ACM Communications in Computer Algebra, Vol 47, No. 3, Issue 185, September 2013

Towards Parallel General-Size Library Generation for Polynomial Multiplication

Lingchuan Meng and Jeremy Johnson
Department of Computer Science
Drexel University
Philadelphia, PA 19104
lm433@drexel.edu,jjohnson@drexel.edu

Fast Fourier Transforms (FFTs) are at the core of many operations in scientific computing. In computer algebra, FFTs are used for *fast polynomial and integer arithmetic and other modular methods*. FFT-based polynomial multiplication outperforms multiplication based on classical and Karatsuba-based algorithms. Computer algebra libraries, such as `modpn` [3], provide hand-optimized low-level routines implementing fast algorithms for multivariate polynomial computations over finite fields, in support of higher-level code. Such libraries do not fully utilize the underlying hardware, and in order to take advantage of platform-dependent optimizations, automated performance tuning that supports general input sizes should be incorporated.

Recently, we extended [2] the use of SPIRAL from fixed-size code generation to general input size library generation to produce a modular FFT [1] library. By incorporating and extending the new library generation mechanism in SPIRAL [4], the generated library provides similar speedup as the fixed-size code, which is an order of magnitude faster over the original implementations in `modpn`, and allows arbitrary input sizes. Additional parallelism exploiting multi-core architecture leading to further speedup also has been implemented. This addition required adding new rules and a new transform definition and parameterization in the library generation framework in order to generate *recursive function closure* in the resulting library. The backend was also extended to enable the generation of scalar and vectorized code for modular arithmetic.

Let the n-point modular DFT matrix be $\mathbf{ModDFT}_{n,p,\omega} = \left[\omega_n^{k\ell}\right]_{0\le k,\ell<n}$, where ω_n is a primitive nth root of unity in $\mathbb{Z}_p$. Let $n = rs$, then the divide and conquer step in the Cooley-Tukey algorithm can be represented as the parameterized matrix factorization:

$$\mathbf{ModDFT}_{n,p,\omega} = (\mathbf{ModDFT}_{r,p,\omega_r} \otimes \mathrm{I}_s)\, \mathrm{T}^n_s (\mathrm{I}_r \otimes \mathbf{ModDFT}_{s,p,\omega_s})\, \mathrm{L}^n_r,$$

where T^n_s is a diagonal matrix containing *twiddle factors*; the *stride permutation* matrix L^n_r permutes the input vector as $is + j \mapsto jr + i, 0 \le i < r, 0 \le j < s$; I_s is the $s \times s$ identity matrix; and the *tensor product* is defined as $A \otimes B = [a_{k,l}B]\,, A = [a_{k,l}]\,.$

The tensor product serves as the key construct in SPIRAL and its many fast algorithms, in that it captures loops, data independence, and parallelism concisely. For instance, Fig. 1 shows that it produces substructures that can be interpreted as vector and parallel operations. Furthermore, the formulae can be transformed to adapt to a given vector length and number of cores; and permutations can be manipulated to obtain desired data access patterns. *Rewriting systems* and *hardware tags* have been developed in SPIRAL to fully exploit two levels of parallelism: *vector parallelism* and *thread parallelism*.

We report experimental data comparing the performance of hand-optimized FFTs from the `modpn` library, fixed-size FFTs and general size parallel FFT library generated by SPIRAL. Performance is reported in Gops (giga-ops) or billions of operations per second (higher is better). As shown in Fig. 2, all SPIRAL generated codes are faster than the hand-optimized implementation in `modpn` by an order of magnitude. The

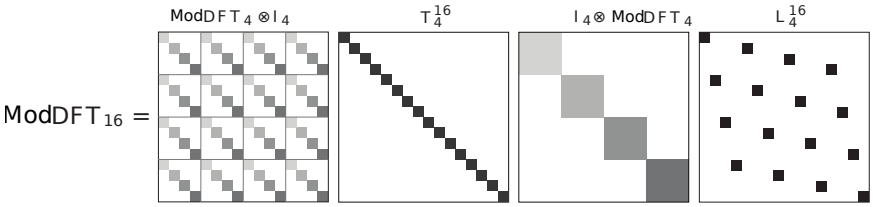

Figure 1: Representation of the matrix factorization based on the Cooley-Tukey algorithm. Shades of gray represent values that belong to the same tensor substructure

performance of general size library's scalar and vector codes are within 81% to 91% of that of corresponding fixed-size codes. For large sizes, the library code is up to 1.5 time faster than the fixed-size code, due to the use of thread level parallelism.

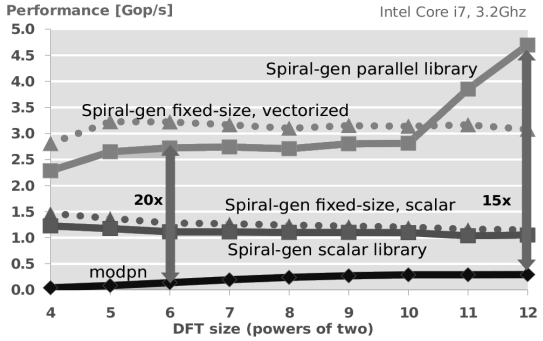

Figure 2: Performance comparison

To eventually generate an optimized parallel library for polynomial multiplication, we are developing additional algorithms for modular FFT, including Prime-factor algorithm and Rader's algorithm. We have also developed a Cooley-Tukey type algorithm for the Truncated Fourier Transform and its inverse (TFT/ITFT) for non-contiguous and non-power-of-two input/output. We have proved block symmetry of the ITFT matrices and derived direct generation formulae that can be used during library generation. Convolutions by definition and the Convolution Theorem have also been implemented in Spiral, whose performance relies on the auto-tuned underlying transforms, such as modular DFT and TFT, and the exploration of hybrid algorithms and automatic tuning of threshold parameters.

References

[1] L. Meng, J. Johnson. Automatic Parallel Library Generation for General-Size Modular FFT Algorithms. To appear in *Proc. of the CASC 2013*, 2013

[2] L. Meng, J. Johnson, F. Franchetti, Y. Voronenko,M. Moreno Maza and Y. Xie. Spiral-Generated Modular FFT Algorithms. In *Proc. of PASCO 2010*, p. 169–170, 2010.

[3] X. Li and M. Moreno Maza: Efficient implementation of polynomial arithmetic in a multiple-level programming environment. In *Proc. Intl. Congress of Mathematical Software*, p. 12–23, Springer, 2006.

[4] Y. Voronenko. Library Generation for Linear Transforms. PhD. thesis, Electrical and Computer Engineering, Carnegie Mellon University, 2008