Spiral-Generated Modular FFT Algorithms

[Extended Abstract]

Lingchuan Meng
Drexel University
Philadelphia, PA, USA
lm433@drexel.edu

Jeremy R. Johnson
Drexel University
Philadelphia, PA, USA
jjohnson@cs.drexel.edu

Franz Franchetti
Carnegie Mellon University
Pittsburgh, PA, USA
franzf@ece.cmu.edu

Yevgen Voronenko
Carnegie Mellon University
Pittsburgh, PA, USA
yvoronen@ece.cmu.edu

Marc Moreno Maza
University of Western Ontario
London, Canada
moreno@csd.uwo.ca

Yuzhen Xie
Massachusetts Insitute of Technology
Cambridge, MA, USA
yxie@csail.mit.edu

ABSTRACT

This paper presents an extension of the SPIRAL system to automatically generate and optimize FFT algorithms for the discrete Fourier transform over finite fields. The generated code is intended to support modular algorithms for multivariate polynomial computations in the `modpn` library used by Maple. The resulting code provides an order of magnitude speedup over the original implementations in the `modpn` library, and the SPIRAL system provides the ability to automatically tune the FFT code to different computing platforms.

Categories and Subject Descriptors

D.1.2 [**Software**]: Programming Techniques, Automatic Programming; G.4 [**Mathematics of Computing**]: Mathematical Software, Efficiency; I.1.3 [**Computing Methodologies**]: Symbolic and Algebraic Manipulation, Languages and Systems

General Terms

Algorithms, Performance

Keywords

FFT, modular arithmetic, code generation, vectorization, high performance computing, autotuning

1. INTRODUCTION

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 modular methods (i.e. computation by homomorphic images). In recent years, the use of fast arithmetic has become prevalent and has stimulated the development of software libraries, such as `modpn` [FLMS06, LM06, LMP09] providing hand-optimized low-level routines implementing fast algorithms for multivariate polynomial computations over finite fields, in support of higher-level code. The `modpn` library has been integrated into the computer algebra system MAPLE and runs on all computer platforms supported by MAPLE. The implementation techniques employed in `modpn` are often platform-dependent, since cache size, associativity properties and register sets have a significant impact. In order to take advantage of platform-dependent optimizations, in the context of quickly evolving hardware acceleration technologies, automated performance tuning has become necessary and should be incorporated into the `modpn` library.

SPIRAL [www.spiral.net] is a library generation system that automatically generates platform-tuned implementations of digital signal processing algorithms with an emphasis on fast transforms. Currently, SPIRAL can generate highly optimized fixed-point and floating-point FFTs for a variety of platforms with automatic tuning, and has support for vectorization, threading, and distributed memory parallelization. The code produced is competitive with the best available code for these platforms and SPIRAL is used by Intel for its IPP (Integrated Performance Primitives) and MKL (Math Kernel Library) libraries.

In this work, SPIRAL was extended to generate algorithms for FFT computation over finite fields. This addition required adding a new data type, several new rules and a new transform definition. In addition, the backend was extended to enable the generation of scalar and vectorized code for modular arithmetic. With these enhancements, the SPIRAL machinery can be applied to modular transforms needed by the `modpn` library. In this paper we present preliminary results showing that the code generated by SPIRAL is approximately eleven times faster than the original FFT code in `modpn`.

2. SPIRAL

The SPIRAL system [PMJ05] uses a mathematical framework for representing and deriving algorithms. Algorithms are expressed symbolically as sparse matrix factorizations

PASCO 2010 Grenoble, France

and are derived using rewrite rules; additional rules are used to symbolically manipulate algorithms into forms that take advantage of the underlying hardware, including vectorization [FVP08] and parallelism [FVP06]. The sequence of applications of rules is encoded as a *ruletree* which can be translated into a formula and compiled with a special-purpose compiler into efficient code. A search engine with a feedback loop is used to tune implementations to particular platforms. New transforms are added by introducing new symbols and their definitions, and new algorithms can be generated by adding new rules.

SPIRAL was developed for floating point and fixed point computation; however, many of the transforms and algorithms carry over to finite fields. For example, the DFT of size n is defined when there is a primitive nth root of unity and many factorizations of the DFT matrix depend only on properties of primitive nth roots. In this case, the same machinery in SPIRAL can be used for generating and optimizing modular transforms. All that is needed is support for new data types and code generation and the addition of new transforms and rules.

For the modular FFT, we added a modular data type, support for modular arithmetic and code generation, and defined the n-point modular DFT

$$\mathbf{ModDFT}_{n,p,\omega} = \left[\omega_n^{k\ell}\right]_{0\le k,\ell<n},$$

where ω_n is a primitive nth root of unity in Z_p, and the Cooley-Tukey factorization (rewrite rule)

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

where $n = rs$, $\otimes$ denotes the Kronecker or tensor product, T is a diagonal matrix called the twiddle matrix, and L is a special permutation matrix called stride permutation.

3. PERFORMANCE RESULTS

This section reports on preliminary experimental data comparing the performance of hand coded FFTs from the `modpn` library and FFTs automatically generated by SPIRAL. SPIRAL generated algorithms using the Cooley-Tukey rule, and used dynamic programming to select an "optimal" recursive breakdown strategy. Dynamic programming is only a heuristic since an optimal algorithm of a given size can depend on the context in which it is called; however, experience shows that it makes good choices. All experiments were performed on an Intel Core i7 965 quad-core processor running at 3.2 GHz with 12 GB of RAM. Generated code was compiled with gcc version 4.3.4-1 with optimization set to O3. Vector code used SSE4.1 with 4-way 32 bit integer vectors. Since there is no vector version of integer division with remainder, in order to vectorize our SPIRAL generated FFT code on the Core i7, Montgomery's trick [Mont85] was used. Initial experiments were performed using 32 bit integers and 16 bit primes. Figure 1 compares the performance of power of two FFTs of size 4 through 4096 using the original `modpn` code and scalar and vectorized code generated by SPIRAL, where performance is reported in Gops (giga-ops) or billions of operations per second (higher is better), which is calculated assuming that DFT of size N takes a total of $(3/2)N\lg(N)$ additions, subtractions and nontrivial multiplications. The SPIRAL generated vector code is between 11 to 23 times faster than the original `modpn` code.

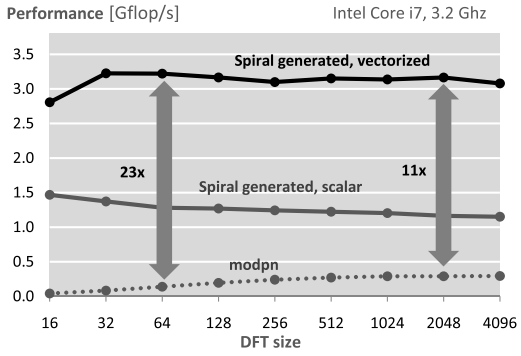

Figure 1: Performance Comparison

4. REFERENCES

[FLMS06] A. Filatei, X. Li, M. Moreno Maza, and É. Schost. Implementation techniques for fast polynomial arithmetic in a high-level programming environment. In *Proc. ISSAC'06*, pp 93–100, New York, NY, USA, 2006. ACM Press.

[FVP06] Franz Franchetti, Yevgen Voronenko and Markus Püschel, "FFT Program Generation for Shared Memory: SMP and Multicore," Proc. Supercomputing (SC), 2006.

[FVP08] Franz Franchetti, Yevgen Voronenko and Markus Püschel, "A Rewriting System for the Vectorization of Signal Transforms," Proc. High Performance Computing for Computational Science (VECPAR), Lecture Notes in Computer Science, Springer, Vol. 4395, pp. 363-377, 2006.

[LM06] X. Li and M. Moreno Maza. Efficient implementation of polynomial arithmetic in a multiple-level programming environment. In A. Iglesias and N. Takayama, editors, *Proc.* International Congress of Mathematical Software - ICMS 2006, pp 12–23. Springer, 2006.

[LMP09] Xin Li, Marc Moreno Maza, and Wei Pan. Computations modulo regular chains. In *ISSAC '09: Proceedings of the 2009 international symposium on Symbolic and algebraic computation*, pp 239–246, New York, NY, USA, 2009. ACM.

[Mont85] P. L. Montgomery. Modular Multiplication Without Trial Division. Mathematics of Computation, vol. 44, no. 170, pp. 519–521, 1985.

[PMJ05] Markus Püschel, José M. F. Moura, Jeremy Johnson, David Padua, Manuela Veloso, Bryan Singer, Jianxin Xiong, Franz Franchetti, Aca Gacic, Yevgen Voronenko, Kang Chen, Robert W. Johnson, and Nick Rizzolo SPIRAL: Code Generation for DSP Transforms Proceedings of the IEEE special issue on "Program Generation, Optimization, and Adaptation," Vol. 93, No. 2, 2005, pp. 232-275.

[www.spiral.net] SPIRAL PROJECT WEBSITE. http://www.spiral.net, 2010.