IMPLICITLY DEALIASED CONVOLUTIONS ON SHARED MEMORY ARCHITECTURES[*]

MALCOLM ROBERTS[†] AND JOHN C. BOWMAN[‡]

submitted to SIAM J. Sci. Comput.

Abstract. Implicit dealiasing is a more efficient method, compared to zero padding, for computing in-place linear convolutions via fast Fourier transforms. For one-dimensional inputs, the memory requirements are identical to conventional zero-padded convolutions, but for large data lengths, implicit dealiasing is typically slightly faster. In two and three dimensions, the decoupling of work buffers and data storage results in significant memory savings. The Fourier transformed values are not stored in a block image for future processing but instead are processed and discarded as they are generated. This leads to greater data locality, resulting in a dramatic increase in execution speed. A generalized multithreaded implementation of implicit dealiasing that accepts an arbitrary number of input and output vectors is presented, along with an improved 1D Hermitian convolution that avoids the loop dependency inherent in previous work. An alternate data format that can accommodate a Nyquist mode and enhance cache efficiency is also proposed.

Key words. implicit dealiasing, zero padding, convolution, fast Fourier transform, Hermitian, Nyquist mode, multithreading, parallelization, shared memory, correlation

AMS subject classifications. 65R99, 65T50

1. Introduction.

The convolution is an important operator in a wide variety of applications ranging from statistics, signal processing, image processing, and the numerical approximation of solutions to nonlinear partial differential equations. The convolution of $F = \{F_k\}_{k\in\mathbb{Z}}$ and $G = \{G_k\}_{k\in\mathbb{Z}}$ is denoted $F * G = \{(F * G)_k\}_{k\in\mathbb{Z}}$, where $(F * G)_k = \sum_{p\in\mathbb{Z}} F_p G_{k-p}$. In many practical applications, the inputs are of finite length m, with $F = \{F_k\}_{k=0}^{m-1}$ and $G = \{G_k\}_{k=0}^{m-1}$, for which the linear convolution $F * G$ is by $(F * G)_k = \sum_{p=0}^{k} F_p G_{k-p}, \quad k = 0, \dots, m-1$. Computing such a convolution directly requires $\mathcal{O}(m^2)$ operations, and roundoff error is a significant problem when m is large. It is therefore preferable to make use of the convolution theorem and harness the power of the fast Fourier transform (FFT), mapping the convolution to a component-wise multiplication. The FFT requires $\mathcal{O}(m \log m)$ [4, 5] operations and the multiplication requires $\mathcal{O}(m)$ operations, reducing the computational complexity to $\mathcal{O}(m \log m)$, while improving the numerical accuracy [6].

Since the FFT considers the inputs F and G to be periodic, the direct application of the convolution theorem results in a circular convolution, due to the indices being computed modulo m. Removing these extra aliases from the periodic convolution to produce a linear convolution is called *dealiasing*. We give a brief overview of the dealiasing requirements for different types of convolutions in Section 2.

The standard method for dealiasing FFT-based convolutions is to pad the inputs with a sufficient number of zero values such that the aliased contributions are all zero, as shown in Figure 1. In Section 3, we review the alternative method of implicit dealiasing [2]. In Section 4 we show for dimensions greater than one that implicit dealiasing uses much less memory and is much faster than explicit dealiasing. Implicitly dealiased convolution routines are publicly available in the open-source software library `FFTW++` [3] (version 2.02).

[*] This work was supported by the Natural Sciences and Engineering Research Council of Canada.

[†]IRMA, University of Strasbourg, 7 rue René-Descartes,67084 Strasbourg Cedex.

[‡]Department of Mathematical and Statistical Sciences, University of Alberta, Edmonton, Alberta T6G 2G1, Canada.

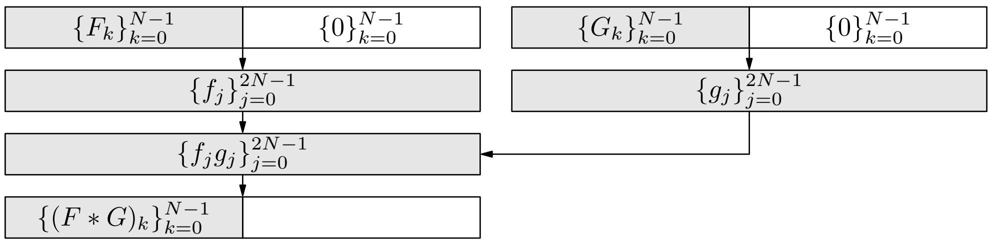


FIG. 1. *Computing a 1D convolution via explicit zero padding.*

In this paper, we generalize implicit dealiasing to handle an arbitrary number of input and output vectors. This allows implicit dealiasing to be efficiently applied to, for example, autocorrelations and pseudospectral simulations of nonlinear partial differential equations (e.g. in hydrodynamics and magnetohydrodynamics). We also discuss key technical improvements that allow implicit dealiasing to be fully multithreaded. Conclusions and future directions for research are given in Section 5.

2. Dealiasing requirements for convolutions. The type of input data and application determines important properties of how the convolution is computed. In this work, we consider the case where the elements of F and G are complex. If the inverse Fourier transform f of F is real valued, then F exhibits a Hermitian symmetry that can be used to reduce storage requirements. Moreover, complex-to-real/real-to-complex Fourier transforms make use of Hermitian symmetry to reduce the computational complexity of the transform by a factor of about two.

For input data of length m that is complex, the data is padded with m zeroes for a total FFT length of $2m$, which we refer to as 1/2 *padding*. If the input data is multidimensional with size $m_1 \times \ldots \times m_d$, then the data must be zero padded to $2m_1 \times \ldots \times 2m_d$, increasing the buffer size by a factor of 2^d.

In the case where the complex input data has Hermitian symmetry, it is convenient to shift the zero wavenumber in the transformed data to the middle of the array. In this case, the inputs are $F = \{F_k\}_{k=-m+1}^{m-1}$ and $G = \{G_k\}_{k=-m+1}^{m-1}$, and their convolution is $(F*G)_k = \sum_{p=k-m+1}^{m-1} F_p G_{k-p}$, where $k = -m+1, \ldots, m-1$. This reduces the amount of zero padding that is necessary: data of length $2m-1$ needs to be padded only to length $3m-2$ (normally extended to $3m$); this is called 2/3 *padding* [8]. The buffer size for d-dimensional Hermitian data is thus increased by a factor of $(3/2)^d$.

A binary convolution can be generalized to an n-ary operator $*(F_1, \ldots, F_n)$, with $*(F_1, \ldots, F_n)_k = \sum_{p_1,\ldots,p_n} F_{p_1} \cdots F_{p_n} \delta_{p_1+\ldots+p_n,k}$, where δ is the Kronecker delta. Although this may be computed as a series of binary convolutions, the padding must still be extended to avoid aliases from the constructive beating of the modes p_1, p_2, p_3, …, p_n. As a result, n-ary convolutions benefit greatly from implicit dealiasing.

A common application of the convolution is the autoconvolution, which can be used to determine the periodicity of a signal. In this case, there is only one input, and the efficient computation of the autoconvolution involves just two FFTs instead of the three FFTs required for computing the convolution of two separate inputs f and g. In general, we consider a convolution operation that takes A inputs and produces B outputs, where the multiplication performed in the transformed space can be an arbitrary component-wise operation. In order to make use of 1/2 padding or 2/3 padding (for noncentered or centered inputs, respectively), the multiplication operator must be quadratic. If the multiplication operator is of higher degree, one must extend the padding to remove undesired aliases.

To compute a convolution with A inputs and B outputs using the convolution theorem, one performs A backward FFTs to transform the inputs, applies the appropriate multiplication operation on the transformed data, and then performs B forward FFTs to produce the final outputs, for a total of $A+B$ FFTs. For the standard binary convolution, there are two inputs and one output, with the multiplication operation $(x,y) \to (xy)$, which is applied element-by-element. An autoconvolution can be computed with just two transforms using $A = B = 1$ and the operation $x \to x^2$. To compute the nonlinear term of the two-dimensional incompressible Navier–Stokes vorticity equation, the operation is $(u_x, u_y, \partial\omega/\partial x, \partial\omega/\partial y) \to (u_x \partial\omega/\partial x + u_y \partial\omega/\partial y)$, where $\boldsymbol{u} = (u_x, u_y)$ is the 2D velocity, $\omega = \hat{\boldsymbol{z}} \cdot \boldsymbol{\nabla} \times \boldsymbol{u}$ is the z-component of the vorticity, and a total of 5 FFTs are required. For three-dimensional magnetohydrodynamic (MHD) flows the operation is $(\boldsymbol{u}, \boldsymbol{\omega}, \boldsymbol{B}, \boldsymbol{j}) \to (\boldsymbol{u} \times \boldsymbol{\omega} + \boldsymbol{j} \times \boldsymbol{B}, \boldsymbol{u} \times \boldsymbol{B})$, where $\boldsymbol{u}$ is the (3D) velocity, $\boldsymbol{\omega} = \boldsymbol{\nabla} \times \boldsymbol{u}$ is the vorticity, $\boldsymbol{B}$ is the magnetic field, and $\boldsymbol{j}$ is the current density. Computing the nonlinear term of the MHD equations in this fashion requires 12 backward and 6 forward FFTs. For the Navier–Stokes and MHD equations, the operation is quadratic in its arguments and the convolution is binary ($n = 2$). Since the wavevectors are symmetric about the Fourier origin, 2/3 padding can be used.

3. One-dimensional implicitly dealiased convolutions.

Implicit padding allows one to dealias convolutions without having to write, read, and multiply by explicit zero values. This is accomplished by implicitly incorporating the zero values into the top level of a decimated-in-frequency FFT. The extra memory previously used for padding now appears as a decoupled work buffer. One-dimensional implicitly dealiased convolutions therefore have the same memory requirements as explicitly padded convolutions. Although in one dimension implicit padding is slightly more efficient than explicit zero padding, the key advantage in one dimension is that user input data does not need to be copied to an enlarged zero-padded buffer before performing the FFT. Let us now describe the optimized 1D building blocks that will be used in Section 4 to construct higher-dimensional implicitly dealiased convolutions that are much more compact and efficient than their explicit counterparts.

3.1. Complex convolution.

The Fourier origin for standard convolutions is located at the first (zero) index of the array. Therefore, input data vectors of length m must be padded with zeros to length $N \geq 2m-1$ to prevent modes with wavenumber $m-1$ from beating together to contaminate mode $N = 0 \bmod N$. However, since FFT sizes with small prime factors in practice yield the most efficient implementations, it is normally desirable to extend the padding to $N = 2m$.

In terms of the Nth primitive root of unity, $\zeta_N \doteq \exp(2\pi i/N)$ (here $\doteq$ is used to emphasize a definition), the unnormalized backward discrete Fourier transform of a complex vector $\{F_k : k = 0, \ldots, N-1\}$ may be written as $f_j \doteq \sum_{k=0}^{N-1} \zeta_N^{jk} F_k$, where $j = 0, \ldots, N-1$. The fast Fourier transform method exploits the properties that $\zeta_N^r = \zeta_{N/r}$ and $\zeta_N^N = 1$.

On taking $N = 2m$ with $F_k = 0$ for $k \geq m$, one can easily avoid looping over the unphysical zero Fourier modes by decimating in wavenumber: for $\ell = 0, 1, \ldots, m-1$:

$$(1) \qquad f_{2\ell} = \sum_{k=0}^{m-1} \zeta_{2m}^{2\ell k} F_k = \sum_{k=0}^{m-1} \zeta_m^{\ell k} F_k, \quad f_{2\ell+1} = \sum_{k=0}^{m-1} \zeta_{2m}^{(2\ell+1)k} F_k = \sum_{k=0}^{m-1} \zeta_m^{\ell k} \zeta_{2m}^{k} F_k.$$

This requires computing two subtransforms, each of size m, for an overall compu-

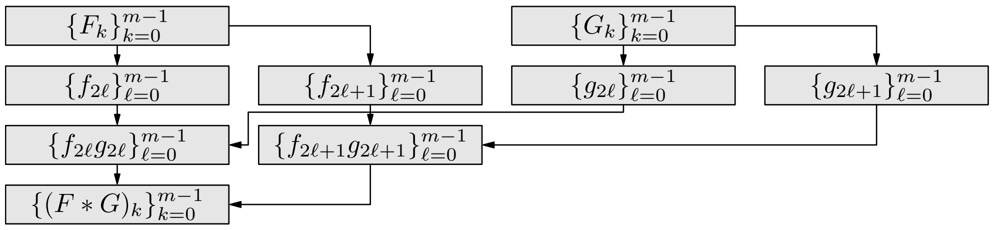

$\{F_k\}_{k=0}^{m-1}$
$\{G_k\}_{k=0}^{m-1}$
$\{f_{2\ell}\}_{\ell=0}^{m-1}$
$\{f_{2\ell+1}\}_{\ell=0}^{m-1}$
$\{g_{2\ell}\}_{\ell=0}^{m-1}$
$\{g_{2\ell+1}\}_{\ell=0}^{m-1}$
$\{f_{2\ell}g_{2\ell}\}_{\ell=0}^{m-1}$
$\{f_{2\ell+1}g_{2\ell+1}\}_{\ell=0}^{m-1}$
$\{(F*G)_k\}_{k=0}^{m-1}$

FIG. 2. *Computing a 1D convolution via implicit dealiasing.*

tational scaling of order $2m\log_2 m = N\log_2 m$. The odd and even terms of the convolution can then be computed separately (without the need for a bit reversal stage), multiplied term by term, and finally transformed again to Fourier space using the (scaled) forward transform

$$
\begin{aligned}
2mF_k &= \sum_{j=0}^{2m-1} \zeta_{2m}^{-kj} f_j = \sum_{\ell=0}^{m-1} \zeta_{2m}^{-k2\ell} f_{2\ell} + \sum_{\ell=0}^{m-1} \zeta_{2m}^{-k(2\ell+1)} f_{2\ell+1} \\
&= \sum_{\ell=0}^{m-1} \zeta_{m}^{-k\ell} f_{2\ell} + \zeta_{2m}^{-k} \sum_{\ell=0}^{m-1} \zeta_{m}^{-k\ell} f_{2\ell+1}, \qquad k = 0,\dots,m-1. \qquad (2)
\end{aligned}
$$

These algorithms were combined in Function cconvof Ref. [2] to compute a dealiased binary convolution of unpadded length m for the inputs f and g. Figure 2 illustrates the data flow for the method of implicit dealiasing. For each input, two arrays of size m are used instead of one array of size $2m$. This seemingly trivial distinction is the key to the improved efficiency and reduced storage requirements of the higher-dimensional implicit convolutions described in Section 4. Moreover, in the 1D complex case, each of the six complex Fourier subtransforms of size m could be done out of place. Since implicit dealiasing does not compute the entire inverse Fourier transformed image at once, we included in our implementation a facility for determining the spatial coordinates of each point as it is processed. For example, this could be used for generating an image in x space of the inverse transformed data.

As is in our previous work [2], we calculate the ζ_N^k factors with a single complex multiply, using two short pre-computed tables $H_a = \zeta_N^{as}$ and $L_b = \zeta_N^b$, where $k = as+b$ with $s = \lfloor\sqrt{m}\rfloor$, $a = 0,1,\dots,\lceil m/s\rceil - 1$, and $b = 0,1,\dots,s-1$. Since these one-dimensional tables require only $\mathcal{O}(\sqrt{m})$ complex words of storage and our focus is on higher-dimensional convolutions anyway, we do not account for them in our storage estimates.

Function cconvin is a generalization of Function cconv of Ref. [2] allowing for A inputs and B outputs. For the standard binary convolution, $A = 2$ and $B = 1$, and the multiplication operator is $(x, y) \to xy$. For autoconvolutions, $A = B = 1$ and the multiplication operator is $x \to x^2$, whereas autocorrelations use $x \to x\bar{x}$, where $\bar{x}$ is the complex conjugate of x.

Out-of-place FFTs are often more efficient than their in-place counterparts. It is not possible to make use of out-of-place FFTs for explicitly dealiased convolutions without allocating additional memory, but the situation is different with implicitly dealiased convolutions. For example, in Function cconvof Ref. [2] all FFTs were out of place. Function cconvin has A in-place FFTs and $A + 2B$ out-of-place FFTs but the function is more general, and in some cases (e.g. autoconvolutions, where

$A = B = 1$), there does not appear to be any way to make the FFTs out of place without allocating additional storage.

However, when $A > B$, the multiplication operator will free buffers that can then be reused, and it is possible to compute the convolution with all FFTs out of place. The idea is that the input and work buffers can be processed separately, and, after applying the multiplication operator, the data in the last work buffer is no longer needed. We make use of this buffer to implement out-of-place transforms. In Function cconvout, we present an algorithm for an implicitly dealiased convolution in which all $2A + 2B$ FFTs of length m are out of place.

```
Input: vectors $\{\mathsf{f}_a\}_{a=0}^{A-1}$
Output: vectors $\{\mathsf{f}_b\}_{b=0}^{B-1}$
for $a = 0$ to $A - 1$ do
    $\mathsf{u}_a \leftarrow \texttt{fft}^{-1}(\mathsf{f}_a)$
$\{\mathsf{u}_b\}_{b=0}^{B-1} \leftarrow \texttt{mult}(\{\mathsf{u}_a\}_{a=0}^{A-1})$
for $a = 0$ to $A - 1$ do
    parallel for $k = 0$ to $m - 1$ do
        $\mathsf{f}_a[k] \leftarrow \zeta_{2m}^{k}\mathsf{f}_a[k]$
    $\mathsf{f}_a \leftarrow \texttt{fft}^{-1}(\mathsf{f}_a)$
$\{\mathsf{f}_b\}_{b=0}^{B-1} \leftarrow \texttt{mult}(\{\mathsf{f}_a\}_{a=0}^{A-1})$
for $b = 0$ to $B - 1$ do
    $\mathsf{f}_b \leftarrow \texttt{fft}^{-1}(\mathsf{f}_b)$
    $\mathsf{u}_b \leftarrow \texttt{fft}^{-1}(\mathsf{u}_b)$
    parallel for $k = 0$ to $m - 1$ do
        $\mathsf{f}_b[k] \leftarrow \mathsf{f}_b[k] + \zeta_{2m}^{-k}\mathsf{u}_b[k]$
return $\{\mathsf{f}_b/(2m)\}_{b=0}^{B-1}$
```

Function cconvin returns the in-place implicitly dealiased 1D convolution of the complex vectors $\{f_a\}_{a=0}^{A-1}$ using the multiplication operator $\texttt{mult} : \mathbb{C}^A \to \mathbb{C}^B$. Each of the FFT transforms is multithreaded, with A of the $A + 2B$ transforms performed out of place.

```
Input: vectors $\{\mathsf{f}_a\}_{a=0}^{A-1}$
Output: vectors $\{\mathsf{f}_b\}_{b=0}^{B-1}$
for $a = 0$ to $A - 1$ do
    $\mathsf{u}_a \leftarrow \texttt{fft}^{-1}(\mathsf{f}_a)$
$\{\mathsf{u}_b\}_{b=0}^{B-1} \leftarrow \texttt{mult}(\{\mathsf{u}_a\}_{a=0}^{A-1})$
for $a = 0$ to $A - 1$ do
    parallel for $k = 0$ to $m - 1$ do
        $\mathsf{f}_a[k] \leftarrow \zeta_{2m}^{k}\mathsf{f}_a[k]$
$\mathsf{u}_{A-1} \leftarrow \texttt{fft}^{-1}(\mathsf{f}_{A-1})$
for $a = A - 2$ to $0$ do
    $\mathsf{f}_{a+1} \leftarrow \texttt{fft}^{-1}(\mathsf{f}_a)$
$\{\mathsf{f}_b\}_{b=0}^{B-1} \leftarrow \texttt{mult}(\{\mathsf{f}_a\}_{a=1}^{A-1} \cup \{\mathsf{u}_{A-1}\})$
for $b = 0$ to $B - 1$ do
    $\mathsf{f}_b \leftarrow \texttt{fft}^{-1}(\mathsf{f}_{b+1})$
    $\mathsf{u}_{A-1} \leftarrow \texttt{fft}^{-1}(\mathsf{u}_b)$
    parallel for $k = 0$ to $m - 1$ do
        $\mathsf{f}_b[k] \leftarrow \mathsf{f}_b[k] + \zeta_{2m}^{-k}\mathsf{u}_{A-1}[k]$
return $\{\mathsf{f}_b/(2m)\}_{b=0}^{B-1}$
```

Function cconvout returns the in-place implicit dealiased 1D convolution of the complex vectors $\{f_a\}_{a=0}^{A-1}$ using the multiplication operator $\texttt{mult} : \mathbb{C}^A \to \mathbb{C}^B$, with $A > B$. All $2A + 2B$ FFTs are out of place.

For complex-to-complex convolutions with $A = 2$ inputs and $B = 1$ outputs, Function cconvout provides a speedup over Function cconvin of about 5% percent for most problem sizes. Thanks to improvements in the loop structure in the pre-and post-processing stages in Function cconvout, we also have a speedup of a few percent over Function cconv from Ref. [2]. As shown in Figure 3(a), the computation time of Function cconvout is approximately 10% faster than a comparable explicitly-padded algorithm.

3.1.1. Multithreaded Complex 1D Binary Convolutions. We parallelize Functions cconvin and cconvout using OpenMP in our pre/post-processing phases and in the multiplication operator, while taking advantage of the multithreading built into FFTW. In Figure 3(a), we compare the speed of the implicit and explicit algorithms using four threads. Using one thread, the implicit method is 1.07–1.28 times faster than the explicit method for $m \geq 256$, whereas using four threads the performance improvement is a factor of 1.03–2.7 for $m \geq 8192$. The error bars in the timing figures indicate the lower and upper one-sided standard deviations as given in Ref. [2]. In

Figure 3(b) we show the parallel scaling of the implicit method, which, for $m \geq 512$ has a parallel efficiency of 50–85%, giving a speedup of a factor of 2–3.4.

Both the `FFTW-3.3.4` library and the convolution layer we built on top of it were compiled with the GCC 5.3.1 20151207 compiler. Our library was compiled with the optimizations `-fopenmp -fomit-frame-pointer -fstrict-aliasing -ffast-math -msse2 -mfpmath=sse -march=native` and executed on a 64-bit 3.4GHz Intel i7-2600K processor with 8MB cache. Like the `FFTW` library, our algorithms were vectorized with specialized single-instruction multiple-data (SIMD) code.

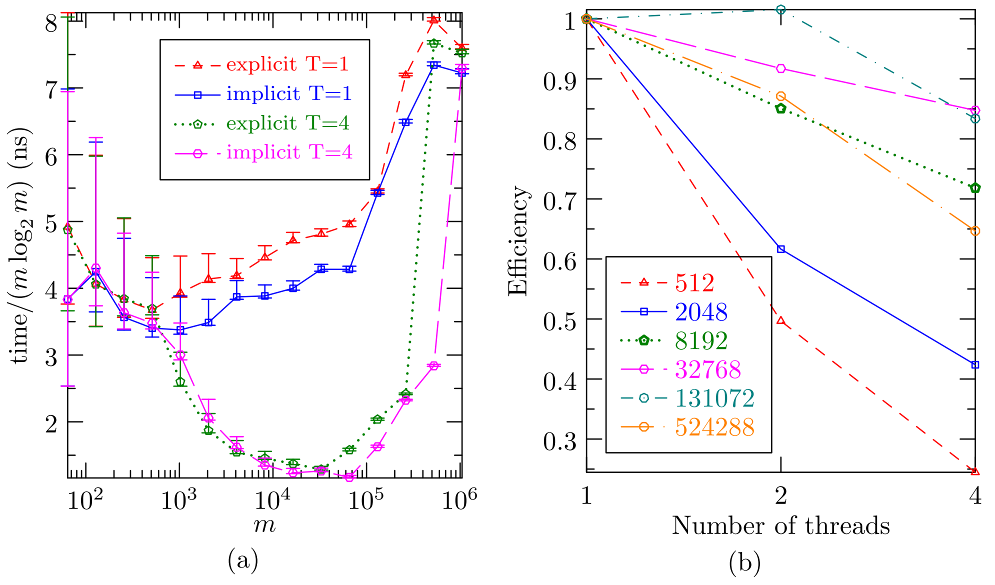


FIG. 3. *In-place 1D complex convolutions of length m: (a) Comparison of computation times for explicit and implicit dealiasing using $T = 1$ thread and $T = 4$ threads; (b) parallel efficiency of implicit dealiasing versus number of threads for various values of m.*

3.2. Centered data formats. In this work, we extend the case where $N = 2m - 1$, treated previously in Ref. [2], to general N. In addition to allowing for an even number of real data points, the new $F_{-m}, \dots, F_{m-1}$ data layout can yield significant performance improvements, even with the additional mode F_{-m} set to zero. We refer to $\{F_{-m+1}, \dots, F_{m-1}\}$ as the *compact format* and $\{F_{-m}, \dots, F_{m-1}\}$ as the *noncompact format.* In particular, in the noncompact case, a fully multithreaded implementation is possible (cf. Procedure fft1padBackward) since it doesn't have the loop dependency inherent in Procedure fft0padBackward.[1] Another advantage of the noncompact format is that it is consistent with the output of a real-to-complex FFT.

Although the noncompact format has slightly smaller storage requirements, on some architectures with more than one (typically a power of two) memory banks, stride resonances can significantly hurt performance if successive multidimensional array accesses fall on the same memory bank. A similar effect can occur on modern architectures due to cache associativity. It is therefore useful in the centered case to allow the user to choose between the two data formats.

[1]We correct here a sequencing error in the pseudocode for Procedure fft0padBackward in Ref. [2]. Minor typographical errors also appeared on page 388 ($0 \dots N$ should be $0 \dots N - 1$) and on p. 400 ($2m_1 - 1$ should be $2m_z - 1$).

In the compact (noncompact), format, it is convenient to shift the Fourier origin so that the $k = 0$ mode is indexed as array element $m - 1$ (m). This shift, which can be built into implicitly dealiased convolution algorithms at no extra cost, allows for more convenient coding of wavenumber loops since the high-wavenumber cutoff is naturally aligned with the array boundaries.

In the compact case, where $N = 2m - 1$, one needs to pad to $N \geq 3m - 2$ to prevent modes with wavenumber $m - 1$ from beating together to contaminate the mode with wavenumber $-m + 1$. The ratio of the number of physical to total modes, $(2m-1)/(3m-2)$, is then asymptotic to $2/3$ for large m [8]. For explicit padding, one usually chooses the padded vector length N to be a power of 2, with $m = \lfloor (n+2)/3 \rfloor$. However, for implicit padding, it is advantageous to choose m itself to be a power of 2 since the algorithm reduces to computing FFTs of length m and the most efficient FFT algorithms tend to be those for power-of-two data lengths. Moreover, it is convenient to pad implicitly slightly beyond $3m - 2$, to $N = 3m$, as this allows the use of a radix 3 subdivision at the outermost level.

In the case of an even number, $2m$, of spatial data points, one must use the noncompact data format, with modes running from $-m$ to $m - 1$. When $N = 3m$, the most negative (Nyquist) wavenumber $-m$ can then constructively beat with itself, producing an alias in mode $-m + (-m) = -2m = m \bmod 3m$, which is equivalent to itself modulo $2m$. To remove this alias we set the Nyquist mode to zero at the end of the convolution, after accounting for its effects on the other modes. We note that there are no aliases in $\{-m, \ldots, 2m-1\}$ arising from the interaction of mode $-m$ with any of the other modes. As we will see in Section 3.2.1, those interactions can (and in fact, should) be retained. We split the coefficient for $k = -m$ equally between F_{-m} and its equivalent F_m (modulo $2m$); in Section 3.2.1, we will see that this is important for maintaining Hermitian symmetry and the corresponding reality of the spatial field.

We now describe a noncompact implicitly dealiased centered Fourier transform; the compact case is obtained by setting $F_{-m} = F_m = 0$. Suppose then that $F_k = 0$ for $k > m$. On decomposing $j = (3\ell + r) \bmod N$, where $r \in \{-1, 0, 1\}$, the substitution $k' = m + k$ allows us to write the backward transform as

$$f_{3\ell+r} = \sum_{k=-m}^{m} \zeta_m^{\ell k} \zeta_{3m}^{rk} F_k = \sum_{k'=0}^{m-1} \zeta_m^{\ell k'} \zeta_{3m}^{r(k'-m)} F_{k'-m} + \sum_{k=0}^{m} \zeta_m^{\ell k} \zeta_{3m}^{rk} F_k = \sum_{k=0}^{m-1} \zeta_m^{\ell k} w_{k,r},$$

where, on recombining F_m into F_{-m},

$$w_{k,r} \doteq \begin{cases} F_0 + \operatorname{Re} \zeta_3^{-r} F_{-m} & \text{if } k = 0, \\ \zeta_{3m}^{rk} (F_k + \zeta_3^{-r} F_{k-m}) & \text{if } 1 \leq k \leq m - 1. \end{cases}$$

The forward transform is then

$$3mF_k = \sum_{r=-1}^{1} \zeta_{3m}^{-rk} \sum_{\ell=0}^{m-1} \zeta_m^{-\ell k} f_{3\ell+r}, \qquad k = -m + 1, \ldots, m - 1,$$

with the Nyquist mode F_{-m} set to zero. The use of the remainder $r = -1$ instead of $r = 2$ allows us to exploit the optimization $\zeta_{3m}^{-k} = \overline{\zeta_{3m}^{k}}$ in Eqs. (3.2) and (3.2). The number of complex multiplies needed to evaluate Eq. (3.2) for $r = \pm 1$ can be reduced by computing the intermediate complex quantities $A_k \doteq \zeta_{3m}^{k} \bigl(\operatorname{Re} F_k + \zeta_3^{-1} \operatorname{Re} F_{k-m}\bigr)$ and $B_k \doteq i\zeta_{3m}^{k} \bigl(\operatorname{Im} F_k + \zeta_3^{-1} \operatorname{Im} F_{k-m}\bigr)$, where $\zeta_3^{-1} = (-\frac{1}{2}, -\frac{\sqrt{3}}{2})$, so that for $k > 0$,

$w_{k,1} = A_k + B_k$ and $w_{k,-1} = \overline{A_k - B_k}$. The resulting (corrected) backwards transform is given in Procedure `fft0padBackward`; its inverse, Procedure `fft0padForward`, is given in Ref. [2].

```
Input: vector $\mathsf{f}$
Output: vector $\mathsf{f}$, vector $\mathsf{u}$
$\mathsf{u}[0] \leftarrow \mathsf{f}[m-1]$
for $k = 1$ to $m-1$ do
    $\mathsf{A} \leftarrow \zeta_{3m}^k \left[\operatorname{Re}\mathsf{f}[m-1+k] + \left(-\frac{1}{2}, -\frac{\sqrt{3}}{2}\right)\operatorname{Re}\mathsf{f}[0]\right]$
    $\mathsf{B} \leftarrow i\zeta_{3m}^k \left[\operatorname{Im}\mathsf{f}[m-1+k] + \left(-\frac{1}{2}, -\frac{\sqrt{3}}{2}\right)\operatorname{Im}\mathsf{f}[0]\right]$
    $\mathsf{C} \leftarrow \mathsf{f}[m-1+k] + \mathsf{f}[0]$
    $\mathsf{f}[0] \leftarrow \mathsf{f}[k]$
    $\mathsf{f}[k] \leftarrow \mathsf{C}$
    $\mathsf{f}[m-1+k] \leftarrow \mathsf{A} + \mathsf{B}$
    $\mathsf{u}[k] \leftarrow \overline{\mathsf{A} - \mathsf{B}}$
$\mathsf{f}[0,\ldots,m-1] \leftarrow \texttt{fft}^{-1}(\mathsf{f}[0,\ldots,m-1])$
$\mathsf{u}[m] \leftarrow \mathsf{f}[m-1]$
$\mathsf{f}[m-1] \leftarrow \mathsf{u}[0]$
$\mathsf{f}[m-1,\ldots,2m-2] \leftarrow \texttt{fft}^{-1}(\mathsf{f}[m-1,\ldots,2m-2])$
$\mathsf{u}[0,\ldots,m-1] \leftarrow \texttt{fft}^{-1}(\mathsf{u}[0,\ldots,m-1])$
```

Procedure `fft0padBackward(f,u)` stores the shuffled $3m$-padded centered backward Fourier transform values of a compact-format vector of length $2m-1$ in f and an auxiliary vector u of length $m+1$.

```
Input: vector $\mathsf{f}$
Output: vector $\mathsf{f}$, vector $\mathsf{u}$
$\mathsf{A} \leftarrow \mathsf{f}[0]$
$\mathsf{f}[0] \leftarrow \mathsf{f}[m] + 2\mathsf{A}$
$\mathsf{f}[m] \leftarrow \mathsf{f}[m] - \mathsf{A}$
$\mathsf{u}[0] \leftarrow \mathsf{f}[m]$
parallel for $k = 1$ to $m-1$ do
    $\mathsf{A} \leftarrow \zeta_{3m}^k \left[\operatorname{Re}\mathsf{f}[m+k] + \left(-\frac{1}{2}, -\frac{\sqrt{3}}{2}\right)\operatorname{Re}\mathsf{f}[k]\right]$
    $\mathsf{B} \leftarrow i\zeta_{3m}^k \left[\operatorname{Im}\mathsf{f}[m+k] + \left(-\frac{1}{2}, -\frac{\sqrt{3}}{2}\right)\operatorname{Im}\mathsf{f}[k]\right]$
    $\mathsf{f}[k] \leftarrow \mathsf{f}[k] + \mathsf{f}[m+k]$
    $\mathsf{f}[m+k] \leftarrow \mathsf{A} + \mathsf{B}$
    $\mathsf{u}[k] \leftarrow \overline{\mathsf{A} - \mathsf{B}}$
$\mathsf{f}[0,\ldots,m-1] \leftarrow \texttt{fft}^{-1}(\mathsf{f}[0,\ldots,m-1])$
$\mathsf{f}[m,\ldots,2m-1] \leftarrow \texttt{fft}^{-1}(\mathsf{f}[m,\ldots,2m-1])$
$\mathsf{u}[0,\ldots,m-1] \leftarrow \texttt{fft}^{-1}(\mathsf{u}[0,\ldots,m-1])$
```

Procedure `fft1padBackward(f,u)` stores the shuffled $3m$-padded centered backward Fourier transform values of a noncompact-format vector f of length $2m$ in f and an auxiliary vector u of length m. The Fourier origin is at position m.

```
**Input**: vector f, vector u
**Output**: vector f
$\mathsf{f}[0,\dots,m-1] \leftarrow \mathtt{fft}(\mathsf{f}[0,\dots,m-1])$
$\mathsf{f}[m,\dots,2m-1] \leftarrow \mathtt{fft}(\mathsf{f}[m,\dots,2m-1])$
$\mathsf{u}[0,\dots,m-1] \leftarrow \mathtt{fft}(\mathsf{u}[0,\dots,m-1])$
$\mathsf{f}[m] \leftarrow \mathsf{f}[0] + \mathsf{f}[m] + \mathsf{u}[0]$
$\mathsf{f}[0] \leftarrow 0$
parallel for $k = 1$ to $m-1$ do
    $\mathsf{A} \leftarrow \mathsf{f}[k]$
    $\mathsf{f}[k] \leftarrow \mathsf{A} + \left(-\frac{1}{2}, \frac{\sqrt{3}}{2}\right)\zeta_{3m}^{-k}\mathsf{f}[m+k] + \left(-\frac{1}{2}, -\frac{\sqrt{3}}{2}\right)\zeta_{3m}^{k}\mathsf{u}[k]$
    $\mathsf{f}[m+k] \leftarrow \mathsf{A} + \zeta_{3m}^{-k}\mathsf{f}[m+k] + \zeta_{3m}^{k}\mathsf{u}[k]$
return $\mathsf{f}/(3\mathrm{m})$
```

Procedure `fft1padForward(f,u)` returns the inverse of `fft1padBackward(f,u)`, with f[0] (the Nyquist mode for spatial data of even length) set to zero.

3.2.1. Centered Hermitian Implicitly Padded 1D FFT. The Fourier transform of real data satisfies the Hermitian symmetry $F_{-k} = \overline{F_k}$. This implies that the Fourier coefficient corresponding to $k = 0$ is real. There is a further consequence of this symmetry when the length N of the discrete transform $\sum_{j=0}^{N-1} \zeta_N^{-kj} f_j$ is even. Due to the periodicity of the discrete transform in N, the highest frequency (Nyquist) mode must also be real: $F_{N/2} = F^*_{-N/2} = F^*_{N/2}$. Letting $m = \lfloor N/2 \rfloor$, in the case where N is even, the $2m$ modes can therefore be indexed as $\{F_{-m+1}, \dots, F_m\}$ where F_0 and F_m are real. An odd number $N = 2m - 1$ of modes is indexed as $\{F_{-m+1}, \dots, F_{m-1}\}$.

A one-dimensional convolution of Hermitian data only requires the data corresponding to non-negative wavenumbers. In the compact case, with modes in $\{-m + 1, \dots, m - 1\}$, the unsymmetrized physical data needs to be padded with at least $m - 1$ zeros, just as in Section 3.2. Hermitian symmetry thus necessitates padding the m non-negative wavenumbers with at least $c \doteq \lfloor m/2 \rfloor$ zeros. The resulting 2/3 padding ratio (for even m) turns out to work particularly well for developing implicitly dealiased centered Hermitian convolutions. As in the centered case, we again choose the Fourier size to be $N = 3m$.

In the noncompact case, it is sufficient to retain the modes $\{0, \dots, m\}$. One could of course treat this as compact data of size $m + 1$, but in the frequently occurring case where $m = 2^p$ this would require the computation of subtransforms of length $2^p + 1$ instead of 2^p. The most efficient available FFT algorithms are typically those of size 2^p.

Fortunately, the choice $N = 3m$ also works for the noncompact case, provided the entry for the Nyquist mode, which must be real, is zeroed at the end of the convolution. For example, direct autoconvolution of the Hermitian data $\{(1, 0), (2, 3), (4, 0)\}$ yields $\{(59, 0), (20, -18), (3, 12)\}$. With $m = 3$, the compact implicitly dealiased convolution in Procedure `conv` yields identical results. For $m = 2$, the noncompact version yields the correct values for the first m elements $\{(59, 0), (20, -18)\}$ only if the data $(3, 12)$ for the Nyquist mode at $k = m$ is taken into account.

Let us now describe a noncompact implicitly dealiased Hermitian implicitly padded FFT; the compact case can then be obtained by setting $F_m = 0$. Given that $F_k = 0$ for $k > m$, the backward (complex-to-real) transform appears as Eq. (3.2), but now with

$$w_{k,r} \doteq \begin{cases} F_0 + \operatorname{Re} \zeta_3^{-r} F_m & \text{if } k = 0, \\ \zeta_{3m}^{rk} \left(F_k + \zeta_3^{-r} \overline{F_{m-k}} \right) & \text{if } 1 \le k \le m-1. \end{cases}$$

We note for $k > 0$ that $w_{k,r}$ obeys the Hermitian symmetry $w_{k,r} = \overline{w_{m-k,r}}$, so that the Fourier transform $\sum_{k=0}^{m-1} \zeta_m^{\ell k} w_{k,r}$ in Eq. (3.2) will indeed be real valued. This allows us to build a backward implicitly dealiased centered Hermitian transform using three complex-to-real Fourier transforms of the first $c+1$ components of $w_{k,r}$ (one for each $r \in \{-1, 0, 1\}$). The forward transform is given by $3mF_k = \sum_{r=-1}^{1} \zeta_{3m}^{-rk} \sum_{\ell=0}^{m-1} \zeta_m^{-\ell k} f_{3\ell+r}$ for $k = 0, \ldots, m-1$. Since $f_{3\ell+r}$ is real, a real-to-complex transform can be used to compute the first $c+1$ frequencies of $\sum_{\ell=0}^{m-1} \zeta_m^{-\ell k} f_{3\ell+r}$; the remaining $m-c-1$ frequencies are then computed using Hermitian symmetry.

3.2.2. Multithreaded Hermitian 1D Binary Convolution. The innermost operation of a recursive multidimensional convolution is a 1D convolution. An in-place implicitly padded Hermitian convolution was previously described in Ref. [2]. However inter-loop dependencies in the outermost loop prevented it from being multithreaded. Furthermore, to facilitate an in-place implementation, the transformed values for $r = 1$ were awkwardly stored in reverse order in the upper half of the input vector, exploiting the quadratic real-space multiplication operator. By unrolling the outer loop of the in-place Hermitian 1D convolution, these deficiencies of Function conv in Ref. [2] can be eliminated, resulting in a fully multithreaded implementation.

In the case of a binary convolution with two input vectors and one output vector, a fully in-place convolution requires a total of nine Hermitian Fourier transforms of size m, for an overall computational scaling of $\frac{9}{2}Km \log_2 m$ operations, in agreement with the leading-order scaling of an explicitly padded centered Hermitian convolution.

In Procedure `pretransform`, we unroll two iterations of the loop from Procedure `build` in Ref. [2] to read, process, and write the entries for the elements indexed by k, $m-k$, $c+1-k$, and $m-c-1+k$ simultaneously, for $k = 1, \ldots, \lceil c/2 \rceil$, as shown in Figure 4. The implicitly padded transformed data for remainders $r = 0$ and $r = 1$ is stored in the input data vector f, whereas the data for remainder $r = -1$ is stored in the auxiliary vector u. In the frequently encountered case where $m = 2c$, the values at position $c-1$ and c overlap for the remainders $r = 0$ and $r = 1$, whereas in the case when $m = 2c+1$ there is only one overlapping value, at index c. These overlapping values are handled with temporary storage registers R, S, F, and G in Function conv. In the noncompact case, any Nyquist inputs $f[m]$ and $g[m]$ are properly accounted for, but on output $f[m]$ is set to zero, for consistency with Hermitian symmetry. In the listed pseudocode, an asterisk $(*)$ denotes a pointwise multiply.

We see in Function `conv` that seven out of the nine Fourier transforms can be performed out of place using the same amount of memory ($6c+2$ words in the compact case) as required to compute a centered Hermitian convolution with explicit padding.

3.3. General Hermitian 1D Convolution. We now generalize the multithreaded 1D Hermitian convolution (Function conv) discussed in the previous section to handle a general convolution with A inputs and B outputs. Function conv computes the frequently encountered case of a binary convolution with one output, so that $A = 2$ and $B = 1$. In Ref. [2], we extended implicit dealiasing to the case $A = 2M$, where the multiplication operator is a dot product. For example, a pseudospectral simulation of the incompressible Navier–Stokes equation has four inputs and one output, with multiplication operator $(u_x, u_y, \partial\omega/\partial x, \partial\omega/\partial y) \to (u_x \partial\omega/\partial x + u_y \partial\omega/\partial y)$.

```
Input: vector f, vector g
Output: vector f
if $m = 2c$ then
    $\mathsf{F} \leftarrow \zeta_{3m}^{-1}\left[\mathsf{f}[1] + \overline{\left(-\frac{1}{2}, \frac{\sqrt{3}}{2}\right)\mathsf{f}[m-1]}\right]$
    $\mathsf{G} \leftarrow \zeta_{3m}^{-1}\left[\mathsf{g}[1] + \overline{\left(-\frac{1}{2}, \frac{\sqrt{3}}{2}\right)\mathsf{g}[m-1]}\right]$

pretransform(f,u)
pretransform(g,v)

u ← crfft(u)
v ← crfft(v)
v ← v * u
u ← rcfft(v)

if noncompact then
    $\mathsf{f}[0] \leftarrow \mathsf{f}[0] - 2\mathsf{f}[m]$
    $\mathsf{g}[0] \leftarrow \mathsf{g}[0] - 2\mathsf{g}[m]$
v ← crfft(f[$0, \dots, c$])
R ← Re f[0]
S ← Re g[0]
f[$0, \dots, c$] ← crfft(g[$0, \dots, c$])
v ← v * f[$0, \dots, c$]
f[$0, \dots, c$] ← rcfft(v)

f[$m-c-1$] ← R
g[$m-c-1$] ← S

if noncompact then
    $\mathsf{f}[m-c-1] \leftarrow \mathsf{f}[m-c-1] - \mathsf{f}[m]$
    $\mathsf{g}[m-c-1] \leftarrow \mathsf{g}[m-c-1] - \mathsf{g}[m]$
if $m = 2c$ then
    g[$c$] ← G
    f[$c$] ← F

v ← crfft(g[$m-c-1, \dots, m-1$])
g[$m-c-1, \dots, m-1$] ← crfft(f[$m-c-1, \dots, m-1$])
g[$m-c-1, \dots, m-1$] ← g[$m-c-1, \dots, m-1$] * v
v ← rcfft(g[$m-c-1, \dots, m-1$])

posttransform(f,v,u)

return f/(3m)
```

Function `conv` uses Procedures `pretransform` and `posttransform` to compute an in-place implicitly dealiased convolution of centered Hermitian vectors f and g of length m ($m+1$) in the compact (noncompact) format, using temporary vectors u and v of length $c+1$, where $c = \lfloor m/2 \rfloor$.

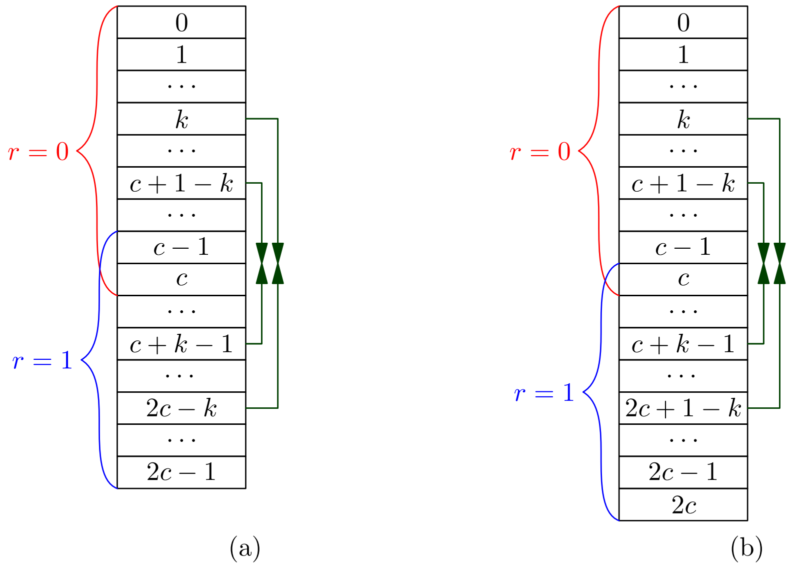


FIG. 4. *Loop unrolling for the Hermitian 1D convolution when (a) $m = 2c$ and (b) $m = 2c+1$.*

```
**Input**: vector $\mathsf{f}$
**Output**: vector $\mathsf{f}$, vector $\mathsf{u}$
**if** noncompact **then** $\mathsf{u}[0] \leftarrow \mathsf{f}[0] - \mathsf{f}[m]$
**else** $\mathsf{u}[0] \leftarrow \mathsf{f}[0]$
**if** $m = 2c$ **then** $\mathsf{F} \leftarrow \mathsf{f}[c]$
$d \leftarrow (c+1)/2$
**parallel for** $k = 1$ **to** $d$ **do**
    $\mathsf{a} \leftarrow \zeta_{3m}^{-k}\left[\operatorname{Re}\mathsf{f}[k] + \left(-\frac{1}{2}, \frac{\sqrt{3}}{2}\right)\operatorname{Re}\mathsf{f}[m-k]\right]$
    $\mathsf{b} \leftarrow -i\zeta_{3m}^{-k}\left[\operatorname{Im}\mathsf{f}[k] + \left(\frac{1}{2}, -\frac{\sqrt{3}}{2}\right)\operatorname{Im}\mathsf{f}[m-k]\right]$
    $\mathsf{A} \leftarrow \zeta_{3m}^{k-c-1}\left[\operatorname{Re}\mathsf{f}[c+1-k] + \left(-\frac{1}{2}, \frac{\sqrt{3}}{2}\right)\operatorname{Re}\mathsf{f}[m-c-1+k]\right]$
    $\mathsf{B} \leftarrow -i\zeta_{3m}^{k-c-1}\left[\operatorname{Im}\mathsf{f}[c+1-k] + \left(\frac{1}{2}, -\frac{\sqrt{3}}{2}\right)\operatorname{Im}\mathsf{f}[m-c-1+k]\right]$
    $\mathsf{u}[k] \leftarrow \mathsf{a} - \mathsf{b}$
    $\mathsf{u}[c+1-k] \leftarrow \mathsf{A} - \mathsf{B}$
    $\mathsf{f}[k] \leftarrow \mathsf{f}[k] + \overline{\mathsf{f}[m-k]}$
    $\mathsf{f}[c+1-k] \leftarrow \mathsf{f}[c+1-k] + \overline{\mathsf{f}[m-c-1+k]}$
    $\mathsf{f}[m-c-1+k] \leftarrow \overline{\mathsf{a}+\mathsf{b}}$
    $\mathsf{f}[m-k] \leftarrow \overline{\mathsf{A}+\mathsf{B}}$
**if** $m = 2c$ **then**
    $\mathsf{u}[c] \leftarrow \operatorname{Re}\mathsf{F} + \sqrt{3}\operatorname{Im}\mathsf{F}$
    $\mathsf{f}[c] \leftarrow 2\operatorname{Re}\mathsf{F}$
```

Procedure `pretransform(f,u)` prepares the arrays to be Fourier transformed in Function `conv` from an unpadded vector f of m $(m+1)$ values in the compact (noncompact) format, and an auxiliary vector u of length $c+1$, where $c = \lfloor m/2 \rfloor$. The Fourier origin is at position 0.

```
Input: vector $\mathsf{f}$, vector $\mathsf{v}$, vector $\mathsf{u}$
Output: vector $\mathsf{f}$
if noncompact then $\mathsf{f}[m] \leftarrow 0$
parallel for $k = 1$ to $(m-1)/2$ do
    $\mathsf{f}[m-k] \leftarrow \overline{\mathsf{f}[k]} + \left(-\frac{1}{2}, -\frac{\sqrt{3}}{2}\right)\zeta_{6c}^{k}\overline{\mathsf{v}[k]} + \left(-\frac{1}{2}, \frac{\sqrt{3}}{2}\right)\zeta_{6c}^{-k}\overline{\mathsf{u}[k]}$
    $\mathsf{f}[k] \leftarrow \mathsf{f}[k] + \zeta_{6c}^{-k}\mathsf{v}[k] + \zeta_{6c}^{k}\mathsf{u}[k]$
if $m = 2c$ then $\mathsf{f}[c] \leftarrow \mathsf{F} + \left(\frac{1}{2}, -\frac{\sqrt{3}}{2}\right)\mathsf{v}[c] + \left(\frac{1}{2}, \frac{\sqrt{3}}{2}\right)\mathsf{u}[c]$
```

Procedure `posttransform(f,v,u)` is called by Function `conv` to combine all three contributions into an implicitly-dealiased Hermitian convolution. The vector f has length m $(m+1)$ in the compact (noncompact) format, and the auxiliary vectors u and v have length $c+1$, where $c = \lfloor m/2 \rfloor$.

However, Function conv cannot be efficiently extended to the autoconvolution case with $A = B = 1$, nor to pseudospectral simulations of magnetohydrodynamic flows with $A = 12$ and $B = 6$ [9].

In order to compute general convolutions, we introduce Function `convin`, which takes A inputs and produces B outputs using a general quadratic multiplication operator `mult` $: \mathbb{R}^A \to \mathbb{R}^B$. In the pseudocode, the portions of the arrays corresponding to remainders $r = 0$ and $r = 1$ are shown in Figure 4 and distinguished by the superscripts 0 and 1. Since these portions overlap, additional code is required to save and restore the overlapping elements in the actual in-place implementation. For simplicity, the handling of Nyquist modes at wavenumber m is also ignored in the pseudocode.

While Function `convin` handles general inputs and outputs, all of the FFTs must be performed in place: in the case of, for example, an autoconvolution, there is no free buffer available that would enable us to use out-of-place transforms. The $3A + 3B$ FFTs in Function `convin` are therefore all in place. In Function `convout`, we present a convolution algorithm for the case $A \geq 2B$ that performs just A transforms in place and the remaining $2A + 3B$ FFTs out of place. Using one thread, Functions `conv`, `convin`, and `convout` have virtually identical performance. All three functions have fully parallelized pre- and post-processing phases and use `FFTW`'s built-in parallel FFTs, which are much more efficient when the transforms are out of place. Both Functions `convin` and `convout` can take either compact or noncompact inputs, and thus can benefit from the performance advantage discussed in Section 3.2 of the noncompact data format.

As seen in Fig. 5, the efficiency of the resulting implicitly dealiased centered Hermitian convolution is comparable to an explicit implementation. For each algorithm, we benchmark only those vector lengths that yield optimal performance. We tested the algorithm with one thread and four threads. With one thread, explicitly dealiased convolutions are faster for small problem sizes, but implicitly dealiased convolutions are faster on larger problem sizes since the problem can be divided into smaller pieces that fit better in the cache. On average, the implicit version is 10% faster than the explicit version with one thread for $m \geq 1024$ and 20% faster with four threads for $m \geq 4096$, as shown in Fig. 5(a). This performance difference between implicit and explicit versions remains unchanged when using four threads. We demonstrate the parallel efficiency of the implicit routine in Fig. 5(b) using one, two, and four threads. For $m \geq 8192$, the parallel efficiency of the implicit method is between 50% and 70% using four threads, giving a speedup of a factor of 2–2.8.

```
Input: vectors $\{\mathsf{f}_a\}_{a=0}^{A-1}$
Output: vectors $\{\mathsf{f}_b\}_{b=0}^{B-1}$

for $a = 0$ to $A-1$ do
    pretransform $(\mathsf{f}_a, \mathsf{u}_a)$
    $\mathsf{f}_a^0 \leftarrow \mathtt{fft}^{-1}(\mathsf{f}_a^0)$
    $\mathsf{f}_a^1 \leftarrow \mathtt{fft}^{-1}(\mathsf{f}_a^1)$
    $\mathsf{u}_a \leftarrow \mathtt{fft}^{-1}(\mathsf{u}_a)$

$\{\mathsf{f}_b^0\}_{b=0}^{B-1} \leftarrow \mathtt{mult}(\{\mathsf{f}_a^0\}_{a=0}^{A-1})$
$\{\mathsf{f}_b^1\}_{b=0}^{B-1} \leftarrow \mathtt{mult}(\{\mathsf{f}_a^1\}_{a=0}^{A-1})$
$\{\mathsf{u}_b\}_{b=0}^{B-1} \leftarrow \mathtt{mult}(\{\mathsf{u}_a\}_{a=0}^{A-1})$

for $b = 0$ to $B-1$ do
    $\mathsf{f}_b^0 \leftarrow \mathtt{fft}(\mathsf{f}_b^0)$
    $\mathsf{f}_b^1 \leftarrow \mathtt{fft}(\mathsf{f}_b^1)$
    $\mathsf{u}_b \leftarrow \mathtt{fft}(\mathsf{u}_b)$

    posttransform $(\mathsf{f}_b^0, \mathsf{f}_b^1, \mathsf{u}_b)$

return $\{\mathsf{f}_b/(3m)\}_{b=0}^{B-1}$
```

Function `convin` returns the implicitly dealiased 1D Hermitian convolution of length m (m+1) in the compact (noncompact) format, using the multiplication operator `mult` : $\mathbb{R}^A \to \mathbb{R}^B$. All $3A + 3B$ FFTs are in place.

```
Input: vectors $\{\mathsf{f}_a\}_{a=0}^{A-1}$
Output: vectors $\{\mathsf{f}_b\}_{b=0}^{B-1}$

for $a = 0$ to $A-1$ do
    pretransform $(\mathsf{f}_a, \mathsf{u}_a)$

for $a = 0$ to $A-1$ do
    $\mathsf{u}_a \leftarrow \mathtt{fft}^{-1}(\mathsf{u}_a)$
$\{\mathsf{u}_b\}_{b=0}^{B-1} \leftarrow \mathtt{mult}(\{\mathsf{u}_a\}_{a=0}^{A-1})$

$\mathsf{u}_{A-1} \leftarrow \mathtt{fft}^{-1}(\mathsf{f}_{A-1}^0)$
for $a = A-2$ to $0$ do
    $\mathsf{f}_{a+1}^0 \leftarrow \mathtt{fft}^{-1}(\mathsf{f}_a^0)$
$\{\mathsf{f}_b^0\}_{b=0}^{B-1} \leftarrow \mathtt{mult}(\{\mathsf{f}_a^0\}_{a=0}^{A-1} \cup \{\mathsf{u}_{A-1}\})$

$\mathsf{u}_{A-1}^1 \leftarrow \mathtt{fft}^{-1}(\mathsf{f}_{A-1}^1)$
for $a = A-2$ to $0$ do
    $\mathsf{f}_{a+1}^1 \leftarrow \mathtt{fft}^{-1}(\mathsf{f}_a^1)$
$\{\mathsf{f}_b^1\}_{b=0}^{B-1} \leftarrow \mathtt{mult}(\{\mathsf{f}_a^1\}_{a=0}^{A-1} \cup \{\mathsf{u}_{A-1}\})$

for $b = 0$ to $B-1$ do
    $\mathsf{u}_b \leftarrow \mathtt{fft}(\mathsf{f}_{b+1}^1)$
    $\mathsf{f}_b^0 \leftarrow \mathtt{fft}(\mathsf{f}_{b+1}^0)$
    $\mathsf{u}_{b+B} \leftarrow \mathtt{fft}(\mathsf{u}_b)$

for $b = 0$ to $B-1$ do
    posttransform $(\mathsf{f}_b^0, \mathsf{u}_b, \mathsf{u}_{b+B})$

return $\{\mathsf{f}_b/(3m)\}_{b=0}^{B-1}$
```

Function `convout` returns the implicitly dealiased 1D Hermitian convolution of length m (m+1) in the compact (noncompact) format, for $A \geq 2B$, with A in-place and $2A + 3B$ out-of-place FFTs.

4. Higher-dimensional convolutions. A d-dimensional convolution can be computed by performing an FFT of size $m_1 \times \ldots \times m_d$, applying the appropriate multiplication on the transformed data, and then inverting the FFT. Equivalently, one can perform an inverse FFT in the first dimension, followed by m_1 convolutions of dimension $d-1$, and then an FFT in the first dimension. Using this decomposition, one can reuse the work buffer for each subconvolution, thereby reducing the total memory demand relative to the explicit 2D dealiasing requirement.This is illustrated for a two-dimensional, 1/2 padded convolution in Figure 6. The input buffer f is implicitly padded and inverse Fourier transformed in the x direction to produce the data shown in the long horizontal boxes. An implicitly padded inverse FFT is then performed in the y direction, column-by-column using a one-dimensional work buffer, to produce a single double column of the Fourier transformed image, depicted in yellow. The Fourier transformed columns for F and G are then multiplied pointwise and stored in the column for F. At this point, the y forward padded FFT can then be performed and result is stored back in the lower-half of the column, joining the

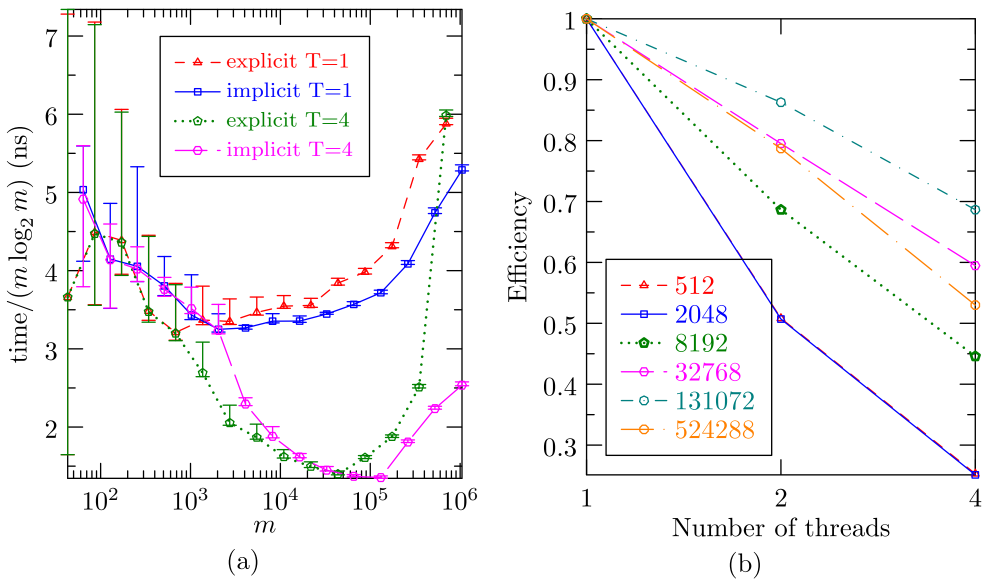


Fig. 5. *In-place 1D Hermitian convolutions of length m: (a) Comparison of computation times for explicit and implicit dealiasing using $T = 1$ thread and $T = 4$ threads; (b) parallel efficiency of implicit dealiasing versus number of threads for various values of m.*

previously processed data shown in red immediately to its left. The process is repeated on the remaining columns, shifting and reusing the yellow work buffer. Once all the columns have been processed, a forwards padded FFT in the x direction produces the final convolution in the left-hand half of the F buffer.

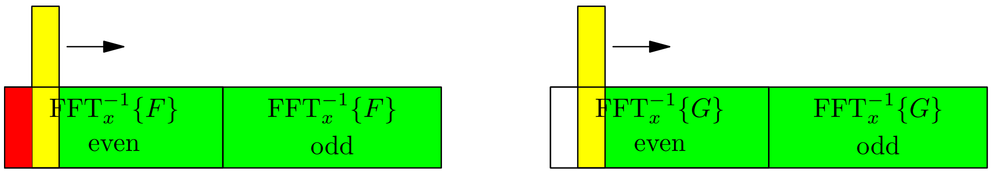

$\mathrm{FFT}_x^{-1}\{F\}$ even
$\mathrm{FFT}_x^{-1}\{F\}$ odd
$\mathrm{FFT}_x^{-1}\{G\}$ even
$\mathrm{FFT}_x^{-1}\{G\}$ odd

Fig. 6. *The reuse of memory in a 2D complex implicitly dealiased convolution: after applying a 1D y convolution to the double yellow column, the upper half is reused for the next column.*

The reuse of subconvolution work memory allows the convolution to be computed using less total memory: for 1/2 padded convolutions, the memory requirements per input are about twice the aliased version; this represents a memory savings of a factor of 2^{d-1} as compared to explicit padding; for 2/3 padded convolutions, the memory savings factor is $(3/2)^{d-1}$. In addition to having reduced memory requirements, implicitly dealiased multi-dimensional convolutions are significantly faster than their explicit counterparts, due to better data locality and cache management, along with the fact that transforms of data known to be zero are automatically avoided.

In the following subsections, we show that the algorithms developed in Section 3 can be used as building blocks to construct efficient implicitly padded higher-dimensional convolutions.

4.1. Complex 2D convolution. Pseudocode for the implicitly padded transforms described by Eqs. (1)–(2) was given in Ref. [2] as Procedures `fftpadBackward` and `fftpadForward`. In order to compute a 2D convolution in parallel, the loops in these procedures were parallelized, and parallel FFTs were used. Since the input and output of these routines are multi-dimensional and the required FFT is one-dimensional, we use the `FFTW` multiple 1D FFT routine. A parallel version of this routine is available in the `FFTW` library, but we found that the parallel performance was sometimes lacking. This was somewhat surprising, as there exists a simple algorithm to parallelize such problems: if one wishes to perform M FFTs using T threads, one can simply divide the M FFTs among the T threads, with any remaining r FFTs distributed among the first r threads. At run time, we test for the possibility that this decomposition is faster than `FFTW`'s parallel multiple FFT and use whichever algorithm runs faster. This yields a significant improvement in the parallel performance of our convolutions.

As shown in Fig. 7(a), the resulting implicit 2D algorithm dramatically outperforms the explicit version: using one thread, the mean speedup is a factor of 1.5, with a maximum speedup of 1.8. Using four threads, the mean speedup over the parallel explicit version is approximately 2.7, with a maximum speedup of more than a factor of four. Fig. 7(b) shows the parallel efficiency of the 2D implicitly dealiased complex convolution for a variety of problem sizes. The parallel efficiency for the implicit routine ranges from 60% to 90% with four threads, for a speedup of 2.4–3.6 relative to one thread. The explicit routine has a parallel efficiency between 45% and 90%. Notably, the 2D explicit version has poor parallel performance for problem sizes of 512^2 and above.

Because the same temporary arrays u and v are used for each column of the convolution, the memory requirement is $2Am_xm_y + TAm_y$ complex words using T threads, far less than the $4Am_xm_y$ complex words needed for an explicitly padded convolution, assuming that $T < 2m_x$.

4.2. Centered Hermitian 2D convolution. In two dimensions, the Fourier-centered Hermitian symmetry appears as $F_{-k,-\ell} = \overline{F_{k,\ell}}$. This symmetry is exploited in the centered Hermitian convolution algorithm shown for the noncompact case in Function conv2. As with the 1D Hermitian convolution, one has the option to use a compact or noncompact data format. For the compact data format, the array has dimensions $\{-m_x+1, \dots, m_x-1\} \times \{0, \dots, m_y-1\}$, whereas the noncompact version has dimensions $\{-m_x, \dots, m_x-1\} \times \{0, \dots, m_y\}$. One can also perform convolutions on data that is compact in one direction and noncompact in the other. For serial computations, the best performance typically is achieved when the x direction is compact and the y direction is noncompact, so that each dimension is odd, to reduce cache associativity issues. However, when running on more than one thread, the noncompact format should be used in the x direction since Procedures fft1padBackward and fft1padForward are fully multithreaded.

While the noncompact case requires slightly more memory than the compact case, one advantage of the noncompact version is that the output of the Fourier transform of $2m_x \times 2m_y$ real values corresponds to the modes $\{-m_x, \dots, m_x-1\} \times \{0, \dots, m_y\}$, where the Fourier origin has been shifted in the x direction to the middle of the array. Moreover, one is able to use the extra memory in the x direction for temporary storage, and having $m_y + 1$ variables in the y direction avoids latency issues with cache associativity when m_y is a power of two. These factors combine to give the noncompact format a performance advantage over the compact one: the noncompact

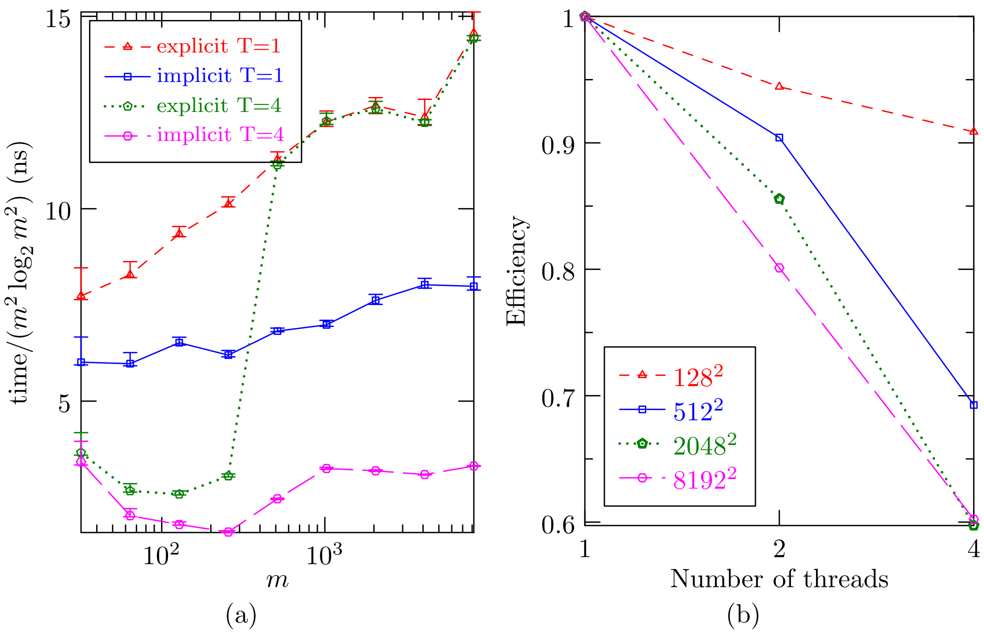


FIG. 7. *In-place 2D complex convolutions of size $m \times m$: (a) Comparison of computation times for explicit and implicit dealiasing using $T = 1$ thread and $T = 4$ threads; (b) parallel efficiency of implicit dealiasing versus number of threads for various values of m.*

case is 10% faster on average than the compact case when using one thread, and 30% faster on average when using four threads.

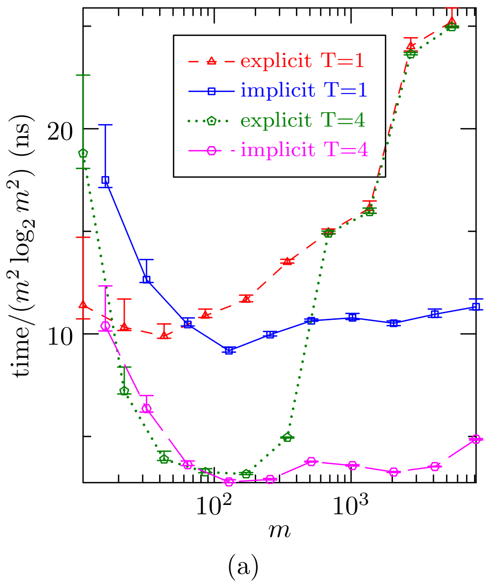
For the noncompact case, the explicit version requires storage for $9Am_x(m_y+1)/2$ complex words (assuming $A \geq B$). The implicit version using T threads requires storage for $3Am_x(m_y+1)+TA(\lfloor m_y/2 \rfloor+1)$ complex words, which is much less than the explicit case when $m_x \gg T$. As shown in Fig. 8(a), implicit padding again yields a dramatic improvement in speed. Note that the optimal problem size for the explicit version is $\lceil \frac{2}{3} 2^p \rceil$, whereas the implicit version has optimal problem size 2^p, so direct comparison of the two methods using optimal problem sizes is not possible. Instead, we compare the methods using a linear interpolation of the execution time rescaled by the computational complexity of the problem. Using this measure, the implicit version is on average 1.6 times faster than the explicit version when using one thread, and 4.3 times faster than the explicit version when using four threads. In Fig. 8(b) we show that the parallel efficiency of the implicit version is between 75% and 85% efficiency when using four threads, giving a speedup of a factor of 3–3.4. As in the 2D complex case, the explicit version does not parallelize well for large problem sizes.

```
Input: matrix $\{f_a\}_{a=0}^{A-1}$
Output: matrix $\{f_b\}_{b=0}^{B-1}$
for $a = 0$ to $A-1$ do
    parallel for $j = 0$ to $m_y - 1$ do
        fftpadBackward($f_a^T[j], U_a^T[j]$)
parallel for $i = 0$ to $m_x - 1$ do
    cconv($\{f_a[i]\}_{a=0}^{A-1}$)
    cconv($\{U_a[i]\}_{a=0}^{A-1}$)
for $b = 0$ to $B-1$ do
    parallel for $j = 0$ to $m_y - 1$ do
        fftpadForward($f_b^T[j], U_b^T[j]$)
return f
```

Function cconv2 returns the in-place implicitly dealiased convolution of $m_x \times m_y$ matrices $\{f_a\}_{a=0}^{A-1}$ in $\{f_b\}_{b=0}^{B-1}$, using A temporary $m_x \times m_y$ matrices $\{U_a\}_{a=0}^{A-1}$.

```
Input: matrix $\{f_a\}_{a=0}^{A-1}$
Output: matrix $\{f_b\}_{b=0}^{B-1}$
for $a = 0$ to $A-1$ do
    parallel for $j = 0$ to $m_y$ do
        fft1padBackward($f_a^T[j], U_a^T[j]$)
parallel for $i = 0$ to $2m_x - 1$ do
    conv($\{f_a[i]\}_{a=0}^{A-1}$)
parallel for $i = 0$ to $m_x - 1$ do
    conv($\{U_a[i]\}_{a=0}^{A-1}$)
for $b = 0$ to $B-1$ do
    parallel for $j = 0$ to $m_y$ do
        fft1padForward($f_b^T[j], U_b^T[j]$)
return f
```

Function conv2 returns the in-place implicitly dealiased centered Hermitian convolution of $2m_x \times (m_y+1)$ matrices $\{f_a\}_{a=0}^{A-1}$ in the noncompact data format, using A temporary $m_x \times (m_y+1)$ matrices $\{U_a\}_{a=0}^{A-1}$.



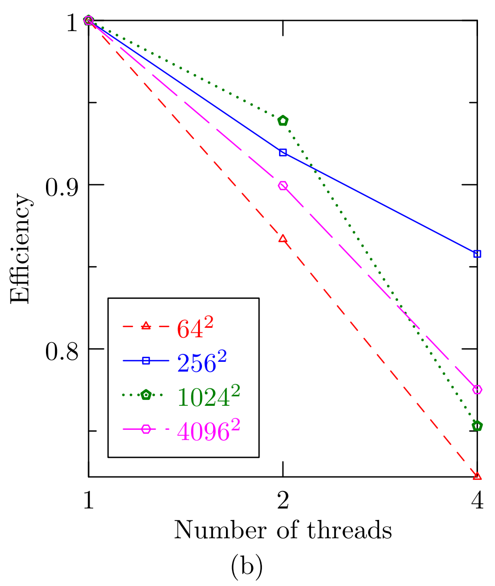


FIG. 8. *In-place 2D Hermitian convolutions of size $2m\times(m+1)$: (a) Comparison of computation times for explicit and implicit dealiasing using $T = 1$ thread and $T = 4$ threads; (b) parallel efficiency of implicit dealiasing versus number of threads for various values of m.*

4.3. Complex 3D convolution. The decoupling of the 2D work arrays in Function cconv2 facilitates the construction of an efficient 3D implicit complex convolution, as described in Function cconv3. For A inputs with dimensions $m_x \times m_y \times m_z$, the explicit version requires $8Am_xm_ym_z$ complex words, whereas the implicit version with T threads requires $2Am_xm_ym_z + TAm_ym_z + TAm_z$ complex words, which is approximately one quarter the storage requirements for the explicit version when $m_x \gg T$. As shown in Fig. 9(a) implicitly dealiased convolutions are much faster than their explicit counterparts. Using one thread, the implicit version is on average twice as fast as the explicit version and more than five times faster on average when comparing execution times over four threads. Fig. 9(b) shows the parallel efficiency of the implicit version, which is between 71% and 86% efficient when using four threads, giving a speedup of a factor of 2.84–3.44 over one thread. The explicit version has reasonable parallel efficiency for small problem sizes, but this drops to near 25% on four threads for problem size $m \geq 64$.

```
Input: ${\mathsf{f}_a}_{a=0}^{A-1}$
Output: ${\mathsf{f}_b}_{b=0}^{B-1}$
$\mathsf{R} \leftarrow \{0,\dots,m_y-1\} \times \{0,\dots,m_z-1\}$
for $a = 0$ to $A-1$ do
    parallel foreach $(j,k) \in R$ do
        fftpadBackward($\mathsf{f}_a^T[k][j], \mathsf{U}_a^T[k][j]$)
parallel for $i = 0$ to $m_x - 1$ do
    cconv2($\{\mathsf{f}_a[i]\}_{a=0}^{A-1}$)
    cconv2($\{\mathsf{U}_a[i]\}_{a=0}^{A-1}$)
for $b = 0$ to $B-1$ do
    parallel foreach $(j,k) \in R$ do
        fftpadForward($\mathsf{f}_b^T[k][j], \mathsf{U}_b^T[k][j]$)
return f
```

Function `cconv3` returns the in-place implicitly dealiased complex convolution of $m_x \times m_y \times m_z$ matrices $\{\mathsf{f_a}\}_{\mathsf{a}=0}^{\mathsf{A}-1}$, using A temporary $m_x \times m_y \times m_z$ matrices $\{\mathsf{U_a}\}_{\mathsf{a}=0}^{\mathsf{A}-1}$.

```
Input: $\{\mathsf{f}_a\}_{a=0}^{A-1}$
Output: $\{\mathsf{f}_b\}_{b=0}^{B-1}$
$\mathsf{R} \leftarrow \{0,\dots,2m_y\} \times \{0,\dots,m_z\}$
for $a = 0$ to $A-1$ do
    parallel foreach $(j,k) \in \mathsf{R}$ do
        fft1padBackward($\mathsf{f}_a^T[k][j], \mathsf{U}_a^T[k][j]$)
parallel for $i = 0$ to $2m_x - 1$ do
    conv2($\{\mathsf{f}_a[i]\}_{a=0}^{A-1}$)
parallel for $i = 0$ to $m_x - 1$ do
    conv2($\{\mathsf{U}_a[i]\}_{a=0}^{A-1}$)
for $b = 0$ to $B-1$ do
    parallel foreach $(j,k) \in R$ do
        fft1padForward($\mathsf{f}_b^T[k][j], \mathsf{U}_b^T[k][j]$)
return f
```

Function `conv3` returns the in-place implicitly dealiased Hermitian convolution of $2m_x \times 2m_y \times (m_z+1)$ matrices $\{\mathsf{f_a}\}_{\mathsf{a}=0}^{\mathsf{A}-1}$, using A temporary $m_x \times m_y \times (m_z+1)$ matrices $\{\mathsf{U_a}\}_{\mathsf{a}=0}^{\mathsf{A}-1}$.

4.4. Centered Hermitian 3D convolution. As with the 1D and 2D cases, we offer compact and noncompact versions of the 3D Hermitian convolution, and users can choose formats that are compact/noncompact in each direction separately. For serial computations, the best performance typically is achieved when the x and y directions are compact and the z direction is noncompact, so that each dimensions is odd, in the interest of cache associativity. However, just as for 2D Hermitian convolutions, when running on more than one thread, the x direction should be made noncompact to obtain optimal multithreading efficiency.

Pseudocode for the noncompact algorithm is given in Function conv3. The noncompact version again offers a performance advantage over the compact version, with the single-threaded compact and noncompact cases roughly equal in execution time when compared using one thread, and the noncompact case offering between a 1% and 10% performance advantage when compared using four threads.

In the noncompact format, the memory requirements for an explicit 3D Hermitian convolution with A inputs is $\frac{27}{2}Am_xm_y(m_z+1)$ complex words, whereas the implicit version requires only $6Am_xm_y(m_z+1) + TAm_y(m_z+1) + TA(\lfloor m_z/2\rfloor + 1)$ complex words using T threads. We did not implement a high-performance version of the explicit routine, so instead we show the execution time of the implicit routine using one and four threads in Fig. 10 (a). The parallel efficiency is shown in Fig. 10(b) and ranges between 70% and 90%, which translates to a speedup of a factor of 2.8–3.6

using four threads instead of one.

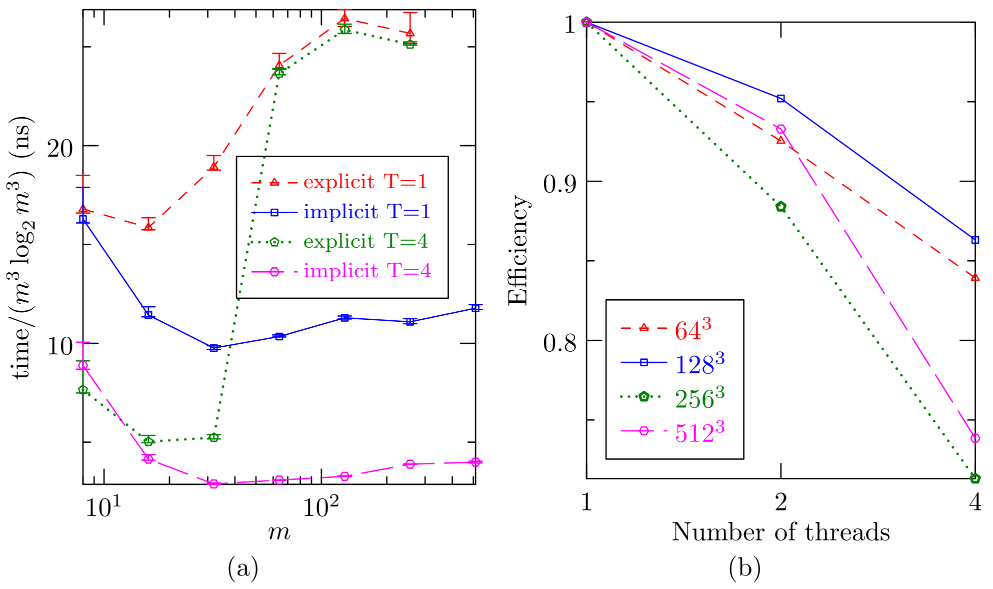


FIG. 9. *In-place 3D complex convolutions of size $m \times m \times m$: (a) Comparison of computation times for explicit and implicit dealiasing using $T = 1$ thread and $T = 4$ threads; (b) parallel efficiency of implicit dealiasing versus number of threads for various values of m.*

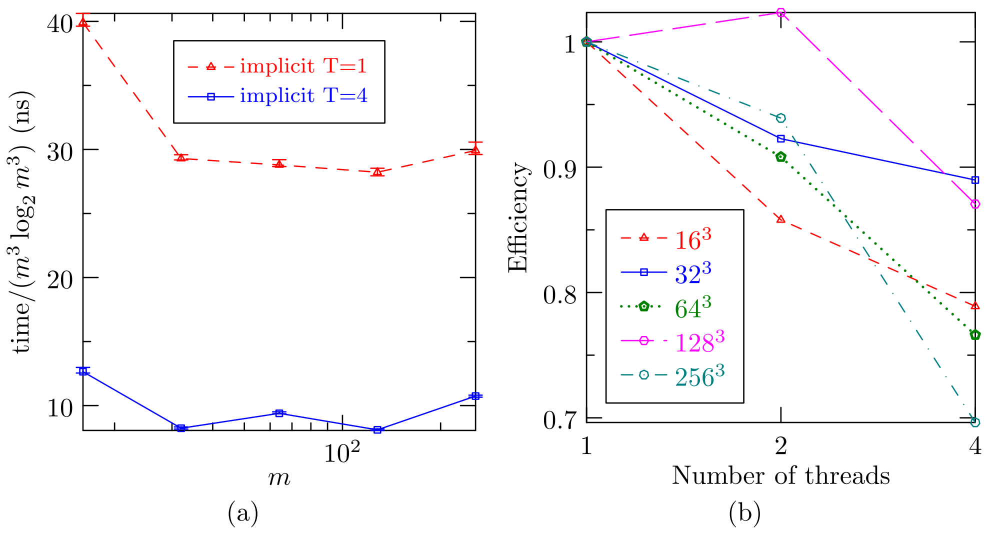


FIG. 10. *Implicitly dealiased in-place 3D Hermitian convolutions of size $2m \times 2m \times (m + 1)$: (a) computation times using $T = 1$ thread and $T = 4$ threads; (b) parallel efficiency versus number of threads for various values of m.*

5. Concluding remarks. In this work we developed an efficient method for computing implicitly dealiased convolutions parallelized over multiple threads. Methods were developed for noncentered complex data and centered Hermitian-symmetric data with inputs in one, two, and three dimensions. We showed how more general multiplication operators can be supported, to allow for the efficient computation of autoconvolutions, correlations (which are identical to convolutions for Hermitian-symmetric data), and general nonlinearities in pseudospectral simulations.

Implicitly dealiased convolutions require less memory, are faster, and have greater parallel efficiency than their explicitly dealiased counterparts. The decoupling of temporary storage and user data means that even in one dimension, users can save memory by not having to copy their data to a separate buffer. In higher dimensions, this decoupling allows one to reuse work memory. By avoiding the need to compute the entire Fourier image at once, one obtains a dramatic reduction in total memory use. Moreover, the resulting increased data locality significantly enhances performance, particularly under parallelization. For example, a 3D implicitly dealiased complex convolution runs about twice as fast as an explicitly dealiased convolution on one thread, and over five times faster on four threads. For large problem sizes, an implicit complex convolution requires one-half of the memory needed for a zero-padded convolution in two dimensions and one-quarter in three dimensions. In the centered Hermitian case, the memory use in two dimensions is $2/3$ of the amount used for an explicit convolution and $4/9$ of the corresponding storage requirement in three dimensions.

An upcoming paper will discuss the implementation of implicit dealiasing on distributed-memory architectures, using hybrid `MPI/OpenMP`. Implicit dealiasing of higher-dimensional convolutions over distributed memory benefits significantly from the reduction of communication costs associated with the smaller memory footprint. It also provides a natural way of overlapping communication (during the transpose phase) with FFT computation. We also plan to implement implicit dealiasing on graphics processor units, either using the `OpenCL` programming language or the recently released `Vulkan` application programming interface.

In future work, we wish to develop specialized implicit convolutions of real data for applications in signal processing, such as computing cross correlations and autocorrelations. We are also exploring novel applications of implicitly dealiased convolutions for computing sparse Fourier transforms [7], fractional phase Fourier (chirp-z) transforms [1], and partial Fourier transforms [10].

REFERENCES

[1] D. H. Bailey and P. N. Swarztrauber, *The fractional Fourier transform and applications*, SIAM review, 33 (1991), pp. 389–404.

[2] J. C. Bowman and M. Roberts, *Efficient dealiased convolutions without padding*, SIAM J. Sci. Comput., 33 (2011), pp. 386–406.

[3] ———, `FFTW++`: *A fast Fourier transform* C^{++} *header class for the* `FFTW3` *library*. http://fftwpp.sourceforge.net, May 6, 2010.

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

[5] C. F. Gauss, *Nachlass: Theoria interpolationis methodo nova tractata*, in Carl Friedrich Gauss Werke, vol. 3, Königliche Gesellschaft der Wissenschaften, Göttingen, 1866, pp. 265–327.

[6] D. Gottlieb and S. A. Orszag, *Numerical Analysis of Spectral Methods: Theory and Applications*, Society for Industrial and Applied Mathematics, Philadelphia, 1977.

[7] H. Hassanieh, P. Indyk, D. Katabi, and E. Price, *Simple and practical algorithm for sparse Fourier transform*, in Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2012, pp. 1183–1194.

[8] S. A. Orszag, *Elimination of aliasing in finite-difference schemes by filtering high-wavenumber components*, Journal of the Atmospheric Sciences, 28 (1971), p. 1074.

[9] M. Roberts, M. Leroy, J. Morales, W. Bos, and K. Schneider, *Self-organization of he-*

lically forced MHD flow in confined cylindrical geometries, Fluid Dynamics Research, 46 (2014), p. 061422.

[10] L. Ying and S. Fomel, *Fast computation of partial Fourier transforms*, Multiscale Modeling and Simulation, 8 (2009), pp. 110–124.