Scientific Programming 21 (2013) 181–195
DOI 10.3233/SPR-130373
IOS Press

A framework for low-communication 1-D FFT [1]

Ping Tak Peter Tang *, Jongsoo Park, Daehyun Kim and Vladimir Petrov
Intel Corporation, Santa Clara, CA, USA

Abstract. In high-performance computing on distributed-memory systems, communication often represents a significant part of the overall execution time. The relative cost of communication will certainly continue to rise as compute-density growth follows the current technology and industry trends. Design of lower-communication alternatives to fundamental computational algorithms has become an important field of research. For distributed 1-D FFT, communication cost has hitherto remained high as all industry-standard implementations perform three all-to-all internode data exchanges (also called global transposes). These communication steps indeed dominate execution time. In this paper, we present a mathematical framework from which many single-all-to-all and easy-to-implement 1-D FFT algorithms can be derived. For large-scale problems, our implementation can be twice as fast as leading FFT libraries on state-of-the-art computer clusters. Moreover, our framework allows tradeoff between accuracy and performance, further boosting performance if reduced accuracy is acceptable.

Keywords: FFT, low communication, hybrid convolution theorem, Poisson summation formula

1. Introduction

FFT is ubiquitous in modern technology. Although there are many FFT algorithms (see [23,34] for example), they all factor the Discrete Fourier Transform (DFT) matrix algebraically into sparse factors, thereby reducing an $O(N^2)$ arithmetic cost to that of $O(N \log N)$. This arithmetic cost reduction has been instrumental in many advances in computing since the Cooley–Tukey paper appeared (see [6,8,17,21]). While arithmetic cost reduction has long been a paradigm in computer science, a new paradigm is emerging as we enter a new era where raw arithmetic speed reaches a teraflop on a single die and parallelism in the form of multicore and multinode is prevalent. The reduction of communication cost – the cost of moving data – has now become a key focus of many research activities (see [1,3,4,10,32] and references). In the context of high-performance computing in a distributed-memory environment, internode data movement is the dominating communication cost. The trend of ever increasing number of nodes only exacerbates the situation as communication bandwidth typically scales sublinearly with node counts in large-scale systems.[2]

Indeed, in high-performance distributed-data 1-D FFT, internode communication in the form of all-to-all data exchanges (also often called global transposes) can account for anywhere from 50% to over 90% of the overall running time (cf. Section 7) as all industry-standard algorithms and software execute three instances of global transposes (see [2,16,33] for example). Triple-all-to-all is a fundamental algorithmic requirement of standard "in-order" 1-D FFT– meaning that the natural order of input and output data are preserved. Relaxing the triple-all-to-all requirement is thus a worthwhile pursuit in FFT theory. The numbers of global transposes can be reduced if out-of-order data can be accommodated such as when FFT is used to compute a convolution. When in-order FFTs are needed or preferred, however, implementation optimizations are carried out in order to mitigate the performance consequences of these three global transposes [11,22,26,30]. Although there are "no-interprocessor-communication" FFT algorithms,

[1] This paper received the Best Paper Award at the SC2012 conference and is published here with permission from IEEE.

* Corresponding author. E-mail: peter.tang@intel.com.

[2] Bandwidth of the Jaguar system scales as (node count)$^{2/3}$ for example.

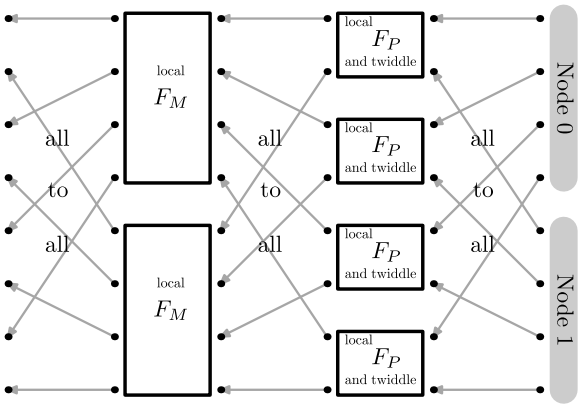
they applied to either two- and higher-dimensional FFTs [18,24] or consider a different computational model and/or require a substantially higher arithmetic cost. The works in [25,27], for example, do not count the communication cost incurred when each processor accesses the entire input data or reorders out-of-order results back into natural order. They would also require $O(N^{3/2})$ computation cost as opposed to $O(N \log N)$. Most recently, the work on sparse FFT [19,20] would likely reduce the amount of communication or computation. Nonetheless, our focus here is dense FFT for mainstream scientific computing.

To the best of our knowledge, the work presented in [14] is the only single-all-to-all, in-order, $O(N \times \log N)$ algorithm for distributed 1-D FFT. Necessarily, [14] needs to derive a new DFT factorization. Since that factorization contains a dense factor, the sophisticated fast multipole method (FMM) was employed so as to achieve an $O(N \log N)$ arithmetic economy. The scientific community appears not to have adopted the work: there is a lack of follow-up research and leading industrial software libraries such as FFTW [16] or Intel® Math Kernel Library (Intel® MKL) [31] do not implement that algorithm.

The main contributions of this paper are as follows. (1) We derive a new framework from which a large number of single-all-to-all in-order DFT factorizations can be derived. For many of our factorizations, all factors are well-approximated by sparse matrices, leading to $O(N \log N)$ algorithms. The computations needed are straightforward linear-algebra operations involving regular data-access patterns. (2) We illustrate the easy-to-implement nature of our framework by presenting the key implementation and optimization steps of one algorithm. (3) We demonstrate that our low-communication 1-D FFT outperforms industry-leading FFT libraries on cutting-edge distributed systems by as much as twofold, depending on the specific computer system and data size. Our contributions advance FFT's state of the art, in theory and in practice.

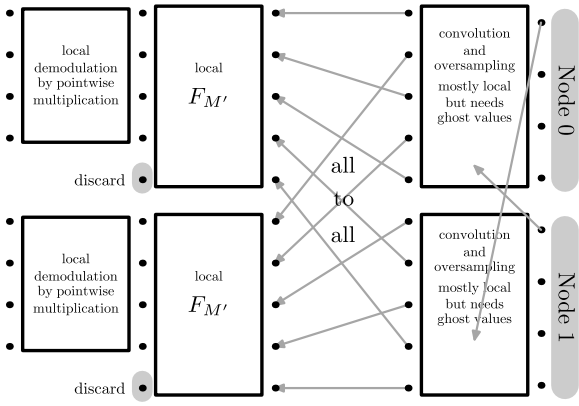
2. Overview

A traditional parallel FFT for $N = MP$ data points typically decomposes, at the highest level, the computation into two tasks. The first task computes M sets of length-P FFTs, followed by elementwise scaling commonly called twiddle-factor multiplication. The second task computes P sets of length-M FFTs:



As depicted above, this decomposition fundamentally requires three all-to-all steps if data order is to be preserved. Recursive applications of this (or variants thereof) decomposition leads to an $O(N \log N)$ arithmetic complexity, but cannot undo the triple-all-to-all requirement.

In contrast, we devised a new decomposition that requires only one all-to-all step. The new decomposition consists of two main tasks. The first task is a convolution process (or filtering in signal processing language). This task generally results in more data points $N' = M'P > N$. We call this inflation oversampling. The oversampling amount is a design parameter chosen independent of N. Our favorite choice of 25% is by no means the only option. The second task computes P sets of length-M' FFTs, followed by elementwise scaling which we call demodulation. Straightforward applications of standard FFT methods to subsequent subproblems lead to $O(N \log N)$ arithmetic complexity, but preserve the single-all-to-all property. While internode communication is also required during convolution, that amount is negligible as each node merely needs an insignificant amount of data from its next-door neighbor.



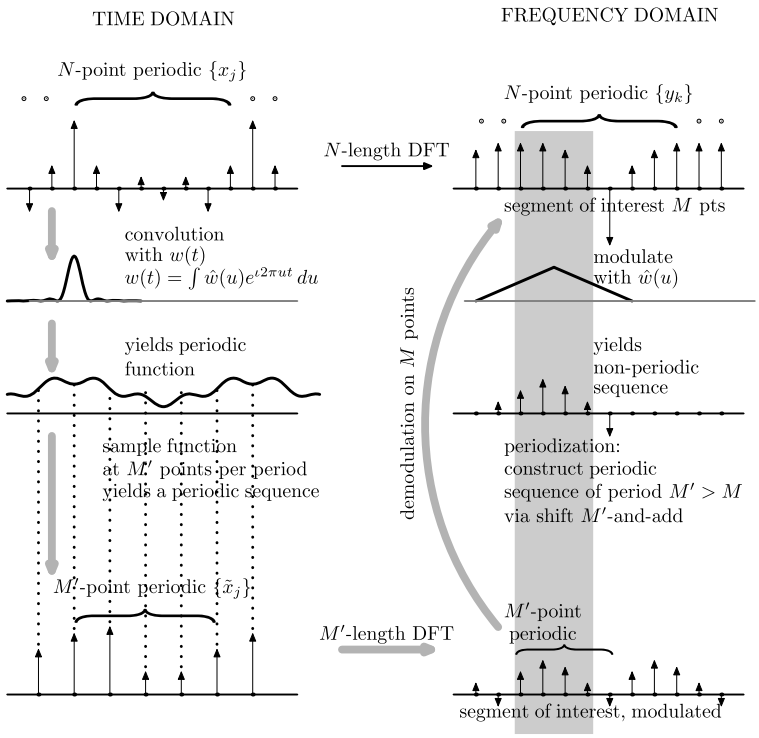


Fig. 1. An overview of pursuing a frequency segment via convolution, sampling, DFT, and demodulation. Note correspondence with the picture on the left.

Consider an N-length DFT as a transform of N time-domain (input) data points x_j, $0 \leqslant j < N$, into N frequency-domain (output) data points y_k, $0 \leqslant k < N$. Our new decomposition is based on a direct pursuit of a contiguous "segment of interest" in the frequency domain of $M < N$ data points, for example, y_k, $0 \leqslant k < M$. To accomplish this, we construct M' time-domain values, $M' > M$, from which the segment of interest can be determined. The basic approach is depicted in Fig. 1: Consider both $\{x_j\}$ and $\{y_k\}$ to be (infinite) periodic sequences, both of period N. If the sequence $\{y_k\}$ were available, a period-M' periodic sequence in the frequency domain can be constructed by (1) modulation (pointwise multiplication) with a bandpass filter to essentially eliminate the data not of interest, and (2) periodization (via shift-and-add) of the outcome of the modulation, thus creating a period-M' periodic sequence containing a modulated form of the segment of interest. Since only $\{x_j\}$, and not $\{y_k\}$, is available, we work in the time domain and carry out actions that correspond to the two previous steps. These actions produce data points $\{\tilde{x}_j\}$ from which the frequency segment of interest can be computed via a standard length-M' FFT, followed by demodulation.

The validity and practicality of Fig. 1 are established rigorously in Sections 3 and 4. Having each node compute in parallel a different segment-of-interest is the basic structure of an in-order parallel algorithm. Nevertheless, coordination and computational economization are needed to eventually derive practical single-all-to-all O($N \log N$) algorithms. Section 5 accomplishes these. The remaining of the paper presents one specific implementation, demonstrates the performance advantage our low-communication framework, and discusses future work.

3. Computing segment of interest – Theory

The standard convolution theorem states that the Fourier transform of convolution is the pointwise product of Fourier transforms. In the context of functions, Fourier transform stands for continuous Fourier transform and convolution stands for linear convolution [28]. In the context of finite vectors, Fourier transform stands for Discrete Fourier Transform and convolution stands for cyclic convolution [6]. Our framework is built on a new variant (embodied in Fig. 1): a hybrid convolution theorem that concerns the mixed operations of finite vectors with functions.

Definition 1. Let x and y be any N-length complex-valued vectors and $M > 0$ be any positive integer.

(1) *Window function*: A function $w : \mathbb{R} \to \mathbb{C}$ is a window function if it has continuous derivatives of all orders and that it decays to zero exponentially fast as $|t| \to \infty$. In particular, $\hat{w}(u) = \int w(t) \exp(-\iota 2\pi u t)\, \mathrm{d}t$, w's continuous Fourier transform, is also a window function.[3]

(2) *Convolution in time domain*: The sequence x convolved with an arbitrary window function w, $x * w$, is the function:

$$(x * w)(t) \stackrel{\text{def}}{=} \sum_{\ell=-\infty}^{\infty} w\left(t - \frac{\ell}{N}\right) x_\ell \quad \text{for all } t \in \mathbb{R},$$

where $x_\ell \stackrel{\text{def}}{=} x_{\ell \bmod N}$ whenever ℓ is outside $[0, N-1]$.

(3) *Sampling in time domain*: Let f be any complex-valued function whose domain contains $[0, 1]$. We define $\text{Samp}(f; 1/M)$, sampling, as the operator that produces the M-vector:

$$\text{Samp}\left(f; \frac{1}{M}\right) \stackrel{\text{def}}{=} \left[f(0), f\left(\frac{1}{M}\right), \ldots, f\left(1 - \frac{1}{M}\right)\right]^T.$$

(4) *Modulation in frequency domain*: The sequence y modulated by an arbitrary window function $\hat{w}$, $y \cdot \hat{w}$, is an infinite sequence z:

$$y \cdot \hat{w} \stackrel{\text{def}}{=} z, \quad z_k = y_k \cdot \hat{w}(k), k = 0, \pm 1, \pm 2, \ldots.$$

Again, $y_k \stackrel{\text{def}}{=} y_{k \bmod N}$ whenever k is outside $[0, N-1]$.

(5) *Periodization in frequency domain*: Given any absolutely summable infinite sequence $\{z_k\}$ (that is, $\sum |z_k| < \infty$), the periodization operator $\text{Peri}(z; M)$ produces the M-length vector z' where

$$\text{Peri}(z; M) \stackrel{\text{def}}{=} z', \quad z'_k = \sum_{j=-\infty}^{\infty} z_{k+jM}$$

for $k = 0, 1, \ldots, M-1$. Note that the last formula can be extended for all values of k, and the resulting infinite sequence of z'_k would indeed be periodic.

[3] The function $\exp(-\sigma t^2)$ for $\sigma > 0$ is an example.

For the rest of the paper, let $x = [x_0, x_1, \ldots, x_{N-1}]^T$ be the time-domain input vector of complex elements and $y = F_N x$ be x's DFT in the frequency domain:

$$y_k = \sum_{j=0}^{N-1} x_j \exp(-\iota 2\pi jk/N), \quad k = 0, 1, \ldots, N-1.$$

The following theorem is the foundation to our new framework for computing y. The proof is given in the Appendix.

Theorem 1 (Hybrid convolution). *Let w be an arbitrary window function whose continuous Fourier transform we denote by $\hat{w}$. Then for any integer $M > 0$,*

$$F_M \frac{1}{M} \text{Samp}(x * w; 1/M) = \text{Peri}(y \cdot \hat{w}; M).$$

Let us aim at obtaining $y^{(0)} = [y_0, y_1, \ldots, y_{M-1}]^T$ for some integer $M < N$. The idea is to seek a window function $\hat{w}(u)$ and an integer $M' \geqslant M$ such that $y^{(0)}$ can be easily derived (exactly or accurately) from $\text{Peri}(y \cdot \hat{w}; M')$, which in turn can be computed as $F_{M'} \frac{1}{M'} \text{Samp}(x * w; 1/M')$.

Let $\hat{w}$ be a window function such that $|\hat{w}(u)| > 0$ on $[0, M-1]$, and that $|\hat{w}(u)|$ is very small outside of $(-\delta - 1, M')$ for some integer $M' = M + \delta \geqslant M$. Consider the M'-vector

$$\tilde{y} = \text{Peri}\big(y \cdot \hat{w}; M'\big).$$

Then for $k = 0, 1, \ldots, M-1$,

$$\tilde{y}_k = y_k \hat{w}(k) + \sum_{|\ell| > 0} y_{k+\ell M'} \hat{w}\big(k + \ell M'\big) \approx y_k \hat{w}(k).$$

Thus, $y^{(0)} \approx \hat{W}^{-1} \mathrm{P}_{\text{roj}}^{M', M} \tilde{y}$, where $\hat{W}$ is the diagonal matrix

$$\hat{W} = \text{diag}\big(\hat{w}(0), \hat{w}(1), \ldots, \hat{w}(M-1)\big),$$

and $\mathrm{P}_{\mathrm{roj}}^{M',M}$ is the projection operator that takes an M'-vector and returns the top M elements. Using Theorem 1,

$$\tilde{y} = \mathrm{Peri}\big(y \cdot \hat{w}; M'\big) = F_{M'}\tilde{x}, \tilde{x} = \frac{1}{M'}\,\mathrm{Samp}\left(x * w; \frac{1}{M'}\right),$$

where $w(t) = \int \hat{w}(u)\exp(\iota 2\pi u t)\,\mathrm{d}u$, the inverse Fourier transform of $\hat{w}$. As the process of producing $\tilde{x}$ is clearly linear in x, $\tilde{x}$ must be of the form $\tilde{x} = C_0 x$ where C_0 is an M'-by-N matrix. We arrive at following:

$$y^{(0)} \approx \hat{W}^{-1}\mathrm{P}_{\mathrm{roj}}^{M',M}F_{M'}C_0 x. \tag{1}$$

Note how Eq. (1) corresponds to Fig. 1: C_0 corresponds to the actions in the time domain, and $\hat{W}^{-1}\mathrm{P}_{\mathrm{roj}}^{M',M}$ corresponds to demodulation only on M points. Equation (1) has two issues. First, $y^{(0)}$ is not obtained exactly because $\tilde{y}_k$ is not exactly $y_k \cdot \hat{w}(k)$ for k in $[0, M-1]$: $\tilde{y}_k$ may be "contaminated" with y outside of $[0, M-1]$. This kind of contamination is rightfully called aliasing as values from "higher frequency domains" (outside of $[0, M-1]$) are being brought inside.[4] Second, the matrix C_0 is generally dense, and the associated arithmetic cost in computing $y^{(0)}$ would be unacceptably high. The next section shows that, by suitable choices of M' and $\hat{w}$, C_0 can be well-approximated by various sparse matrices and all errors, including aliasing, can be made acceptably small.

4. Computing segment of interest – Practice

This section develops a computation-efficient variant of Eq. (1) and quantify the approximation accuracy. Let us start with some intuitive discussions. The entries of the matrix C_0 are closely related to the values of the time-domain window function $w(t)$. If $w(t)$ decays to zero so rapidly that it is negligible except in a small region, it is likely that C_0 can be approximated well by a sparse matrix. In general then, w's counter part in the frequency domain, $\hat{w}(u)$, needs to be smooth and slow varying. On the other hand, $|\hat{w}(u)|$ needs to be small outside of $(-\delta - 1, M')$ in order to attenuate the aliasing effect pointed out earlier. If we aim for both properties, but set $M' = M$, then $|\hat{w}(M-1)|$ would inevitably be small. This is undesirable: Demodulation requires division by $|\hat{w}(k)|$, $0 \leqslant k \leqslant M-1$, and division by small numbers tend to magnify errors of all kinds. Consequently, we have to set $M' > M$, and call this action oversampling. This term is appropriate because sampling in the time domain must now be carried out at a rate $1/M'$, instead of $1/M$ if we were able to limit ourselves strictly to the frequency band $[0, M-1]$. We now examine the specifics.

Let $N = MP$ and that an oversampling length $M' = M(1+\beta) = M + \delta > M$ has been chosen. (A choice of $\beta = 1/4$ is rather typical.) We begin the construction of $\hat{w}$ with choosing a reference window function $\hat{H}(u)$ such that (a) $|\hat{H}(u)| > 0$ on $[-1/2, 1/2]$, (b)

$$\kappa \stackrel{\mathrm{def}}{=} \max_{u,v\in[-1/2,1/2]}\left|\frac{\hat{H}(u)}{\hat{H}(v)}\right|$$

is moderate (for example, less than 10^3), and (c)

$$\varepsilon^{(\mathrm{alias})} \stackrel{\mathrm{def}}{=} \frac{\int_{|u|\geqslant 1/2+\beta}|\hat{H}(u)|\,\mathrm{d}u}{\int_{-1/2}^{1/2}|\hat{H}(u)|\,\mathrm{d}u}$$

is small (for example, floating-point rounding error). To simplify our presentation, all numerical results presented in this paper are based on the two-parameter, (τ, σ), reference window function which is the convolution (smoothing) of a rectangular function (perfect bandpass filter) with a Gaussian function:

$$\hat{H}(u) = \frac{1}{\tau}\int_{-\tau/2}^{\tau/2}\exp\big(-\sigma(u-t)^2\big)\,\mathrm{d}t. \tag{2}$$

Let $H(t)$ denote the inverse Fourier transform of $\hat{H}(u)$.[5] For a given threshold $\varepsilon^{(\mathrm{trunc})}$ (for example, around floating-point rounding error), determine a corresponding integer B such that

$$\int_{|t|\geqslant B/2}\big|H(t)\big|\,\mathrm{d}t \leqslant \varepsilon^{(\mathrm{trunc})}\int_{-\infty}^{\infty}\big|H(t)\big|\,\mathrm{d}t.$$

[4] Along the same line, if $\hat{w}$ is exactly zero outside of $(-\delta-1, M')$, then "$\approx$" can be replaced by simple equality. See [7] for an example of a compact-support window function.

[5] Note that both $\hat{H}$ and H have simple closed-form representations. $\hat{H}$ in terms of differences of two erfc functions and H in terms of product of a sinc with a Gaussian.

Once a reference window function is chosen, the problem-size-specific window $\hat{w}$ is configured via simple translation, dilation, and phase shift:

$$\hat{w}(u) \stackrel{\text{def}}{=} \exp(\iota\pi BPu/N)\hat{H}\big((u - M/2)/M\big).$$

Thus $|\hat{w}(u)|$ on $[0, M] \approx [0, M-1]$ corresponds to $|\hat{H}(u)|$ on $[-1/2, 1/2]$, and $|\hat{w}(u)|$ on $(-\infty, -\delta - 1] \cup [M', \infty)$ corresponds to $|\hat{H}(u)|$ on $|u| \geqslant 1/2 + \beta$. Moreover,

$$\sum_{\ell=0}^{BP-1} \left| w\left(-\frac{\ell}{N}\right)\right| \approx \sum_{\ell=-\infty}^{\infty} \left| w\left(\frac{\ell}{N}\right)\right| \tag{3}$$

to within $1 - \varepsilon^{(\text{trunc})}$ in relative accuracy.

Turning to the matrix C_0, recall that $C_0 x = (1/M')\,\text{Samp}(x * w; 1/M')$. Let us denote C_0's entries by c_{jk}, with indices starting at 0. From Definition 1,

$$c_{jk} = \frac{1}{M'} \sum_{\ell=-\infty}^{\infty} w\left(\left(\frac{j}{M'} - \frac{k}{N}\right) - \ell\right). \tag{4}$$

In particular, if M' divides N, that is, $\frac{1}{M'} = \frac{L}{N}$ for some integer L, Eq. (4) shows that $c_{jk} = c_{j+1k'}$ where $k' = (k + L) \bmod N$. In other words, the matrix C_0 is completely specified by its first row, with each subsequent row being its previous row circular-shifted by L positions to the right. Furthermore, using Eq. (3), that first row is approximated by

$$\begin{aligned}&\frac{1}{M'}\left[w(0), w\left(-\frac{1}{N}\right), \ldots, w\left(\frac{1-BP}{N}\right), \ldots\right]\\ &\quad\approx \frac{1}{M'}\left[w(0), w\left(-\frac{1}{N}\right), \ldots, w\left(\frac{1-BP}{N}\right),\right.\\ &\qquad\left. 0, 0, \ldots, 0\right].\end{aligned}$$

If M' does not divide N, but $\frac{1}{M'} = \frac{L}{N}\frac{\nu}{\mu}$ where ν/μ is a fraction in irreducible form, then similar derivation from Eq. (4) shows that C_0 is completely determined by its first μ rows. The $(j + \mu)$th row is the jth row circular-right-shifted by νL positions. Section 6 will exploit this property.

In general, row-j of C_0 can be approximated as

$$\text{row-}j \text{ of } C_0 \approx \big[\mathbf{w}_{j,0}^T, \mathbf{w}_{j,1}^T, \ldots, \mathbf{w}_{j,M}^T\big],$$

where each $\mathbf{w}_{j,k}^T$ is a P-vector, and that all but a stretch of B of these $\mathbf{w}_{j,k}^T$ are zeros. Denoting the approximation matrix by $C_0^{(\text{trunc})}$ yields the computation-efficient variant of Eq. (1) that we seek.

$$y^{(0)} \approx \hat{W}^{-1}\mathrm{P}_{\text{roj}}^{M',M} F_{M'} C_0^{(\text{trunc})} x. \tag{5}$$

The computation exhibited by this factorization involves a regular matrix–vector product and a standard FFT. Exploiting the sparsity of $C_0^{(\text{trunc})}$, the operation count in computing $y^{(0)}$ via Eq. (5) is easily seen to be

$$\mathrm{O}\big(M' \log M'\big) + \mathrm{O}\big(N'B\big),$$

where $N' = N(1 + \beta)$. The nonzero block in the first row zero of $C_0^{(\text{trunc})}$ is shifted right in a regular pattern as one traverses down the rows. Hence, $C_0^{(\text{trunc})} x$ can be computed with one pass over the vector x.

A straightforward analysis shows that the error introduced by the approximate nature of this factorization is of the form

$$\begin{aligned}&\frac{\|\text{computed } y^{(0)} - y^{(0)}\|}{\|y\|}\\ &\quad= \mathrm{O}\big(\kappa\big(\varepsilon^{(\text{fft})} + \varepsilon^{(\text{alias})} + \varepsilon^{(\text{trunc})}\big)\big).\end{aligned}$$

This characterization explains the effects of aliasing, truncation, and the computational error $\varepsilon^{(\text{fft})}$ of the underlying FFT software. The κ term plays the familiar role of a condition number. Finally, the κ and $\varepsilon^{(\text{alias})}$ terms also underline the need for oversampling. We omit the error-bound derivation, but numerical results shown later will serve as supporting evidences.

5. Low-communication FFT framework

This section achieves the main goal of the paper: a family of single-all-to-all, in-order, $\mathrm{O}(N \log N)$ DFT factorizations. The previous section shows how a first segment of y, $y^{(0)} = [y_0, y_1, \ldots, y_{M-1}]^T$, can be obtained. In essence, the low-communication FFT algorithm obtains the entire y by efficiently gathering all the other segments, each computed using the same method.

Let $N = MP$ and denote the sth segment by

$$y^{(s)} = [y_{sM}, y_{sM+1}, \ldots, y_{(s+1)M-1}]^T$$

for $s = 0, 1, \ldots, P - 1$. It is well known that DFT of a phase-shifted x results in a translated y. In particular, the first segment of $F_N(\Phi_s x)$ is the sth segment, $y^{(s)}$,

of y if

$$\Phi_s = \mathrm{diag}\big(1, \omega^s, \omega^{2s}, \ldots, \omega^{(N-1)s}\big),$$
$$\omega = \mathrm{e}^{-\iota 2\pi M/N} = \mathrm{e}^{-\iota 2\pi/P}.$$

Hence the computation in the previous section will yield $y^{(s)}$ when applied verbatim, except on the input vector $\Phi_s x$. Note that $\omega^P = 1$ and the diagonal entries in Φ_s repeat themselves after P elements. In other words, Φ_s is a block diagonal matrix with M diagonal blocks, each being the diagonal matrix $\mathrm{diag}(\boldsymbol{\omega}_s)$ where $\boldsymbol{\omega}_s = [1, \omega^s, \omega^{2s}, \ldots, \omega^{(P-1)s}]$. In the succinct Kronecker product notation,[6] $\Phi_s = I_M \otimes \mathrm{diag}(\boldsymbol{\omega}_s)$, where I_M is the dimension-M identity matrix.

We have thus derived

$$\begin{aligned} y^{(s)} &\approx \hat{W}^{-1} \mathrm{P}_{\mathrm{roj}}^{M',M} F_{M'} C_0^{(\mathrm{trunc})} \\ &\quad \times \big[I_M \otimes \mathrm{diag}(\boldsymbol{\omega}_s)\big] x \\ &= \hat{W}^{-1} \mathrm{P}_{\mathrm{roj}}^{M',M} F_{M'} C_s^{(\mathrm{trunc})} x, \end{aligned}$$

where $C_s^{(\mathrm{trunc})}$ denotes $C_0^{(\mathrm{trunc})}[I_M \otimes \mathrm{diag}(\boldsymbol{\omega}_s)]$, which is an M'-by-N matrix. Stacking all the $y^{(s)}$ together yields

$$y \approx I_P \otimes \big(\hat{W}^{-1} \mathrm{P}_{\mathrm{roj}}^{M',M} F_{M'}\big) C^{(\mathrm{trunc})} x,$$

where

$$C^{(\mathrm{trunc})} \stackrel{\mathrm{def}}{=} \begin{bmatrix} C_0^{(\mathrm{trunc})} \\ C_1^{(\mathrm{trunc})} \\ \vdots \\ C_{P-1}^{(\mathrm{trunc})} \end{bmatrix}.$$

Applying the $C^{(\mathrm{trunc})}$ matrix row-by-row as-is, however, is problematic: It has high communication cost, as each $C_s^{(\mathrm{trunc})}$ block needs the entire input x, and it has high arithmetic cost, as the total operation count is $\mathrm{O}(N'PB)$, which increases linearly with P. Fortunately, a simple rearrangement solves both problems.

Let $\mathrm{P}_{\mathrm{erm}}^{\ell,n}$ where ℓ divides n denotes the stride-ℓ permutation: $w = \mathrm{P}_{\mathrm{erm}}^{\ell,n} v \Leftrightarrow v_{j+k\ell} = w_{k+j(n/\ell)}$, for all $0 \leqslant j < \ell$ and $0 \leqslant k < n/\ell$. Now consider $\mathrm{P}_{\mathrm{erm}}^{P,N'}$ and $\mathrm{P}_{\mathrm{erm}}^{M',N'}$, which are inverses of each other. Thus

$$C^{(\mathrm{trunc})} = \mathrm{P}_{\mathrm{erm}}^{P,N'} \mathrm{P}_{\mathrm{erm}}^{M',N'} C^{(\mathrm{trunc})} = \mathrm{P}_{\mathrm{erm}}^{P,N'} A,$$

where

$$A = \mathrm{P}_{\mathrm{erm}}^{M',N} \begin{bmatrix} C_0^{(\mathrm{trunc})} \\ C_1^{(\mathrm{trunc})} \\ \vdots \\ C_{P-1}^{(\mathrm{trunc})} \end{bmatrix} = \begin{bmatrix} A_0 \\ A_1 \\ \vdots \\ A_{M'-1} \end{bmatrix}.$$

The jth block, A_j, of A is the P-by-N matrix consisting of gathering all the jth rows of the $C_s^{(\mathrm{trunc})}$ matrices. However, row-j of matrix $C_s^{(\mathrm{trunc})}$ is

$$\begin{aligned} &\big[\, \mathbf{w}_{j,0}^T \quad \mathbf{w}_{j,1}^T \quad \cdots \quad \mathbf{w}_{j,M}^T \,\big] \big(I_M \otimes \mathrm{diag}(\boldsymbol{\omega}_s)\big) \\ &\quad = \boldsymbol{\omega}_s \big[\, \mathrm{diag}(\mathbf{w}_{j,0}^T) \quad \mathrm{diag}(\mathbf{w}_{j,1}^T) \\ &\qquad \cdots \quad \mathrm{diag}(\mathbf{w}_{j,M}^T) \,\big]. \end{aligned}$$

Stacking these sth rows, $s = 0, 1, \ldots, P-1$ (where j is fixed) gives A_j, which is

$$\begin{aligned} &F_P \big[\, \mathrm{diag}(\mathbf{w}_{j,0}^T) \quad \mathrm{diag}(\mathbf{w}_{j,1}^T) \quad \cdots \quad \mathrm{diag}(\mathbf{w}_{j,M}^T) \,\big] \\ &\quad = F_P W_j, \end{aligned}$$

because $\boldsymbol{\omega}_s$ is the sth row of the DFT matrix F_P. Hence $A = (I_{M'} \otimes F_P) W$, where W is the stacking of W_j for $j = 0$ up to $M' - 1$. The matrices W_j are sparse: They are block diagonal matrices each with only B nonzero blocks. Similar to $C_0^{(\mathrm{trunc})}$, the blocks of contiguous nonzeros elements of W are shifted right in a regular pattern as one traverses down blocks of rows (see Fig. 4).

We have achieved our main goal:

$$\begin{aligned} y &\approx I_P \otimes \big(\hat{W}^{-1} \mathrm{P}_{\mathrm{roj}}^{M',M} F_{M'}\big) \\ &\quad \times \mathrm{P}_{\mathrm{erm}}^{P,N'} (I_{M'} \otimes F_P) W x. \end{aligned} \tag{6}$$

This is a family of factorization: There are many choices for window functions, oversampling rate, and accuracy characteristics $\varepsilon^{(\mathrm{alias})}$ and $\varepsilon^{(\mathrm{trunc})}$. The only global all-to-all pattern in this factorization is $\mathrm{P}_{\mathrm{erm}}^{P,N'}$. The arithmetic operation count is readily seen as $\mathrm{O}(N' \log M') + \mathrm{O}(N' \log P) + \mathrm{O}(N'B)$, which is

$$\mathrm{O}\big(N' \log N'\big) + \mathrm{O}\big(N'B\big).$$

[6] These notations, see [23] for example, have proved effective in aiding FFT implementations on modern architectures. For example, $I_p \otimes M_{\ell \times m}$ for an $\ell \times m$ matrix M stands for perfect p-parallel matrix operations each involving a local m-vector.

In summary, compared to a traditional factorization, the new factorization requires only one global all-to-all communication between N' data points as opposed to three such steps, each involving N data points. On the other hand, the new factorization costs more arithmetic operations, by a factor of $1 + \beta$ over the standard method, plus another additional $\mathrm{O}(N'B)$. As the next two sections demonstrate, this tradeoff is unequivocally beneficial in high-performance large-data computations.

6. Implementation

To implement is to translate the formula in Eq. (5),

$$I_P \otimes \left(\hat{W}^{-1}\mathrm{P}_{\mathrm{roj}}^{M',M} F_{M'}\right)\mathrm{P}_{\mathrm{erm}}^{P,N'}(I_{M'} \otimes F_P)Wx,$$

into computer code. We point out two important aspects of our coding.

(a) *Parallelism*: A Kronecker product of the form $I_\ell \otimes A$ expresses parallelism naturally. It says that ℓ copies of the matrix A are to be applied independently on ℓ contiguous segments of stride-one data. Furthermore, because $I_{\ell m} \otimes A = I_\ell \otimes (I_m \otimes A)$, multiple levels of parallelism can be realized readily. Figure 2 gives a scenario on how parallelism can be harnessed. The figure depicts the case where the number of segments P equals the number of processor nodes. In general, P can be a multiple of number of processor nodes, increasing the granularity of parallelism. We employ a hybrid parallel programming model consisting of MPI processes and OpenMP threads, configured appropriately to exploit the specific hardware resources of nodes, sockets, cores and (hardware) threads.
By its very nature, permutation is a parallel operation as each source data can be sent to its distinct destination independently. Nevertheless, a practical implementation has to realize this parallelism without needlessly inflating the number of threads or messages used. The Kronecker product notation can be used once again to describe $\mathrm{P}_{\mathrm{erm}}^{P,N'}$ in a way that maps well to actual implementation. Figure 3 illustrates the general idea with the example of $P = 2$, $N' = 12$. In essence, node-local permutations gather data destined for the same foreign processor node into contiguous memory locations, thus decreasing the number of messages required.

(b) *Compute efficiency*: Referring to Fig. 2, the two major computational tasks are FFTs (the groups of F_P and $F_{M'}$) and convolution (the matrix–vector product Wx). The arithmetic operations of the first task is comparable to that of a standard FFT $F_N x$. One can view that the arithmetic operations incurred in Wx is the extra price our algorithm pays in order to reduce communication. While that arithmetic operations count is nontrivial, we manage to complete them effi-

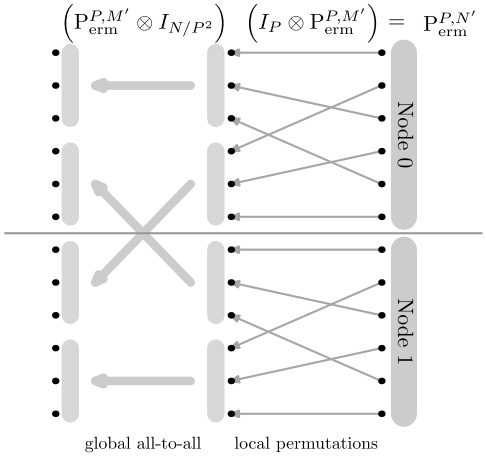


Fig. 3. The global data reordering involves local permutation followed by an all-to-all communication. The latter can be implemented via the MPI all-to-all primitive, or by other techniques such as non-blocking send-receive.

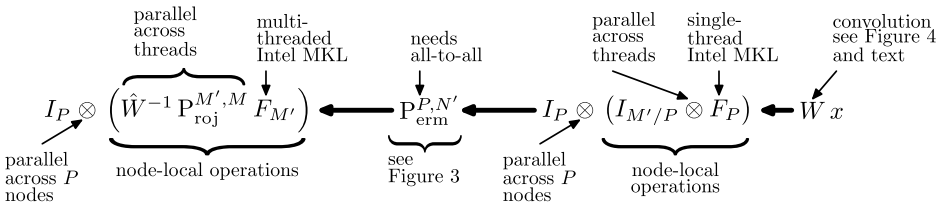


Fig. 2. An implementation scenario at-a-glance. Number of segments equals number of nodes. Hybrid parallelism is employed: MPI process mapped to internode parallelism, and OpenMP threads maps to multicore. Intel MKL FFTs, single threaded as well as multithreaded, are used as building blocks.

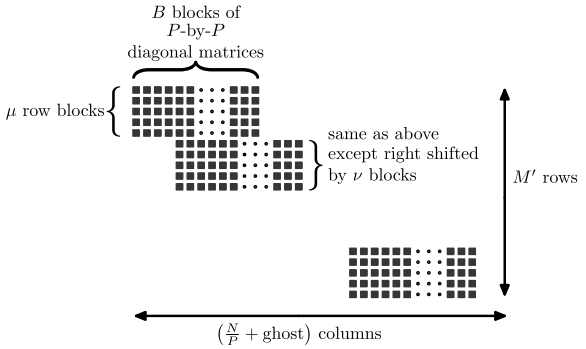


Fig. 4. The convolution matrix on a node. The dimension is M'-by-$(M+(B-\nu)P)$. The μ and ν are related to the oversampling factor $\mu/\nu = 1 + \beta$. For $\beta = 1/4$, $\mu = 5$ and $\nu = 4$. Input data is the $M = N/P$ elements local to the node, and $(B - \nu)P$ elements from its adjacent node. This amount is typically less than 0.01% of M. The matrix is structured as shown. Each P-by-P block is a diagonal matrix. The entire matrix has μPB distinct elements, a fact that facilitates various standard locality optimization strategies.

ciently through a number of standard optimization techniques we now describe.

W times x is carried out by each node computing in parallel by a local matrix–vector product. The local matrix is highly structured, depicted in Fig. 4. The computation loop on a node consists of $M'/(\mu P)$ chunks of operations, each of which is μP independent length-B inner products. The pseudo code of a straightforward implementation is:

```
loop_a over M'/(μP) chunks
// same matrix elements
   loop_b over μ rows
      // same input range
      loop_c over B blocks
         // inner product
         loop_d over P elements in
         each block
```

We apply a series of standard optimizations [35]:

- *Loop unrolling and interchange*: in order to keep partial sums of inner products in registers while exploiting SIMD parallelism, we unroll `loop_d` by the width of the native SIMD instructions and interchange the order of `loop_c` and `loop_d`.
- *Unroll-and-jam*: for the locality of accessing matrix elements and input (both in cache and register levels), we unroll `loop_a` twice and `loop_b` μ times, and we jam them into `loop_d`.
- *Loop fusion and array contraction*: we fuse the convolution computation with the succeeding loops for FFT and local permutation. We reuse a small array across iterations for passing data between the convolution, FFT, and local permutation steps.

7. Evaluation

We compare the performance of our low-communication FFT (hereafter called SOI) with several industry-standard FFT libraries such as Intel MKL [31], FFTW [16] and FFTE [33]. Comparisons are done on two different hardware platforms whenever possible.

7.1. Setup

Most of the experiments are run on a cluster named Endeavor which is part of the computing infrastructure within Intel Corporation. We are able to run on Endeavor SOI as well as all the other software libraries mentioned before. In addition, Professor Eric Polizzi of the University of Massachusetts compared SOI with Intel MKL on a system called the Gordon Cluster. The two systems have similar compute nodes but different interconnect fabrics. While both clusters use InfiniBand as their interconnect backbone, Endeavor uses a fat-tree topology whereas Gordon uses that of a 3-D torus. The former offers an aggregated peak bandwidth that scales linearly up to 32 nodes while the bandwidth scalability of the latter becomes more challenged at 32 nodes and beyond. Table 1 tabulates the configuration of the two systems.

Performance tests are done using the weak scaling model. We use 2^{28} double-precision complex data points per node. Performance is reported in GFLOPS, which is $5N \log N$ divided by execution time. The numbers reported correspond to the maximum performance observed among ten or more runs per problem-size/number-of-nodes/software choice. To be conservative, we make at least three times as many runs on non-SOI software per problem-size/number-of-nodes than on SOI.

7.2. Performance – Full accuracy

The signal-to-noise (SNR) ratio of our double-precision SOI is around 290 dB, which is 20 dB (one digit) lower than standard FFTs (Intel MKL, FFTW, etc.). This slight decrease of accuracy is expected as

Table 1
System configuration

Compute node	
Sock. × core × SMT	2 × 8 × 2
SIMD width	8 (single precision), 4 (double precision)
Clock (GHz)	2.60
Micro-architecture	Intel® Xeon® E5-2670
DP GFLOPS	330
L1/L2/L3 Cache (KB)	64/256/20,480, L1/L2:core, L3:socket
DRAM (GB)	64
Interconnect	
Fabric	QDR InfiniBand 4×
Topology	Endeavor: Two-level 14-ary fat tree Gordon: 4-ary 3-D torus, concentration factor 16
Libraries	
SOI	8 segment/process, $\beta = 1/4$, SNR = 290 dB
MKL	v10.3, 2 processes per node MPI + OpenMP
FFTE	used in HPCC 1.4.1
FFTW	v3.3, MPI + OpenMP and `FFTW_MEASURE`

SOI uses an underlying FFT as building block and incurs a small amount of additional error due to the condition number κ as well as an increased number of floating-point operations. The B value is set to 72 for a pair of (τ, σ) parameters obtained in the fashion outlined in Section 4.

All comparisons in this section are done with SOI set to this accuracy level. Figure 5 presents the set of comparisons on the Endeavor cluster with fat-tree InfiniBand interconnect. SOI's advantage is apparent despite the relatively high scalability of the interconnect fabric. On the Gordon cluster, comparison is made between SOI and Intel MKL only. See Fig. 6. Note the additional performance gain over Endeavor from 32 nodes onwards. This phenomenon is consistent with the narrower bandwidth due to a 3-D torus topology.

7.3. Performance – Accuracy tradeoff

Our framework has a unique capability of trading accuracy for speed. By allowing the condition number κ (of the W matrix) to gradually increase, faster-decay convolution window functions w can be obtained, which in turn leads to a smaller B value. Figure 7 illustrates this capability using 64 nodes on Gordon. As a reference, Intel MKL typically offers a SNR

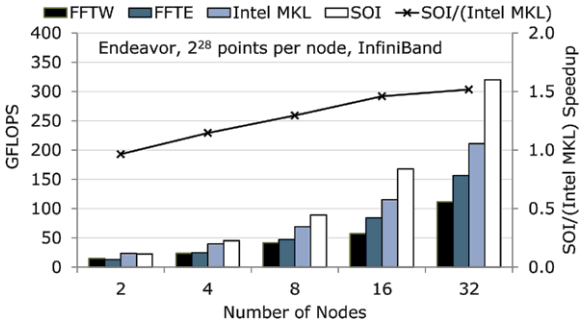


Fig. 5. The bar graph shows the best performance among multiple runs in GFLOPS calibrated by the left y-axis. The line graph, calibrated by the right y-axis, shows the speed up of SOI over Intel MKL, the fastest among the non-SOI algorithms. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-130373.)

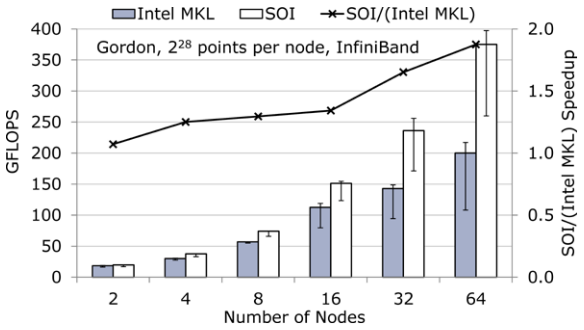


Fig. 6. Performance comparison between Intel MKL and SOI carried out by Professor Eric Polizzi on the XSEDE Gordon supported by NSF grant number OCI-1053575. The bar graph shows the best performance among multiple runs in GFLOPS calibrated by the left y-axis. Due to a large range of reported performance, we include in the figure the 90% confidence interval based on normal distribution. The line graph, calibrated by the right y-axis, shows the speed up of SOI over Intel MKL. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-130373.)

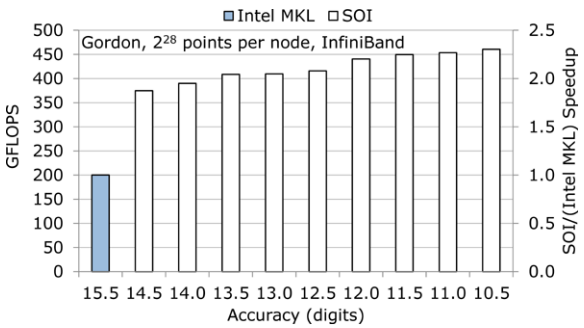


Fig. 7. Performance boost by decreasing accuracy requirement. Results are obtained on 64-node Gordon. The bar shows performance and speed up, calibrated by the left and right y-axes, respectively. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-130373.)

value of 310 dB, about 15.5 digits. Full-accuracy SOI is about 14.5 digits. Figure 7 shows the additional performance gain as the accuracy requirement is relaxed. We believe applications can exploit this feature offered by SOI. It is plausible that 10-digit-accurate FFT suffices for many applications. Furthermore, in the context of iterative algorithms where FFT is computed in an inner loop, full accuracy is typically unnecessary until very late in the iterative process. Note also that at an accuracy level of 10 digits, SOI outperforms Intel MKL by more than twofold–which is the likely the best speedup achievable by a 6-digit-accurate single-precision Intel MKL.

7.4. Performance analysis and projection

Two key factors contribute to SOI's speed advantage. First and foremost, SOI requires just one step of all-to-all communication whereas its peers require three. In the situation where communication is much slower than computation, one would expect SOI at an oversampling rate of $1+\beta$ to outperform non-SOI implementation by about a factor of $3/(1+\beta)$. We tested this reasoning by comparing SOI with Intel MKL on Endeavor again, but using a 10 Gigabit Ethernet interconnect. Figure 8 matches our expectation practically perfectly. The speed up factors lie in the interval $[2.3, 2.4]$, near the theoretical value of $3/(1+\beta) = 3/1.25 = 2.4$.

Second, convolution can be implemented with relatively high computation efficiency. As pointed out previously in Section 5, convolution represents "extra" arithmetic cost peculiar to SOI; and this amount is non-trivial. In our test of 2^{28} data points per node and running SOI to full accuracy on 32 nodes, arithmetic operations needed in convolution is almost fourfold that of a regular FFT. In other words, SOI is about fivefold as expensive in terms of arithmetic operations count. Fortunately, while FFT's computational efficiency is notoriously low – often hovering around 10% of a machine's peak performance – Section 6 shows that convolution is quite amenable to programming optimization. We profiled our implementation and found that convolution computation reaches about 40% of the processor's peak performance. That is consistent with our other profiling data that the total convolution time in SOI is about the same as that of the FFT computation time within it. Thus, our full-accuracy SOI implementation takes about twice, not five times, as much computation time than a regular FFT. As we have seen, this penalty is more than offset by our savings in communication time.

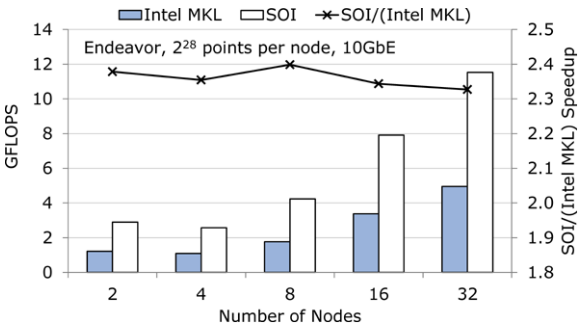


Fig. 8. This figure illustrates the low-communication advantage of SOI over those that require three all-to-all steps. On Endeavor with a 10 Gigabit Ethernet interconnect, one would expect SOI, 25% oversampling, to have a 2.4× speed advantage. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-130373.)

We can model the execution time of SOI and of Intel MKL, the latter being a representative of non-SOI algorithm. Consider the problem size of $N = 2^{28}n$, where n is the number of processor nodes. Furthermore, let the nodes be connected by a k-ary 3-D torus with a concentration factor of 16 (16 compute nodes connected to one switch), giving the relationship of $n = 16k^3$. As functions of n, let $T_{\text{fft}}(n)$ and $T_{\text{conv}}(n)$ denote the execution time of node-local FFT and convolution computations, and $T_{\text{mpi}}(n)$ be the latency of one all-to-all exchange of N data points. We model each of these functions as follows.

On the one hand, $T_{\text{fft}}(1)$ for Intel MKL can be measured, and on the other hand,

$$T_{\text{fft}}(1) = \mathrm{O}\big(2^{28}\log\big(2^{28}\big)\big) = \alpha\log\big(2^{28}\big).$$

We therefore obtain α through the measured execution time. At n nodes, we model $T_{\text{fft}}(n)$ by $\mathrm{O}(2^{28}n \times \log(2^{28}n))/n$ and subsequently by

$$T_{\text{fft}}(n) \approx \alpha\big(\log\big(2^{28}\big) + \log(n)\big).$$

Next, because node-local convolution requires $\mathrm{O}(MB)$ operations where M is the number of data points per node, $T_{\text{conv}}(n)$ remains roughly constant regardless of n in our weak scaling scenario. Let T_{conv} be the time we measured from running SOI. We will model $T_{\text{conv}}(n)$ by cT_{conv} where c lies in $[0.75, 1.25]$, reflecting possible deviation from T_{conv} one way or the other. Finally, we consider $T_{\text{mpi}}(n)$. Using the Gordon cluster as a reference, we model switch-to-switch (global) channels with three 4× QDR InfiniBand links and

node-to-switch (local) channels with one 4× QDR InfiniBand link. We assume we achieve the theoretical peak bandwidths: 120 Gbit/s for global channels and 40 Gbit/s for local channels. The MPI communication time is bounded by the local channel bandwidths for $n \leqslant 128$, or by the bisection bandwidth otherwise.[7] These parameters yield specific values for $T_{\text{mpi}}(n)$.

Putting these ingredients together with the obvious model

$$T_{\text{soi}}(n) \approx T_{\text{fft}}\big((1+\beta)n\big) + cT_{\text{conv}} + (1+\beta)T_{\text{mpi}}(n),$$

and

$$T_{\text{mkl}}(n) \approx T_{\text{fft}}(n) + 3T_{\text{mpi}}(n),$$

we obtain

$$\text{speedup}(n) \approx \frac{T_{\text{fft}}(n) + 3T_{\text{mpi}}(n)}{T_{\text{fft}}((1+\beta)n) + cT_{\text{conv}} + (1+\beta)T_{\text{mpi}}(n)}.$$

Figure 9 is the resulting projection. This projection is relevant as 3-D torus networks can be found in realistic supercomputers such as the Jaguar system in Oak Ridge National Laboratory with about 18K nodes. As a final remark, while convolution in our current SOI implementation is measured at 40% efficiency, we believe there is room for improvement: We have hitherto only spent a modest effort in optimizing the convolution code, and there might also exist better convolution algorithm for our purpose. The upper envelope of $c = 0.75$ correspond to an improvement of convolution efficiency to 50%, which is quite likely realizable.

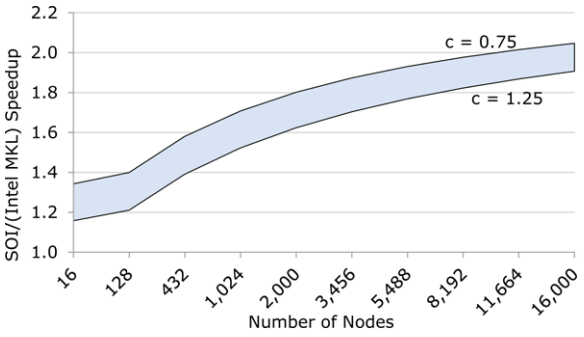


Fig. 9. Speed up projection on a hypothetical 3-D torus network. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/SPR-130373.)

[7] We compute the communication time bound by bisection bandwidth as (Data transferred across a bisection)/(Bisection bandwidth), where (Data transferred across a bisection) = (the total data transferred)/2 due to symmetry and (Bisection bandwidth) = $4n/k$ [9].

8. Conclusion

We have introduced a new FFT framework, derived from it a class of approximation-based DFT factorizations, and implemented one specific algorithm based on these factorizations that exhibited exemplary performance. Algorithms based on our framework have many desirable properties: efficient – as they save communication cost significantly, versatile – as they admit many design choices such as window function, oversampling rate, and accuracy-performance tradeoff, and simple – as they involve merely standard linear-algebra and DFT kernels. The framework is more general than what we have hitherto presented. To entice continued interest from the readers, we wish to elaborate briefly here in this concluding section two facets: window functions and Theorem 1.

The choice of window function is a key design component. It determines, among other things, the condition number κ and aliasing error $\varepsilon^{(\text{alias})}$. These two parameters control the ultimate accuracy that is achievable. In this paper, we have focused exclusively on the class of two-parameter window functions given in Eq. (2). The two-parameter space contains some window functions with $\kappa \approx 1$ and $\varepsilon^{(\text{alias})}$ close to machine roundoff while β is kept small at $1/4$. Had we used a simple one-parameter Gaussian function, $\exp(-\sigma t^2)$, one can show that the accuracy will be limited to 10 digits at best if β is kept at $1/4$. Achieving full double-precision accuracy would require β be set to 1. Another kind of window functions $\hat{w}$, those with compact support (cf. [7]), can eliminate aliasing error completely and may be desirable for some applications. Theoretically, our DFT factorizations can be made exact with these window functions. We can also rederive the factorization in [14] by one particular compactly supported window. Consider $\hat{w}$ that is 1 on $[0, M-1]$ and zero outside $(-1, M)$. With no oversampling or truncation, our framework corresponds to an exact factorization

$$F_N = (I_P \otimes F_M)\mathrm{P}_{\text{erm}}^{P,N}(I_M \otimes F_P)W^{(\text{exact})}.$$

The entries of $W^{(\text{exact})}$ are those c_{jk} in Section 3

$$\begin{aligned} c_{jk} &= \frac{1}{M}\sum_{\ell=-\infty}^{\infty} w\left(\left(\frac{j}{M}-\frac{k}{N}\right)-\ell\right) \\ &= \frac{1}{M}\sum_{\ell=-\infty}^{\infty} \exp\bigl(\iota 2\pi\ell(j/M-k/N)\bigr)\hat{w}(\ell) \\ &= \frac{1}{M}\sum_{\ell=0}^{M-1} \omega^{\ell}, \quad \omega = \mathrm{e}^{\iota 2\pi(j/M-k/N)}. \end{aligned}$$

This shows that $W^{(\text{exact})}$ is a permuted form of the matrix M in the factorization used in [14]. In this sense, that factorization is included in our framework. Nevertheless, our main approach is to avoid this kind of abrupt-changing $\hat{w}$ so that $W^{(\text{exact})}$ is amenable to simple sparse approximation. Without this property, the fast multipole method had to be employed in [14] to carry out the matrix–vector product with the matrix M. In general, design of window functions with various properties to varying extent such as locality, smoothness, fast decay rate, and computation ease is still a lively subject.

The convolution theorem is the enabler of our framework. Theorem 1, presented here in a simplified and limited setting, is in fact an instantiation of a more general set up we devised elsewhere where the objects in question are not finite vectors but sequences, even infinite ones, of delta functions. In that setting, the Fourier integral (continuous Fourier transform) is the same as DFT. Sampling and periodization can be expressed, respectively, as multiplication and convolution with generalized functions of the form $\sum_{\ell=-\infty}^{\infty}\delta(t-\ell\Delta)$ where $\delta(t)$ is the Dirac delta function. The general convolution theorem would resemble the standard one: Fourier transform of convolution is pointwise product of Fourier transforms. We proved that theorem via a suitable application of the Poisson Summation Formula. Using that general convolution theorem, a large body of the work generally known as nonuniform FFTs (cf. [12,13,15,29]) can be rederived. Further applications of these convolution theorems should be explored.

We are encouraged by the performance of our algorithm, and even more so by the versatility of our framework. Next steps and opportunities are many: design better window functions, program faster implementations, generalize to higher-dimensional FFTs – to name just a few.

Appendix. Proof of the convolution theorem

Let $y = F_N x$, $\tilde{x} = \frac{1}{M}\,\mathrm{Samp}(x * w; 1/M)$ and $\tilde{y} = F_M\tilde{x}$. We will prove $\tilde{y} = \mathrm{Peri}(y\cdot\hat{w}; M)$. The crucial tool we need is the Poisson Summation Formula [36] (PSF)

$$\sum_{n=-\infty}^{\infty} w(n+\alpha) = \sum_{n=-\infty}^{\infty} \hat{w}(n)\mathrm{e}^{\iota 2\pi\alpha n},$$

which holds for our window functions w and $\hat{w}$ and any fixed $\alpha\in\mathbb{R}$. (For simplicity's sake, we have defined our window more stringently than is necessary for PSF to be valid [5].) Starting from the definition of $\tilde{x}$, rearranging sums and applying PSF give

$$\begin{aligned} \tilde{x}_j &= \frac{1}{M}\sum_{\ell=-\infty}^{\infty} w\left(\frac{j}{M}-\frac{\ell}{N}\right)x_\ell \\ &= \frac{1}{M}\sum_{\ell=0}^{N-1} x_\ell \sum_{p=-\infty}^{\infty} w\left(\frac{j}{M}-\frac{pN+\ell}{N}\right) \\ &= \frac{1}{M}\sum_{\ell=0}^{N-1} x_\ell \sum_{p=-\infty}^{\infty} \hat{w}(p)\mathrm{e}^{\iota 2\pi(\frac{j}{M}-\frac{\ell}{N})p} \\ &= \frac{1}{M}\sum_{p=-\infty}^{\infty} \hat{w}(p)\mathrm{e}^{\iota 2\pi jp/M}\sum_{\ell=0}^{N-1} x_\ell \mathrm{e}^{-\iota 2\pi\ell p/N} \\ &= \frac{1}{M}\sum_{p=-\infty}^{\infty} y_p\hat{w}(p)\mathrm{e}^{\iota 2\pi jp/M}. \end{aligned}$$

The last equation holds because $y = F_N x$ by definition. Turning now to $\tilde{y} = F_N\tilde{x}$, we have

$$\begin{aligned} \tilde{y}_k &= \sum_{j=0}^{M-1} \tilde{x}_j \mathrm{e}^{-\iota 2\pi jk/M} \\ &= \frac{1}{M}\sum_{j=0}^{M-1} \mathrm{e}^{-\iota 2\pi jk/M}\sum_{p=-\infty}^{\infty} y_p\hat{w}(p)\mathrm{e}^{\iota 2\pi jp/M} \\ &= \frac{1}{M}\sum_{p=-\infty}^{\infty} y_p\hat{w}(p)\sum_{j=0}^{M-1} \mathrm{e}^{-\iota 2\pi j(k-p)/M} \\ &= \frac{1}{M}\sum_{p=-\infty}^{\infty}\sum_{m=0}^{M-1} y_{pM+m}\hat{w}(pM+m)A_{k,m}, \end{aligned}$$

where

$$A_{k,m} = \sum_{j=0}^{M-1} \mathrm{e}^{-\iota 2\pi j(k-m)/M} = \begin{cases} M & \text{if } m = k, \\ 0 & \text{otherwise.} \end{cases}$$

Consequently,

$$\tilde{y}_k = \sum_{p=-\infty}^{\infty} y_{pM+k} \hat{w}(pM + k),$$

which yields $\tilde{y} = \mathrm{Peri}(y * \hat{w}; M)$ and completes the proof.

Disclaimer

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. Configurations: Refer to Table 1. For more information go to http://www.intel.com/performance.

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3 and SSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804.

References

[1] E. Agullo, C. Coti, J. Dongarra, T. Herault and J. Langem, QR factorization of tall and skinny matrices in a grid computing environment, in: *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, 2010.

[2] D.H. Bailey, FFTs in external or hierarchical memory, *Journal of Supercomputing* **4**(1) (1990), 23–35.

[3] G. Ballard, J. Demmel, O. Holtz and O. Schwartz, Graph expansion and communication costs of fast matrix multiplication, in: *Proceedings of the 23rd ACM Symposium on Parallelism in Algorithms and Architecture (SPAA)*, 2011.

[4] G. Ballard, J. Demmel, O. Holtz and O. Schwartz, Minimizing communication in numerical linear algebra, *SIAM Journal of Matrix Analysis and Applications* **32** (2012), 866–901.

[5] J.J. Benedetto and G. Zimmermann, Sampling multipliers and the Poisson summation formula, *The Journal of Fourier Analysis and Applications* **3** (1997), 505–523.

[6] E.O. Brigham, *The Fast Fourier Transform and Its Applications*, Prentice-Hall, New York, 1988.

[7] O.P. Bruno, C.A. Geuzaine, J.A. Monro, Jr. and F. Reitich, Prescribed error tolerances within fixed computational times for scattering problems of arbitrarily high frequency: the convex case, *Philosophical Transactions of the Royal Society of London A* **362** (2004), 629–645.

[8] J.W. Cooley and J. Tukey, An algorithm for the machine computation of complex Fourier series, *Mathematics of Computation* **19**(2) (1965), 297–301.

[9] W.J. Dally and B.P. Towles, *Principles and Practices of Interconnection Networks*, Morgan Kaufmann, 2004.

[10] J. Demmel, M. Hoemmen, M. Mohiyuddin and K. Yelick, Avoiding communication in sparse matrix computations, in: *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, 2008.

[11] J. Doi and Y. Negishi, Overlapping methods of all-to-all communication and FFT algorithms for torus-connected massively parallel supercomputers, in: *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, November 2010.

[12] A. Dutt and V. Rokhlin, Fast Fourier transform for nonequispaced data, *SIAM Journal on Scientific Computing* **14**(6) (1993), 1368–1393.

[13] A. Dutt and V. Rokhlin, Fast Fourier transform for nonequispaced data, II, *Applied and Computational Harmonic Analysis* **2** (1995), 85–100.

[14] A. Edelman, P. McCorquodale and S. Toledo, The future fast Fourier transform?, *SIAM Journal on Scientific Computing* **20**(3) (1999), 1094–1114.

[15] J.A. Fessler and B.P. Sutton, Nonuniform fast Fourier transforms using min-max interpolation, *IEEE Transactions on Signal Processing* **51**(2) (2003), 560–574.

[16] M. Frigo and S.G. Johnson, The design and implementation of FFTW, *Proceedings of the IEEE* **93** (2005), 216–231.

[17] W.M. Gentleman and G. Sande, Fast Fourier transforms – for fun and profit, in: *Proceedings of Fall Joint Computer Conference (AFPIS)*, November 1966, pp. 563–578.

[18] I. Gertner and M. Rofheart, A parallel algorithm for 2-d DFT computation with no interprocessor communication, *IEEE Transactions on Parallel and Distributed Systems* **1** (1990), 377–382.

[19] H. Hassanieh, P. Indyk, D. Katabi and E. Price, Simple and practical algorithm for sparse Fourier transform, in: *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2012, pp. 1183–1194.

[20] H. Hassanieh, P. Indyk, D. Katabi and E. Price, Nearly optimal sparse Fourier transform, in: *ACM Symposium on Theory of Computing (STOC)*, 2012, pp. 563–577.

[21] M. Heidman, D. Johnson and C. Burrus, Gauss and the history of the Fast Fourier transform, *IEEE ASSP Magazine* **1** (1984), 14–21.

[22] S. Kumar, Y. Sabharwal, R. Garg and P. Heidelberger, Optimization of all-to-all communication on the Blue Gene/L supercomputer, in: *The 37th IEEE International Conference on Parallel Processing*, 2008, pp. 320–329.

[23] C.V. Loan, *Computational Frameworks for the Fast Fourier Transforms*, SIAM, Philadelphia, 1992.

[24] C. Lu, M. An, Z. Qian and R. Tolimieri, A hybrid parallel M-D FFT algorithm without interprocessor communication, *IEEE International Conference on Acoustics, Speech and Signal Processing* **3** (1993), 281–284.

[25] F. Marino and E.E. Swartzlander, Parallel implementation of multidimensional transforms without interprocessor communication, *IEEE Transactions on Computers* **48**(9) (1999), 951–961.

[26] R.A. Na'mneh, D.W. Pan and R. Adhami, Communication efficient adaptive matrix transpose algorithm for FFT on symmetric multiprocessors, in: *Proceedings of Southeastern Symposium on System Theory*, 2005, pp. 312–315.

[27] R.A. Namneh, W.D. Pan and S.-M. Yoo, Parallel implementations of 1-D fast Fourier transform without interprocessor communication, *International Journal of Computers and Applications* **2** (2007), 180–186.

[28] A. Papoulis, *The Fourier Integral and Its Applications*, McGraw-Hill, 1984.

[29] D. Potts, G. Steidl and M. Tasche, Fast Fourier transforms for nonequispaced data: A tutorial, in: *Modern Sampling Theory: Mathematics and Applications*, J.J. Benedetto and P. Ferreira, eds, 2001, pp. 249–274.

[30] D.S. Scott, Efficient all-to-all communication patterns in hypercube and mesh topologies, in: *Proceedings of the Sixth Distributed Memory Conference*, 1991, pp. 398–403.

[31] software.intel.com/en-us/articles/intel-mkl.

[32] F. Song, H. Ltaief, B. Hadri and J. Dongarra, Scalable tile communication-avoiding QR factorization on multicore cluster systems, in: *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, November 2010.

[33] D. Takahashi, A parallel 1-D FFT algorithm for the Hitachi SR8000, *Parallel Computing* **29** (2003), 679–690.

[34] R. Tolimieri, M. An and C. Lu, *Algorithms for Discrete Fourier Transform and Convolution*, Springer-Verlag, New York, 1989.

[35] M. Wolfe, *High Performance Compilers for Parallel Computing*, Addison-Wesley, 1996.

[36] A. Zygmund, *Trigonometric Series*, 2nd edn, Cambridge Univ. Press, 1968.