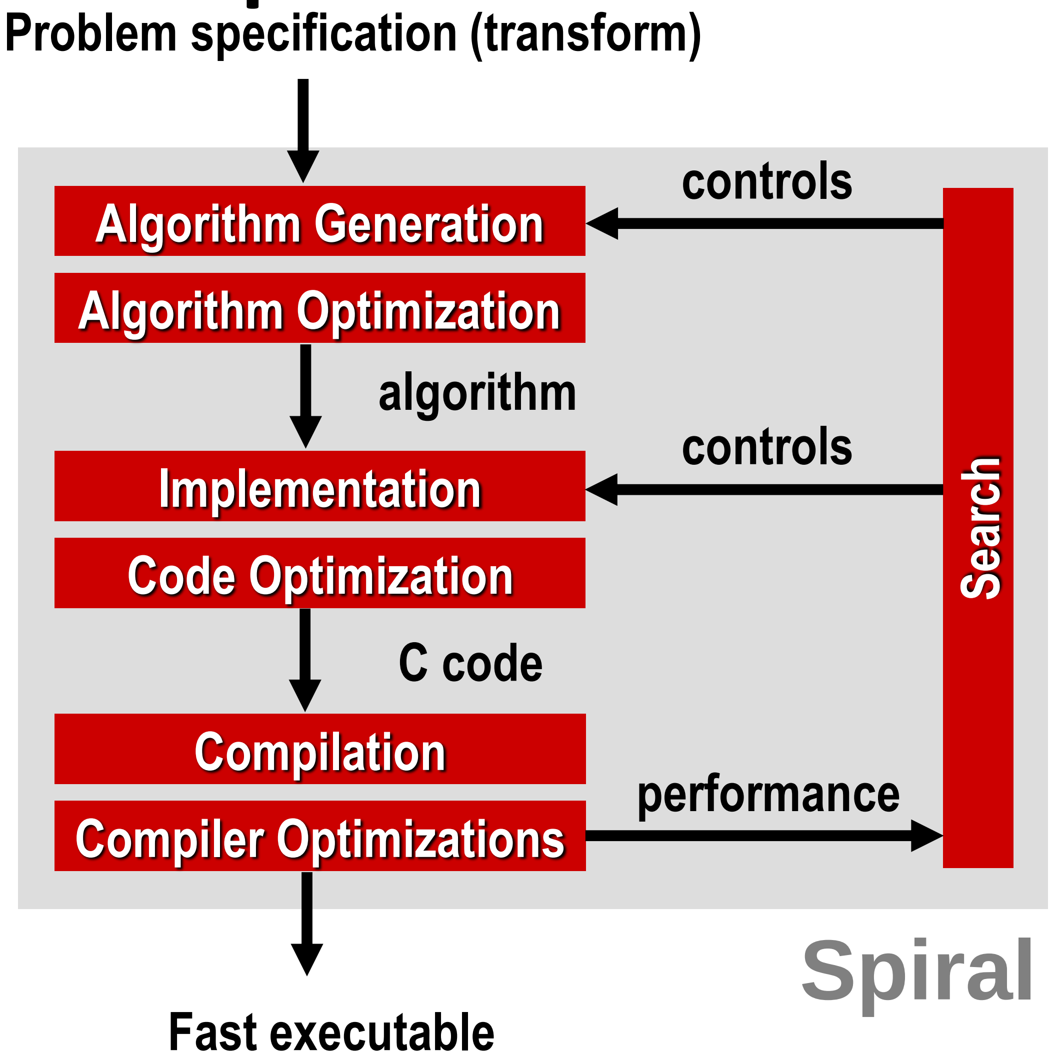
Kernel:
problem size,
algorithm choice

Program Generation in Spiral

Spiral:
Complete automation of the implementation and optimization task

Basic ideas:
Declarative representation of algorithms

Rewriting systems to generate and optimize algorithms at a high level of abstraction



Markus Püschel, José M. F. Moura, Jeremy Johnson, David Padua, Manuela Veloso, Bryan Singer, Jianxin Xiong, Franz Franchetti, Aca Gacic, Yevgen Voronenko, Kang Chen, Robert W. Johnson, and Nick Rizzolo: **SPIRAL: Code Generation for DSP Transforms.** Special issue, Proceedings of the IEEE 93(2), 2005

Some Kernels as Operator Formulas

Linear Transforms

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

Viterbi Decoding

$$\begin{aligned}
\underset{\mathrm{vec}(v)}{\underline{\mathbf{Vit}}} &\to \underbrace{\left(\prod(L \times I) \circ (I \otimes C)\right) \circ Id}_{\mathrm{vec}(v)}\\
&\to \left(\prod \underbrace{(L \times I) \circ (I \otimes C)}_{\mathrm{vec}(v)}\right) \circ Id\\
\pounds &\to \left(\prod(L \otimes I_v \times I) \circ (I \otimes C \otimes I_v) \circ (\vec{L} \times I)\right) \circ Id\\
&\to \prod(L \otimes I_v \times I) \circ (I \otimes (B \otimes I_v)) \circ (\vec{L} \times I)
\end{aligned}$$

Matrix-Matrix Multiplication

$$\begin{aligned}
&\mathsf{MMM}_{1,1,1} \to (\cdot)_1\\
&\mathsf{MMM}_{m,n,k} \to (\otimes)_{m/m_b \times 1} \otimes \mathsf{MMM}_{m_b,n,k}\\
&\mathsf{MMM}_{m,n,k} \to \mathsf{MMM}_{m,nb,k} \otimes (\otimes)_{1 \times n/nb}\\
&\mathsf{MMM}_{m,n,k} \to ((\Sigma_{k/k_b} \circ (\cdot)_{k/k_b}) \otimes \mathsf{MMM}_{m,n,k_b}) \circ\\
&\qquad\qquad ((L^{mk/k_b}_{k/k_b} \otimes I_{k_b}) \times I_{kn})\\
&\mathsf{MMM}_{m,n,k} \to (L^{mn/n_b}_m \otimes I_{n_b}) \circ\\
&\qquad\qquad ((\otimes)_{1 \times n/n_b} \otimes \mathsf{MMM}_{m,n_b,k}) \circ\\
&\qquad\qquad (I_{km} \times (L^{kn/n_b}_{n/n_b} \otimes I_{n_b}))
\end{aligned}$$

Synthetic Aperture Radar (SAR)

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

Same Approach for Different Paradigms

Threading:

$$\underbrace{\mathrm{DFT}_{mn}}_{\mathsf{smp}(p,\mu)} \to \underbrace{\big((\mathrm{DFT}_m \otimes \mathrm{I}_n)\mathrm{T}^{mn}_n(\mathrm{I}_m\otimes \mathrm{DFT}_n)\,\mathrm{L}^{mn}_m\big)}_{\mathsf{smp}(p,\mu)}$$

$$\ldots$$

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

$$\ldots$$

$$\to \big((\mathrm{L}^{mp}_m\otimes \mathrm{I}_{n/p\mu})\otimes_\mu \mathrm{I}_\mu\big)\big(\mathrm{I}_p\otimes_\| (\mathrm{DFT}_m\otimes \mathrm{I}_{n/p})\big)\big((\mathrm{L}^{mp}_p\otimes\mathrm{I}_{n/p\mu})\otimes_\mu\mathrm{I}_\mu\big)$$

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

Vectorization:

$$\underbrace{\big(\mathrm{DFT}_{mn}\big)}_{\mathsf{vec}(\nu)} \to \underbrace{\big((\mathrm{DFT}_m\otimes\mathrm{I}_n)\mathrm{T}^{mn}_n(\mathrm{I}_m\otimes\mathrm{DFT}_n)\,\mathrm{L}^{mn}_m\big)}_{\mathsf{vec}(\nu)}$$

$$\ldots$$

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

$$\ldots$$

$$\to \big(\mathrm{I}_{mn/\nu}\otimes \underbrace{\mathrm{L}^{2\nu}_\nu}_{\mathsf{sse}}\big)\big(\overline{\mathrm{DFT}_m\otimes\mathrm{I}_{n/\nu}}\,\vec\otimes\,\mathrm{I}_\nu\big)\underbrace{\big(\overline{\mathrm{T}^{mn}_n}\big)^{\nu}}_{\mathsf{sse}}$$

$$\Big(\mathrm{I}_{m/\nu}\otimes(\overline{\mathrm{L}^{n}_\nu}\,\vec\otimes\,\mathrm{I}_\nu)\big(\mathrm{I}_{n/\nu}\otimes(\mathrm{L}^{2\nu}_\nu\,\vec\otimes\,\mathrm{I}_\nu)(\mathrm{I}_2\otimes\underbrace{\mathrm{L}^{\nu^2}_\nu}_{\mathsf{sse}})(\mathrm{L}^{2\nu}_2\,\vec\otimes\,\mathrm{I}_\nu)\big)(\overline{\mathrm{DFT}_n}\,\vec\otimes\,\mathrm{I}_\nu)\Big)$$

$$\big((\mathrm{L}^{mn}_m\otimes\mathrm{I}_2)\,\vec\otimes\,\mathrm{I}_\nu\big)\big(\mathrm{I}_{mn/\nu}\otimes\underbrace{\mathrm{L}^{2\nu}_2}_{\mathsf{sse}}\big)$$

GPUs:

$$\underbrace{\big(\mathrm{DFT}_{r^k}\big)}_{\mathsf{gpu}(t,c)} \to \underbrace{\left(\prod_{i=0}^{k-1}\mathrm{L}^{r^k}_r\big(\mathrm{I}_{r^{k-1}}\otimes\mathrm{DFT}_r\big)\Big(\mathrm{L}^{r^k}_{r^{k-i-1}}\big(\mathrm{I}_{r^i}\otimes\mathrm{T}^{r^{k-i}}_{r^{k-i-1}}\big)\underbrace{\mathrm{L}^{r^k}_{r^{i+1}}}_{\mathsf{vec}(c)}\Big)\right)\mathrm{R}^{r^k}_r}_{\mathsf{gpu}(t,c)}$$

$$\ldots$$

$$\to \left(\prod_{i=0}^{k-1}\big(\mathrm{L}^{r^n/2}_r\,\vec\otimes\,\mathrm{I}_2\big)\Big(\mathrm{I}_{r^{n-1}/2}\otimes_\times \underbrace{\big(\mathrm{DFT}_r\,\vec\otimes\,\mathrm{I}_2\big)\mathrm{L}^{2r}_r}_{\mathsf{shd}(t,c)}\Big)\mathrm{T}_i\right)$$

$$\big(\mathrm{L}^{r^n/2}_r\,\vec\otimes\,\mathrm{I}_2\big)\big(\mathrm{I}_{r^{n-1}/2}\otimes_\times \underbrace{\mathrm{L}^{2r}_r}_{\mathsf{shd}(t,c)}\big)\big(\mathrm{R}^{r^{n-1}}_r\,\vec\otimes\,\mathrm{I}_r\big)$$

Verilog for FPGAs:

$$\underbrace{\big(\mathrm{DFT}_{r^k}\big)}_{\mathsf{stream}(r^s)} \to \underbrace{\left[\prod_{i=0}^{k-1}\mathrm{L}^{r^k}_r\big(\mathrm{I}_{r^{k-1}}\otimes\mathrm{DFT}_r\big)\Big(\mathrm{L}^{r^k}_{r^{k-i-1}}\big(\mathrm{I}_{r^i}\otimes\mathrm{T}^{r^{k-i}}_{r^{k-i-1}}\big)\mathrm{L}^{r^k}_{r^{i+1}}\Big)\right]\mathrm{R}^{r^k}_r}_{\mathsf{stream}(r^s)}$$

$$\ldots$$

$$\to \left[\prod_{i=0}^{k-1}\underbrace{\mathrm{L}^{r^k}_r}_{\mathsf{stream}(r^s)}\ \underbrace{\big(\mathrm{I}_{r^{k-1}}\otimes\mathrm{DFT}_r\big)}_{\mathsf{stream}(r^s)}\ \underbrace{\Big(\mathrm{L}^{r^k}_{r^{k-i-1}}\big(\mathrm{I}_{r^i}\otimes\mathrm{T}^{r^{k-i}}_{r^{k-i-1}}\big)\mathrm{L}^{r^k}_{r^{i+1}}\Big)}_{\mathsf{stream}(r^s)}\right]\underbrace{\mathrm{R}^{r^k}_r}_{\mathsf{stream}(r^s)}$$

$$\ldots$$

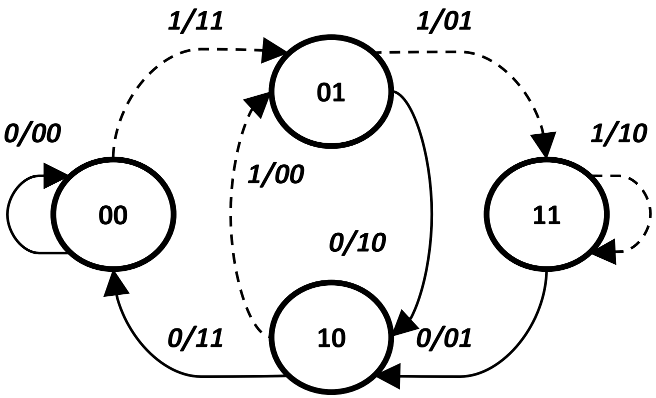
$$\to \left[\prod_{i=0}^{k-1}\underbrace{\mathrm{L}^{r^k}_r}_{\mathsf{stream}(r^s)}\ \big(\mathrm{I}_{r^{k-s-1}}\otimes_s(\mathrm{I}_{r^{s-1}}\otimes\mathrm{DFT}_r)\big)\ \underbrace{\mathrm{T}'_i}_{\mathsf{stream}(r^s)}\right]\underbrace{\mathrm{R}^{r^k}_r}_{\mathsf{stream}(r^s)}$$

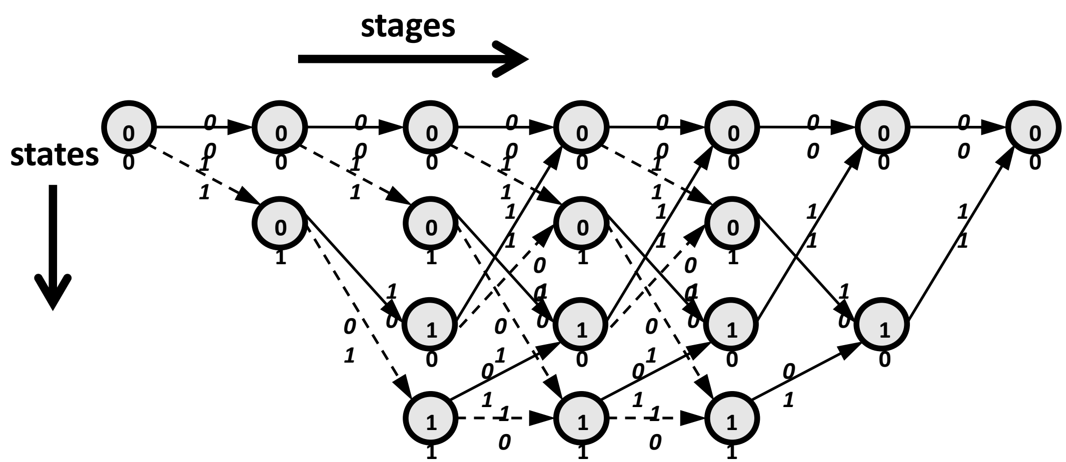
Organization

- Spiral
- Generating software Viterbi decoders
- Performance results
- Summary

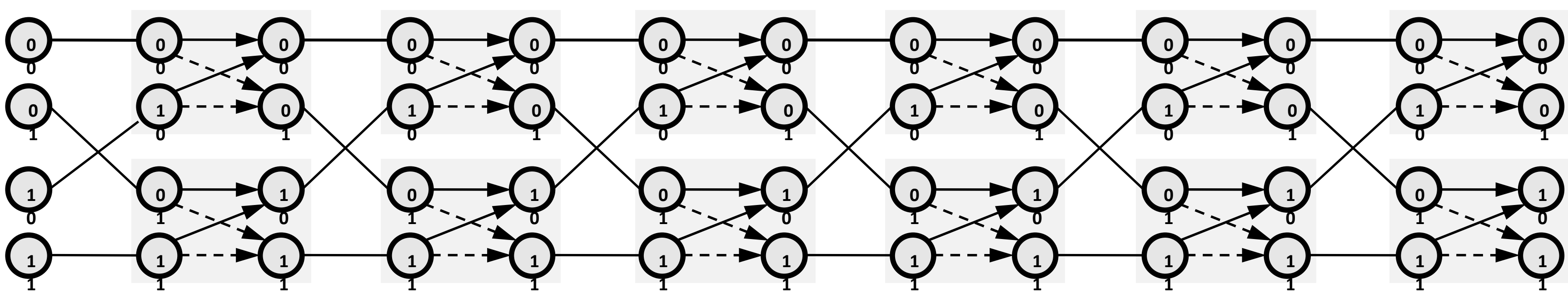
Structure of Viterbi Decoders

State machine





Viterbi trellis (data flow)



Key observation: similarity to Walsh-Hadamard transform (WHT)

Viterbi Language (VL)

VL in Backus-Naur Form (BNF)

<op>	::=	$F_{K,F}$	*Viterbi forward pass*
	|	I_n	*identity*
	|	L_n^{mn}	*stride permutation*
	|	$B_{i,j}$	*Viterbi butterfly*
	|	<op> <op>	*composition*
	|	$\prod$ <op>	*iterative composition*
	|	<op> $\otimes$ <op>	*tensor product*

Viterbi decoder forward pass in VL

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

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

Compiling VL To Code

construct	code
$y = (CD)x$	`t = D(x);` `y = C(t);`

$y = \prod_{i=0}^{l-1} C^i x$	`y = C(l-1, x);`
	`for (i=l-2;i>=0;i--)`
	`   y = C(i, y);`

$y = (I_m \otimes_j C_n^j)x$	`for (j=0;j<m;j++)`
	`   y[j*n:j*n+n-1] =`
	`      C(j, x[j*n:j*n+n-1]);`

$y = L_m^{mn} x$	`for (i=0;i<m;i++)`
	`   for (j=0;j<n;j++)`
	`      y[i+m*j]=x[n*i+j];`

$y = B_{i,j} x$	*see equation last slide*

Vectorization Through Rewriting

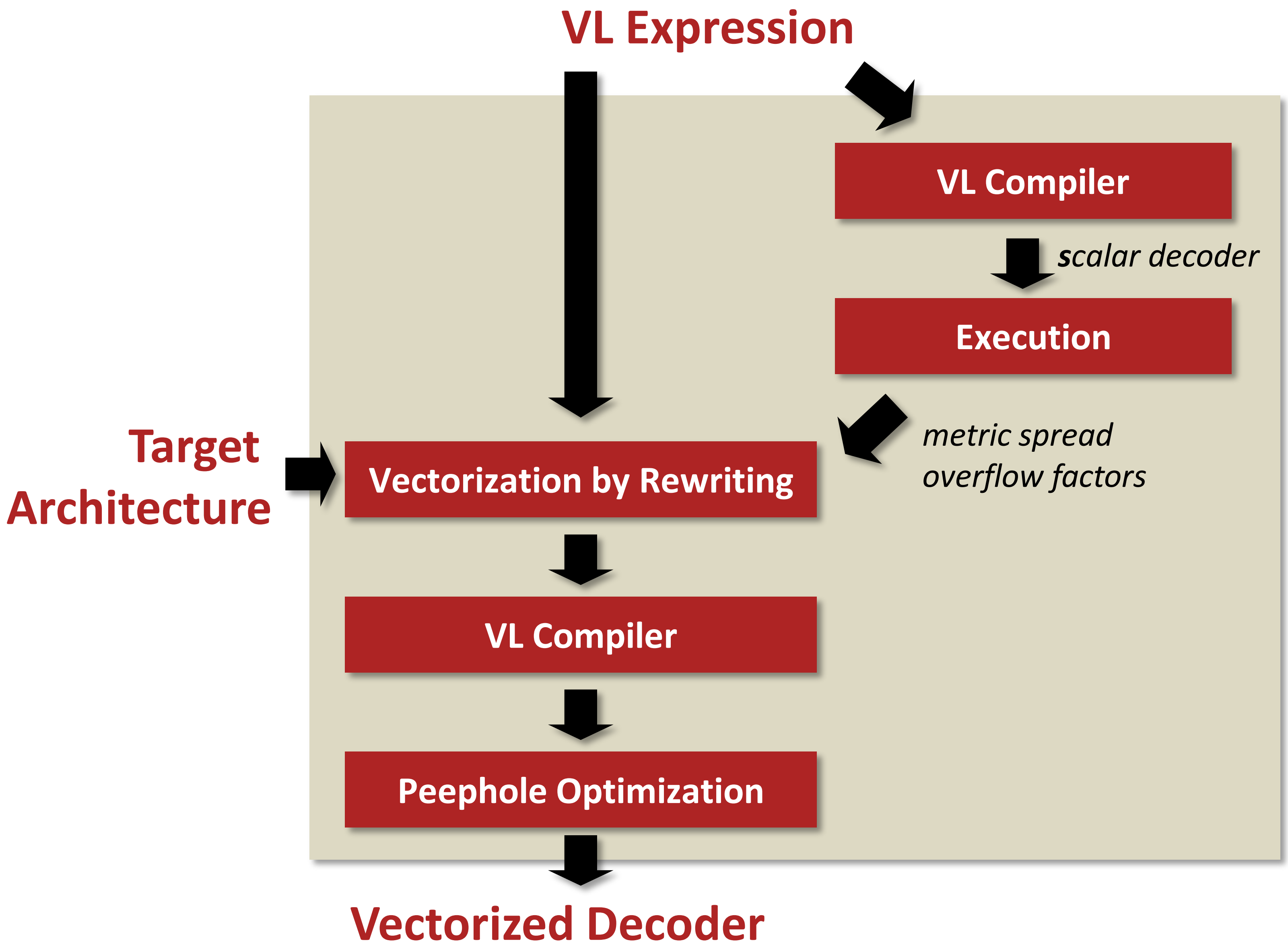
Vectorization Rule Set

$$
\begin{aligned}
\underline{CD}_{\nu} &\rightarrow \underline{C}_{\nu}\;\underline{D}_{\nu} \\
\underline{\prod C}_{\nu} &\rightarrow \prod \underline{C}_{\nu} \\
\underline{I_m \otimes_j C^j}_{\nu} &\rightarrow I_m \otimes_j \underline{C^j}_{\nu} \\
\underline{C \otimes \mathrm{I}_{\nu}}_{\nu} &\rightarrow C \bar{\otimes} \mathrm{I}_{\nu} \\
\underline{B_{i,l\nu+j} \otimes_j \mathrm{I}_{\nu}}_{\nu} &\rightarrow \vec{B}^{\nu}_{i,l} \\
\underline{\mathrm{L}^{2\nu}_{\nu}}_{\nu} &\rightarrow \vec{\mathrm{L}}^{2\nu}_{\nu}
\end{aligned}
$$

Vectorized Viterbi Decoder

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

VL Compilation System

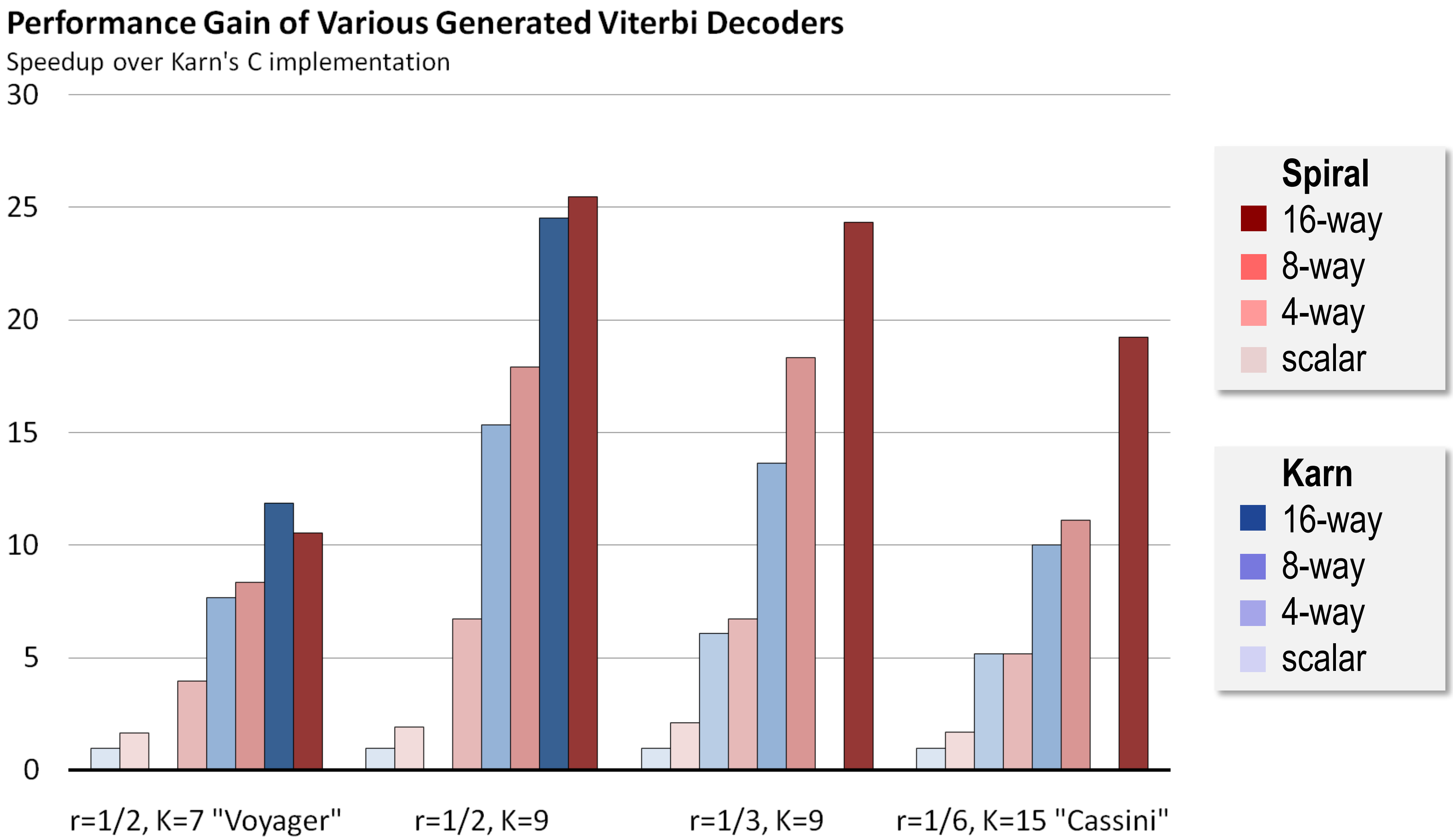


Organization

- Spiral
- Generating Software Viterbi Decoders
- **Performance results**
- Summary

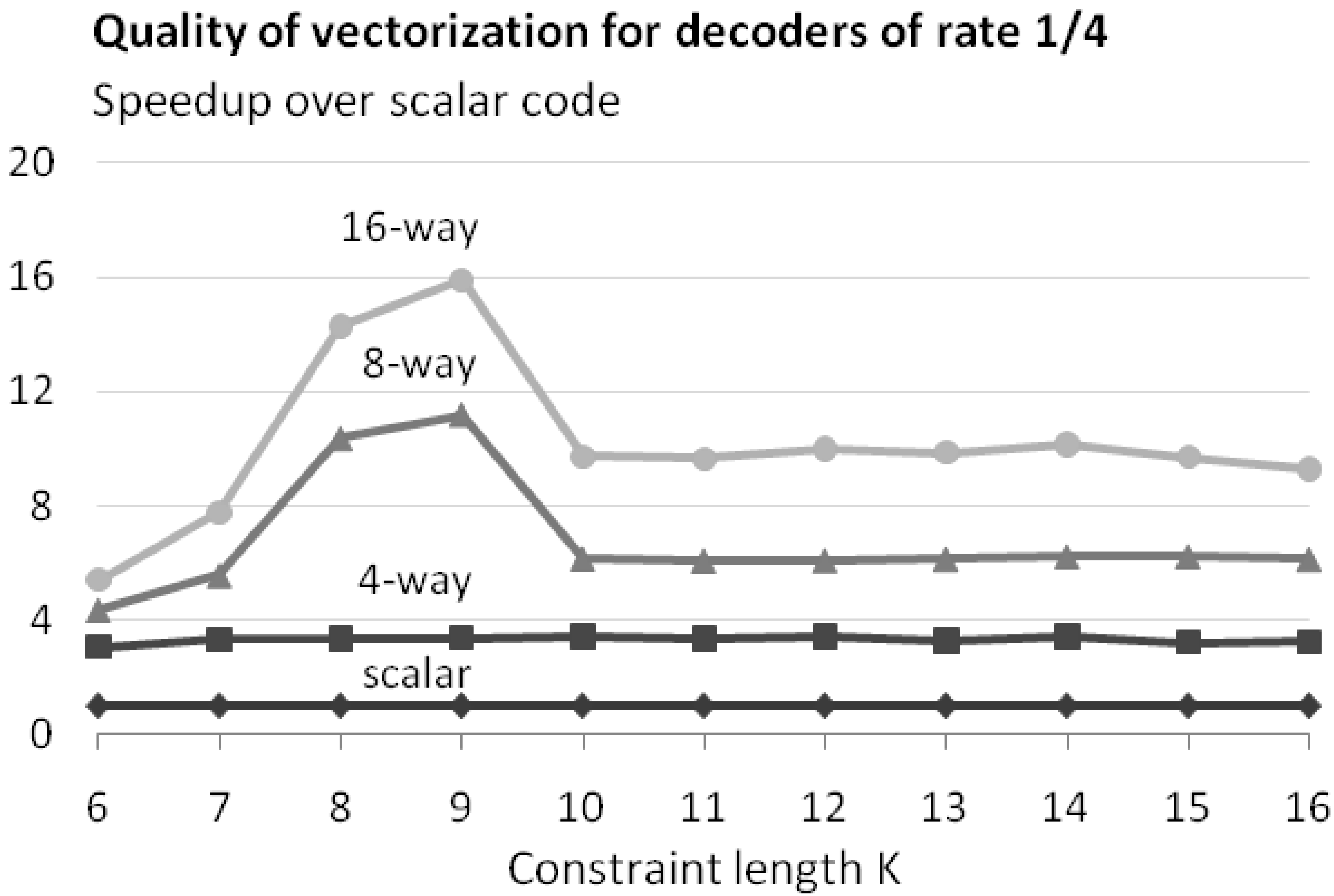
Comparison to Hand-Tuned Code

Karn's implementation: hand-written assembly for 4 specific Viterbi codes



Single core of Core2 Extreme (quad-core), 3 GHz, Intel C++ compiler 10.0

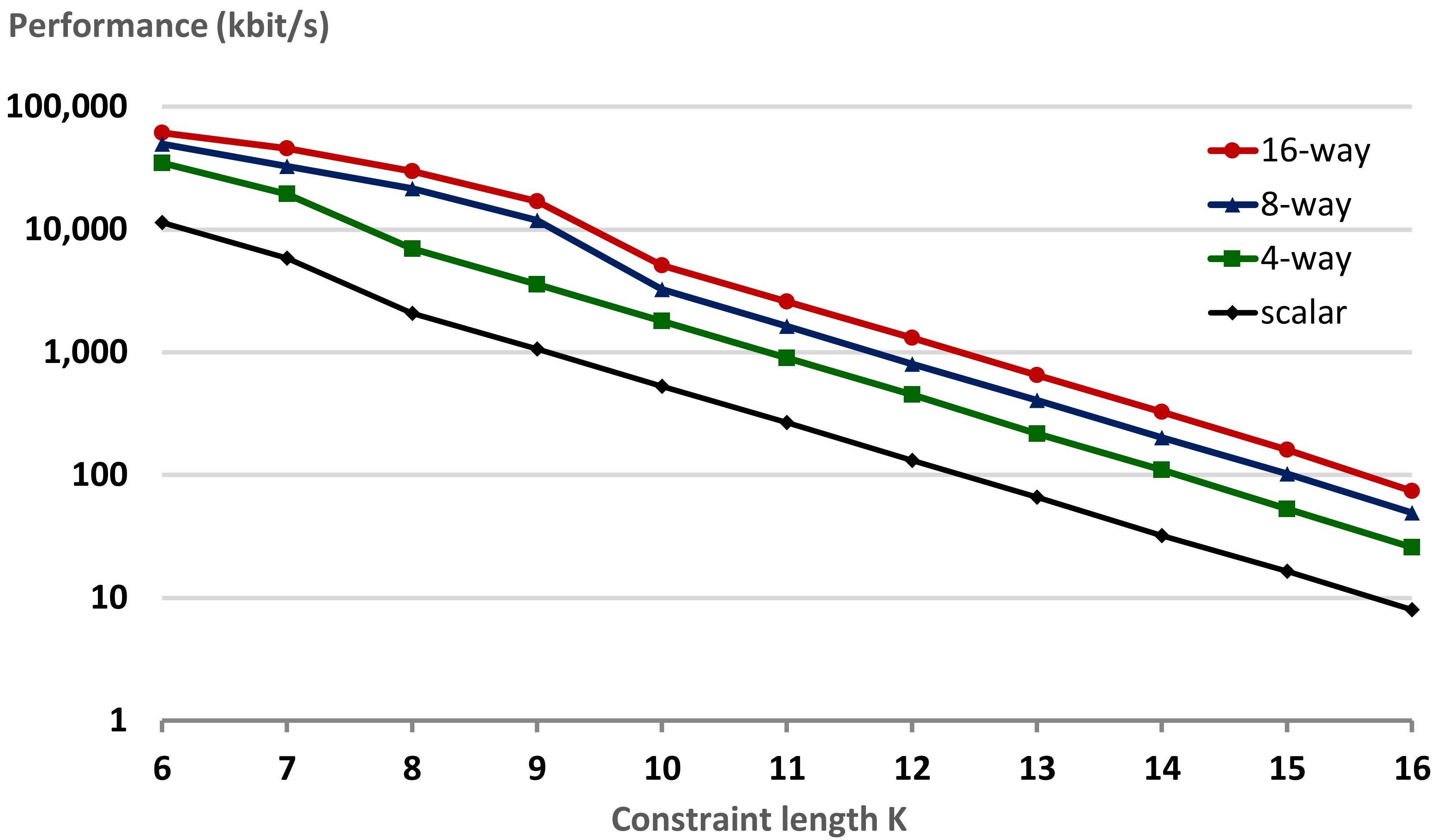
Vectorization Speed-Up



Single core of Core2 Extreme (quad-core), 3 GHz, Intel C++ compiler 10.0

Data Rate Results

Decoders for rate 1/4



Single core of Core2 Extreme (quad-core), 3 GHz, Intel C++ compiler 10.0

Organization

- Spiral
- Generating Software Viterbi Decoders
- Performance results
- **Summary**

Summary

- **Platforms are powerful yet complicated**
  optimization will stay a hard problem

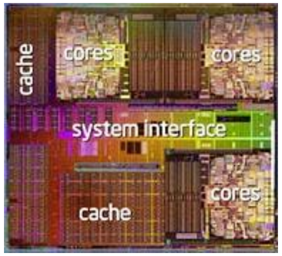
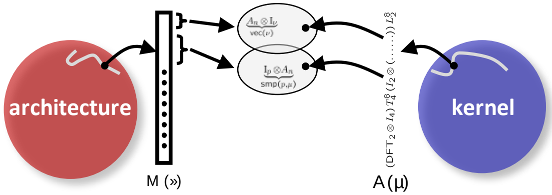


Image: Intel

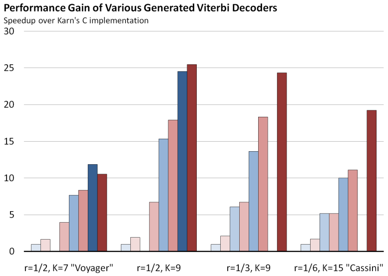
- **Automatic generation of Viterbi decoder**
  from high-level specification

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

- **Spiral: program generation and autotuning**
  can provide full automation



- **Performance of Spiral's Viterbi decoders**
  is competitive with expert hand tuning



(Part of the) Spiral Team

www.spiral.net