FFT Program Generation for the Cell BE*

Srinivas Chellappa, Franz Franchetti, and Markus Püschel

Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh PA 15213, USA
{schellap, franzf, pueschel}@ece.cmu.edu

1 Introduction

The Cell BE chip-multiprocessor (CMP) is designed for high-density floating point computation required in multimedia, visualization, and other similar applications. Its innovative design includes multiple SIMD vector cores (called synergistic processing elements, or SPEs) with explicitly managed per-core local memory and inter-core communication. The computational power of a single Cell BE chip is impressive: its single-precision peak performance is 204.8 Gflop/s for the 8 SPEs alone. However, the same features that allow for high theoretical performance make it difficult and time consuming to design and optimize specific real-world computational kernels for the Cell. Instead of using automated tools, these programs must explicitly address multithreading, SIMD vectorization, and data streaming in order to extract maximum performance.

In this paper we address the automation of program optimization for the Cell: we extend the program generation system Spiral [1] to support the Cell processor. Spiral automates the production, platform adaptation, and optimization of signal processing transform libraries and targets SIMD vector extensions [2], shared memory multicore CPUs [3], graphics processors (GPUs), and FPGAs. It is based on a domain specific, declarative, mathematical language to describe the algorithms, and uses rewriting to parallelize and optimize algorithms at a high level of abstraction.

We extend Spiral in two steps: First, we extend Spiral's SIMD vector program generation capabilities to support the SIMD instructions set of the Cell SPE. Second, we extend Spiral to support explicit DMA transfers and single-program-multiple-data multithreaded code to generate parallel multi-SPE implementations. The performance of our single-threaded code is comparable to the best available hand tuned code. We obtain about a 2x speed-up for a parallel 4-SPE implementation computing a single small DFT (16 to 4,096 data points) with all data resident in the SPEs' local stores. The presented work is preliminary. The final version of this paper will contain updated results.

Related Work. Several other projects have implemented specialized FFT libraries tuned for the Cell [4–7]. Their main focus is to optimize DFT computations of specific sizes where the data resides in main memory. FFTW [7] uses automated search techniques to produce its Cell FFT library.

* This work was supported by NSF through awards 0325687, 0702386, by DARPA through the Department of Interior grant NBCH1050009, and by Mercury Computer Systems, Inc.

2 Spiral

Spiral is a program generator for linear transforms including the discrete Fourier transform (DFT), the Walsh-Hadamard transform (WHT), discrete cosine and sine transforms, filters, and others. For a given transform (e.g., DFT of size 384), Spiral autonomously generates different algorithms, represented in a declarative form as mathematical formulas, and their implementations to find the best match for the target platform.

Fig. 1 shows the design of the Spiral system. Spiral uses *breakdown rules* to break down larger transforms into smaller kernels based on recursion. A large space of algorithms (formulas) for a single transform may be obtained using these breakdown rules. A formula thus obtained is structurally optimized to match the architecture using a rewriting system. The output is C code for a specific implementation of the transform. The performance of this implementation is measured and fed back into Spiral's formula generation and rewriting system. This allows Spiral to search over a large algorithm space. The final output of this feedback loop is highly optimized C code.

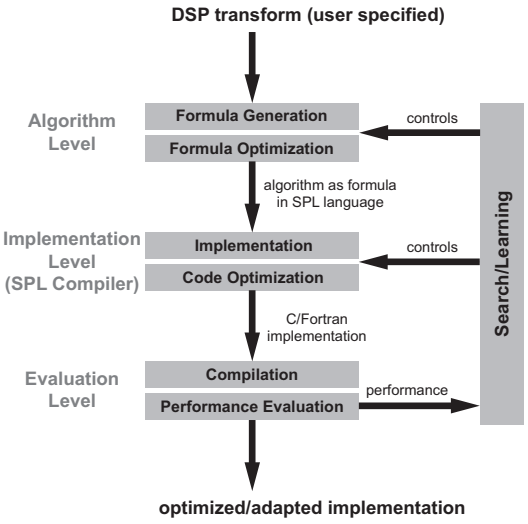

Fig. 1. Spiral's program generation system.

Formula representation. A linear transform in Spiral is represented by a transform matrix M, where performing the matrix-vector multiplication $y = Mx$ transforms the input vector x into the output vector y. Algorithms for transforms can be viewed as structured factorizations of the transform matrices. Such structures are expressed in Spiral using its own signal processing language (SPL), which is based on standard matrix operators in addition to the Kronecker (tensor) product. The Kronecker product $\otimes$ is defined as:

$$A \otimes B = [a_{k\ell}B],\ A = [a_{k\ell}].$$

Based on this, the well known Cooley-Tukey FFT algorithm's corresponding breakdown rule in Spiral is:

$$\mathrm{DFT}_{mn} \rightarrow (\mathrm{DFT}_m \otimes I_n) D_{m,n} (I_m \otimes \mathrm{DFT}_n) L^{nm}_m \tag{1}$$

where I_n is the identity matrix, $D_{m,n}$ the diagonal matrix, and L^{nm}_m, a stride permutation matrix.

3 Parallelization and Vectorization for the Cell

The key observation is that the tensor product representation of the transform algorithms in Spiral can be mapped to components of the target architecture. The tensor product can be viewed as a program loop. A loop featuring the appropriate structure can be implemented using multiple threads or SIMD vector instructions. For instance, in (1) the construct $I_n \otimes \mathrm{DFT}_m$ is an inherently parallel n-way loop, while the construct $\mathrm{DFT}_m \otimes I_n$ is easily translated into a SIMD vector loop.

We use formula rewriting to manipulate vector loops into parallel loops and vice versa, when mapping a formula fragment to the multicore and SIMD parallelism present in the Cell. To guide this rewriting process, we introduce the *tags* "simd(ν)" to denote ν-way SIMD vectorization and "threads(p, μ)" to denote p-way multithreaded code that sends packets of size μ. Using these tags allows us to generate vectorized, multithreaded code that is optimized for large DMA packet sizes for efficient inter-core communication.

Vectorization. We first discuss SIMD vectorization for the Cell, which is based on the ideas presented in [2]. Consider the construct $I_n \otimes \mathrm{DFT}_m$ in (1), which cannot immediately be translated into an efficient SIMD vector program. However, the rewriting rule

$$\underbrace{I_n \otimes \mathrm{DFT}_m}_{\text{simd}(\nu)} \rightarrow \underbrace{L^{mn}_n}_{\text{simd}(\nu)} \left((\mathrm{DFT}_m \otimes I_{n/\nu}) \overrightarrow{\otimes} I_\nu \right) \underbrace{L^{mn}_m}_{\text{simd}(\nu)}$$

translates it into a perfectly vectorizable construct (denoted using the symbol $\overrightarrow{\otimes}$), at the expense of data permutations which are handled subsequently by other rewriting rules.

Parallelization. Our goal is to generate multithreaded Cell code that computes a single 1D DFT in parallel across multiple SPEs. Such code requires all-to-all inter-core communication, which must be explicitly setup using DMA calls. Furthermore, DMA requests that use larger packet sizes achieve higher interconnect bandwidths on the Cell's EIB [8]. We generate optimized multithreaded code with large packet sizes using formula rewriting. The tag "threads(p, μ)" steers the rewriting process to formulas that are p-way parallel and have a packet size of μ. For instance, the rule

$$\underbrace{\mathrm{DFT}_m \otimes I_n}_{\text{threads}(p,\mu)} \rightarrow \left((L^{mp}_p \otimes I_{n/\mu p}) \bar{\otimes} I_\mu \right) \left(I_p \otimes_{\parallel} (\mathrm{DFT}_m \otimes I_{n/p}) \right) \left((L^{mp}_p \otimes I_{n/\mu p}) \bar{\otimes} I_\mu \right) \tag{2}$$

manipulates the first factor in (1) into a load balanced, parallel version that uses the specified DMA packet size for inter-core communication. In (2), $\bar{\otimes}$ is later translated into DMA transfers, and $\otimes_{\parallel}$, into a parallel loop. Other formula fragments are handled by similar rules.

4 Experimental Results and Conclusion

We evaluated our generated single-precision 1D DFT implementations on a single SPE and on 4 SPEs of a PlayStation 3, running at 3.2 GHz, for input and output vectors

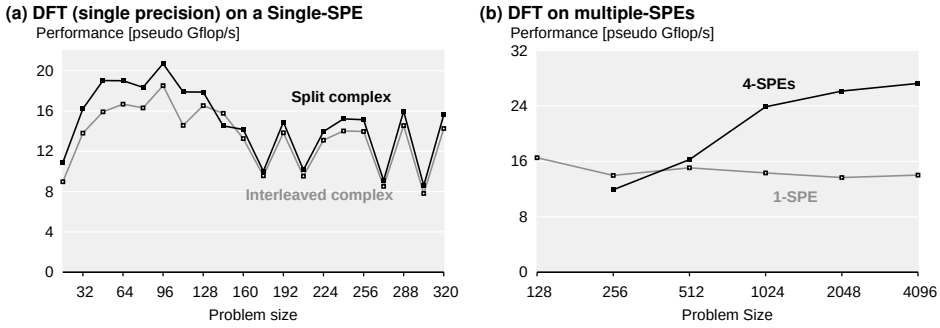

Fig. 2. Performance results for the DFT on a PlayStation 3 (3.2 GHz Cell). Higher is better.

that are resident in the SPEs' local stores. In Fig. 2 (a) we display the single-core performance of our generated DFT kernels for both the split-complex and the interleaved-complex data formats. We generated DFT kernels of sizes up to 320 that are multiples of 16. Spiral generated code achieves 16-20 Gflop/s which is is comparable to the best reported single-core performance (22 Gflop/s for size 8,192 in [5]). In Fig. 2 (b) we compare our generated multithreaded 1D DFT code (that uses a block-cyclic data format) run on 4 SPEs to the single-SPE kernel performance. For $n = 512$ we reach the break-even point (the runtime is about 5,000 cycles), and for $n = 4,096$ we see close to a 2x speed-up, leading to 27 Gflop/s for a single, non-streamed, 1D DFT.

Conclusion. We presented preliminary work that extends the Spiral framework to automatically generate DFT code for the Cell BE. Our DFT libraries have performance competitive with hand-tuned code on a single SPE, and speed up small DFT sizes by using multiple SPEs. We are currently examining algorithmic manipulations to increase communication bandwidth, and to enable double-buffering to hide communication costs.

References

1. Püschel, M., Moura, J.M.F., Johnson, J., Padua, D., Veloso, M., Singer, B.W., Xiong, J., Franchetti, F., Gačić, A., Voronenko, Y., Chen, K., Johnson, R.W., Rizzolo, N.: SPIRAL: Code generation for DSP transforms. Proceedings of the IEEE **93**(2) (2005) 232–275 Special issue on *Program Generation, Optimization, and Adaptation*.
2. Franchetti, F., Püschel, M.: A SIMD vectorizing compiler for digital signal processing algorithms. In: Proc. IEEE Int'l Parallel and Distributed Processing Symposium. (2002)
3. Franchetti, F., Voronenko, Y., Püschel, M.: FFT program generation for shared memory: SMP and multicore. In: Supercomputing (SC). (2006)
4. Bader, D.A., Agarwal, V.: FFTC: Fastest fourier transform for the IBM Cell Broadband Engine. In: 14th IEEE International Conference on High Performance Computing. (2007)
5. Cico, L., Cooper, R., Greene, J.: Performance and Programmability of the IBM/Sony/Toshiba Cell Broadband Engine Processor. In: Proc. of (EDGE) Workshop. (2006)
6. Chow, A.C., Fossum, G.C., Brokenshire, D.A.: A programming example: Large FFT on the Cell Broadband Engine. Technical report (May 2005)
7. Frigo, M., Johnson, S.G.: The design and implementation of FFTW3. Proceedings of the IEEE **93**(2) (2005) 216–231 Special issue on "Program Generation, Optimization, and Adaptation".
8. Kistler, M., Perrone, M., Petrini, F.: Cell multiprocessor communication network: Built for speed. IEEE Micro **26**(3) (2006) 10–23