High Performance Linear Transform Program Generation for the Cell BE *

Srinivas Chellappa, Franz Franchetti, Markus Püschel
{schellap, franzf, pueschel}@ece.cmu.edu
Electrical and Computer Engineering, Carnegie Mellon University

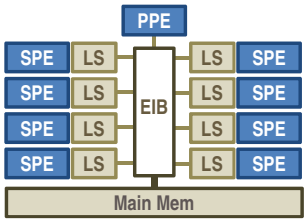

Figure 1: Cell processor.

Introduction

The Cell BE is among a new generation of multicore processors including the Intel Larrabee and the Tilera TILE64 that provide an impressive peak fixed or floating point performance for scientific, signal processing, visualization, and other engineering applications. As shown in Fig. 1, the Cell uses simple in-order cores designed specifically for numerical computing, and requires explicit memory management to achieve maximal performance, which make programming and optimizing a challenge. In this paper, we extend Spiral [7], a program generation system, to generate highly optimized linear transform programs for the Cell BE. In doing so, as presented in [2], we extend Spiral's architectural paradigms to include support for distributed memory architectures like the Cell that allow hiding memory costs using multibuffering techniques.

We focus on fixed-size code for the 1D complex discrete Fourier transform (DFT), but also generate code for variants including transforms that work on real input, 2D input, and for other transforms including the discrete Cosine and Sine transforms. We generate code for various usage scenarios, including latency optimized and throughput optimized code, and our system can handle various complex data formats and data distribution formats. The performance of Spiral generated code for the Cell is comparable to, and in many cases better than existing implementations, where available.

Spiral. Spiral automates the generation of platform adapted high-performance libraries with a focus on the domain of linear transforms. Spiral provides a range of functionality difficult to match with hand written libraries, with generated programs comparing well against the performance of hand-optimized code.

Spiral uses a domain-specific, declarative, mathematical language to both represent algorithms, and to model the architecture at a high level. It uses rewriting to transform algorithms at a high level of abstraction to "fit" the target architecture. The input to Spiral is a high-level functional description of the required transform (e.g., DFT of size 8,192 on 4 SPEs); the output is a highly optimized C program implementation.

Related work. Various other works including Cico et al. [4], Chow [3], FFTC [1], FFTW [5], and IBM [6] have produced single-SPE and multi-SPE FFT libraries for the Cell, with various functionality and performance limitations. Spiral is among the class of several automatic program generation and performance tuning systems.

Parallelization and Streaming

Generation of high-performance code for the Cell involves making efficient use of the Cell architecture's parallelism (SIMD and multicore), and its support for hiding memory costs using explicit DMA accesses and multibuffering. We extend Spiral in a general manner to support these concepts, and provide a separate Cell-specific implementation layer, allowing us to easily port our techniques to future architectures that implement these architectural paradigms.

We perform parallelization and streaming using formula manipulation and rewriting. Linear transforms are represented in Spiral as matrices where performing matrix-vector multiplication on the input vector with the transform matrix transforms it into the output vector. Algorithms for transforms are viewed as structured factorizations of the transform matrices, represented using Spiral's internal signal processing language (SPL) [7], which is based on the Kronecker or tensor product formalism. For instance, the well known Cooley-Tukey recursive breakdown of the 1D DFT is represented in SPL as:

$$\mathrm{DFT}_{mn} \rightarrow (\mathrm{DFT}_n \otimes \mathrm{I}_m) T_{n,m} (\mathrm{I}_n \otimes \mathrm{DFT}_m)\, \mathrm{L}^{nm}_n . \quad (1)$$

Key to the SPL representation is the fact that the tensor product can be viewed as a program loop where certain loop properties are made explicit. This allows us to derive high-performance implementations of the loop that can be mapped to the hardware architecture.

A problem is specified by tagging a transform with with hardware specifications (e.g., the number of SPEs to parallelize for). Formula identities transform SPL formulas into base case mathematical equivalents in Σ-SPL(see [2] for details) that are structurally suited for parallelization and/or streaming, from which optimized C code is generated. We show examples of formula manipulation only at the SPL level in this paper due to space constraints.

Parallelization. We reuse Spiral's SIMD paradigm for the Cell. We develop a multicore paradigm that uses explicit DMA-based inter-core communication with large packet sizes to exchange data using the fast on-chip interconnect. As

*This work was supported by NSF through awards 0325687, 0702386, by DARPA (DOI grant NBCH1050009), ARO grant W911NF0710416, and by Mercury Computer Systems. We also thank Mercury Computer Systems for allowing us to evaluate our libraries on their PowerXCell 8i based product.

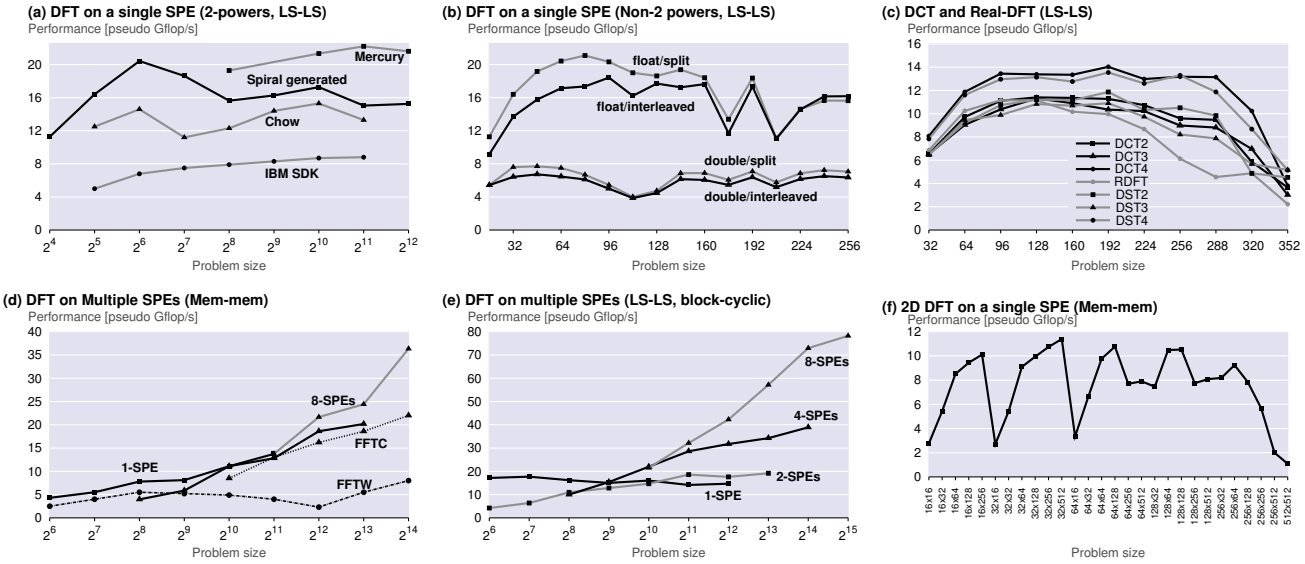

Figure 2: Performance results for latency optimized 1D and 2D linear transforms. Higher is better.

an example, we show how our system parallelizes (1) to execute on p SPEs:

$$\underbrace{\mathrm{DFT}_{mn}}_{\mathrm{spe}(p,\mu)} = \left(\mathrm{I}_p \otimes_{\parallel} (\mathrm{DFT}_m \otimes \mathrm{I}_{n/p})\right)^{\left((\mathrm{L}^{mp}_p \otimes \mathrm{I}_{n/p\mu}) \otimes \mathrm{I}_\mu\right)} T^{mn}_m$$
$$\left(\mathrm{I}_p \otimes_{\parallel} (\mathrm{I}_{m/p} \otimes \mathrm{DFT}_n)\, \mathrm{L}^{mn/p}_{m/p}\right)\left((\mathrm{L}^{pn}_p \otimes \mathrm{I}_{m/p\mu}) \otimes \mathrm{I}_\mu\right).$$

Above, we use conjugation $A^P = P^{-1}AP$ of a matrix A by a permutation matrix P. The above rewrites (1) into factors that correspond to the base cases for parallelization: $\mathrm{I}_p \otimes_{\parallel} A_m$ represents execution of A_m on p processors in parallel, and the $P \otimes \mathrm{I}_\mu$ terms symbolize global permutations, which are then translated into all-to-all inter-SPE DMA communication with packet size μ.

Streaming. We also extend Spiral's formalism to support a streaming paradigm. Streaming or multibuffering is the idea of hiding memory access costs by explicitly managing memory to overlap computation with memory accesses. Our paradigm works in two scenarios. In the first scenario, we generate throughput-optimized code for small to mid-sized transforms. In the second scenario, we generate code for transforms too large to fit on-chip, that must be streamed in and out of the chip to be computed in parts. We use a similar approach for streaming as for parallelism, not shown here due to space constraints.

Experimental Results

Fig. 2 shows our performance results, presented as normalized inverse runtime measured in pseudo Gflop/s. Evaluation was done on a single Cell processor of 3.2GHz IBM Cell Blade QS20, and double precision implementations on a PowerXCell 8i based system.

Fig. 2(a) and Fig. 2(b) show single-core performance of Spiral generated DFT kernels and others when available. Fig. 2(c) shows other transforms performed on a single-SPE. Fig. 2(d) compares our multi-SPE code to other libraries. We achieve close to 80 Gflop/s when using a block-cyclic data distribution format shown in Fig. 2(e). Fig. 2(f) shows 2D DFT kernels on a single-SPE some of which are too large to fit on-chip, performed using multibuffering techniques.

Current Work. Multi-SPE versions of Fig. 2(e) and Fig. 2(f), and extending Fig. 2(c) to include large DFT sizes using multibuffering are work-in-progress.

Conclusion

As the number of cores in emerging multicore architectures scale, accessing off-core memory comes at an expense. Further, inter-core contention for off-chip memory resources increases, forcing chip designers to use alternative memory access techniques which in turn increase programming burden. We address this in the context of high performance program generation for linear transform libraries by extending Spiral to support the general architectural paradigms (rather than develop a Cell-specific system) inherent in such emerging multicore architectures.

References

[1] D. A. Bader and V. Agarwal. FFTC: Fastest Fourier transform for the IBM Cell Broadband Engine. In *IEEE Intl. Conference on High Performance Computing*, pages 172–184, 2007.

[2] S. Chellappa, F. Franchetti, and M. Püschel. Computer generation of fast Fourier transforms for the cell broadband engine. In *International Conference on Supercomputing (ICS) (to appear)*, 2009.

[3] A. C. Chow. Fast Fourier transform SIMD code generators for synergistic processor element of Cell processor. In *Workshop on Cell Systems and Applications*, 2008.

[4] L. Cico, R. Cooper, and J. Greene. Performance and Programmability of the IBM/Sony/Toshiba Cell Broadband Engine Processor. In *Proc. of (EDGE) Workshop*, 2006.

[5] M. Frigo and S. G. Johnson. The design and implementation of FFTW3. *Proc. of the IEEE, special issue on "Program Generation, Optimization, and Adaptation"*, 93(2):216–231, 2005.

[6] IBM. *Fast Fourier Transform Library Programmer's Guide and API Reference*, 3.0 edition, 2008.

[7] M. Püschel, J. M. F. Moura, J. Johnson, D. Padua, M. Veloso, B. Singer, J. Xiong, F. Franchetti, A. Gacic, Y. Voronenko, K. Chen, R. W. Johnson, and N. Rizzolo. SPIRAL: Code generation for DSP transforms. *Proceedings of the IEEE, special issue on "Program Generation, Optimization, and Adaptation"*, 93(2):232– 275, 2005.