Short Vector SIMD Code Generation for DSP Algorithms*

Franz Franchetti† Markus Püschel‡ José M. F. Moura‡ Christoph W. Ueberhuber†

Abstract

Short vector SIMD instructions on recent general purpose microprocessors, such as SSE on Pentium III and 4, offer a high potential speed-up but require a very high level of programming expertise. We present a compiler that generates vectorized code for digital signal processing algorithms such as the fast Fourier transform (FFT). The input to our compiler is a mathematical description of the algorithm in the language SPL. SPL is used in the SPIRAL system to automatically generate and optimize code for a given computing platform. The output of our compiler is a C function enhanced with vector instructions. Interfacing our compiler with SPIRAL yields speed-ups of more than a factor of 2 in several important cases including the FFT and the DCT used in the JPEG compression standard. For the FFT, our automatically generated code is competitive with the hand-coded Intel Math Kernel Library.

Short Vector SIMD Instructions

One of the recent trends in the evolution of general purpose microprocessors is the introduction of short vector SIMD (single instruction multiple data) instructions into their instruction set architecture (ISA) to improve the performance of multi-media applications. Examples of SIMD extensions supporting both integer operations and floating-point operations include the Intel Streaming SIMD Extensions (SSE and SSE2), AMD 3DNow! (plus extensions) and the Motorola AltiVec extensions. Each of these ISA extensions is based on the packing of large registers (64-bits or 128-bits) with smaller data types and providing instructions for the parallel operation on these subwords within one register.

SIMD instructions have the potential to substantially speed up code for applications that exhibit the fine-grain parallelism necessary for using SIMD instructions. Examples of such applications include DSP algorithms such as the fast Fourier transform (FFT). Because of hardware restrictions and the specific structure of DSP algorithms, their efficient vectorization and implementation is a difficult problem that requires a high level of programming expertise. Main challenges include:

- Non-unit stride accesses typical for DSP algorithms and complex data types produce data access patterns that prevent a straightforward vectorization.
- A general purpose vectorizing compiler does not have access to the full structure of a DSP algorithm, and thus cannot find a satisfactory vectorization.
- No standard API exists for different extensions across architectures; thus assembly coding is required.

To overcome these problems we have developed a compiler that *generates* vectorized code for DSP algorithms using a *mathematical description* of the algorithm as input. This way, we have access to all structural information and can use formal manipulations to exhibit vectorizable parts. Furthermore, we have explicit access to all data access patterns, which allows us to carry out expensive load and store operations efficiently. Finally, we achieve portability, by using our own API (consisting of C macros) that is built only on commonly available vector instructions.

*This work was supported by DARPA through research grant DABT63-98-1-0004 administered by the Army Directorate of Contracting and by the Special Research Program SFB F011 "AURORA" of the Austrian Science Fund FWF.

†Department of Applied and Numerical Mathematics, Technical University of Vienna, Vienna, Austria

‡Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA

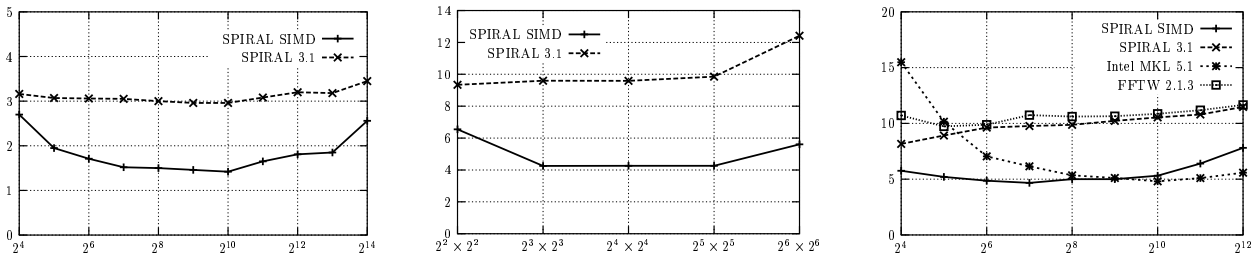

Figure 1: From left to right ($n = 2^k$): normalized runtimes for a real WHT of size $n = 2^4, \ldots, 2^{14}$, a real 2-D DCT of size $n \times n$, $n = 2^2, \ldots, 2^6$, and a complex FFT of size $n = 2^4, \ldots, 2^{12}$. The platform is a Pentium III, running Windows 2000, using the Intel C compiler 5.0.

A SIMD Vectorizing Compiler for DSP Algorithms

Our approach is based on the mathematical framework developed in SPIRAL. SPIRAL is a generator for platform-adapted libraries of DSP transforms. SPIRAL represents fast algorithms for DSP transforms as *formulas* in a symbolic mathematical language called SPL (signal processing language). These formulas are (1) automatically generated from a transform specification [2], and (2) automatically translated into optimized code in a high-level language like C or Fortran using the SPL compiler [3]. Platform adaptation is achieved by intelligently searching, for a given transform, the large space of possible algorithms, i.e., SPL formulas, for the fastest one.

In this paper we present an extension of the SPL compiler that generates vectorized code. The compiler first manipulates a given formula to exhibit maximal vectorizable components, then C code is generated containing inlined vector instructions for these components. By interfacing our compiler with SPIRAL we can then readily *search* for the best algorithm for the given platform. Figure 1 shows some runtime results for our automatically generated code for the WHT (Walsh-Hadamard transform), the 2-D DCT (discrete cosine transform) used in the JPEG image compression standard, and the FFT. Compared to C code generated by SPIRAL, or by FFTW [1] for FFTs, we achieve speed-ups of up to a factor of 2.1. For FFT sizes less than 1024 our generated code outperforms the hand-coded Intel Math Kernel Library.

We summarize the key points. (1) In contradistinction to other approaches, we are *not* implementing (and vectorizing) a specific transform algorithm (as, e.g., the FFT), but generate code for the mathematical constructs the algorithm is built from. This way, every algorithm using these constructs is automatically included. (2) Our generated code can not be produced by a general purpose vectorizing compiler, since it misses the high level information provided by the formula representation. (3) Using SPIRAL, the code generation (including verification) and algorithmic optimization is entirely automatic from the transform specification.

References

[1] M. Frigo and S. G. Johnson. FFTW: An adaptive software architecture for the FFT. In *Proc. ICASSP*, volume 3, pages 1381–1384, 1998. `http://www.fftw.org`.

[2] M. Püschel, B. Singer, M. Veloso, and J. M. F. Moura. Fast Automatic Generation of DSP Algorithms. In *Proc. ICCS*, LNCS 2073, pages 97–106. Springer, 2001.

[3] J. Xiong, J. Johnson, R. Johnson, and D. Padua. SPL: A Language and Compiler for DSP Algorithms. In *Proc. PLDI*, pages 298–308, 2001.