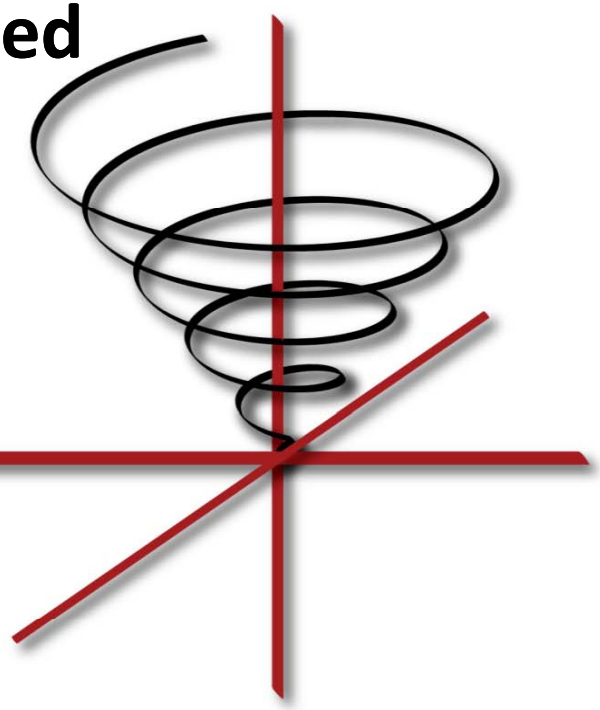
SPIRAL
www.spiral.net

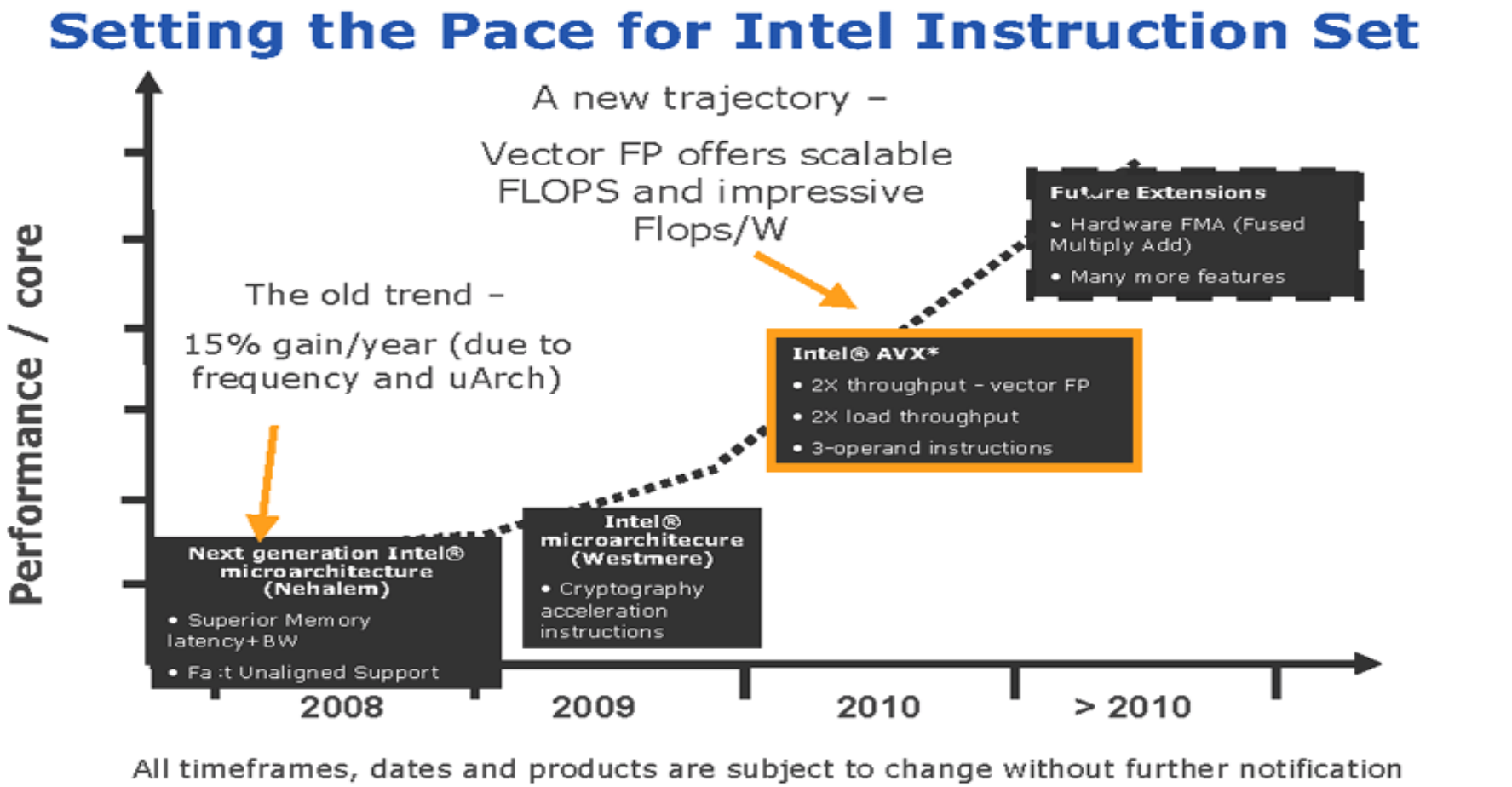
Automatic Generation of Vectorized Fast Fourier Transform Libraries for the Larrabee and AVX Instruction Set Extension



Daniel McFarlin
Franz Franchetti
Markus Püschel
Carnegie Mellon University

HPEC, September 2009, Lexington, MA, USA
This work was supported by DARPA DESA program, NSF-NGS/ITR, NSF-ACR, and Intel
Daniel McFarlin was supported by NPSC and NDSEG Fellowships

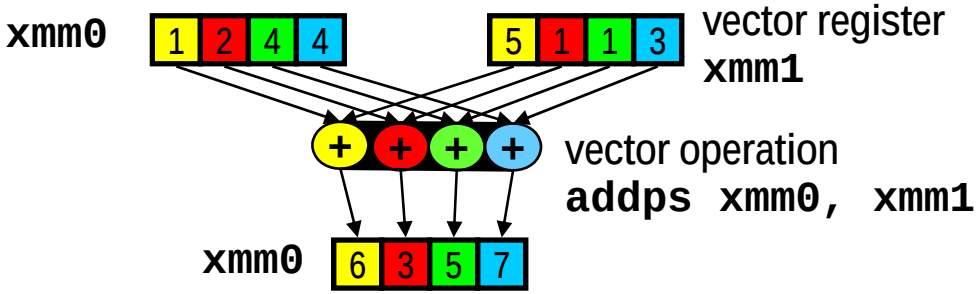
Performance Trends



Vector Units Will Dominate Performance on CPUs, GPUs and GPGPUs

Vectorization Challenges

- **Must extract fine-grain data level parallelism**

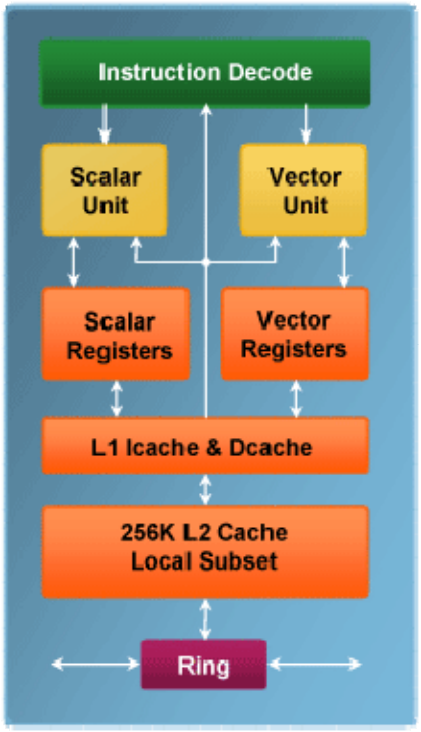


- **Problems:**
  - Not standardized
  - Compiler vectorization limited
  - Low-level issues (data alignment,…)
  - Reordering data kills runtime

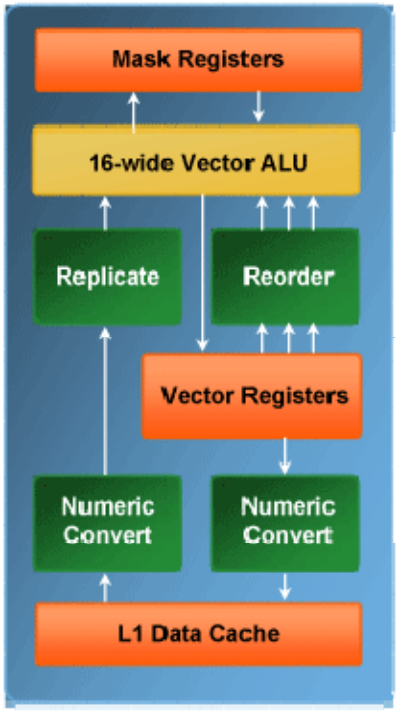
- **Intel MMX**
- **AMD 3DNow!**
- **Intel SSE**
- **AMD Enhanced 3DNow!**
- **Motorola AltiVec**
- **AMD 3DNow! Professional**
- **Itanium**
- **Intel XScale**
- **Intel SSE2**
- **AMD-64**
- **IBM BlueGene/L PPC440FP2**
- **Intel Wireless MMX**
- **Intel SSE3**
- **Intel AVX**
- **Intel LRBni**

One can easily slow down a program by vectorizing it

Larrabee



Processor Core



Vector Unit

- Many enhanced in-order x86 cores
  - Simple Cores, Complex Instructions
- Completely New Vector Instructions
  - 512-bit vectors (16 x 32-bit floats)
  - Ternary, Fused multiply-add/sub
  - Writemasking/Predication
  - Load-op (third operand from mem)
  - Broadcasting
  - Swizzling (pre-defined shuffle ops)

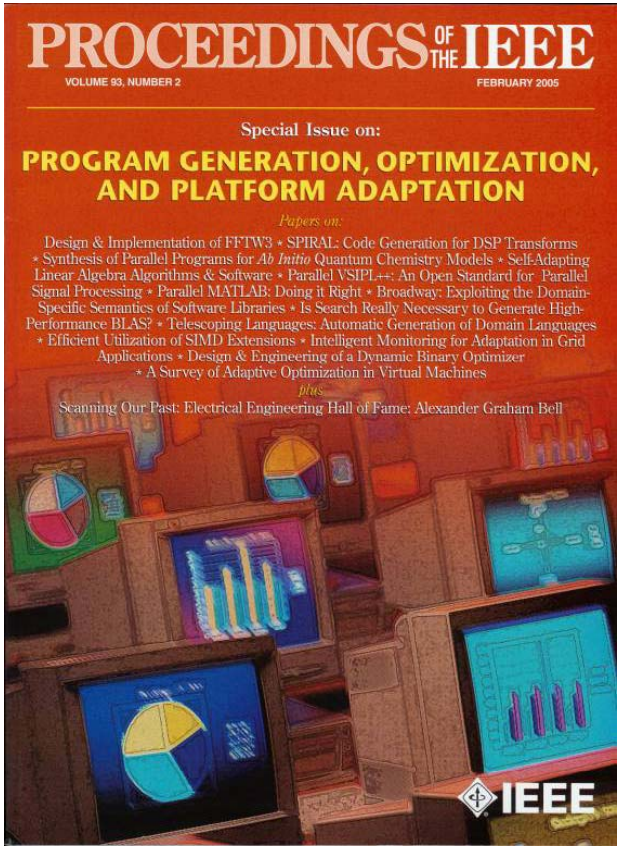
Code Must Be Vectorized to Attain High Performance

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: Spiral
  - …others
  - New compiler techniques

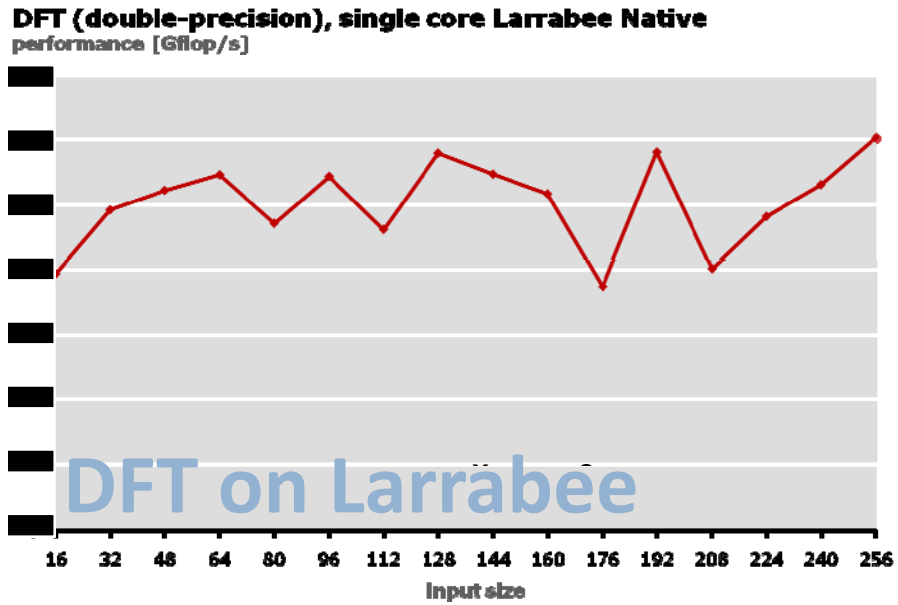
New challenge: ubiquitous parallelism



Proceedings of the IEEE special issue, Feb. 2005

Pre-Silicon Optimization: Larrabee and AVX

```
void dft64(float  *Y, float  *X) {
    __m512 U912, U913, U914, U915, U916, U917, U918, U919, U920, U921, U922, U923, U924, U925,...
 __m512  *a2153, *a2155;
    a2153 = ((__m512  *) X);  s1107 = *(a2153);
    s1108 = *((a2153 + 4));   t1323 = _mm512_add_ps(s1107,s1108);
    t1324 = _mm512_sub_ps(s1107,s1108);
    ...
    U926 = _mm512_swizupconv_r32(_mm512_set_1to16_ps(0.70710678118654757),_MM_SWIZ_REG_CDAB);
    s1121 = _mm512_madd231_ps(_mm512_mul_ps(_mm512_mask_or_pi(
        _mm512_set_1to16_ps(0.70710678118654757),0xAAAA,a2154,U926),t1341),
        _mm512_mask_sub_ps(_mm512_set_1to16_ps(0.70710678118654757),0x5555,a2154,U926),
        _mm512_swizupconv_r32(t1341,_MM_SWIZ_REG_CDAB));
    U927 = _mm512_swizupconv_r32(_mm512_set_16to16_ps(0.70710678118654757, (-0.70710678118654757),
        0.70710678118654757, (-0.70710678118654757), 0.70710678118654757, (-0.70710678118654757),
        0.70710678118654757, (-0.70710678118654757), 0.70710678118654757, (-0.70710678118654757),
        0.70710678118654757, (-0.7071067811865475
        0.70710678118654757, (-0.7071067811865475
    ...
    s1166 = _mm512_madd231_ps(_mm512_mul_ps(_mm51
        0.70710678118654757, (-0.7071067811865475
        0.70710678118654757, (-0.7071067811865475
        0.70710678118654757, (-0.7071067811865475
        0.70710678118654757, (-0.7071067811865475
        0xAAAA,a2154,U951),t1362),
        _mm512_mask_sub_ps(_mm512_set_16to16_ps(0
        (-0.70710678118654757), 0.707106781186547
        (-0.70710678118654757), 0.707106781186547
        (-0.70710678118654757), 0.707106781186547
        (-0.70710678118654757), 0.707106781186547
        _mm512_swizupconv_r32(t1362,_MM_SWIZ_REG_
    ...
}
```



Not actual data (NDA)

Outline

- General Vectorization Challenges
- Spiral
- Vectorization in Spiral
- Vectorization Challenges with AVX and Larrabee
- Results
- Conclusions

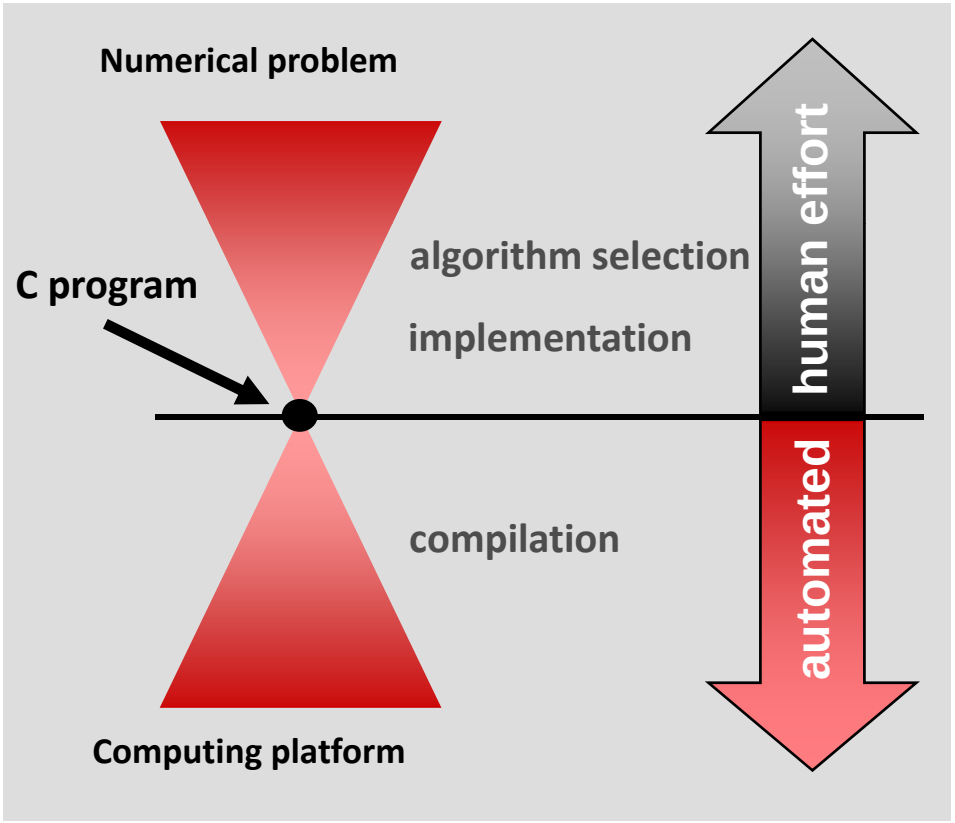
Spiral

- **Library generator for linear transforms (DFT, DCT, DWT, filters, ….) *and recently more …***
- **Wide range of platforms supported: scalar, fixed point, vector, parallel, Verilog, GPU**
- **Research Goal: "Teach" computers to write fast libraries**
  - Complete automation of implementation and optimization
  - Conquer the "high" algorithm level for automation
- **When a new platform comes out: Regenerate a retuned library**
- **When a new platform paradigm comes out (e.g., CPU+GPU): Update the tool rather than rewriting the library**

Intel uses Spiral to generate parts of their MKL and IPP libraries

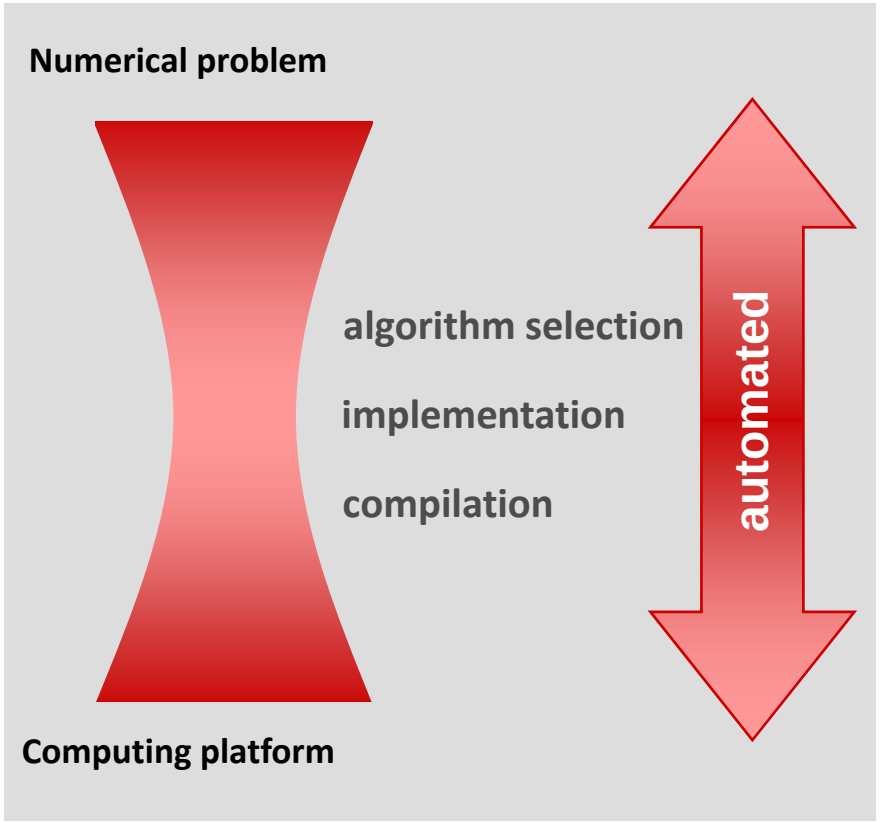
Vision Behind Spiral

Current



- C code a singularity: Compiler has no access to high level information

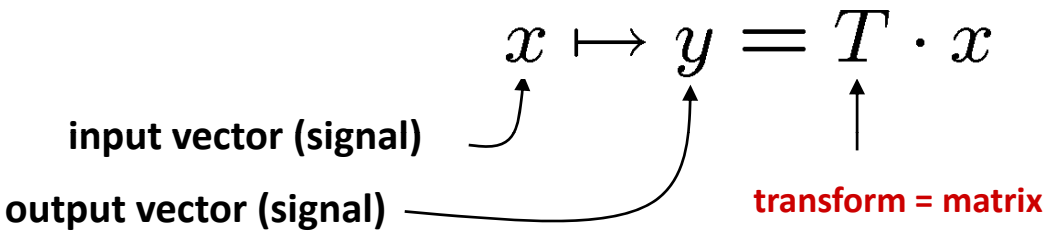
Future



- Challenge: conquer the high abstraction level for complete automation

What is a (Linear) Transform?

- **Mathematically: Matrix-vector multiplication**



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

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

Transform Algorithms: Example 4-point FFT

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)

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

Kronecker product — Identity — Permutation

- Algorithms reduce arithmetic cost $O(n^2) \rightarrow O(n\log(n))$
- Product of structured sparse matrices
- Mathematical notation exhibits structure: SPL (signal processing language)

Program Generation in Spiral (Sketched)

SPIRAL www.spiral.net

Transform
user specified

$$\mathrm{DFT}_8$$

Fast algorithm in SPL
many choices

$$(\mathrm{DFT}_2 \otimes I_4)\, T^8_4\, (I_2 \otimes ((\mathrm{DFT}_2 \otimes I_2) \cdot T^4_2\, (I_2 \otimes \mathrm{DFT}_2)\, L^4_2))\, L^8_2$$

∑-SPL:

$$\sum \left(S_j\, \mathrm{DFT}_2\, G_j\right) \sum \left(\sum \left(S_{k,l}\, \mathrm{diag}\left(\mathrm{t}_{k,l}\right) \mathrm{DFT}_2\, G_l\right) \sum \left(S_m\, \mathrm{diag}\left(\mathrm{t}_m\right) \mathrm{DFT}_2\, G_{k,m}\right)\right)$$

C Code:

```
void sub(double *y, double *x) {
  double f0, f1, f2, f3, f4, f7, f8, f10, f11;
  f0 = x[0] - x[3];
  f1 = x[0] + x[3];
  f2 = x[1] - x[2];
  f3 = x[1] + x[2];
  f4 = f1 - f3;
  y[0] = f1 + f3;
  y[2] = 0.7071067811865476 * f4;
  f7 = 0.9238795325112867 * f0;
  f8 = 0.3826834323650898 * f2;
  y[1] = f7 + f8;
  f10 = 0.3826834323650898 * f0;
  f11 = (-0.9238795325112867) * f2;
  y[3] = f10 + f11;
}
```

Optimization at all abstraction levels

- **parallelization vectorization**
- **loop optimizations**
- **constant folding scheduling ……**

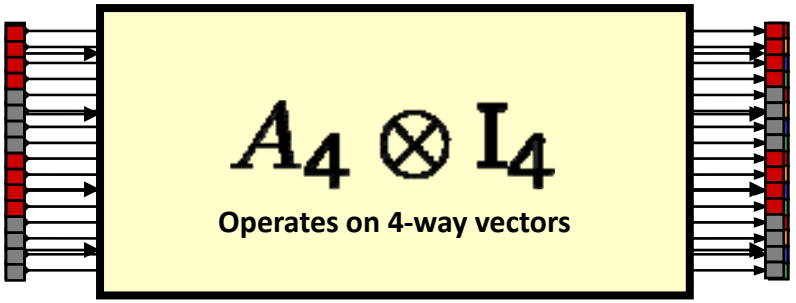
Vectorization In Spiral

- **Naturally vectorizable construct**

  Franchetti and Püschel (IPDPS 2002/2003)

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

vector length (any two-power)



- **Rewriting rules to vectorize formulas**

  Introduces data reorganization (permutations)

$$\mathrm{I}_n \otimes A^{k\times m} \rightarrow \mathrm{I}_{n/\nu} \otimes \mathbf{L}^{k\nu}_{\nu} \left(A^{k\times m} \otimes \mathrm{I}_\nu\right) \mathbf{L}^{m\nu}_{m}$$

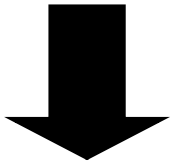
$$\mathbf{L}^{m\nu}_{m} \rightarrow \left(\mathrm{I}_{m/\nu} \otimes \mathbf{L}^{\nu^2}_{\nu}\right)\left(\mathbf{L}^{m}_{m/\nu} \otimes \mathrm{I}_\nu\right)$$

vector construct
further rewriting
base permutation

Vectorized DFT

Standard FFT

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



Automatic formula rewriting

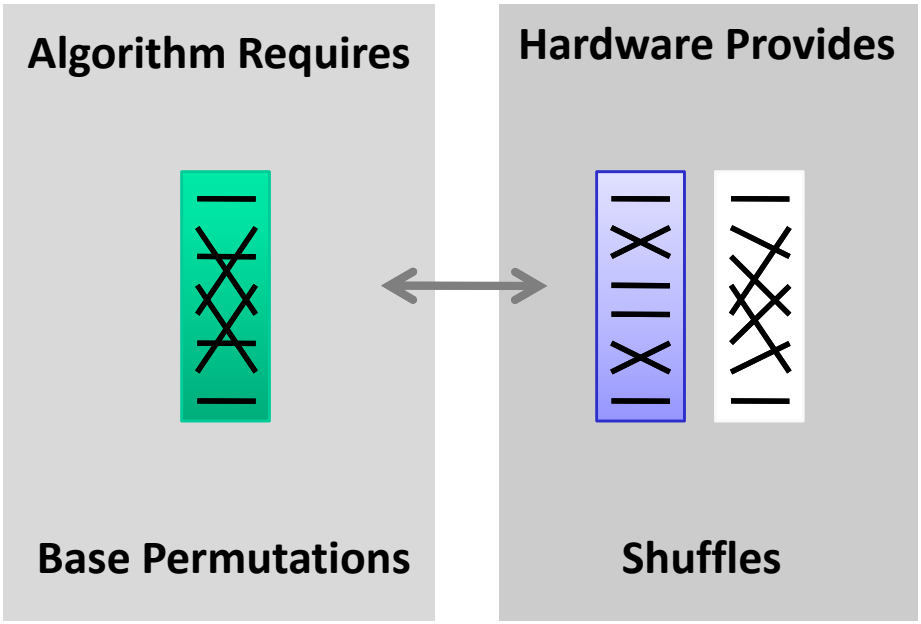
$$\mathbf{DFT}_{mn} \rightarrow \left(\mathrm{I}_{\frac{mn}{\nu}} \otimes \mathrm{L}^{2\nu}_{\nu}\right)\left(\overline{\mathbf{DFT}_m \otimes \mathrm{I}_{\frac{n}{\nu}}} \otimes \mathrm{I}_\nu\right)\overline{\mathrm{T}}'^{mn}_n$$
$$\left(\mathrm{I}_{\frac{m}{\nu}} \otimes \left(\mathrm{L}^{2n}_{\nu} \otimes \mathrm{I}_\nu\right)\left(\mathrm{I}_{\frac{2n}{\nu}} \otimes \mathrm{L}^{\nu^2}_{\nu}\right)\left(\mathrm{I}_{\frac{n}{\nu}} \otimes \mathrm{L}^{2\nu}_{2} \otimes \mathrm{I}_\nu\right)\left(\overline{\mathbf{DFT}}_n \otimes \mathrm{I}_\nu\right)\right)\left(\mathrm{L}^{\frac{mn}{\nu}}_{\frac{m}{\nu}} \otimes \mathrm{L}^{2\nu}_{2}\right)$$

Vectorized arithmetic

Architecture specific data reorganization

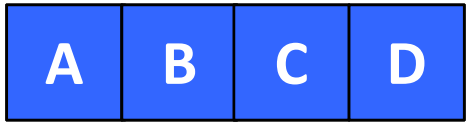
$$\mathrm{L}^{2\nu}_{2},\ \mathrm{L}^{2\nu}_{\nu},\ \mathrm{L}^{\nu^2}_{\nu}$$

Base Permutations To Code



Goal: Find Fast Instruction Sequences That Implement Base Permutations

Base Permutation Example: $L^{2\nu}_{\nu}$ ν=4



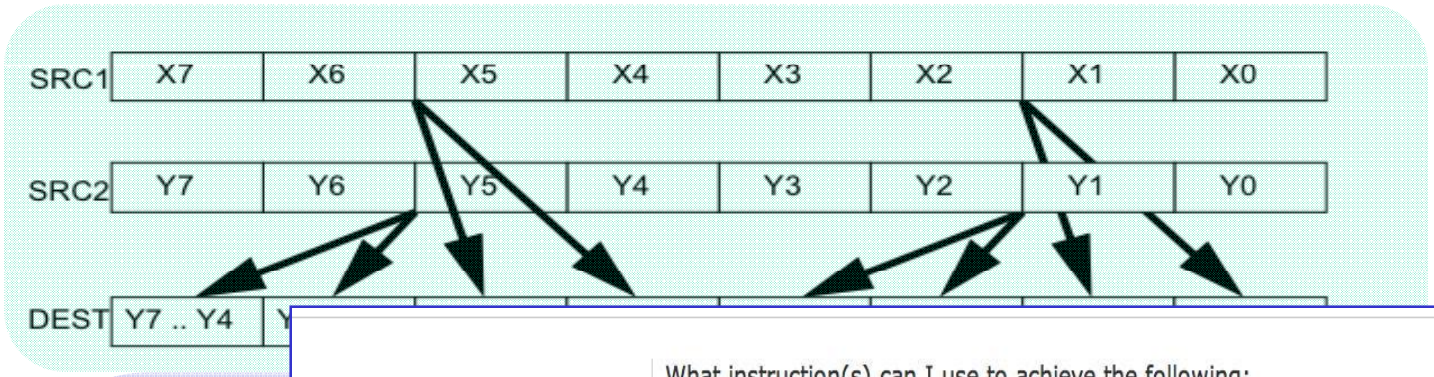
$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

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

```
y0 = _mm_unpacklo_ps(x0, x1);

y1 = _mm_unpackhi_ps(x0, x1);
```

A Tale of Two Shuffles



Some Date

Anonymous

What instruction(s) can I use to achieve the following:
_M512 va ={a0,a1,a2,a3, a4,a5,a6,a7, a8,a9,a10,a11, a12,a13,a14,a15} ;
_M512 vb ={b0,b1,b2,b3,b4,b5,b6,b7,b8,b9,b10,b11,b12,b13,b14,b15} ;

and I want

_M512 vc ={a0,b0, a1,b1, a2,b2, a3,b3, a4,b4, a5,b5, a6,b6, a7,b7};

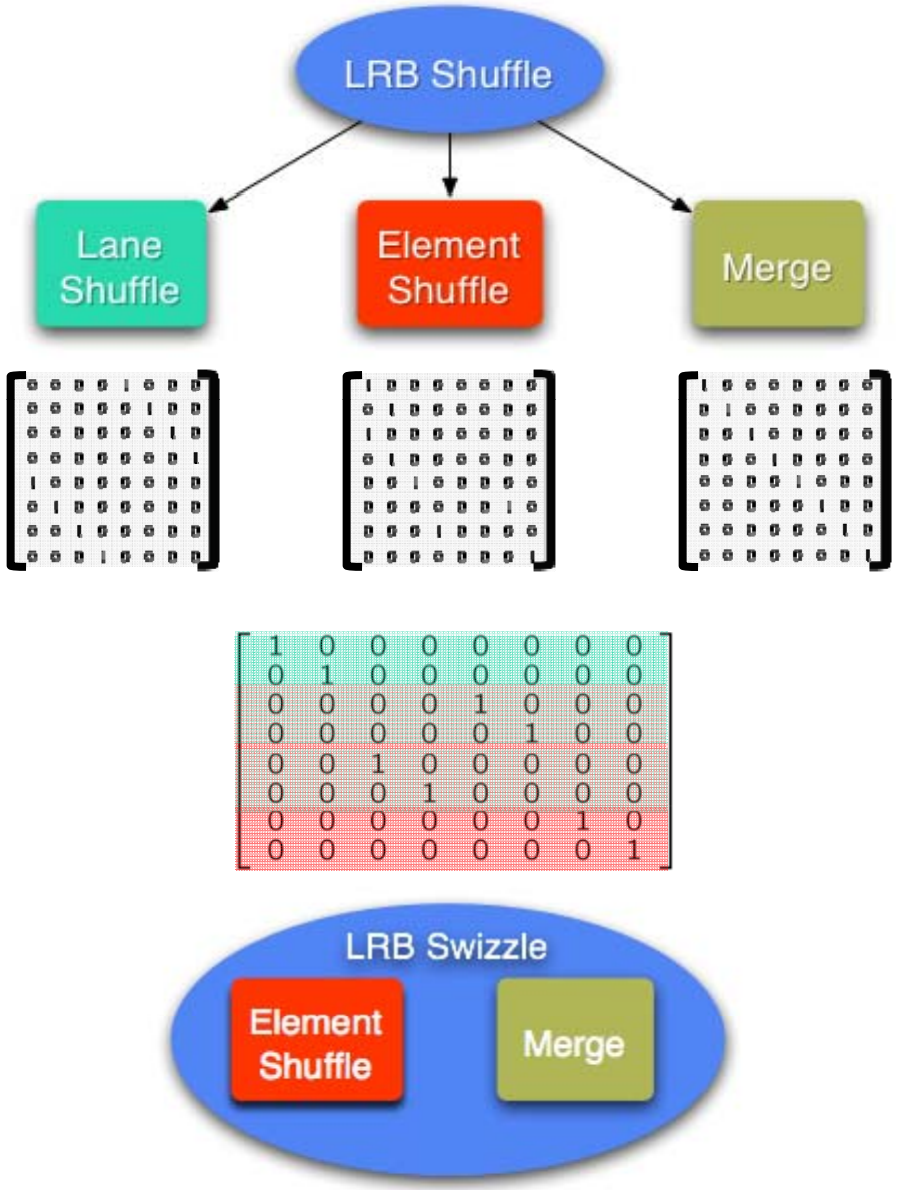
What if I want vc to have some random selection of the float values in va and vb?
The shuffle instruction (_mm512_shuf128x32) does not seem to be able to do that.

Implementing Base Permutation Matrices

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

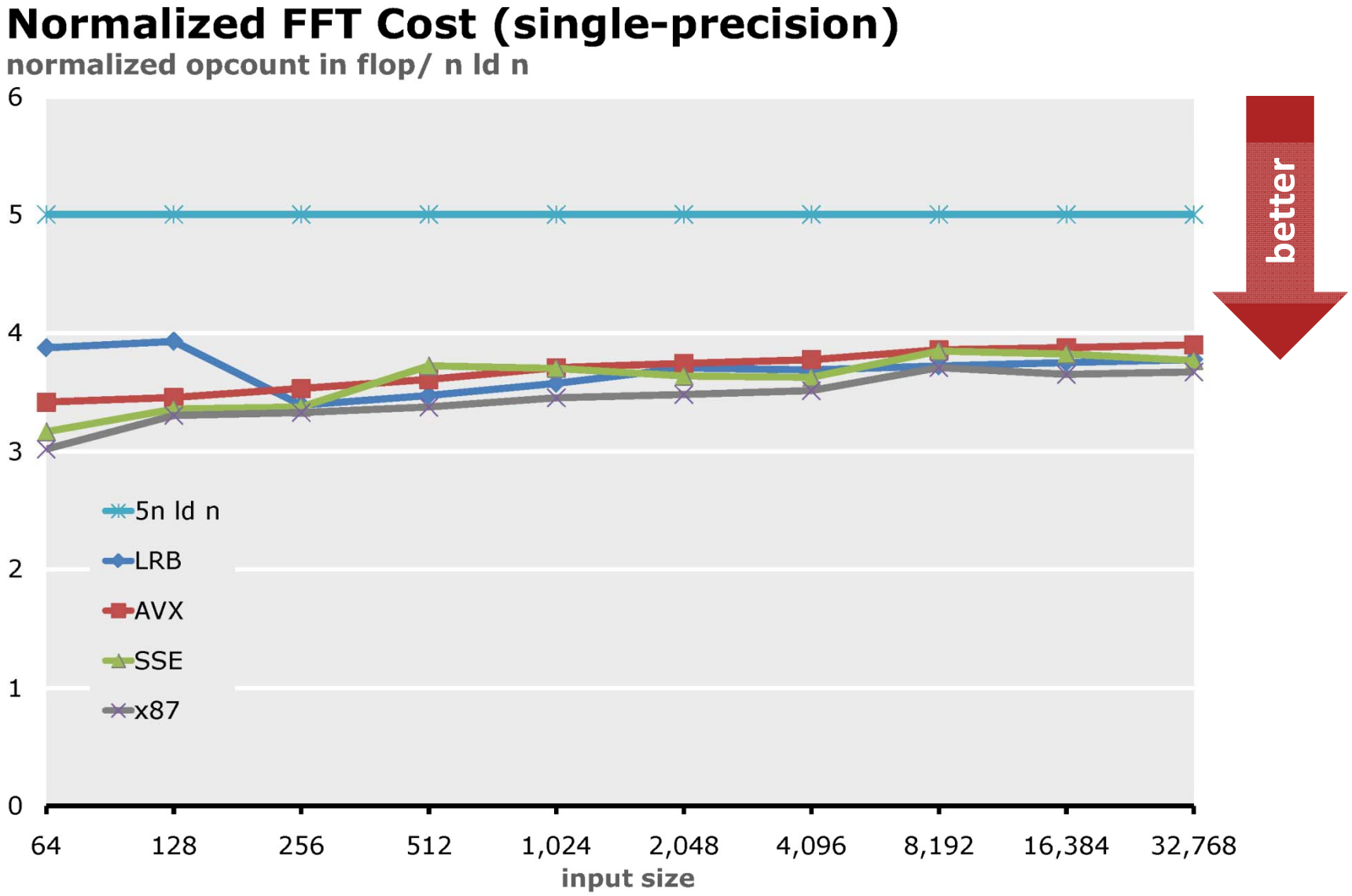
- **Equivalent to Factoring A Binary Matrix**
  - Computationally Intractable
- **Encode instructions as matrices**
  - Generate and multiply matrix sequences
  - Minimize sequence length and cost
  - Four Billion Variants of the LRB Shuffle Instruction!

Search Space Optimization

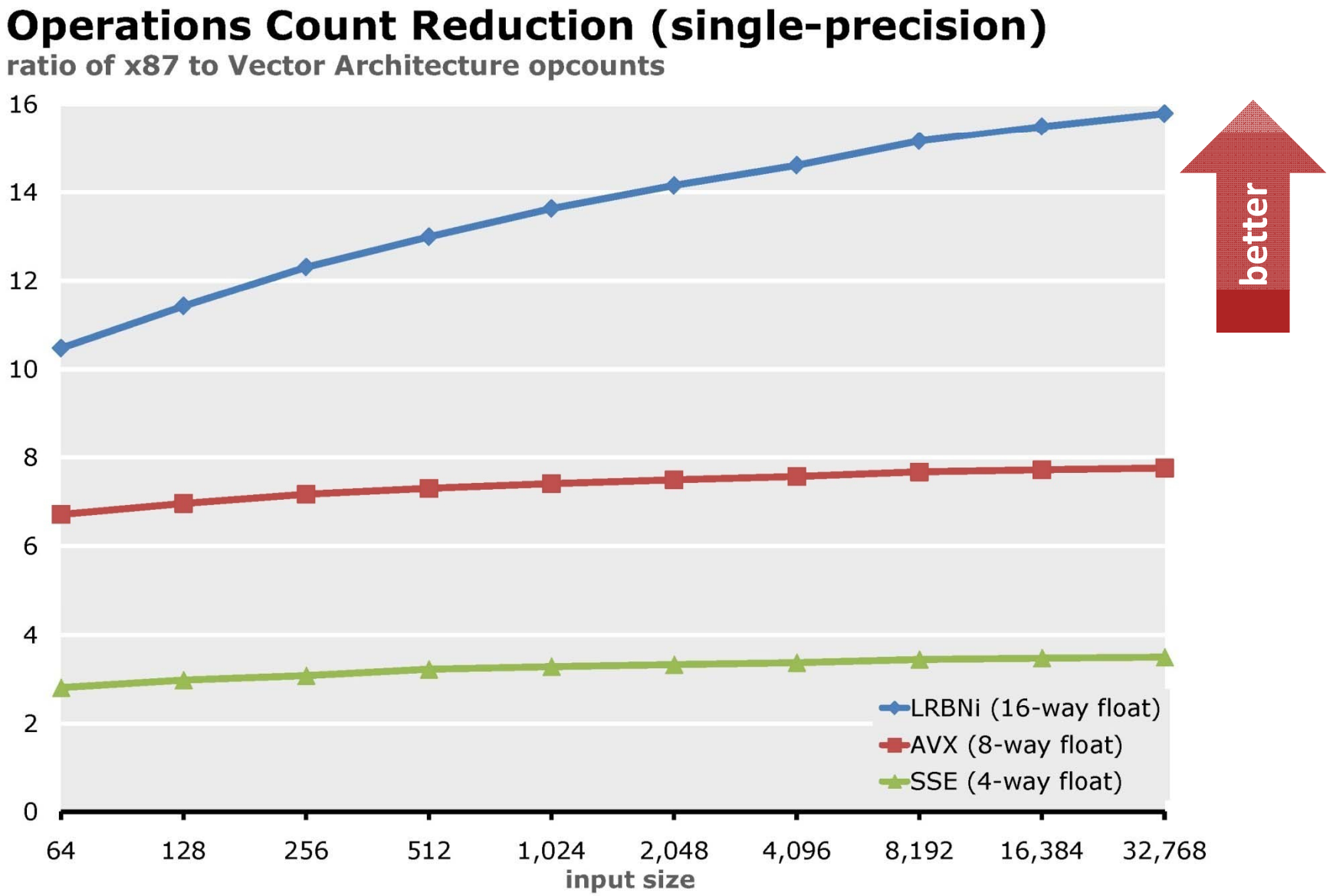


- **Micro-Op Reduction**
  - Use heuristics to filter µ-ops
- **Fast Binary Matrix Multiply**
- **Parallelize Search Space**
- **Strength Reduction**
  - Micro-Op Fusion
  - Pattern Matching
  - Prefix Trees
    - unpackhi/lo

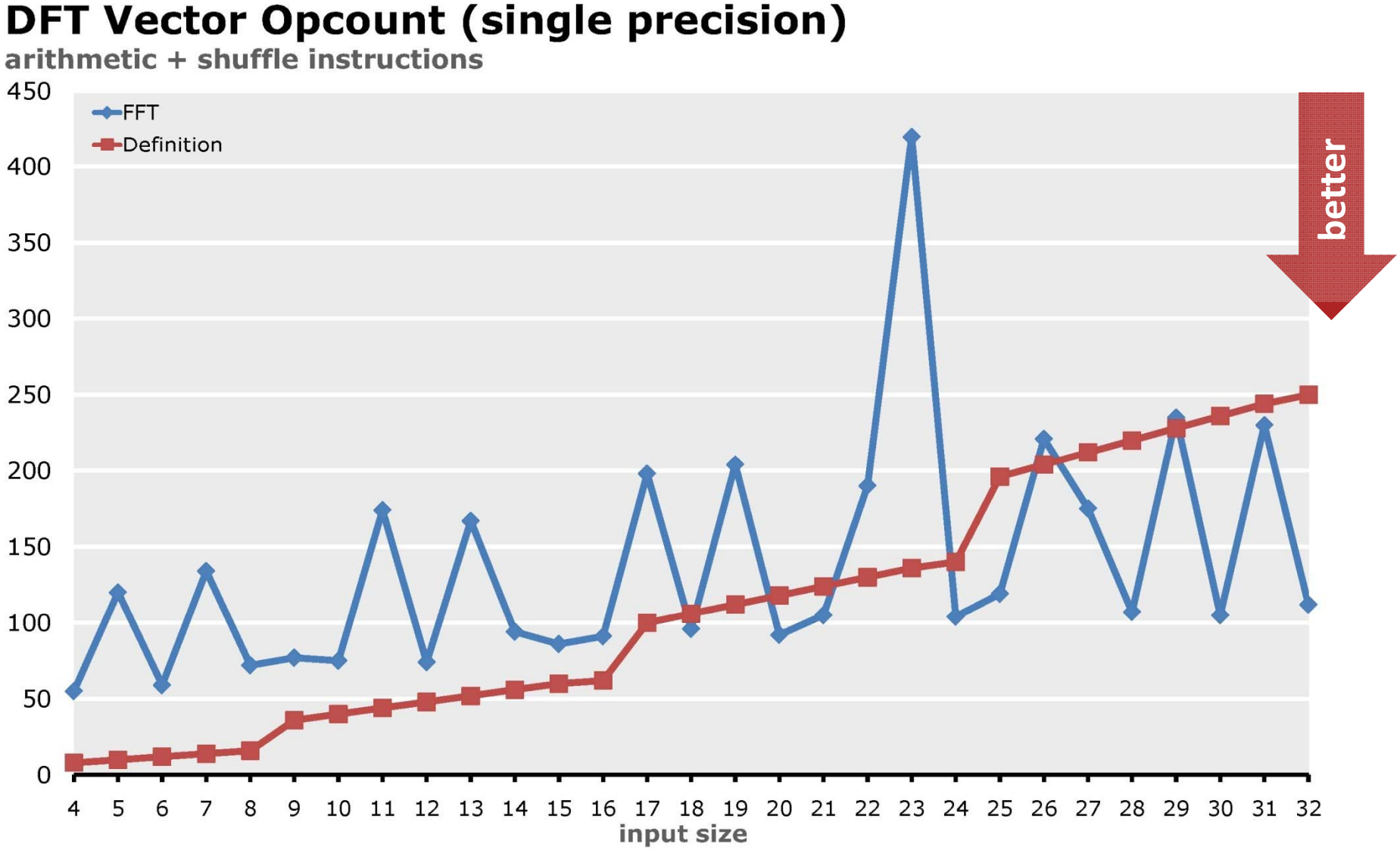
Results: Numerical Cost



Results: Vectorization Efficiency



Results: Flexible Algorithms



Conclusions

- New processors increase vector ISA power but also ISA complexity
- Reordering operations dominate vector performance
- Minimize reordering op cost through optimized search
- Fully Automate Spiral Vectorization Framework
- Enable production of High Quality Code