SIAM J. Sci. Comput.
Vol. 20, No. 3, pp. 1094–1114

THE FUTURE FAST FOURIER TRANSFORM?[*]

ALAN EDELMAN[†], PETER McCORQUODALE[‡], AND SIVAN TOLEDO[§]

Abstract. It seems likely that improvements in arithmetic speed will continue to outpace advances in communication bandwidth. Furthermore, as more and more problems are working on huge datasets, it is becoming increasingly likely that data will be distributed across many processors because one processor does not have sufficient storage capacity. For these reasons, we propose that an inexact DFT such as an approximate matrix-vector approach based on singular values or a variation of the Dutt–Rokhlin fast-multipole-based algorithm may outperform any exact parallel FFT. The speedup may be as large as a factor of three in situations where FFT run time is dominated by communication. For the multipole idea we further propose that a method of "virtual charges" may improve accuracy, and we provide an analysis of the singular values that are needed for the approximate matrix-vector approaches.

Key words. parallel Fourier transforms, fast multipole method

AMS subject classifications. 65T20, 65Y05, 65Y20

PII. S1064827597316266

1. Introduction.

In future high-performance parallel computers, improvements in floating-point performance are likely to continue to outpace improvements in communication bandwidth. Therefore, important algorithms for the future may trade off arithmetic for reduced communication. Indeed, with the increasing popularity of networks of workstations and clusters of symmetric multiprocessors, even on present machines it may be worthwhile to make this tradeoff.

Traditional research into algorithmic design for the fast Fourier transform (FFT) focuses on memory and cache management and organization. All such algorithms are in effect variations of the original algorithm of Cooley and Tukey [7]. A few important variants are the Stockham framework [6], which reorders data at each step, the Bailey method [4], which minimizes the number of passes through external data sets, Swarztrauber's method [18] for hypercubes and vector supercomputers, and the recent algorithm by Cormen and Nicol [8], which reorganizes data for out-of-core algorithms. Many other important references may be found in Van Loan [20]. In this paper, we believe that we are first to propose a parallel Fourier transform algorithm that would not be exact in the absence of roundoff error.

In our distributed-memory model, we assume that the input vector is stored in natural order, with each processor holding a contiguous portion of the data. The output vector should be distributed the same way. In this model, the standard approach to the parallel FFT is known as the "six-step framework" [20, pp. 173–174], consisting of (1) a global bit reversal or shuffle, (2) local FFTs, (3) a global transpose, (4) multiplication by twiddle factors, (5) local FFTs, and (6) a global shuffle or bit

[*]Received by the editors February 7, 1997; accepted for publication (in revised form) August 12, 1997; published electronically January 29, 1999. A preliminary version of this paper was presented at the Eighth SIAM Conference on Parallel Processing, Minneapolis, MN, 1997. The work of the second and third authors was supported by NSF grants 9501278-DMS and 9404326-CCR.

http://www.siam.org/journals/sisc/20-3/31626.html

[†]Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139-4307 (edelman@math.mit.edu),

[‡]Lawrence Berkeley National Laboratory, Berkeley, CA 94720 (PWMcCorquodale@lbl.gov).

[§]Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304 (toledo@parc.xerox.com). The work was done while this author was at the IBM T. J. Watson Research Center, Yorktown Heights, NY.

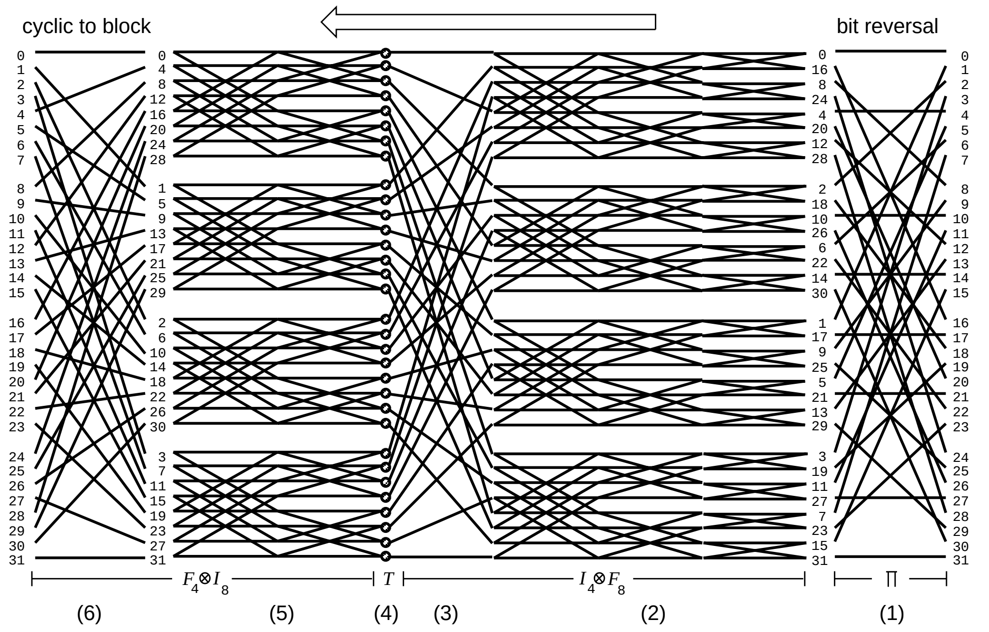

FIG. 1. *Communication pattern in parallel FFT of length* 32 *over* 4 *processors, using the* 6*-step framework based on the factorization* $F_{32} = (F_4 \otimes I_8)T(I_4 \otimes F_8)\Pi$ *of* (8) *in section* 4. *The step numbers are indicated at the bottom of the figure.*

reversal. The global shuffles in steps (1) and (6) each require an amount of communication equivalent to the transpose in step (3). They may be saved if the order is not important. The communication pattern is as indicated in Figure 1, which is based on Gupta et al. [15].

This paper presents a method which can save up to a factor of three in communication cost, by using an approximate algorithm that essentially combines the three global transposes into one. Accuracy can be extended to full machine precision with negligible effect on communication complexity.

The paper is organized as follows. Section 2 contains a mathematical discussion of the singular-value ideas that explain why a reduction in communication is possible. Section 3 introduces a matrix-vector multiply algorithm that uses an offline singular-value analysis. Section 4 introduces our parallel fast multipole algorithm, an equispaced variation of the nonequispaced Fourier transform proposed by Dutt and Rokhlin [10]. Section 5 discusses the results of numerical experiments.

The main contributions of this work are

- the proposal that these algorithms in the parallel context may in fact be faster than the traditional algorithms;
- a mathematical analysis of why these methods work in terms of singular values and their connection to the prolate matrix;
- a portable prototype MPI code that demonstrates the accuracy of the algorithm; and
- an improvement of the Dutt–Rokhlin algorithm that our experiments show often yields two additional digits of accuracy in the equispaced case.

We conclude the introduction by pointing out an interesting fact about all exact Fourier transforms that became clear to one of us while working at Thinking Machines

Corporation. The FFT always starts with long-distance butterfly moves. To be precise, assuming the data are stored in standard (serial) order and need to end up that way (often this can be avoided!), then to perform the first set of butterflies, communication with long strides must occur. This is true no matter whether we use decimation in time, decimation in frequency, or any inverse FFT.

2. Mathematical insights.

In this section, we do not yet present our algorithms, but rather provide mathematical explanations as to why we expect that more communication-efficient algorithms may be found. The most important underlying critical idea is the notion of near-rank deficiency. The operators that represent the relationship between the input on one processor and the output on another processor are nearly rank-deficient. Therefore, as is well known to those who wish to compress images [3], this represents an opportunity to replace the operator with its more economical rank-deficient cousin, thereby gaining speedups on parallel supercomputers. We shall see later that the existence of a multipole algorithm is really a way of taking advantage of this fact.

We can mathematically press further and ask for an explanation of why we are lucky enough to be in this near-rank deficiency situation at all. The answer to such a question may be found in an understanding of the link between our linear operator and its continuous limiting form. Such an understanding is closely related to the mathematics of prolate functions, which we shall explain in this section.

The DFT of $x \in \mathbb{C}^n$ is $y = F_n x$, where

$$[F_n]_{jk} = \exp(-2\pi i jk/n) \qquad (0 \leq j, k \leq n-1).$$

Let $F_{n|p}$ denote the top-left $m \times m$ submatrix of the unitary matrix $\frac{1}{\sqrt{n}} F_n$, where $m = n/p$ is an integer.

PROPOSITION 1. *The singular values of $F_{n|p}$ are strictly between 0 and 1.*

Proof. Since $F_{n|p}$ is a submatrix of a unitary matrix, its singular values are at most 1. Moreover, $F_{n|p}$ is a Vandermonde matrix and hence nonsingular.

The CS decomposition [11, p. 77] of $\frac{1}{\sqrt{n}} F_n$ shows that if any singular value of $F_{n|p}$ is equal to 1, then 0 occurs as a singular value of a rectangular block of F_n. But this is not possible because the rectangular block would be a section of a Vandermonde matrix, which has full rank. □

PROPOSITION 2. *If $p = 2$, then singular values occur in sine-cosine pairs, that is, $\sigma_j^2 + \sigma_{m+1-j}^2 = 1$.*

Proof. If $\frac{1}{\sqrt{n}} F_n$ is split into four blocks, then all four blocks differ from $F_{n|2}$ by a diagonal unitary matrix, and hence they have the same singular values. Then the CS decomposition shows that the singular values must occur in sine-cosine pairs. □

For any p, the singular values of $F_{n|p}$ have an interesting property suggested by the plot in Figure 2: a fraction $1/p$ of them are close to 1, and the rest are close to 0.

This is a remarkable property of sections of the Fourier matrix. By contrast, if we take a random unitary matrix (with Haar measure) and plot the singular values of a section, we find that for $p = 4$ the singular values appear to be uniformly distributed on the interval $(0, \sqrt{3}/2)$, as shown in Figure 3. A discussion of the behavior of singular values of a section with $p \neq 4$ is beyond the scope of this paper.

An exact statement of the near-rank deficiency of a Fourier matrix section is obtained by bounding the number of singular values away from 0 and 1.

THEOREM 3. *For fixed p and $0 < \epsilon < \frac{1}{2}$, let $S_n(a, b)$ represent the number of*

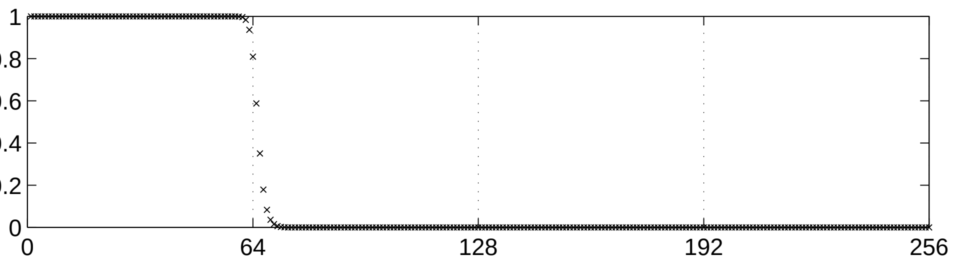

FIG. 2. *Singular values of $F_{1024|4}$, computed with* MATLAB.

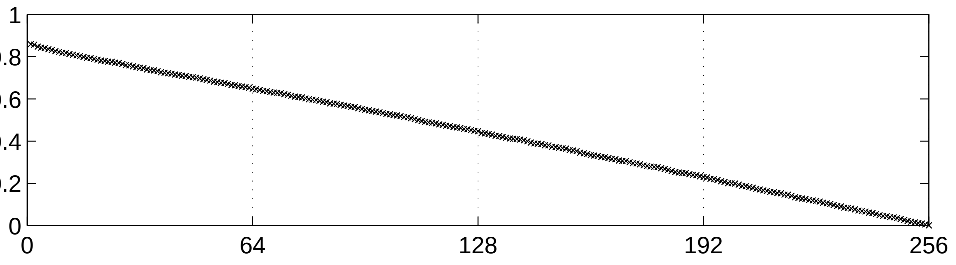

FIG. 3. *Singular values of a 256×256 section of a random 1024×1024 unitary matrix, computed with* MATLAB.

singular values of $F_{n|p}$ in the interval (a, b). Then asymptotically, with $m = n/p$,

$$\begin{aligned} S_n(0,\epsilon) &\sim \left(1 - \frac{1}{p}\right) m, \\ S_n(\epsilon, 1-\epsilon) &\sim O(\log n), \\ S_n(1-\epsilon, 1) &\sim \frac{1}{p} m. \end{aligned}$$

Proof. Up to unitary phase factors, $F_{n|p}^* F_{n|p}$ is equal to the $m \times m$ matrix $G_{n|p} = [g_{jk}]$, where

$$g_{jk} = \begin{cases} \frac{1}{p} & \text{if } j = k, \\ \frac{\sin(\pi(k-j)/p)}{n \sin(\pi(k-j)/n)} & \text{if } j \neq k. \end{cases} \tag{1}$$

The singular values $\sigma_1, \ldots, \sigma_m$ of $F_{n|p}$ are the positive square roots of the eigenvalues of $G_{n|p}$. Take the Frobenius norm of $G_{n|p}$:

$$\begin{aligned} \|G_{n|p}\|_F^2 = \sum_{j,k} |g_{jk}|^2 &= \frac{m}{p^2} + \frac{1}{n^2} \sum_{j \neq k} \frac{\sin^2\left(\frac{\pi(k-j)}{p}\right)}{\sin^2\left(\frac{\pi(k-j)}{n}\right)} = \frac{m}{p^2} + \frac{2}{n^2} \sum_{l=1}^{m-1} (m-l) \frac{\sin^2\left(\frac{\pi l}{p}\right)}{\sin^2\left(\frac{\pi l}{n}\right)} \\ &= \frac{m}{p^2} + \frac{2}{n^2} m \sum_{l=1}^{m-1} \frac{\sin^2\left(\frac{\pi l}{p}\right)}{\sin^2\left(\frac{\pi l}{n}\right)} - \frac{2}{n^2} \sum_{l=1}^{m-1} l \frac{\sin^2\left(\frac{\pi l}{p}\right)}{\sin^2\left(\frac{\pi l}{n}\right)} \\ &\sim \frac{m}{p^2} + \left(\frac{m}{p} - \frac{m}{p^2}\right) - \frac{1}{\pi^2} \ln \frac{m}{p} \\ &\sim \frac{n}{p^2} - \frac{1}{\pi^2} \ln n. \end{aligned}$$

Since the eigenvalues of $G_{n|p}$ are σ_j^2, we have, by the trace formula,

$$\sum_{j=1}^{m} \sigma_j^2 = m\frac{1}{p} = \frac{n}{p^2}. \tag{2}$$

But σ_j^2 are also the singular values of $G_{n|p}$, so

$$\sum_{j=1}^{m} \sigma_j^4 = \|G_{n|p}\|_F^2 \sim \frac{n}{p^2} - \frac{1}{\pi^2}\ln n. \tag{3}$$

Subtracting (3) from (2), then

$$\sum_{j=1}^{m} \sigma_j^2(1-\sigma_j^2) \sim \frac{1}{\pi^2}\ln n. \tag{4}$$

Since $0 < \sigma_j < 1$, (4) implies $S_n(\epsilon, 1-\epsilon) = O(\log n)$. And then from (2) and (3), we also have $S_n(0,\epsilon) \sim m/p$ and $S_n(1-\epsilon, 1) \sim m(1-1/p)$. $\square$

A better understanding of the transition region is suggested by examining the generating function [13] $g(x) = \sum_{l=-\infty}^{\infty} \widehat{g}(l)e^{2\pi i l x}$ of the *infinite* Toeplitz matrix with entries $g_{jk} = \widehat{g}(k-j)$ given by (1), for $j,k \in \mathbb{Z}$. In this case, we have

$$g(x) = \frac{1}{n}\sum_{j=1}^{m} \delta\left(x - \frac{2j-1-m}{2n}\right),$$

which is a comb function having $m = n/p$ spikes equally spaced in the interval $(-\frac{1}{2p}, \frac{1}{2p})$. This function can be viewed as a discrete version of the periodic window function defined on $[-\frac{1}{2}, \frac{1}{2}]$ as

$$w(x) = \begin{cases} 1 & \text{if } x \in \left[-\frac{1}{2p}, \frac{1}{2p}\right], \\ 0 & \text{if } x \in \left[-\frac{1}{2}, -\frac{1}{2p}\right) \cup \left(\frac{1}{2p}, \frac{1}{2}\right], \end{cases}$$

which is the generating function of an infinite Toeplitz matrix with entries

$$w_{jk} = \widehat{w}(k-j) = \frac{\sin(\pi(k-j)/p)}{\pi(k-j)}.$$

The finite version W_p of this matrix is known as the *prolate matrix* [21], and it has been well studied in the signal-processing literature. Observe that $G_{n|p}$ converges elementwise to W_p:

$$\lim_{n\to\infty} g_{jk} = w_{jk}.$$

Slepian [17] observed that the prolate matrix W_p also has eigenvalues clustered near 0 and 1, with a fraction $1/p$ of them near 1 and the rest near 0. In the transition region near $j = m/p$, the jth largest eigenvalue is given asymptotically by

$$\lambda_j(W_p) \sim \frac{1}{1+\exp(\pi\beta)},$$

TABLE 1
A listing of eigenvalues in a transition interval for the matrices $G_{n|p}$ and W_p, of order n/p, and the approximation (5).

p	n	n/p	j	$\lambda_j(G_{n\|p})$	$\lambda_j(W_p)$	approx
2	512	256	127	0.8774	0.8643	0.8588
			128	0.6545	0.6465	0.6460
			129	0.3455	0.3535	0.3540
			130	0.1226	0.1357	0.1412
2	1024	512	255	0.8575	0.8453	0.8408
			256	0.6425	0.6356	0.6352
			257	0.3575	0.3644	0.3648
			258	0.1425	0.1547	0.1592
4	1024	256	63	0.8774	0.8743	0.8681
			64	0.6545	0.6525	0.6521
			65	0.3454	0.3473	0.3479
			66	0.1226	0.1257	0.1319

where $j = \lceil m/p + \frac{\beta}{\pi}\log m \rceil$. In the region $0.2 < \lambda_j < 0.8$, Slepian found that a good approximation [17, eqns. 61–62] is

$$\lambda_j(W_p) \sim \frac{1}{1 + \exp\left(\pi\widehat{\beta}\right)}, \tag{5}$$

where

$$\widehat{\beta} = \frac{\pi(j - 1/2 - m/p)}{\log(8m|\sin(\pi/p)|) + \gamma}$$

and $\gamma = 0.5772156649$ is the Euler–Mascheroni constant. Table 1 shows that approximation (5) is a good predictor of the eigenvalues of the prolate matrix, W_p, which are also close to the eigenvalues of $G_{n|p}$.

These results on the eigenvalue distribution of the prolate matrix come from finding the asymptotics of discrete prolate spheroidal functions, which are solutions of a differential equation with a differential operator related to a tridiagonal matrix that commutes with the prolate matrix. A similar analysis may, in principle, be applicable to $G_{n|p}$. Grünbaum [14] makes the first step in this direction by finding a tridiagonal matrix that commutes with $G_{n|p}$.

3. Algorithm 1: A matrix-vector algorithm. Given that the sections $F_{n|p}$ are nearly rank-deficient, we may borrow an idea from image compression [3] and take a singular-value decomposition. Our first algorithm for the DFT involves nothing more than matrix-vector multiplication by SVD matrices. In the equation $y = F_n x$, if we write F_n as p^2 blocks of size $m = n/p$ and if p^2 divides n, then

$$\begin{pmatrix} y(0\colon m-1) \\ y(m\colon 2m-1) \\ \vdots \\ y(n-m\colon n-1) \end{pmatrix} = \sqrt{n} \begin{pmatrix} F_{n|p} & \cdots & D^{p-1}F_{n|p} \\ F_{n|p}D & \cdots & D^{p-1}F_{n|p}D \\ \vdots & & \vdots \\ F_{n|p}D^{p-1} & \cdots & D^{p-1}F_{n|p}D^{p-1} \end{pmatrix} \begin{pmatrix} x(0\colon m-1) \\ x(m\colon 2m-1) \\ \vdots \\ x(n-m\colon n-1) \end{pmatrix}, \tag{6}$$

where

$$D = \operatorname{diag}(1, \eta, \eta^2, \ldots, \eta^{m-1}), \qquad \eta = \exp(-2\pi i/p).$$

Since the number of significant singular values of $F_{n|p}$ is asymptotically only m/p, this suggests the idea of using compression to reduce communication.

The singular-value decomposition of $F_{n|p}$ is written

$$F_{n|p} = U\Sigma V^*,$$

where U and V are $m \times m$ unitary matrices, and Σ is a diagonal matrix of singular values. Let Σ_k denote the matrix consisting of the first k rows of Σ, containing the k largest singular values. The value of k depends on the accuracy desired, but for fixed accuracy and fixed p, the results of the previous section tell us $k = m/p + O(\log m)$.

Let U_k be the first k columns of U. We shall use the approximation

$$F_{n|p} \approx U_k \Sigma_k V^*. \tag{7}$$

If we precompute the $k \times m$ matrices

$$A^{(r)} = \Sigma_k V^* D^r \qquad (0 \le r \le p-1)$$

and the $m \times k$ matrices

$$B^{(r)} = D^r U_k \qquad (0 \le r \le p-1),$$

then we have the following algorithm. The input vector x is distributed across the p-processors, as are the intermediate vectors $u^{(r)}$ and $v^{(r)}$ (of length pk), and the output vector y.

ALGORITHM 1. Matrix-vector DFT.

```
1. **foreach** processor $s = 0 : p-1$:
   **for** $r = 1 : p$
       $v^{(r)}(sm\colon (s+1)m-1) \leftarrow A^{(r)} x(sm\colon (s+1)m-1)$
   **end**
2. **foreach** processor $s = 0 : p-1$:
   **for** $r = 1 : p$
       Send $v^{(r)}(sm\colon (s+1)m-1)$ to processor $r$;
       Receive $u^{(r)}(sm\colon (s+1)m-1)$ from processor $r$.
   **end**
3. **foreach** processor $s = 0 : p-1$:
   $y(sm\colon (s+1)m-1) \leftarrow \sum_{r=0}^{p-1} B^{(r)} u^{(r)}(sm\colon (s+1)m-1)$
```

3.1. Accuracy. Let $\widetilde{F}_n$ denote an approximation to F_n obtained by replacing $F_{n|p}$ in (6) by $U_k \Sigma_k V^*$ from approximation (7).

LEMMA 4. *For all* $x \in \mathbb{C}^n$, $\|F_n x - \widetilde{F}_n x\|_2 / \|F_n x\|_2 \le p(\sum_{j=k+1}^m \sigma_j^2)^{1/2}$.

Proof.

$$\|F_n - \widetilde{F}_n\|_2 \le \|F_n - \widetilde{F}_n\|_F = p\sqrt{n}\|U\Sigma V^* - U_k \Sigma_k V^*\|_F = p\sqrt{n}\left(\sum_{j=k+1}^{m} \sigma_j^2\right)^{1/2},$$

and for $x \in \mathbb{C}^n$, $\|F_n x\|_2 = \sqrt{n}\|x\|_2$. ◻

Table 2 shows k, the number of significant singular values, that must be used with several choices of n and p in order to obtain selected values of relative accuracy. These results were computed on a Sun Ultra Enterprise 5000 with the SunSoft Performance Library.

Table 2
Number of significant singular values required for given relative accuracy.

	Size		Relative accuracy		
p	n	n/p^2	10^{-5}	10^{-8}	10^{-11}
2	1024	256	271	278	284
	2048	512	529	537	544
	4096	1024	1043	1051	1060
4	1024	64	78	84	89
	2048	128	144	151	157
	4096	256	274	282	289
	8192	512	532	540	549
8	1024	16	28	32	36
	2048	32	45	51	55
	4096	64	79	85	91
	8192	128	145	152	159

3.2. Complexity. We first analyze the arithmetic complexity of Algorithm 1, which is based on multiplications of complex matrices. In each of steps 1 and 3, every processor performs p complex matrix-vector multiplications, each requiring $8km$ flops. Hence the total number of flops per processor is $16mkp = 16m^2 + O(m \log m)$.

As for communication, all of it takes place in step 2, where there is an all-to-all personalized communication with each processor sending k scalars to each other processor. The total number of scalars sent by each processor is $(p-1)k = m(1 - 1/p) + O(\log m)$.

The FFT, by comparison, has each processor sending $3m(1-1/p)$ scalars but uses only $5m \lg n$ flops. The matrix-vector multiplication algorithm using the SVD saves as much as a factor of three in communication at the cost of greater arithmetic. In the next section, we show how a different algorithm using the fast multipole method can reduce the arithmetic but maintain this saving in communication.

4. Algorithm 2: Fast multipole approach.

The near-rank deficiency shown in section 2, together with the understanding that the Fourier transform is a highly structured problem, leads to the suggestion that a multipole-based algorithm may be appropriate. The suggestion is subtle, for it is the converse that is really correct. It is well known that we can cluster charges or particles when evaluating potential fields far away. In the language of linear algebra, the linear operator that transforms charge or mass to faraway potentials is approximately low rank. It is therefore intuitively reasonable to attempt to identify nearly low rank matrices that arise from highly structured mathematical problems with the evaluation of some sort of potential field. We shall see that commuting the so-called "twiddle factor" matrix through the "butterfly" operations leads to just this sort of identification.

4.1. Matrix factorizations. For parallel FFT computations over p processors, the standard "six-step framework" [20, pp. 173–174] is based on the radix-p splitting [20, eqn. 2.1.5], a factorization of the Fourier matrix as

$$F_n = (F_p \otimes I_m)T(I_p \otimes F_m)\Pi, \tag{8}$$

where again $m = n/p$ and T is a diagonal matrix of twiddle factors,

$$T = \operatorname{diag}(I_m, \Omega, \Omega^2, \ldots, \Omega^{p-1}),$$

$$\Omega = \operatorname{diag}(1, \omega, \omega^2, \ldots, \omega^{m-1}), \qquad \omega = \exp(-2\pi i/n),$$

and Π is a block-to-cyclic permutation that acts on the columns of the identity matrix as

$$\Pi e_{j+kp} = e_{k+jm} \qquad (0 \leq j \leq p-1, 0 \leq k \leq m-1).$$

Our algorithm will use a factorization

$$F_n = (I_p \otimes F_m)(F_p \otimes I_m)M\Pi. \tag{9}$$

To solve for M, use the fact that $I_p \otimes F_m$ and $F_p \otimes I_m$ commute, so

$$(F_p \otimes I_m)T(I_p \otimes F_m)\Pi = F_n = (F_p \otimes I_m)(I_p \otimes F_m)M\Pi,$$

which gives

$$\begin{aligned} M &= (I_p \otimes F_m)^{-1}T(I_p \otimes F_m) \\ &= \operatorname{diag}(I_m, F_m^{-1}\Omega F_m, \ldots, F_m^{-1}\Omega^{p-1}F_m) \\ &= \operatorname{diag}(I_m, C^{(1)}, \ldots, C^{(p-1)}), \end{aligned} \tag{10}$$

where the matrices $C^{(s)} = \left(c_{jk}^{(s)}\right)$ have elements

$$c_{jk}^{(s)} = \rho^{(s)} \left[\cot\left(\frac{\pi}{m}\left(k - j + \frac{s}{p}\right)\right) + i\right],$$

with $\rho^{(s)} = \frac{1}{m}\exp(-i\pi s/p)\sin(\pi s/p)$.

For fast multiplication by $C^{(s)}$, we can use the one-dimensional fast multipole method (FMM) of Dutt, Gu, and Rokhlin [9]. Dutt and Rokhlin [10] use a similar approach in a serial algorithm to compute the discrete Fourier transform on a nonequispaced set of points.

4.2. General approach. In evaluating $y = F_n x$, we assume that $x \in \mathbb{C}^n$ is stored in block order and $y \in \mathbb{C}^n$ should also end in block order. That is, $x(\mu m\colon(\mu+1)m-1) \in \mathit{Proc}(\mu)$ at the beginning, and $y(\mu m\colon(\mu+1)m-1) \in \mathit{Proc}(\mu)$ at the end of the algorithm.

One possible approach is as follows:

1. Perform the distributed permutation Πx.
2. In processor $\mu = 1 : p-1$, multiply local vector by $C^{(\mu)}$.
3. Do m distributed FFTs of size p, with one element from each processor (corresponds to $F_p \otimes I_m$).
4. In each processor, do a local FFT of size m (corresponds to $I_p \otimes F_m$).

This method requires two distributed permutations, one in step 1 and the other in the distributed FFT in step 3.

We use an alternative approach that avoids performing the distributed permutation Π directly. Instead, we combine steps 1 and 2, doing each of the p multiplications by $C^{(s)}$ matrices in parallel. In terms of total number of scalars sent, the communication requirements are reduced by nearly half. In the description of the algorithm below, each of the vectors $v^{(s)}$, of length m, is distributed blockwise across the p processors, as are x and y, which are of length n.

ALGORITHM 2. Fast multipole DFT (Dutt and Rokhlin [10]).

```
1. for $s = 1 : p-1$
       $\sigma^{(s)} \leftarrow \sum_{k=0}^{m-1} x(s+kp)$
   end
2. for $s = 1 : p-1$
       $v^{(s)} \leftarrow \sum_{k=0}^{m-1} \cot\left(\frac{\pi}{m}\left(k - (0{:}\, m-1) + \frac{s}{p}\right)\right) \cdot x(s+kp)$
       {evaluated approximately using Algorithm 3}
   end
3. $v^{(0)} \leftarrow x(0{:}\, p{:}\, (m-1)p)$
   for $s = 1 : p-1$
       $v^{(s)} \leftarrow \rho^{(s)} \left(i\sigma^{(s)} + v^{(s)}\right)$
   end
4. $\left[v^{(0)}, \dots, v^{(p-1)}\right] \leftarrow \left[v^{(0)}, \dots, v^{(p-1)}\right] F_p$
   {evaluated as local FFTs of length $p$}
   $y \leftarrow \begin{bmatrix} v^{(0)} \\ v^{(1)} \\ \vdots \\ v^{(p-1)} \end{bmatrix}$
   {distributed transpose}
5. $y(\mu m{:}\, (\mu+1)m-1) \leftarrow F_m y(\mu m{:}\, (\mu+1)m-1)$
   {evaluated as local FFT of length $m$ in $Proc(\mu)$}
```

4.3. The fast multipole method. The heart of Algorithm 2 consists of the distributed matrix multiplications of step 2. We view each of these $p-1$ transformations as a mapping of m charges on a circle to values of the potential due to these charges, at points on the circle. The potential due to a charge of strength q, at a point separated from it by a clockwise arc subtended by an angle θ, is given by $q\cot(\theta/2)$. Here the charges and potential evaluation points are both spaced equally along the circumference of the circle. In the sth potential mapping, the charge locations and evaluation points are offset from each other by an arc of length $2\pi s/n$.

Dutt and Rokhlin [10] showed how the nonequispaced Fourier transform can be computed using the FMM. In this article, we are restricted to an equispaced DFT but we use a different set of interpolating functions that offer greater accuracy in this restricted case. We also compute it in parallel, using the method of Greengard and Gropp [12] and Katzenelson [16].

We divide the input, m particles, into 2^h boxes, each containing $b = m/2^h$ particles. In the tree code for the one-dimensional FMM, there will be $h-1$ levels in the tree. At level h, the lowest level, the number of boxes, 2^h, is not necessarily equal to the number of processors, p, but 2^h should be a multiple of p. The number of boxes at the lowest level is chosen so as to minimize the total amount of arithmetic to be performed. At higher levels, numbered $h-1 \geq l \geq 2$, the number of boxes at level l is 2^l, and each box is formed by combining two boxes at the level immediately below it. There are four boxes at the top level, level 2.

The number of interpolation coefficients, t, must also be chosen high enough to obtain sufficient accuracy. In general, t will depend on the size of the problem and the number of particles in each box. Finite machine precision, however, will also provide an upper limit on t beyond which improvements in accuracy are not obtained.

In the following code, we may assume that each box is in a separate processor,

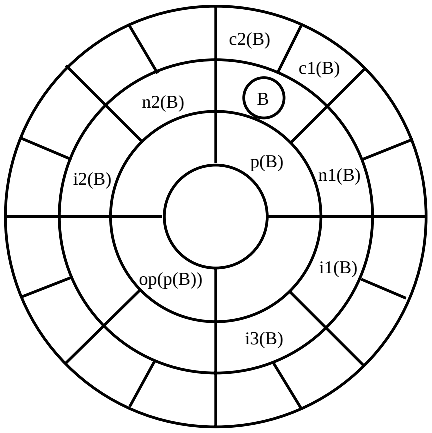

Fig. 4. *Box B at level 3 with children $c1(B)$ and $c2(B)$ at level 4, parent $p(B)$ at level 2, neighbors $n1(B)$ and $n2(B)$, and interaction list $i1(B), i2(B), i3(B)$. At level 2, the "highest" level, box $p(B)$ is opposite box $op(p(B))$.*

although the communication we are concerned about is that between boxes in different processors. Each box B is adjacent to two other boxes, labelled $n1(B)$ and $n2(B)$, and if B is below level 2 it has a parent labeled $p(B)$. A box B at level 2 is opposite to a box labeled $op(B)$.

The *interaction list* of box B is the set of boxes that are children of neighbors of B's parent but are not neighbors of B. The interaction list of each box at levels $l \geq 3$ consists of three boxes. See Figure 4.

For each box k at level l, the algorithm calculates $\Phi_{l,k}$, a far-field expansion containing t interpolation coefficients, representing the potential due to particles inside box k. The algorithm also calculates $\Psi_{l,k}$, a local expansion containing t interpolation coefficients, representing the potential due to particles outside box k and its immediate neighbors.

Algorithm 3. Fast multipole potential solver (Dutt, Gu, and Rokhlin [9]).

```
1. **foreach** box $k$ at bottom level $h$: form the $t$-term far-field expansion $\Phi_{h,k}$.
2. {Form the far-field expansions at all higher levels.}
   **for** $l = h - 1 : -1 : 2$
       **foreach** box $k$ at level $l$:
       $\Phi_{l,k} \leftarrow \mathrm{SHFTF}(\Phi_{l+1,c1(k)}, \pi/2^{l+1}) + \mathrm{SHFTF}(\Phi_{l+1,c2(k)}, -\pi/2^{l+1})$
   **end**
3. {Form the local expansions at every level.}
   **foreach** box $k$ at level 2:
   $\Psi_{2,k} \leftarrow \mathrm{FLIP}(\Phi_{2,op(k)}, 2)$
   **for** $l = 3 : h$
       **foreach** box $k$ at level $l$: $\Psi_{l,k} \leftarrow \mathrm{SHFTL}(\Psi_{l-1,p(k)}, (-1)^k \pi/2^l) +$
       $\mathrm{SHFTL}(\mathrm{FLIP}(\Phi_{l,i1(k)}, l), 2\pi/2^{l-1}) +$
       $\mathrm{SHFTL}(\mathrm{FLIP}(\Phi_{l,i2(k)}, l), -2\pi/2^{l-1}) +$
       $\mathrm{SHFTL}(\mathrm{FLIP}(\Phi_{l,i3(k)}, l), (-1)^k 3\pi/2^{l-1})$
   **end**
4. **foreach** box $k$ at level $h$: Evaluate $\Psi_{h,k}$ at the potential locations of box $k$.
5. **foreach** box $k$ at level $h$: Compute and sum up the direct potentials due to
   particles in boxes $k$, $n1(k)$, and $n2(k)$, at the potential locations of box $k$.
```

6. **foreach** box k at level h: Add up the results from the previous two steps for each potential location of box k.

At each level, the operations SHFTF, FLIP, and SHFTL can be computed by multiplying the vector of coefficients by a real $t \times t$ matrix. The initial far-field expansion of step 1 is obtained by multiplying the vector of b charges by a real $t \times b$ matrix.

4.4. Interpolation functions and accuracy. In their t-term interpolations for the Fourier transform, Dutt and Rokhlin [10] use polynomial functions of $\cot(\theta/2)$ and degree t. (This is equivalent to using a finite series of derivatives of $\cot(\theta/2)$.) To approximate the potential at angular positions $\theta \in [6a, 2\pi - 6a]$, due to charges in the interval $-2a \leq \theta \leq 2a$, the Dutt–Rokhlin interpolation scheme sets $x = 3 \tan a \cot(\theta/2)$ and for $x \in [-1, 1]$ sets

$$\widetilde{f}(x) = \sum_{j=1}^{t} f(c_j) \prod_{k \neq j} \left(\frac{x - c_k}{c_j - c_k} \right), \tag{11}$$

where $c_1, \dots c_n$ are the Chebyshev nodes, $c_j = -\cos((j - 1/2)\pi/t)$.

In our approach, the interpolation functions are the potentials due to t "virtual charges" at fixed positions $\theta'_1, \dots, \theta'_t$ given by $\theta'_j = -2a \cos((j-1)\pi/(t-1))$. Then we have

$$\widetilde{f}(x) = \sum_{j=1}^{t} f(c_j) \prod_{k \neq j} \left(\frac{x - c_k}{c_j - c_k} \right) \prod_{l=1}^{t} \left(\frac{c_j - x'_l}{x - x'_l} \right), \tag{12}$$

with $x'_l = 3 \tan a \cot(\theta'_l/2)$.

For local expansions, where the potential at $\theta \in [-2a, 2a]$ is calculated due to charges in $6a \leq \theta \leq 2\pi - 6a$, the Dutt–Rokhlin scheme uses (11) with $x = \cot(a)/\cot(\theta/2)$. In our approach, we use (12) with "virtual charges" at $\theta'_j = \pi + (6a - \pi)\cos((j-1)\pi/(t-1))$, and $x'_j = \cot(a)/\cot(\theta'_j/2)$.

When using the virtual-charge approach, the shift and flip matrices in the multipole algorithm will depend on the level of the tree, but greater accuracy is obtained with the same number of coefficients.

Using the proof methods as Dutt and Rokhlin [10], we can show that the relative error with the new interpolation functions is $O((1/11)^t)$. This compares with an error bound of $O((1/5)^t)$ using Chebyshev polynomials. The analysis is included in the second author's doctoral dissertation [23].

If $\widetilde{C}^{(s)}$ denotes an approximation to $C^{(s)}$ in (10) and $\widetilde{F}_n$ is the resulting approximation to F_n from (9), then

$$\|F_n x - \widetilde{F}_n x\|_2 \leq \max_s \{\|C^{(s)} - \widetilde{C}^{(s)}\|_2\} \|F_n x\|_2.$$

Therefore the maximum relative 2-norm error in computation of $F_n x$ is

$$\max_s \{\|C^{(s)} - \widetilde{C}^{(s)}\|_2\}, \tag{13}$$

which is plotted in Figure 5 for a problem of size $n = 32,768$ with $p = 4$. These results were obtained by computing the matrix norms with a power method using MATLAB. With both the Chebyshev polynomial and virtual-charge interpolation schemes, using an even number of coefficients, t, the error actually increases over using $t - 1$ coefficients. Hence odd t is recommended.

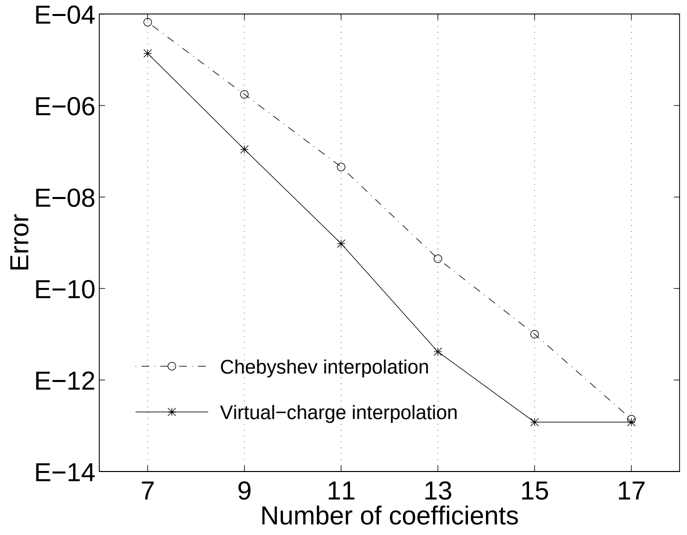

FIG. 5. *Maximum relative* 2*-norm error as a function of number of coefficients used, with Chebyshev polynomial interpolation (dashed line) and virtual-charge interpolation (solid line). The problem size is* $n = 32,768$*, with* $p = 4$ *processors. Results were computed from formula* (13) *using* MATLAB.

The authors have also found that when using double-precision arithmetic, accuracy is not improved for increasing t above 15, because of the effects of roundoff error. In fact, with $t = 15$ for virtual charges or $t = 17$ for Chebyshev, the computed maximum error for a problem of this size is less than the error of 3×10^{-13} that we obtain using random data in MATLAB.

4.5. Arithmetic complexity. We first analyze the arithmetic complexity of Algorithm 3 by itself. The algorithm is based on multiplications of vectors of complex numbers by real matrices. To multiply a real $\mu \times \nu$ matrix by a complex vector of length ν requires $4\mu\nu$ flops. Here is a step-by-step analysis of the number of flops used by each processor in Algorithm 3.

1. $\frac{2^h}{p} 4tb = \frac{4mt}{p}$:
   Level h has 2^h boxes. A vector of length b containing charges is multiplied by a $t \times b$ matrix.
2. $\sum_{l=2:h-1} \lceil \frac{2^l}{p} \rceil 2 \cdot 4t^2 = 8t^2(\lg p - 2 + \frac{2^h - p}{p}) = 8t^2(\frac{m}{bp} + \lg p - 3)$:
   Level l has 2^l boxes. The two SHFTF matrices are $t \times t$.
3. $\lceil \frac{4}{p} \rceil 4t^2 + \sum_{l=3:h} \lceil \frac{2^l}{p} \rceil 4 \cdot 4t^2 \leq 4t^2 + 16t^2(\lg p - 3 + \frac{2^{h+1} - p}{p}) = 16t^2(\frac{2m}{bp} + \lg p - 15/4)$:
   Level l has 2^l boxes. The four matrices are t×t.
4. $\frac{2^h}{p} 4bt = \frac{4mt}{p}$:
   Level h has 2^h boxes. Vector of length t containing coefficients is multiplied by a $b \times t$ matrix.
5. $\frac{2^h}{p} \cdot 3 \cdot 4b^2 = \frac{12mb}{p}$:
   Level h has 2^h boxes. Vector of length b containing charges is multiplied by a $b \times b$ matrix.
6. $\frac{2^h}{p} 2b = \frac{2m}{p}$:

Level h has 2^h boxes. Vectors are of length b.

The total number of flops per processor for each potential problem solved by the fast multipole method is

$$\frac{8mt}{p} + \frac{12mb}{p} + \frac{40t^2m}{bp} + 24t^2 \lg p - 84t^2 = \frac{8mt}{p} + \frac{4m}{p}\left(3b + \frac{10t^2}{b}\right) + 12t^2(2\lg p - 7).$$

We choose b, the number of particles per box, to minimize this quantity. The best choice is $b_{opt} = t\sqrt{10/3}$, but since b should be a power of 2, we choose some $b_{opt}/\sqrt{2} \leq b \leq b_{opt}\sqrt{2}$. Then the number of flops per processor for a potential problem (Algorithm 3) is at most

$$55\frac{mt}{p} + 12t^2(2\lg p - 7).$$

In Algorithm 2 for the DFT, we solve $p-1$ multipole problems in parallel in step 2. Here is a step-by-step analysis of the number of flops used by each processor in Algorithm 2.

1. $(p-1)2m/p$.
2. $(p-1)[55\frac{mt}{p} + 12t^2(2\lg p - 7)]$ (at most).
3. $(p-1)8m/p$.
4. $m \cdot 5p \lg p/p = 5m \lg p$:
   There are m FFT evaluations of size p.
5. $5m \lg m$:
   In each processor there is an FFT evaluation of size m.

Adding up and replacing $m = n/p$, the total number of flops per processor is bounded above by

$$\frac{n}{p}\left[5\lg n + \left(1 - \frac{1}{p}\right)(10 + 55t)\right] + 12t^2(2\lg p - 7)(p-1). \tag{14}$$

For $t = 15$, the number of flops per processor is

$$\frac{n}{p}\left[5\lg n + 835\left(1 - \frac{1}{p}\right)\right] + 2700(2\lg p - 7)(p-1).$$

4.6. Communication complexity. In Algorithm 3, for each multipole problem, the number of scalars sent by any processor in each step is at most

1. 0.
2. $\lg(p/4)t$ at the highest levels of the tree;
   $4t\lg(\frac{m}{bp})$ for sending Φ to neighbors.
3. t, for the top-level flip;
   $(\lg p - 2)4t$, for the other high levels of the tree.
4. 0.
5. $2b$, for sending charges to neighbors.
6. 0.

The total number of scalars sent by a processor in each potential problem is therefore at most

$$2b + t\left(4\lg\left(\frac{m}{b}\right) + \lg p - 9\right).$$

In Algorithm 2 for the DFT, we solve $p-1$ multipole problems in parallel in step 2. Here is a step-by-step analysis of the maximum number of scalars sent by each processor in Algorithm 2:

1. 2.
2. $(p-1)[2b + t(4\lg(\frac{m}{b}) + \lg p - 9)]$
3. 0.
4. $\frac{m}{p}(p-1)$.
5. 0.

So the total number of scalars sent by each processor is

$$(p-1)[m/p + 2b + t(4\lg(m/b) + \lg p - 9)], \tag{15}$$

which for $t = 15$ and $b = 32$ is

$$(p-1)[n/p^2 + 60\lg n - 45\lg p - 371].$$

The total number of messages required to be sent from each processor is at most $2p + 5\lg p - 8$.

5. Experimental results. We have implemented both our new algorithm and a conventional high-performance parallel FFT algorithm in order to assess the accuracy and performance of our algorithm. In this section, the phrase "new algorithm" refers to Algorithm 2 of section 4, and the phrase "conventional parallel FFT" refers to an implementation of the six-step framework described in section 1 and in Figure 1. We use our implementation to show below that the new algorithm is accurate and that it can outperform the performance of conventional parallel FFT algorithms.

Before we proceed to present our experimental results, we would like to state the precise goal of our performance experiments. The experiments are intended to show that the performances of the two algorithms are within a small factor of each other, and that the relative speed of the two algorithms is determined by the communication-to-computation-rates ratio of the parallel computer on which they are executed. When the ratio is high, the conventional algorithm is faster. When the ratio is low, the new algorithm is faster.

Our experiments are *not* intended to show that either of our implementations is a state-of-the-art code that is better than other parallel FFT codes. We do believe, however, that if both implementations are improved to a state-of-the-art level, our new algorithm would still prove faster on machines with fast processors and relatively slow communication network.

5.1. Performance results. This section compares the performance of our implementations of the new algorithm and a conventional high-performance FFT algorithm. Both algorithms are coded in Fortran 77. We use a publicly available FFT package, FFTPACK [19], for performing local FFTs on individual processors, and the message passing interface (MPI) for interprocessor communication. The software is portable and runs without modifications on both the IBM SP2 scalable parallel computer and a cluster of Sun UltraSparc symmetric multiprocessors (SMPs).

The first set of experiments was conducted on an IBM SP2 parallel computer [2]. The machine was configured with so-called thin nodes with 128 Mbytes of main memory. Thin nodes have a 66.7 MHz POWER2 processor [22], 64 Kbytes 4-way set associative level-1 data-cache, no level-2 cache, and a 64-bit-wide main memory bus. They have smaller data paths between the cache and the floating-point units than all other POWER2-based SP2 nodes. The system software that we used includes the AIX version 4.1.3 operating system, Parallel Environment version 2.1 (this includes the message-passing library), and the XLF version 3.2.5 Fortran compiler.

The computation-to-communication balance of the SP2 can be summarized as follows. The peak floating-point performance of POWER2-based nodes is 266 million operations per seconds (Mflops), thanks to two floating-point functional units that can each execute a multiply-add operation in every cycle. While many dense matrix operations run on these nodes at close to peak performance [1], FFT codes run at lower rates. Large power-of-two one-dimensional FFTs from FFTPACK run at 20–30 Mflops, and similar routines from IBM's Engineering and Scientific Subroutine Library (ESSL) run at 75–100 Mflops. When the message-passing libraries use the SP2's high-performance switch (a specialized interconnection network) using the so-called user-space communication protocol, which bypasses the operating system, the communication bandwidth they can achieve is at most 41 Mbytes per second per node. When the libraries use the high-performance switch using the internet protocol (IP), which does not bypass the operating system, the communication bandwidth is at most 13 Mbytes per second per node. When the libraries use IP over Ethernet rather than over the high-performance switch, the bandwidth is even lower, 1.25 Mbytes per second for all the nodes combined.

The running times that are reported here are averages of 10 runs. We ran each experiment 11 times, always discarding the first run, which incurs various startup costs. We also discarded runs in which the running time was more than twice the smallest running time for that experiment, which happened only once. We averaged the other 10 runs (9 in one case). The relative standard deviations were less than 3% when we used user-space communication over the high-performance switch, less than 9% when we used IP over the high-performance switch, and less than 20% when we used IP over Ethernet.

The results of our experiments are summarized in Table 3. The results show that the conventional algorithm is faster when we use the two faster communication mechanisms, and that the new algorithm is faster with the slowest communication mechanism, IP over Ethernet. The absolute running times using Ethernet are very slow. Ethernet is also the only communication mechanism that does not allow additional processors to reduce the absolute running times, since it is a broadcast mechanism in which the total bandwidth does not grow with the number of processors. The high-performance switch allows additional processors to decrease the absolute running times of both algorithms.

Table 3 also shows that the conventional algorithm is more sensitive to degradation in communication bandwidth. For example, on an FFT of 1,048,576 points on 4 processors, the running time of the conventional algorithm increased by 0.932 seconds when we switched from user-space to IP communication over the HPS, but the running time of the new algorithm increased by only 0.423 seconds. The relative increases are 36% and 8%, respectively.

Table 4 describes the experiments with the best communication mechanism in more detail. The table shows that the conventional FFT achieves good speedups. The speedups for the largest problems on 2, 4, and 8 processors are 1.45, 2.66, and 4.77 respectively (where the speedup is defined as $p \cdot T_{\text{fft}}/T$). The volume of communication and the time spent in communication in the new algorithm are smaller by a factor of 2–3 than the volume and time spent by the conventional algorithm. The part spent in computations other than local FFTs is much larger, however, in the new algorithm.

With a flop rate of FR (in flops per second) and a communication bandwidth of BW (in bytes per second), the times we should *expect* for the the conventional parallel

TABLE 3

A comparison of the performance of the two algorithms on an SP2 parallel computer using three communication mechanisms. The table compares the running time of a standard parallel FFT with the running time of the new approximate DFT algorithm. Running times are in seconds. The three communication mechanisms that were used are user-space communication over the high-performance switch (US-HPS), internet protocol over the high-performance switch (IP-HPS), and internet protocol over Ethernet (IP-EN). The last two rows give the minimum and maximum ratios of the timings reported in the table to what we would expect from the sum of (16)–(18) *for* T_C *or* (19)–(21) *for* T_N.

		Communication mechanism					
	Size	US-HPS		IP-HPS		IP-EN	
p	n	standard	new	standard	new	standard	new
2	32768	0.113	0.199	0.164	0.219	0.769	0.443
	65536	0.220	0.399	0.301	0.432	1.526	0.857
	131072	0.471	0.833	0.633	0.888	3.083	1.725
	262144	1.043	1.761	1.354	1.869	6.250	3.535
	524288	2.545	3.987	3.154	4.197	12.976	7.479
4	32768	0.059	0.128	0.109	0.152	1.213	0.602
	65536	0.116	0.268	0.199	0.302	2.368	1.198
	131072	0.220	0.563	0.355	0.614	5.928	2.528
	262144	0.469	1.171	0.719	1.264	11.474	4.902
	524288	1.033	2.441	1.504	2.605	18.540	8.726
	1048576	2.608	5.355	3.540	5.778	37.020	17.000
8	32768	0.031	0.077	0.061	0.101	1.708	1.263
	65536	0.070	0.150	0.114	0.179	3.166	2.117
	131072	0.140	0.296	0.266	0.358	7.225	3.196
	262144	0.265	0.593	0.446	0.681	12.983	5.691
	524288	0.556	1.288	0.866	1.410	22.165	10.097
	1048576	1.172	2.704	1.770	2.924	42.093	17.827
	2097152	2.823	5.926	3.926	6.320	85.428	33.783
	min ratio	5.503	4.984	3.482	4.597	1.193	1.356
	max ratio	10.060	6.890	5.405	6.087	1.639	1.918

FFT are

$$T_{\text{fftloc}} = \left(5\frac{n}{p}\lg n\right)/FR, \tag{16}$$

$$T_{\text{arith}} = \left(6\frac{n}{p}\right)/FR, \tag{17}$$

$$T_{\text{comm}} = \left[3\frac{n}{p}\left(1-\frac{1}{p}\right)\right]\cdot(16\text{ bytes})/BW. \tag{18}$$

The last two rows of Table 4 show the minimum and maximum ratio of the actual times recorded to the times expected with $FR = 266$ Mflops/sec and $BW = 41$ Mbytes/sec.

For the new parallel approximate DFT, (14) and (15) with $t = 16$ give expected times of

$$T_{\text{fftloc}} = \left(5\frac{n}{p}\lg\frac{n}{p}\right)/FR, \tag{19}$$

$$T_{\text{arith}} = \left(890\frac{n}{p}\left(1-\frac{1}{p}\right)\right)/FR, \tag{20}$$

$$T_{\text{comm}} = \left[(p-1)\left(\frac{n}{p^2}+64\lg n-48\lg p-400\right)\right]\cdot(16\text{ bytes})/BW. \tag{21}$$

Table 4

A comparison of the performance of the two algorithms on an SP2 parallel computer. The communication software used the high-performance switch without operating system overhead (US-HPS). Mean times are reported in seconds. The total time is divided into three parts: T_{fftloc} spent in Netlib local FFTs, T_{comm} used for communication, and T_{arith} for other arithmetic. The last two rows give the minimum and maximum ratios of the timings reported in the table to what we would expect from (16)–(21).

		Conventional parallel FFT				New parallel approximate DFT			
Formula →			(16)	(17)	(18)		(19)	(20)	(21)
p	n	T	T_{fftloc}	T_{arith}	T_{comm}	T	T_{fftloc}	T_{arith}	T_{comm}
2	32768	0.113	0.067	0.018	0.029	0.199	0.046	0.141	0.012
	65536	0.220	0.129	0.035	0.055	0.399	0.086	0.292	0.021
	131072	0.471	0.295	0.070	0.105	0.833	0.211	0.583	0.039
	262144	1.043	0.690	0.141	0.212	1.761	0.521	1.166	0.075
	524288	2.545	1.846	0.282	0.417	3.987	1.502	2.338	0.147
4	32768	0.059	0.026	0.008	0.024	0.128	0.020	0.097	0.011
	65536	0.116	0.059	0.015	0.042	0.268	0.047	0.203	0.018
	131072	0.220	0.113	0.030	0.076	0.563	0.088	0.443	0.033
	262144	0.469	0.263	0.060	0.146	1.171	0.213	0.902	0.056
	524288	1.033	0.626	0.121	0.286	2.441	0.533	1.804	0.104
	1048576	2.608	1.724	0.305	0.580	5.355	1.524	3.635	0.197
8	32768	0.031	0.011	0.004	0.016	0.077	0.008	0.056	0.012
	65536	0.070	0.025	0.014	0.031	0.150	0.019	0.112	0.018
	131072	0.140	0.057	0.028	0.055	0.296	0.047	0.223	0.026
	262144	0.265	0.110	0.056	0.100	0.593	0.087	0.464	0.043
	524288	0.556	0.253	0.109	0.193	1.288	0.214	1.003	0.070
	1048576	1.172	0.608	0.219	0.344	2.704	0.519	2.052	0.133
	2097152	2.823	1.683	0.468	0.672	5.926	1.513	4.165	0.249
	min ratio	5.503	9.525	40.588	2.485	4.984	8.659	4.649	2.537
	max ratio	10.060	19.717	79.147	3.813	6.890	17.182	5.526	4.734

As with the conventional parallel FFT, the last two rows of Table 4 show the minimum and maximum ratios of actual to expected times with $FR = 266$ Mflops/sec and $BW = 41$ Mbytes/sec.

We have also conducted experiments on a cluster of 9 Sun Ultra Enterprise 5000 servers connected by an Ethernet switch. These servers use UltraSPARC processors with a peak floating-point performance of 333 Mflops. Although each server contains 8 UltraSPARC processors, our experiments used only 1 processor per server. The maximum observed bandwidth of the Ethernet switch was approximately 1.25 Mbytes/second for all nodes.

Table 5 summarizes the results of our experiments on the Sun Ultra cluster. As on the SP2, we ran each experiment 11 times, discarding the first run, and averaged the other 10 runs. Relative standard deviations for the arithmetic portions were less than 15% in all but 4 cases, which ran as high as 30%. Because of fluctuations in traffic on the cluster, relative standard deviations in communication time were as high as 53%.

5.2. Extrapolation to other machines. Our results have shown that when we use Ethernet as an interconnect for SP2 nodes or UltraSPARC processor servers, our new algorithm outperforms a conventional FFT. While Ethernet cannot be considered an appropriate communication medium for high-performance scientific computing, high-performance machines with similar communication-to-computation-rates ratios do exist and are likely to be popular platforms in the future. (16)–(21) show that the cutoff ratio is 0.036 bytes/flop.

Let us consider a cluster of symmetric multiprocessors connected with a fast

Table 5
A comparison of the performance of the two algorithms on a cluster of servers of UltraSPARC processors. Mean times are reported in seconds. The total time is divided into three parts: T_{fftloc} spent in Netlib local FFTs, T_{comm} used for communication, and T_{arith} for other arithmetic. The last two rows give the minimum and maximum ratios of the timings reported in the table to what we would expect from (16)–(21).

		Conventional parallel FFT				New parallel approximate DFT			
Formula →			(16)	(17)	(18)		(19)	(20)	(21)
p	n	T	T_{fftloc}	T_{arith}	T_{comm}	T	T_{fftloc}	T_{arith}	T_{comm}
2	32768	2.341	0.041	0.015	2.285	1.125	0.030	0.260	0.835
	65536	5.738	0.092	0.031	5.615	2.260	0.070	0.553	1.637
	131072	12.386	0.203	0.063	12.120	5.165	0.158	1.122	3.885
	262144	23.632	0.470	0.126	23.036	9.663	0.381	2.248	7.034
	524288	48.895	0.972	0.254	47.669	21.118	0.795	4.595	15.728
4	32768	1.366	0.016	0.007	1.343	0.558	0.015	0.193	0.351
	65536	5.127	0.041	0.015	5.071	1.957	0.032	0.402	1.524
	131072	10.603	0.092	0.031	10.479	4.341	0.077	0.818	3.447
	262144	21.447	0.200	0.062	21.185	8.403	0.168	1.654	6.581
	524288	43.711	0.444	0.124	43.143	16.878	0.392	3.368	13.119
	1048576	79.827	0.951	0.248	78.628	33.573	0.833	6.860	25.881
8	32768	1.677	0.007	0.003	1.667	1.028	0.006	0.113	0.909
	65536	5.460	0.019	0.006	5.435	2.274	0.014	0.226	2.034
	131072	12.537	0.039	0.014	12.484	5.945	0.031	0.459	5.455
	262144	23.825	0.089	0.031	23.706	10.781	0.076	0.974	9.732
	524288	33.214	0.196	0.062	32.956	13.682	0.167	1.950	11.565
	1048576	61.447	0.480	0.123	60.843	26.096	0.433	3.931	21.732
	2097152	115.096	0.953	0.250	113.893	41.130	0.855	7.896	32.379
	min ratio	1.444	7.588	40.649	1.423	1.384	8.130	11.753	0.910
	max ratio	4.887	13.268	53.776	4.816	5.563	12.942	13.117	4.660

commodity network. Such a configuration might consist, for example, of several Sun Ultra Enterprise servers using 8 UltraSparc processors each, connected by an ATM switch. The peak floating-point performance of each node (if all processors are used) is about 2.5 Gflops. Measurements made by Bobby Blumofe with Sun Sparc workstations connected by a Fore ATM switch have shown that the application-to-application communication bandwidth of the switch is about 5 Mbytes per second per node in one direction (the nominal peak bandwidth of this network is 155 Mbits per second). Even if the network can support 5 Mbytes/sec in both directions, the communication-to-computation-rates ratio is only 0.002 bytes/flop.

A cluster of Digital AlphaServers connected by Digital's GIGAswitch/FDDI network yields a similar communication-to-computation ratio. The nodes can have up to 12 processors each, with peak floating-point performance of 600–874 Mflops each. Digital has measured the bandwidth of the network at about 11.9 Mbytes per second [5]. With nodes consisting of 12 600-Mflops processors each, the ratio is 0.0017 bytes/flop.

The ratio in our SP2 experiments with Ethernet is about 0.0022 bytes/flop when we use 2 nodes, 0.0010 with 4 nodes, and 0.0005 with 8 nodes. The peak performance of each node is 266 Mflops and the measured communication bandwidths are about 580, 270, and 132 Kbytes per second per node with 2, 4, and 8 nodes. In our Ultra cluster experiments, the ratio is approximately 0.002 bytes/flop with 2 nodes, 0.0009 with 4 nodes, and 0.0005 with 8 nodes. Each node has a peak performance of 333 Mflops, and the communication bandwidth is approximately 1.25 Mbytes for all nodes combined.

Since the new algorithm outperformed the conventional FFT by a large margin even on two processors on the SP2, when the ratio is 0.0022 bytes/flop, it seems safe to predict that the new algorithm would outperform a conventional FFT on the above-mentioned clusters whose ratios are even lower.

If we assume that tuning both algorithms would improve the performance of their local computations by a factor of three, say, then the new algorithm would outperform a conventional FFT even if the networks of the clusters improved by a similar factor. This assumption is supported by the fact that a tuned high-performance local FFT routine (in ESSL) is about 3.75 times faster than the publicly available package that we used (FFTPACK).

6. Conclusions.

The results of our experiments on the SP2 and the Ultra cluster have shown that when the communication-to-computation-rates ratio is low, the new algorithm outperforms a conventional parallel FFT by more than a factor of two. Quantitative performance extrapolation indicates that the new algorithm would also be faster on state-of-the-art clusters of symmetric multiprocessors.

The new algorithm is faster when communication dominates the running time of conventional parallel FFTs. When communication is so expensive, both conventional and the new algorithms are not likely to be very efficient when compared to a uniprocessor FFT. That is, their speedups are likely to be modest. There are at least two reasons to believe that the new algorithm would prove itself useful even when speedups are modest. First, in many applications the main motivation to use parallel machines is the availability of large memories, and not necessarily parallel speedups. In other words, it may be necessary to compute FFTs on multiple nodes because the data do not fit within the main memory of one node. Second, an FFT with a small or no speedup can be a part of a larger application which exhibits a good overall speedup. The application might include, for example, FFTs as well as grid computations, which require less communication per floating-point operation than the FFTs. In both cases, accelerating the parallel FFTs contributes to the performance of the application, whereas switching to a single-node FFT is not a viable option.

We believe that improvements in the new algorithms that would reduce the amount of local arithmetic it performs are possible. Such improvements would make the new algorithm faster than a conventional parallel FFT on machines with higher communication-to-computation-rates ratio than the ratios that we have indicated in this paper.

Acknowledgments. We wish to thank Jim Varah for bringing reference [21] and the papers cited therein to our attention. We gratefully acknowledge the use of IBM SP2 parallel computers at MIT and IBM Watson Research Center, and also the prototype Ultra HPC cluster of SMPs being developed at MIT in collaboration with Sun Microsystems, Inc.

REFERENCES

[1] R. C. Agarwal, F. G. Gustavson, and M. Zubair, *Exploiting functional parallelism of POWER2 to design high-performance numerical algorithms*, IBM Journal of Research and Development, 38 (1994), pp. 563–576.
[2] T. Agerwala, J. L. Martin, J. H. Mirza, D. C. Sadler, D. M. Dias, and M. Snir, *SP2 system architecture*, IBM Systems Journal, 34 (1995) pp. 152–184.
[3] H. C. Andrews and C. L. Patterson, *Singular value decomposition (SVD) image coding*, IEEE Trans. Commun., COM-24 (1976) pp. 425–432.
[4] D. H. Bailey, *FFTs in external or hierarchical memory*, J. Supercomput., 4 (1990), pp. 23–35.

[5] E. G. Benson, D. C. P. LaFrance-Linden, R. A. Warren, and S. Wiryaman, *Design of digital's parallel software environment*, Digital Technical Journal, 7 (1995), pp. 24–38.

[6] W. T. Cochrane, J. W. Cooley, J. W. Favin, D. L. Helms, R. A. Kaenel, W. W. Lang, G. C. Maling, D. E. Nelson, C. M. Rader, and P. D. Welch, *What is the fast Fourier transform?* IEEE Trans. Audio and Electroacoustics, AU-15 (1967), pp. 45–55.

[7] J. W. Cooley and J. W. Tukey, *An algorithm for the machine calculation of complex Fourier series*, Math. Comput., 19 (1965), pp. 297–301.

[8] T. H. Cormen and D. M. Nicol, *Performing Out-of-Core FFTs on Parallel Disk Systems*, Technical Report PCS-TR96-294, Dartmouth College, Hanover, NH, 1996.

[9] A. Dutt, M. Gu, and V. Rokhlin, *Fast Algorithms for Polynomial Interpolation, Integration and Differentiation*, Research Report YALEU/DCS/RR-977, Yale University, New Haven, CT, 1993.

[10] A. Dutt and V. Rokhlin, *Fast Fourier transforms for nonequispaced data*, II, Appl. Comput. Harmon. Anal., 2 (1995), pp. 85–100.

[11] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 2nd ed., The Johns Hopkins University Press, Baltimore, London, 1989.

[12] L. Greengard and W. D. Gropp, *A parallel version of the fast multipole method*, Comput. Math. Applic., 20 (1990), pp. 63–71.

[13] U. Grenander and G. Szegő, *Toeplitz Forms and Their Applications*, University of California Press, Berkeley, Los Angeles, 1958.

[14] F. A. Grünbaum, *Eigenvectors of a Toeplitz matrix: discrete version of the prolate spheroidal wave functions*, SIAM J. Alg. Discrete Methods, 2 (1981), pp. 136–141.

[15] S. K. S. Gupta, C.-H. Huang, P. Sadayappan, and R. W. Johnson, *Implementing fast Fourier transforms on distributed-memory multiprocessors using data redistributions*, Parallel Process. Lett., 4 (1994), pp. 477–488.

[16] J. Katzenelson, *Computational structure of the N-body problem*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 787–815.

[17] D. Slepian, *Prolate spheroidal wave functions, Fourier analysis, and uncertainty V: the discrete case*, Bell System Tech. J., 57 (1978), pp. 1371–1430.

[18] P. N. Swarztrauber, *Multiprocessor FFTs*, Parallel Comput., 5 (1987) pp. 197–210.

[19] P. N. Swarztrauber, *Vectorizing the FFT*, in Parallel Computations, G. Rodrigue, ed., Academic Press, New York, 1982, pp. 51–83.

[20] C. F. Van Loan, *Computational Frameworks for the Fast Fourier Transform*, SIAM, Philadelphia, 1992.

[21] J. M. Varah, *The prolate matrix*, Linear Algebra Appl., 187 (1993), pp. 269–278.

[22] S. W. White and S. Dhawan, *POWER*2: *next generation of the RISC System/6000 family*, IBM Journal of Research and Development, 38 (1994), pp. 493–502.

[23] P. McCorquodale, *Fast Multipole-Type Methods in One and Two Dimensions, with Application to Parallel Fourier Transforms*, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA, 1998.