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

Setting the Pace for Intel Instruction Set

A new trajectory –
Vector FP offers scalable FLOPS and impressive Flops/W

The old trend –
15% gain/year (due to frequency and uArch)

Next generation Intel® microarchitecture (Nehalem)
- Superior Memory latency+ BW
- Fast Unaligned Support

Intel® microarchitecture (Westmere)
- Cryptography acceleration instructions

Intel® AVX*
- 2X throughput – vector FP
- 2X load throughput
- 3-operand instructions

Future Extensions
- Hardware FMA (Fused Multiply Add)
- Many more features

All timeframes, dates and products are subject to change without further notification

* Intel® AVX = Intel® Advanced Vector Extensions (Intel® AVX)

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

One can easily slow down a program by vectorizing it
Larrabee

- 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*
Pre-Silicon Optimization: Larrabee and AVX

```c
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, (-0.70710678118654757), 0.70710678118654757, (-0.70710678118654757),
                                0.70710678118654757, (-0.70710678118654757), 0.70710678118654757, (-0.70710678118654757),
                                0.70710678118654757, (-0.70710678118654757)), _MM_SWIZ_REG_CDAB);
    ...
    s1166 = __mm512_madd231_ps(__mm512_mul_ps(__mm512_mask_or_pi(__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, (-0.70710678118654757), 0.70710678118654757, (-0.70710678118654757),
                                0.70710678118654757, (-0.70710678118654757), 0.70710678118654757, (-0.70710678118654757),
                                0.70710678118654757, (-0.70710678118654757)), 0xAAAA, a2154, U951), t1362),
                                __mm512_mask_sub_ps(__mm512_set_16to16_ps(0 (-0.70710678118654757), 0.707106781186547.
                                (-0.70710678118654757), 0.70710678118654757, (-0.70710678118654757), 0.70710678118654757, (-0.70710678118654757),
                                0.70710678118654757, (-0.70710678118654757)), _MM_SWIZ_REG_CDAB);
    ...
}
```

DFT (double-precision), single core Larrabee Native performance [GFlop/s]
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**

- Numerical problem
  - C program
  - algorithm selection
  - implementation
  - compilation

**Future**

- Numerical problem
  - Challenge: conquer the high abstraction level for complete automation
  - algorithm selection
  - implementation
  - compilation
  - automated

- Challenge: conquer the high abstraction level for complete automation

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

- Automated
What is a (Linear) Transform?

- Mathematically: Matrix-vector multiplication

\[ x \xrightarrow{\text{transform}} y = T \cdot x \]

- Example: Discrete Fourier transform (DFT)

\[ \text{DFT}_n = [e^{-2\pi ik\ell/n}]_{0 \leq k, \ell < n} \]
Transform Algorithms: Example 4-point FFT

Cooley/Tukey fast Fourier transform (FFT):

\[
\begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & j & -1 & -j \\
1 & -1 & 1 & -1 \\
1 & -j & -1 & j
\end{bmatrix} = \begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1
\end{bmatrix} \begin{bmatrix}
1 & 1 & 1 & 1 \\
-1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1
\end{bmatrix} \begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1
\end{bmatrix} \begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1
\end{bmatrix}
\]

Fourier transform
Diagonal matrix (twiddles)

\[
\text{DFT}_4 = (\text{DFT}_2 \otimes I_2) \Lambda_2^4 (I_2 \otimes \text{DFT}_2) L_2^4
\]

\begin{itemize}
  \item Algorithms reduce arithmetic cost $O(n^2) \rightarrow O(n\log(n))$
  \item Product of structured sparse matrices
  \item Mathematical notation exhibits structure: SPL (signal processing language)
\end{itemize}
Program Generation in Spiral (Sketched)

Transform
user specified

Fast algorithm
in SPL
many choices

Optimization at all abstraction levels

\[
\text{DFT}_8
\]

\[
(D\text{FT}_2 \otimes I_4) T^8_4 (I_2 \otimes ((D\text{FT}_2 \otimes I_2) \\
\cdot T^4_2 (I_2 \otimes D\text{FT}_2) L^4_2)) L^8_2
\]

\[
\sum (S_j D\text{FT}_2 G_j) \sum \left( \sum (S_{k,l} \text{diag}(t_{k,l}) D\text{FT}_2 G_l) \right) \\
\sum (S_m \text{diag}(t_m) D\text{FT}_2 G_{k,m})
\]

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 * f1;
    f7 = 0.92387953525112867 * f0;
    f8 = 0.3826834323650898 * f2;
    y[1] = f7 + f8;
    f10 = 0.3826834323650898 * f0;
    f11 = (-0.92387953525112867) * f2;
    y[3] = f10 + f11;
}

parallelization
vectorization
loop optimizations
constant folding
scheduling
Vectorization in Spiral

- **Naturally vectorizable construct**
  Franchetti and Püschel (IPDPS 2002/2003)

\[ y = (A \otimes I_\nu)x \]

vector length (any two-power)

- **Rewriting rules to vectorize formulas**
  Introduces data reorganization (permutations)

\[
I_n \otimes A^{k \times m} \rightarrow I_{n/\nu} \otimes L^{k\nu}_\nu \left( A^{k \times m} \otimes I_\nu \right) L^{m\nu}_m
\]

\[
L^{m\nu}_m \rightarrow \left( I_{m/\nu} \otimes L^{\nu^2}_\nu \right) \left( L^{m}_{m/\nu} \otimes I_\nu \right)
\]

vector construct
data reorganization (permutations)
Vectorized DFT

Standard FFT

\[ \text{DFT}_{mn} \rightarrow (\text{DFT}_m \otimes I_n) T^m_n (I_m \otimes \text{DFT}_n) L^m_m \]

Automatic formula rewriting

\[ \text{DFT}_{mn} \rightarrow \left( I_{mn} \otimes L^2_{2\nu} \right) \left( \text{DFT}_m \otimes I^2_n \otimes I_{\nu} \right) T^m_n \]

- Vectorized arithmetic
- Architecture specific
- Data reorganization

\[ L^2_{2\nu}, L^2_{\nu}, L^{\nu^2} \]
Base Permutations To Code

Algorithm Requires

Hardware Provides

Base Permutations

Shuffles

Goal: Find Fast Instruction Sequences That Implement Base Permutations
\[ L_{2^4}^{2^4} \]

Base Permutation Example:

\begin{align*}
A & B & C & D \\
E & F & G & H
\end{align*}

\[
y_0 = \_mm\_unpacklo\_ps(x_0, x_1); \\
y_1 = \_mm\_unpackhi\_ps(x_0, x_1);
\]
A Tale of Two Shuffles

What instruction(s) can I use to achieve the following:

```c
_MM512 va = {a0, a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11, a12, a13, a14, a15};
_MM512 vb = {b0, b1, b2, b3, b4, b5, b6, b7, b8, b9, b10, b11, b12, b13, b14, b15};
```

and I want

```c
_MM512 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_shufflef128x32`) does not seem to be able to do that.
Implementing Base Permutation Matrices

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
= \begin{bmatrix}
? \\
? \\
\end{bmatrix}
\]

- 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 $\mu$-ops

- **Fast Binary Matrix Multiply**

- **Parallelize Search Space**

- **Strength Reduction**
  - Micro-Op Fusion
  - Pattern Matching
  - Prefix Trees
    - unpackhi/lo
Results: Numerical Cost

Normalized FFT Cost (single-precision)

(normalized opcount in flop/ n ld n)

- 5n ld n
- LRB
- AVX
- SSE
- x87

Input size

64 128 256 512 1,024 2,048 4,096 8,192 16,384 32,768

better
Results: Vectorization Efficiency

Operations Count Reduction (single-precision)

ratio of x87 to Vector Architecture opcounts

better

LRBNi (16-way float)
AVX (8-way float)
SSE (4-way float)
Results: Flexible Algorithms

DFT Vector Opcount (single precision)

arithmetic + shuffle instructions

- FFT
- Definition

input size
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