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

Daniel McFarlin, Franz Franchetti, Markus Püschel
{dmcfarli,franzf,pueschel}@ece.cmu.edu
Electrical and Computer Engineering, Carnegie Mellon University

Introduction

The discrete Fourier transform (DFT) and its fast algorithms (fast Fourier transforms or FFTs) are among the most important computational building blocks in signal processing and scientific computing. Consequently, there is a number of high performance DFT libraries available including Intel's Integrated Performance Primitives (IPP), FFTW [6], and libraries generated by Spiral [9, 10]. When optimizing a DFT library, all the latest performance-enhancing processor features have to be used.

Since the introduction of Intel's SSE and AltiVec/VMX on PowerPCs, DFT libraries have to be tuned for single instruction multiple data (SIMD) vector instructions. These instructions pack multiple smaller data words (for instance, four 32-bit floating-point numbers) into wide registers (in our example 128-bit wide). While these instructions provide the potential for tremendous speed-up, using them is challenging: vector instructions impose many restrictions and must be carefully selected to provide actual speed-up. Unavoidable overhead due to data alignment and reorganization often diminishes the performance gains and sometimes make vector code uncompetitive.

Intel recently released the definition of two new vector instruction sets:

- Advanced Vector Extension (AVX) [1], the successor of SSE, defines 256-bit registers that can be used as 8-way single-precision vectors and 4-way double-precision vectors, and defines fused multiply-add (FMA) instructions. It is announced for the Sandy Bridge processor family (2010 timeframe).
- Intel's Larrabee graphics processor is based on the Larrabee native instruction (LRBni) [7, 8] set that defines 512-bit vectors usable as 16-way single-precision or 8-way double-precision vectors. LRBni also defines FMA instructions.

These two new vector instruction sets require a complete redesign of performance libraries, including DFT libraries. The long vector lengths (8 and 16) pose a particular challenge in FFTs, due to their intrinsically complicated data access patterns. The additional FMA instructions (introduced for the first time in the Intel architecture) further complicate matters. The instructions defined by AVX and LRBni are complicated and heavily parameterized, making them powerful, yet hard to use. The challenge posed to library developers by these new instruction sets is compounded by the fact that actual hardware implementing the instruction sets is not yet available. This makes performance optimization very difficult.

Related work. Intel's IPP and MKL, Mercury's SAL, and IBM's ESSL are (assembly level) hand-optimized libraries that provide highly optimized FFT implementations for the respective target processors. These libraries support SSE and AltiVec/VMX, respectively. FFTW [6] provides an adaptive FFT library that supports SSE, 3DNow!, and AltiVec. Vectorizing Compilers like Intel's C++ compiler and the Gnu C compiler provide automatic vectorization [3], which typically fails on FFT code.

SIMD Vectorization in Spiral

In this section we give an overview of how we extended the Spiral library generator to support AVX and LRBni, and how we optimized for these instruction sets at the pre-silicon stage.

Spiral. Spiral automates the generation of high-performance software libraries for the domain of linear transforms including the DFT. It generates software that takes advantage of different forms of parallelism, while at the same time matching the performance of hand-written code. Spiral relies on two fundamental building blocks 1) A domain-specific, declarative, mathematical language to describe algorithms; and 2) the use of rewriting to parallelize and optimize algorithms at a high level of abstraction.

Symbolic vectorization. Spiral applies rewriting to automatically vectorize FFT algorithms symbolically. This enables algorithm transformations that are beyond the reach of compilers, and are challenging for human programmers. A overview on Spiral's SIMD vectorization can be found in [4].

The basic idea is that Spiral performs algorithm-level optimizations to extract maximally vectorizable blocks while only using a small number of vector shuffle blocks. An example result is the short vector Cooley-Tukey FFT [4] that is parameterized by the vector length ν. It shows that for all sizes N with $\nu^2/4 \mid N$, the DFT can be implemented using only vector additions, subtractions, and, multiplications, and a small set of data reorderings described by the following permutations:

$$\mathrm{L}^{2\nu}_{\nu}, \quad \mathrm{L}^{2\nu}_{2}, \quad \mathrm{L}^{\nu^2}_{\nu}, \quad \text{and} \quad \mathrm{L}^{\nu^2/4}_{\nu/2} \otimes \mathrm{I}_2$$

*This work was supported by NSF through awards 0325687, 0702386, by DARPA (DOI grant NBCH1050009), ARO grant W911NF0710416, and by the Intel Corporation. We thank Randi Rost and Scott Buck (Intel Education) and Boris Sabanin and Andrey Bakshaev (Intel IPP) for enabling our Larrabee work.

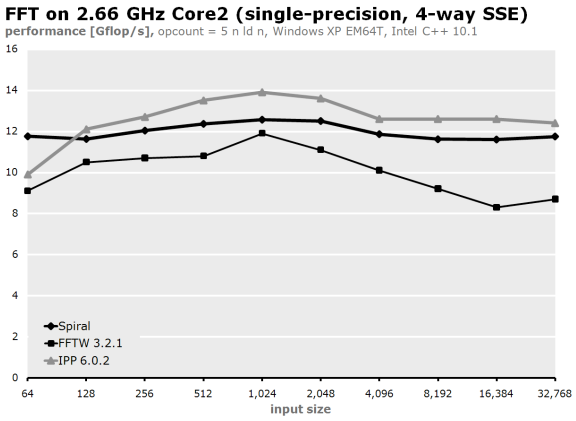

Figure 1: Performance results on a 2.66 GHz Core2 Duo (single core). Higher is better.

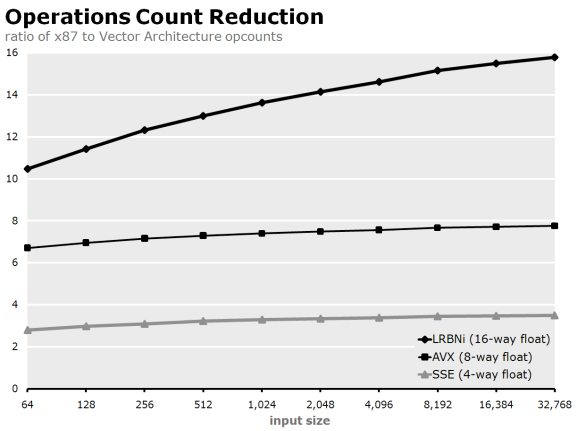

Figure 2: Instruction count reduction of Spiral-generated FFT functions for SSE, AVX, and LRBni over x87. Higher is better.

each of which is done using in-register permutations. Above, L_m^{mn} can be viewed as a transposition of an $m \times n$ matrix.

Extending Spiral to AVX and LRBni. The major effort in extending Spiral to support AVX and LRBni was to find efficient implementations of these permutations. We had to extend the approach we used for SSE [5], since AVX and Larrabee instructions have a much larger parameter space and are much more complex. To find the required short instruction sequences we were running extensive searches.

In addition to finding good permutation implementations, we needed to build an AVX emulator from the instruction specification since no emulation library was available at the time of this work. We targeted the Larrabee Prototype Primitives [7] with Spiral to emulate LRBni instructions. We also extended the vector FMA support in Spiral that was developed for the Cell BE [2].

Experimental Results

We now evaluate the performance improvement achievable with AVX and LRBni. We used Spiral to generate highly optimized SSE implementations for DFTs of size $64, \ldots, 32,768$. We compare these implementations to Intel's IPP and FFTW to establish the quality of our base line. Then we evaluate the operations count reduction by using the 8-way AVX instructions and the 16-way LRBni instructions instead of the 4-way SSE instructions. Since actual hardware is not yet available we instruct Spiral to minimize the instruction count, and use the instruction count reduction as performance metric.

Fig. 1 compares the performance of Spiral-generated SSE implementations to Intel's IPP 6.0.2 and FFTW 3.2.1 on a 2.66 GHz Core2 (65nm), using the Intel C++ compiler 10.1 on Windows XP 64-bit. Spiral-generated FFT functions are within 10% of the respective IPP functions and somewhat faster than FFTW 3.2.1.

Fig. 2 shows the reduction in operations by using various vector instruction sets. We only count arithmetic vector operations and vector shuffle operations, but no memory and indexing operations to not require a target compiler. The reduction gets larger for larger transform sizes since the overhead from shuffle operations is linear in the transform size, and the arithmetic operations count is roughly cut linearly by the vector length. 4-way SSE provides between 2.5x and 3.3x reduction over x87. 8-way AVX provides between 6x and 7.75x reduction. 16-way LRBni provides between 10x and 15.75x reduction. This shows that substantial operations count reduction for FFT implementations is possible by using AVX and LRBni.

Conclusion

In this paper we extended the program generator Spiral to generate optimized vector code for the Larrabee and AVX instruction set. We verify the generated code using software emulation of the new instructions. Spiral's feedback loop optimizing the generated code minimizes instruction counts since actual runtime is not yet available. We show that a substantial reduction in operations count is achievable for FFT functions by using the new instruction sets.

References

[1] Intel Advanced Vector Extensions programming reference, 2008. http://software.intel.com/en-us/avx/.

[2] Srinivas Chellappa, Franz Franchetti, and Markus Püschel. Computer generation of fast Fourier transforms for the cell broadband engine. In *International Conference on Supercomputing (ICS)*, 2009.

[3] Alexandre E. Eichenberger, Peng Wu, and Kevin O'Brien. Vectorization for SIMD architectures with alignment constraints. *SIGPLAN Not.*, 39(6):82–93, 2004.

[4] F. Franchetti, Y. Voronenko, and M. Püschel. A rewriting system for the vectorization of signal transforms. In *Proc. High Performance Computing for Computational Science (VECPAR)*, 2006.

[5] Franz Franchetti and Markus Püschel. Generating SIMD vectorized permutations. In *International Conference on Compiler Construction (CC)*, volume 4959 of *Lecture Notes in Computer Science*, pages 116–131. Springer, 2008.

[6] Matteo Frigo and Steven G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Adaptation".

[7] C++ Larrabee Prototype Library, 2009. http://software.intel.com/en-us/articles/prototype-primitives-guide.

[8] A first look at the Larrabee New Instructions (LRBni), 2009. http://www.ddj.com/hpc-high-performance-computing/216402188.

[9] Markus Püschel, José M. F. Moura, Jeremy Johnson, David Padua, Manuela Veloso, Bryan W. Singer, Jianxin Xiong, Franz Franchetti, Aca Gačić, Yevgen Voronenko, Kang Chen, Robert W. Johnson, and Nick Rizzolo. SPIRAL: Code generation for DSP transforms. *Proc. of the IEEE*, 93(2):232–275, 2005. Special issue on *Program Generation, Optimization, and Adaptation*.

[10] Spiral web site. www.spiral.net.