A RECURSIVE IMPLEMENTATION OF THE DIMENSIONLESS FFT

Jeremy Johnson, Xu Xu

Drexel University
Department of Computer Science
3141 Chestnut Street, Philadelphia, PA 19104, USA
jjohnson@cs.drexel.edu and xx27@drexel.edu

ABSTRACT

A divide and conquer algorithm is presented for computing arbitrary multi-dimensional discrete Fourier transforms. In contrast to standard approaches such as the row-column algorithm, this algorithm allows an arbitrary decomposition, based solely on the size of the transform independent of the dimension of the transform. Only minor modifications are required to compute transforms with different dimension. These modifications were incorporated into the FFTW package so that the algorithm for computing one-dimensional transforms can be used to compute arbitrary dimensional transforms. This reduced the runtime of many multi-dimensional transforms.

1. INTRODUCTION

The divide and conquer construction used by the fast Fourier transform (FFT) allows a discrete Fourier transform (DFT) of size mn to be computed using n transforms of size m followed by m transforms of size n [1]. This construction requires that the input data be accessed at stride and the intermediate data obtained after computing the n transforms of size m be scaled by the so called “twiddle factors”. Multi-dimensional DFTs are normally computed using one-dimensional FFTs along each of the dimensions. For example, let $X(a, b)$ with $0 \leq a < m$ and $0 \leq b < n$ be a function of two variable stored in an $m \times n$ array. The two-dimensional $m \times n$ DFT of X can be calculated by applying m one-dimensional n-point DFTs to the rows of X followed by n one-dimensional m-point DFTs to the columns. The one-dimensional DFTs are computed using the FFT. This approach, called the row-column algorithm, can be generalized to DFTs with arbitrarily many dimensions; however, it has the shortcoming that the divide and conquer construction used by the FFT can only be applied separately to the number of points in each dimension. It does not allow the use of smaller transforms of size equal to an arbitrary factor of the number of data points. The dimensionless FFT [2] allows a multi-dimensional DFT of total size $N = RS$ to be computed using R multi-dimensional DFTs of size S followed by S multi-dimensional DFTs of size R independent of dimension. This is identical to the one-dimensional construction except that a slightly different input permutation is required and the values of the twiddle factors are different. The permutation and twiddle factors depend on the dimension. The original presentation of the dimensionless FFT was based on an iterative algorithm for computing the FFT and was motivated by the desire to produce FFT hardware that could be used for one, two, and three dimensional transforms [3, 4].

Recently there has been efforts to automatically optimize the performance of important signal processing routines such as the FFT [5, 6, 7]. These approaches search for a good decomposition (breakdown strategy) of the FFT in an effort to best utilize the number of registers, cache, and other features of the underlying hardware. A good breakdown can be far more important than saving a few arithmetic operations. The use of the dimensionless FFT allows decomposition sizes that are not available in the row-column algorithm and can provide improved performance for many multi-dimensional DFTs.

In this paper, we show that FFTW [5], one of the fastest public domain FFT packages, can be modified to support a recursive implementation of the dimensionless FFT. FFTW can use many different recursive divide and conquer strategies, and dynamic programming is used to empirically determine the “best” strategy. The desired strategy is stored in a tree data structure called a plan. The plan also stores the necessary twiddle factors. An executor uses the plan to compute the FFT from a collection of small FFTs, called codelets, implemented using straight-line code. To support the dimensionless FFT, extra information must be stored in the plan to keep track of the dimension of the various DFTs that arise, and a set of multi-dimensional FFT codelets must be provided. The plan generator must be extended to produce the twiddle factors required by the dimensionless FFT, and the executor must support one additional parameter needed

This work was supported by DARPA through research grant DABT63-98-1-0004 administered by the Army Directorate of Contracting.

for the generalized permutations that can arise. These changes were incorporated into FFTW and empirical data is presented showing the potential for improved performance as compared to the row-column algorithm currently provided in FFTW.

In the remainder of the paper, we derive a recursive formulation of the dimensionless FFT, show how to modify FFTW to implement the dimensionless FFT, and provide empirical performance data.

2. THE DIMENSIONLESS FFT

In this section we review the row-column algorithm in the mathematical framework best used to derive and understand the dimensionless FFT. An example is used to show how to derive the dimensionless FFT and the general theorem underlying the dimensionless FFT is stated.

Let $X(a), 0 \leq a < N$ be a complex function of N points. The N-point *DFT* of X is defined by

$$Y(b) = \sum_{a=0}^{N-1} \omega_N^{ab} X(a)$$

where $\omega_N = e^{2\pi i/N}$. Let $X(a)$ and $Y(b)$ be represented by column vectors of size N, then the DFT is given by the matrix-vector product $Y = F_N X$, where F_N is the $N \times N$ matrix whose (i,j) element, $0 \leq i,j < N$, is equal to ω_N^{ij}.

Let $X(a_1, \ldots, a_t)$ be a function of t variables, where $0 \leq a_j < n_j$. The t-dimensional $n_1 \times \cdots \times n_t$ DFT of X is

$$Y(b_1, \ldots, b_t) = \sum_{0 \leq a_j < n_j} \omega_{n_1}^{a_1 b_1} \cdots \omega_{n_t}^{a_t b_t} X(a_1, \ldots, a_t)$$

Since the summation indices are independent, this sum can be written as a nested sum.

$$Y(b_1, \ldots, b_t) = \\ \sum_{a_1=0}^{n_1-1} \omega_{n_1}^{a_1 b_1} \left(\cdots \left(\sum_{a_t=0}^{n_t-1} \omega_{n_t}^{a_t b_t} X(a_1, \ldots, a_t) \right) \cdots \right).$$

The nested sum implies that the multi-dimensional DFT can be computed by applying one-dimensional DFTs along each dimension. A sequence of n_t-point DFTs is applied to the functions obtained by fixing the first $t-1$ inputs to X. Then a sequence of n_{t-1}-point DFTs is applied to the resulting function with all but the $(t-1)$-st variable fixed. This process continues until finally a sequence of n_1-point DFTs are applied. In the case of two-dimensions this is called the row-column algorithm, since, if the function $X(a_1, a_2)$ is stored lexicographically as an $n_1 \times n_2$ matrix, the computation proceeds by applying n_2-point DFTs to the rows of X followed by n_1-point DFTs applied to the columns of the partially transformed matrix.

If the functions X and Y are stored lexicographically as column vectors x and y respectively and $\otimes$ denotes the tensor product (also called the Kronecker product) [8], then

$$y = (F_{n_1} \otimes \cdots \otimes F_{n_t})x, \tag{1}$$

and the row-column algorithm corresponds to the matrix factorization

$$(F_{n_1} \otimes \cdots \otimes F_{n_t}) = \prod_{j=1}^{t} (I_{N(j-1)} \otimes F_{n_j} \otimes I_{\bar{N}(j)})$$

where $N(j) = n_1 \cdots n_j$, $\bar{N}(j) = N/N(j)$, $N(0) = 1$, and $N(t) = N$.

Fast algorithms for computing $Y = F_N X$, the FFT and variants, can be obtained from factorizations of F_N [9, 10, 8]. If $N = RS$,

$$F_N = (F_R \otimes I_S) T_S^N (I_R \otimes F_S) L_R^N, \tag{2}$$

where I_N is the N point identity matrix, T_S^N is a diagonal matrix containing the twiddle factors and L_R^N is a permutation matrix that gathers the input at stride R. More precisely, letting $\oplus$ denote the direct sum of matrices,

$$T_S^N = \bigoplus_{j=0}^{R-1} \mathrm{diag}(\omega_N^0, \ldots, \omega_N^{S-1})^j$$

and

$$\mathrm{L}_R^N : i * S + j \mapsto j * R + i \text{ for } 0 \leq i < R, 0 \leq j < S.$$

A multi-dimensional DFT can be computed in exactly the same way using the dimensionless FFT. Let $\mathcal{F}_N = F_{n_1} \otimes \cdots \otimes F_{n_t}$ be an arbitrary multi-dimensional DFT of size $N = n_1 \cdots n_t$. Independent of dimension, if $N = RS$, then there exist a diagonal matrices D, a permutation matrix P, and multi-dimensional DFTs $\mathcal{F}_R$ and $\mathcal{F}_S$ such that

$$\mathcal{F}_N = (\mathcal{F}_R \otimes I_S) D (I_R \otimes \mathcal{F}_S) P. \tag{3}$$

When $N = 2^k$, the permutation P and diagonal D can be calculated easily. The following example illustrates the calculation and shows how the general theorem can be derived. Applying Equation 2 to F_8,

$$F_2 \otimes F_8 = F_2 \otimes (F_2 \otimes I_4) T_4^8 (I_2 \otimes F_4) L_2^8. \tag{4}$$

Writing $F_2 = F_2 I_2 I_2 I_2$ and using the property $AB \otimes CD = (A \otimes C)(B \otimes D)$, Equation 4 implies

$$\begin{aligned} F_2 \otimes F_8 \quad &= \quad (F_2 \otimes (F_2 \otimes I_4))(I_2 \otimes T_4^8) \\ &\quad\quad (I_2 \otimes (I_2 \otimes F_4))(I_2 \otimes L_2^8). \end{aligned}$$

Using associativity of the tensor product and the property $I_m \otimes I_n = I_{mn}$ we obtain

$$\begin{aligned} F_2 \otimes F_8 \quad &= \quad ((F_2 \otimes F_2) \otimes I_4)(I_2 \otimes T_4^8) \\ &\quad\quad (I_4 \otimes F_4)(I_2 \otimes L_2^8). \end{aligned}$$

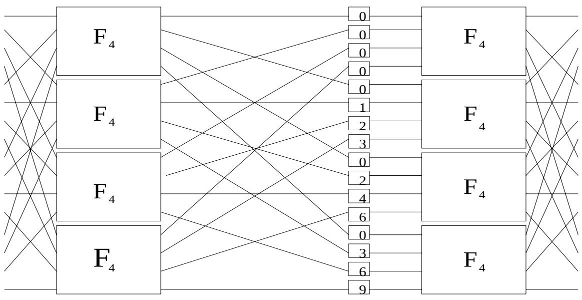

Fig. 1. Cooley-Tukey factorization for F_{16}

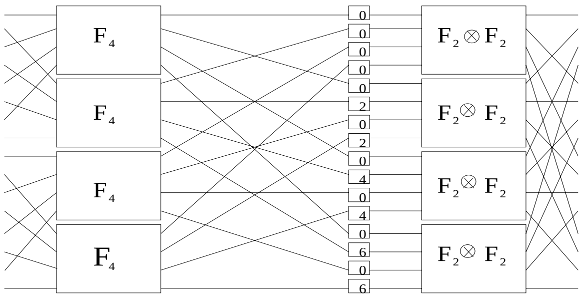

Fig. 2. Cooley-Tukey factorization for $F_2 \otimes F_4 \otimes F_2$

Since the tensor product of diagonal matrices is a diagonal matrix and the tensor product of permutation matrices is a permutation matrix, this example is an instance of Equation 3 with $D = (I_2 \otimes T_4^8)$, $P = (I_2 \otimes L_2^8)$, $\mathcal{F}_R = (F_2 \otimes F_2)$, and $\mathcal{F}_S = F_4$. Figure 2 compares the flow of this computation to the one-dimensional algorithm $F_{16} = (F_4 \otimes I_4)T_4^{16}(I_4 \otimes F_4)L_4^{16}$ in Figure 1. The computation flows from left to right in the figures and the small boxes contain the exponents of ω_{16} in the twiddle factors.

A similar derivation can be used to derive the following theorem.

Theorem 1 (Dimensionless FFT) *Let $N(l) = n_1 n_2 \cdots n_l$ and $\overline{N(l)} = N/N(l)$ with $N(0) = 1$ and $N(t) = N$, and assume $N = 2^K$ with $N = n_1 \times n_2 \times \cdots \times n_t and n_i = 2^{k_i}$. Let $\mathcal{F}_N = F_{n_1} \otimes \cdots \otimes F_{n_t}$, and let $N = RS$. If l is the largest integer such that $N(l-1) < R$ and $n_l = ab$ with $R = N(l-1)a$, then*

$$\mathcal{F}_N = (\mathcal{F}_R \otimes I_s) D (I_R \otimes \mathcal{F}_s) P \tag{5}$$

$\mathcal{F}_R = F_{n_1} \otimes \cdots \otimes F_{n_{l-1}} \otimes F_a$, $\mathcal{F}_S = F_b \otimes F_{n_{l+1}} \otimes \cdots \otimes F_{n_t}$, $D = I_{N(l-1)} \otimes T_b^{n_l} \otimes I_{\overline{N(l)}}$, $P = I_{N(l-1)} \otimes L_a^{n_l} \otimes I_{\overline{N(l)}}$.

3. IMPLEMENTATION AND PERFORMANCE

The factorization required by the dimensionless FFT in Theorem 1 is very similar to the factorization required by the one-dimensional FFT in Equation 2. The only difference

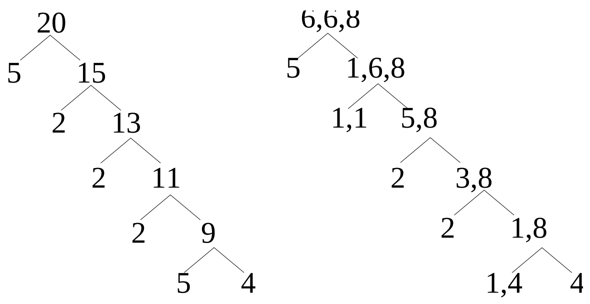

Fig. 3. Plans for $F_{2^{20}}$ and $F_{64} \otimes F_{64} \otimes F_{256}$

is that the twiddle factor T_S^N is replaced by a twiddle factor of the form $I_{N_1} \otimes T_b^n \otimes I_{\bar{N}_1}$ and the stride permutation L_R^N is replaced by a permutation of the form $I_{N_1} \otimes L_a^n \otimes I_{\bar{N}_1}$, where $N = N_1 n \bar{N}_1$ and $n = ab$. This suggests that a recursive one-dimensional FFT implementation could be modified slightly to obtain an implementation of an arbitrary multi-dimensional DFT. In this section we show how to modify FFTW's one-dimensional FFT implementation to obtain a multi-dimensional dimensionless FFT.

FFTW recursively applies Equation 2 to compute the FFT. The sequence of applications of Equation 2 are stored in a tree data structure called a plan. The plan is precomputed and is chosen through a search to provide "the most efficient plan". The executor uses the plan to compute the FFT with the desired algorithm. A node in the plan corresponds to the computation of a DFT. A leaf node is computed with highly tuned straight-line code called a codelet. An internal node is computed with the corresponding application of Equation 2. Plans are restricted to rightmost trees, those trees with the property that all left children are leaf nodes and are computed with codelets. The codelets for left nodes incorporate twiddle factor computation with twiddles stored in the plan data structure. The addressing required by the stride permutations is incorporated into the addressing of the rightmost leaf node. The strides are combined in each recursion step. Figure 3 shows a plan for computing $F_{2^{20}}$ with the corresponding plan for $F_{64} \otimes F_{64} \otimes F_{256}$. Nodes are labeled with the exponents of the transforms.

In order to modify FFTW to support the dimensionless FFT, the plan must include the dimension of the DFT at each node, and must store the generalized twiddle factors. Since the twiddle factors are precomputed this only requires modifying the plan generator to compute the necessary twiddle factors. Second, the addressing parameters used by the executor must be extended to support permutation by $I_{N_1} \otimes L_a^n \otimes I_{\bar{N}_2}$ rather than just stride permutations. Finally since the leaf nodes can correspond to multi-dimensional DFTs, a library of multi-dimensional DFTs must be provided. We used the SPIRAL system [7] to generate the necessary codelets.

The addressing required by the computation of $\mathcal{F}_S$ in $(I_R \otimes \mathcal{F}_S)P$ can be determined by a stride parameter and a

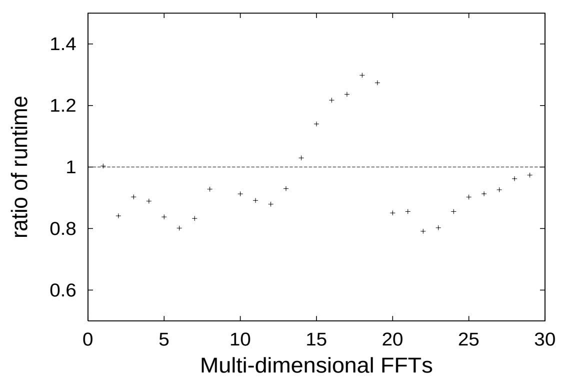

Fig. 4. The runtime ratio of 1D to MD FFT

block parameter. The input to $\mathcal{F}_S$ starting at x with stride parameter s and block parameter m is equal to $x(\lfloor j/m \rfloor sm + j \mod m)$ for $j = 0, \ldots, S-1$. If $P = I_{N_1} \otimes L_a^n \otimes I_{\bar{N}_1}$, then $s = a$ and $m = \bar{N}_1$. If $\mathcal{F}_S$ is recursively split with $S = R'S'$ and $S = N_1' a' b' \bar{N}_1'$, then $s = a'$ and $m = \bar{N}_1'$ if S' divides $\bar{N}_1$, and $s = aa'$ and $m = \bar{N}_1' = \bar{N}_1$ if S' does not divide $\bar{N}_1$. This shows that the addressing parameters can be recursively combined as was the case for stride permutations. The source code is available upon request.

To determine the potential performance improvement we compared the runtime, using FFTW, of the one-dimensional and multi-dimensional FFT using the row-column algorithm. Experiments were performed on a 550 MHz Pentium III processor with 16K L1 cache, 512 K L2 cache and 12MB memory running Mandrake 8.0 Linux. Figure 4 shows the ratio of selected two and three dimensional FFTs of size 2^{20} versus a 2^{20} one-dimensional FFT. Since the dimensionless algorithm for each corresponding multi-dimensional DFT should have the same performance as the 1D algorithm, this figure indicates which transforms would benefit from the dimensionless algorithm.

Points above the line equal to one indicate that the 1D FFT is faster and hence a dimensionless FFT using the 1D plan, shown in Figure 3, should be faster than the row-column algorithm. This was verified using our modification to FFTW. For example, the dimensionless FFT for a two-dimensional $2^{15} \times 2^5$ transform is 1.24 times faster than the row-column algorithm used by FFTW. This algorithm was slightly faster than the 1D FFT due to slightly faster codelets.

In summary we have presented a divide and conquer algorithm for computing multi-dimensional DFTs that works independent of the dimension. The benefit is that the decomposition is not constrained by the number of points in each dimension as is the case for the standard row-column algorithm. The dimensionless algorithm can be implemented with minor modifications to existing one-dimensional FFT programs. This approach was carried out using FFTW and led to performance gains for many multi-dimensional FFTs.

4. REFERENCES

[1] James W. Cooley and J.W. Tukey, "An algorithm for the machine calculation of complex series," *Math. Comput.*, vol. 19, pp. 297–301, 1965.

[2] L. Auslander and J. R. Johnson and R. W. Johnson, "Dimensionless fast Fourier transform method and apparatus," Patent #US6003056, 1999.

[3] L. Auslander and J. R. Johnson and R. W. Johnson, "Dimensionless fast Fourier transforms," Tech. Rep. DU-MCS-97-01, Drexel University, 1997, `http://www.cs.drexel.edu`.

[4] P. Kumhom, J. R. Johnson, and P. Nagvajara, "Design, optimization, and implementation of a universal FFT processor," in *Proc. 13th IEEE International ASIC/SOC Conference*, Washington, DC, Sept. 2000.

[5] M. Frigo and S.G. Johnson, "FFTW: An Adaptive Software Architecture for the FFT," in *ICASSP Conference Proceedings*, 1998, vol. 3, p. 1381.

[6] D. Mirković and S. L. Johnsson, "Automatic Performance Tuning in the UHFFT Library," in *Proc. ICCS*. 2001, LNCS 2073, pp. 71–80, Springer.

[7] J. M. F. Moura, J. Johnson, R. Johnson, D. Padua, V. Prasanna, M. Püschel, and M. M. Veloso, "SPIRAL: Portable Library of Optimized SP Algorithms," 1998, `http://www.ece.cmu.edu/~spiral/`.

[8] C. Van Loan, *Computational Framework for the Fast Fourier Transform*, SIAM, Philadelphia, PA, 1992.

[9] J.R. Johnson, R.W. Johnson, D. Rodriguez, and R. Tolimieri, "A methodology for designing, modifying, and implementing fourier transform algorithms on various architecture," *Circuit, Systems, and Signal Processing*, vol. 9, no. 4, pp. 249–500, 1990.

[10] R. Tolimieri, M. An, and C. Lu, *Algorithms for discrete Fourier transforms and convolution*, Springer, 2nd edition, 1997.