Journal of Scientific Computing manuscript No.
(will be inserted by the editor)

Parallel Implicitly Dealiased Convolutions on Shared Memory Architectures

Malcolm Roberts · John C. Bowman

Received: date / Accepted: date

Abstract Implicit dealiasing is a method for computing in-place linear convolutions via fast Fourier transforms that decouples work memory from input data. It offers easier memory management and, for long one-dimensional input sequences, greater efficiency than conventional zero-padding. Furthermore, for convolutions of multidimensional data, the segregation of data and work buffers can be exploited to reduce memory usage and execution time. This is accomplished by processing and discarding data as it is generated, allowing work memory to be reused, for greater data locality and performance. A generalized multithreaded implementation of implicit dealiasing that accepts an arbitrary number of input and output vectors is presented, along with an improved one-dimensional 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.

Keywords implicit dealiasing · zero padding · convolution · fast Fourier transform · pseudospectral method · Hermitian symmetry · Nyquist mode · multithreading · parallelization · shared memory · correlation

Mathematics Subject Classification (2000) 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 two

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

Malcolm Roberts
Computer Modelling Group Ltd, 3710 33 Street NW, Calgary, Alberta, T2L 2M1 Canada.
E-mail: malcolm.i.w.roberts@gmail.com

John C. Bowman
Department of Mathematical and Statistical Sciences, University of Alberta, Edmonton, Alberta T6G 2G1, Canada. E-mail: bowman@ualberta.ca

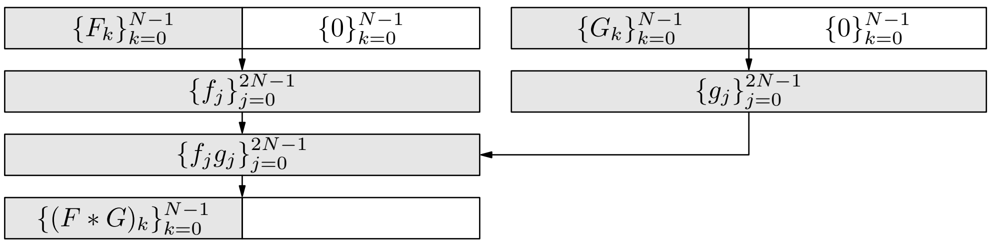


Figure 1 Computing a 1D convolution via explicit zero padding.

sequences $\{F_k\}_{k\in\mathbb{Z}}$ and $\{G_k\}_{k\in\mathbb{Z}}$ is $\sum_{p\in\mathbb{Z}} F_pG_{k-p}$. In practical applications, the inputs $\{F_k\}_{k=0}^{m-1}$ and $\{G_k\}_{k=0}^{m-1}$ are of finite length m, yielding a linear convolution with components $\sum_{p=0}^{k} F_pG_{k-p}$ for $k = 0,\ldots,m-1$. Computing such a convolution directly requires $\mathcal{O}(m^2)$ operations, and roundoff error is a significant problem for large m. It is therefore preferable to make use of the convolution theorem, harnessing the power of the fast Fourier transform (FFT) to map the convolution to a component-wise multiplication. This reduces the computational complexity of a convolution to $\mathcal{O}(m\log m)$ [6,7] while improving numerical accuracy [8].

Since the FFT considers the inputs 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 generalize the method of implicit dealiasing [4] to handle an arbitrary number of input and output vectors, with a general spatial multiplication operator. This allows implicit dealiasing to be efficiently applied to 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. For an efficient in-place implementation of the centered Hermitian convolution, it was necessary to unroll the outer loop partially so that interacting wavenumbers can be simultaneously processed. This loop unrolling offers another advantageous: it removes the loop interdependence that prevented Function `conv` in [4] from being fully parallelized. In Section 4 we demonstrate that multithreaded implicit dealiasing in dimensions greater than one uses much less memory and is much faster than explicit dealiasing. The accomplishments of this work and future directions for research are summarized in Section 5. Implicitly dealiased convolution routines are publicly available in the open-source software library `FFTW++` [5].

2 Dealiasing requirements for convolutions

To compute the standard linear convolution $\sum_{p=0}^{k} F_pG_{k-p}$ for input with $k \in \{0,\ldots,m-1\}$, the data is padded with m zeroes for a total FFT length of $2m$.

We refer to these inputs as *non-centered* and the paddings 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.

For pseudospectral simulations, it is convenient to shift the zero wavenumber in the transformed data to the middle of the array. In this case, the inputs are $\{F_k\}_{k=-m+1}^{m-1}$ and $\{G_k\}_{k=-m+1}^{m-1}$, which we refer to as *centered*, and their convolution has components $\sum_{p=k-m+1}^{m-1} F_p G_{k-p}$ for $k = -m+1, \ldots, m-1$. Convolutions on centered inputs require less padding than on non-centered inputs: data of length $2m-1$ needs to be padded only to length $3m-2$ (normally extended to $3m$); this is called 2/3 *padding* [10]. Explicit zero padding increases the d-dimensional buffer size in this case by a factor of $(3/2)^d$.

A binary convolution can be generalized to an n-ary operator $*(F_1, \ldots, F_n)_k = \sum_{p_1,\ldots,p_n} F_{p_1} \cdots F_{p_n} \delta_{p_1+\ldots+p_n,k}$, where δ is the Kronecker delta. For non-centered inputs, an n-ary convolution could be computed as a sequence of binary convolutions using 1/2 padding. However, for centered inputs with both negative and positive frequencies, each binary convolution would have to be padded further to eliminate all aliased interactions [11]. As a result, n-ary convolutions benefit greatly from implicit dealiasing [4].

We consider a generalized 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.

The choice of multiplication operator determines the type of convolution. Let $\{f_j\}$ be the inverse Fourier transform of $\{F_k\}$. An autoconvolution can be computed with just two transforms using $A = B = 1$ and the operation $f_j \to f_j^2$, while an autocorrelation would use $f_j \to f_j \overline{f_j}$, where $\overline{f_j}$ denotes the complex conjugate of f_j. For the standard binary convolution, there are two inputs and one output, and the multiplication operation is $(f_j, g_j) \to f_j g_j$.

The nonlinear advective term of the 2D incompressible Navier–Stokes vorticity equation can be computed with the operation $(u_x, u_y, \partial\omega/\partial x, \partial\omega/\partial y) \to (u_x \partial\omega/\partial x + u_y \partial\omega/\partial y)$, where $\boldsymbol{u} = (u_x, u_y)$ is the 2D velocity and $\omega = \hat{\boldsymbol{z}} \cdot \boldsymbol{\nabla} \times \boldsymbol{u}$ is the z-component of the vorticity; this requires a total of five FFTs ($A = 4$ and $B = 1$). As shown in Appendix A, it is possible to reduce the FFT count for this case to four, with $A = B = 2$. Similarly, in three dimensions, Basdevant [2] showed that the number of FFTs can be reduced from nine to eight, with $A = 3$ and $B = 5$. For 3D 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 velocity, $\boldsymbol{\omega} = \boldsymbol{\nabla} \times \boldsymbol{u}$ is the vorticity, $\boldsymbol{B}$ is the magnetic field, and $\boldsymbol{j}$ is the current density ($A = 12$, $B = 6$) [12]. For the Navier–Stokes and MHD equations, the operation is quadratic and the convolution is binary ($n = 2$), with a padding ratio of 2/3 (since the Fourier modes are symmetric about the origin). In these pseudospectral applications, the physical space quantities are real valued: one can therefore use real-to-complex Fourier transforms, which are about twice as efficient as their complex counterparts.

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 only slightly more efficient than explicit zero padding on a single thread, it still has the advantage of not requiring the copying of user data to a separate enlarged zero-padded buffer before performing the FFT. We 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 efficient and compact than their explicit counterparts.

3.1 Complex convolution

Dealiasing the standard convolution $\sum_{p=0}^{k} F_p G_{k-p}$ for $k = 0, \dots, m-1$ requires extending the input data with zeros from length m to length $N \geq 2m-1$, thus removing the beating of two modes with wavenumber $m-1$ that would otherwise contaminate mode $N = 0 \bmod N$. One generally chooses $N = 2m$ so that N has a large number of prime factors, resulting in improved FFT performance.

The backward Fourier transform $\{f_j\}_{j=0}^{N-1}$ of the zero-padded input vector $\{F_k\}_{k=0}^{N-1}$ has components $f_j = \sum_{k=0}^{N-1} \zeta_N^{jk} F_k$, where $\zeta_N = \exp(2\pi i/N)$ denotes the N^{th} root of unity. The divide-and-conquer strategy of the fast Fourier transform is based on the property $\zeta_N^r = \zeta_{N/r}$. Since $F_k = 0$ for $k \geq m$, we can compute the even- and odd-indexed terms of $\{f_j\}_{j=0}^{N-1}$ as separate subtransforms:

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

where $\ell = 0, \dots, m-1$. That is, $\{f_j\}_{j=0}^{N-1}$ can be computed with two Fourier transforms of length m depending on the input data $\{F_k\}_{k=0}^{m-1}$, with the even-indexed and odd-indexed parts of the output stored separately. Equation (1) has a (slightly improved) computational complexity of $\mathcal{O}(N \log m)$, while avoiding the outermost bit reversal stage and the inconvenience of explicitly appending m extra zero values to the input data. The (scaled) inverse of Eq. (1) is given by the forward transform

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

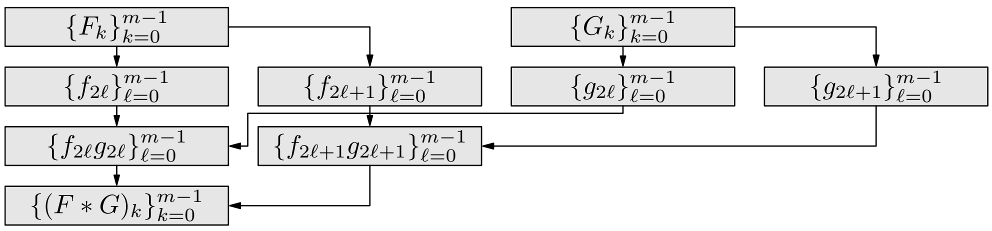

$\{F_k\}_{k=0}^{m-1}$
$\{G_k\}_{k=0}^{m-1}$
$\{f_{2\ell}\}_{\ell=0}^{m-1}$
$\{f_{2\ell+1}\}_{\ell=0}^{m-1}$
$\{g_{2\ell}\}_{\ell=0}^{m-1}$
$\{g_{2\ell+1}\}_{\ell=0}^{m-1}$
$\{f_{2\ell}g_{2\ell}\}_{\ell=0}^{m-1}$
$\{f_{2\ell+1}g_{2\ell+1}\}_{\ell=0}^{m-1}$
$\{(F*G)_k\}_{k=0}^{m-1}$

Figure 2 Computing a 1D convolution via implicit dealiasing.

again using two Fourier transforms of length m. Equations (1) and (2) can be combined to compute a dealiased binary convolution of $\{F_k\}_{k=0}^{m-1}$ and $\{G_k\}_{k=0}^{m-1}$, as shown in Figure 2 and implemented in pseudocode in Function `cconv` of Ref. [4]. For each input, two arrays of size m are used instead of one array of size $2m$. This distinction is the key to the improved efficiency and reduced storage requirements of the higher-dimensional implicit convolutions described in Section 4. In the 1D complex case, each of the six complex Fourier subtransforms of size m can 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. This can be used for generating an image in x space of the inverse transformed data.

As in our previous work [4], 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 $s = \lfloor\sqrt{m}\rfloor$, $a = 0, 1, \ldots, \lceil m/s\rceil - 1$, and $b = 0, 1, \ldots, s - 1$. Since these one-dimensional tables occupy only $\mathcal{O}(\sqrt{m})$ complex words, we do not account for them in our storage estimates.

Out-of-place FFTs are often more efficient than their in-place counterparts, and are more amenable to multithreading. 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 `cconv` of Ref. [4], all FFTs are out of place. Our general function `cconv` in this work has $A + B - 1$ out-of-place FFTs and $A + B + 1$ in-place FFTs.

For $A > B$, the multiplication operator will free buffers that can 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 of the A work buffers is no longer needed. We make use of this buffer to implement out-of-place transforms. In Function `cconvA`, we present an algorithm for an implicitly dealiased convolution with $A > B$ in which all $2A + 2B$ FFTs of length m are out of place. Likewise, when $A < B$, Function `cconvB` shows that all but one of the $2A + 2B$ FFTs can be performed out-of-place.

When $A = 2$ and $B = 1$, Function `cconvA` runs a few percent faster than Function `cconv` from Ref. [4], thanks to improvements in the loop structure in the pre- and post-processing stages. Thus, in one dimension, as seen in Figure 3(a), implicit dealiasing on a single thread is now on average 12% faster than explicit zero padding.

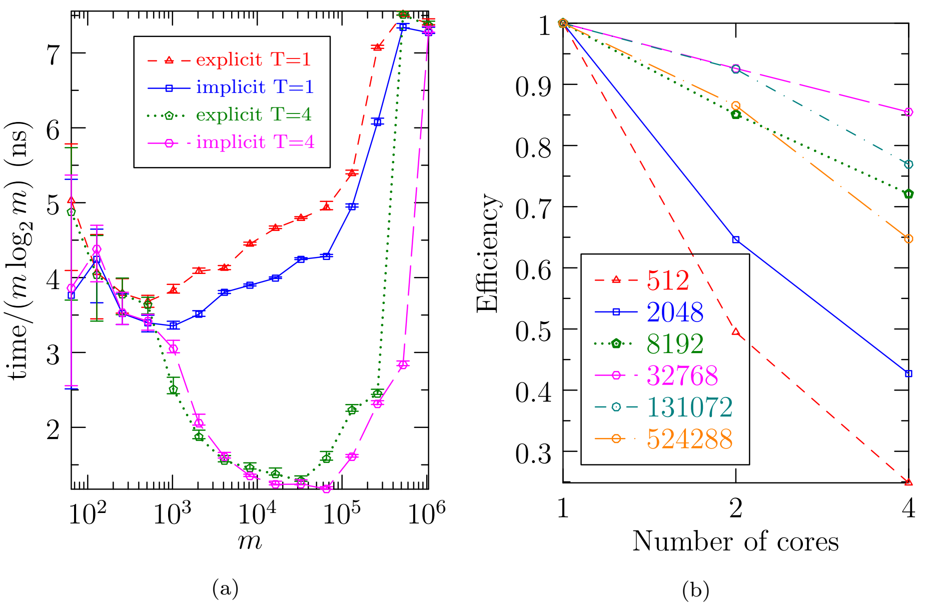


Figure 3 In-place 1D complex convolutions of length m: (a) comparison of computation times for explicit and implicit dealiasing using $T = 1$ thread and $T = 4$ threads; (b) parallel efficiency of implicit dealiasing versus number of threads. For efficiency m is chosen to be a power of two.

3.1.1 Multithreaded Complex 1D Binary Convolutions

We parallelize Functions `cconv`, `cconvA`, and `cconvB` 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 one and four threads. Using one thread, the implicit method is on average 1.12 times faster than the explicit method, whereas using four threads the performance improvement is a factor of 1.04 to 2.6 for $m \geq 8192$. The reason that the explicit version benefits from parellization at smaller m values than implicit dealiasing is a simple consequence of the fact that the vector sizes for explicit dealiasing are twice as large, due to the memory wasted on padding. The error bars in the timing figures indicate the lower and upper one-sided standard deviations, as given in Ref. [4]. The number of samples varied from several million for small data sizes to 20 for larger data sizes. In Figure 3(b), we observe for $m = 2048$ to 524288 that the implicit method with four threads has a parallel efficiency of 43% to 85%, giving a speedup of a factor of 1.7 to 3.4.

Both the `FFTW-3.3.6` library and the convolution layer we built on top of it were compiled with the GCC 5.3.1 20160406 compiler. Our library was compiled with the optimizations `-fopenmp -fomit-frame-pointer -fstrict-aliasing -msse2 -ffast-math -mfpmath=sse -march=native` and executed on a 64-bit 3.4GHz Intel i7-2600K processor with an 8MB cache. Like the `FFTW` library, our algorithms were vectorized with specialized `SSE2` single-instruction multiple-data code.

3.2 Centered data formats

In this work, we extend the treatment of centered Fourier input data $\{F_{-m+1},\ldots,F_{m-1}\}$ for even m from Ref. [4], to all natural numbers m. We also implement an optional new data layout $\{F_{-m},\ldots,F_{m-1}\}$. In addition to handling convolutions of Fourier transformed real-space data of even length, this extended format can yield significant performance improvements, even if the additional mode F_{-m} is simply set to zero. We refer to $\{F_{-m+1},\ldots,F_{m-1}\}$ as the *compact* format and $\{F_{-m},\ldots,F_{m-1}\}$ as the *noncompact* format. In particular, in the noncompact case, a fully multithreaded implementation (Procedure fft1padBackward) is possible since it doesn't have the loop dependency seen in Procedure fft0padBackward.[1] The noncompact format is consistent with the output of a real-to-complex FFT.

```
**Input:** vectors $\{\mathtt{f}_a\}_{a=0}^{A-1}$
**Output:** vectors $\{\mathtt{f}_b\}_{b=0}^{B-1}$
**for** $a=0$ **to** $A-1$ **do**
    $\mathtt{u}_a \leftarrow \mathtt{fft}^{-1}(\mathtt{f}_a)$
$\{\mathtt{u}_b\}_{b=0}^{B-1} \leftarrow \mathtt{mult}(\{\mathtt{u}_a\}_{a=0}^{A-1})$
**parallel for** $k=0$ **to** $m-1$ **do**
    **for** $a=0$ **to** $A-1$ **do**
        $\mathtt{f}_a[k] \leftarrow \zeta_{2m}^{k}\mathtt{f}_a[k]$
**for** $a=0$ **to** $A-1$ **do**
    $\mathtt{f}_a \leftarrow \mathtt{fft}^{-1}(\mathtt{f}_a)$
$\{\mathtt{f}_0\}_{b=0}^{B-1} \leftarrow \mathtt{mult}(\{\mathtt{f}_a\}_{a=0}^{A-1})$
$\mathtt{f}_0 \leftarrow \mathtt{fft}(\mathtt{f}_0)$
$\mathtt{u}_0 \leftarrow \mathtt{fft}(\mathtt{u}_0)$
**parallel for** $k=0$ **to** $m-1$ **do**
    $\mathtt{f}_0[k] \leftarrow \mathtt{f}_0[k] + \zeta_{2m}^{-k}\mathtt{u}_0[k]$
**for** $b=1$ **to** $B-1$ **do**
    $\mathtt{f}_b \leftarrow \mathtt{fft}(\mathtt{f}_b)$
    $\mathtt{u}_0 \leftarrow \mathtt{fft}(\mathtt{u}_b)$
    **parallel for** $k=0$ **to** $m-1$ **do**
        $\mathtt{f}_b[k] \leftarrow \mathtt{f}_b[k] + \zeta_{2m}^{-k}\mathtt{u}_0[k]$
**return** $\{\mathtt{f}_b/(2m)\}_{b=0}^{B-1}$
```

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

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

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

Although the compact 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

[1] We correct here a sequencing error in the pseudocode for Procedure fft0padBackward in Ref. [4]. Minor typographical errors also appeared on page 388 ($0\ldots N$ should be $0\ldots N-1$), page 391 (n should be N), and on p. 400 ($2m_1-1$ should be $2m_z-1$).

```
Input: vectors $\{\mathsf{f}_a\}_{a=0}^{A-1}$
Output: vectors $\{\mathsf{f}_b\}_{b=0}^{B-1}$
for $a = A-1$ to $0$ do
    $\mathsf{u}_a \leftarrow \mathtt{fft}^{-1}(\mathsf{f}_a)$
parallel for $k = 0$ to $m-1$ do
    for $a = A-1$ to $0$ do
        $\mathsf{f}_a[k] \leftarrow \zeta_{2m}^{k}\mathsf{f}_a[k]$
for $a = A-1$ to $0$ do
    $\mathsf{f}_{a+1} \leftarrow \mathtt{fft}^{-1}(\mathsf{f}_a)$
$\{\mathsf{f}_b\}_{b=1}^{B-1} \cup \{\mathsf{u}_{B-1}\} \leftarrow \mathtt{mult}(\{\mathsf{f}_a\}_{a=1}^{A})$
for $b = 0$ to $B-2$ do
    $\mathsf{f}_b \leftarrow \mathtt{fft}(\mathsf{f}_{b+1})$
$\mathsf{f}_{B-1} \leftarrow \mathtt{fft}(\mathsf{u}_{B-1})$
$\{\mathsf{u}_b\}_{b=0}^{B-1} \leftarrow \mathtt{mult}(\{\mathsf{u}_a\}_{a=0}^{A-1})$
$\mathsf{u}_0 \leftarrow \mathtt{fft}(\mathsf{u}_0)$
parallel for $k = 0$ to $m-1$ do
    $\mathsf{f}_0[k] \leftarrow \mathsf{f}_0[k] + \zeta_{2m}^{-k}\mathsf{u}_0[k]$
for $b = 1$ to $B-1$ do
    $\mathsf{u}_0 \leftarrow \mathtt{fft}(\mathsf{u}_b)$
    parallel for $k = 0$ to $m-1$ do
        $\mathsf{f}_b[k] \leftarrow \mathsf{f}_b[k] + \zeta_{2m}^{-k}\mathsf{u}_0[k]$
return $\{\mathsf{f}_b/(2m)\}_{b=0}^{B-1}$
```

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

```
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
    $\mathsf{A} \leftarrow$
        $\zeta_{3m}^{k}\left[\operatorname{Re}\mathsf{f}[m-1+k] + \left(-\frac{1}{2}, -\frac{\sqrt{3}}{2}\right)\operatorname{Re}\mathsf{f}[0]\right]$
    $\mathsf{B} \leftarrow i\zeta_{3m}^{k}\left[\operatorname{Im}\mathsf{f}[m-1+k] +\right.$
        $\left.\left(-\frac{1}{2}, -\frac{\sqrt{3}}{2}\right)\operatorname{Im}\mathsf{f}[0]\right]$
    $\mathsf{C} \leftarrow \mathsf{f}[m-1+k] + \mathsf{f}[0]$
    $\mathsf{f}[0] \leftarrow \mathsf{f}[k]$
    $\mathsf{f}[k] \leftarrow \mathsf{C}$
    $\mathsf{f}[m-1+k] \leftarrow \mathsf{A} + \mathsf{B}$
    $\mathsf{u}[k] \leftarrow \overline{\mathsf{A} - \mathsf{B}}$
$\mathsf{f}[0,\ldots,m-1] \leftarrow \mathtt{fft}^{-1}(\mathsf{f}[0,\ldots,m-1])$
$\mathsf{u}[m] \leftarrow \mathsf{f}[m-1]$
$\mathsf{f}[m-1] \leftarrow \mathsf{u}[0]$
$\mathsf{f}[m-1,\ldots,2m-2] \leftarrow$
  $\mathtt{fft}^{-1}(\mathsf{f}[m-1,\ldots,2m-2])$
$\mathsf{u}[0,\ldots,m-1] \leftarrow \mathtt{fft}^{-1}(\mathsf{u}[0,\ldots,m-1])$
```

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 $m+1$.

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.

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 [10]. With explicit padding, for efficiency reasons one normally chooses the padded vector length N to be a power of 2, with $m = \lfloor (N+2)/3 \rfloor$, while for implicit padding, it is advantageous to choose the subtransform length m to be a power of 2. Moreover, it is convenient to pad implicitly slightly beyond $3m-2$, to $N = 3m$, to support a radix 3 subdivision at the outer level.

In the case of an even number $2m-2$ 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 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

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

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

$$w_{k,r}\doteq\begin{cases}F_0+\operatorname{Re}\zeta_3^{-r}F_{-m} & \text{if } k=0,\\ \zeta_{3m}^{rk}(F_k+\zeta_3^{-r}F_{k-m}) & \text{if } 1\le k\le m-1.\end{cases} \tag{4}$$

Here $\doteq$ is used to emphasize a definition. The forward transform is then

$$3mF_k=\sum_{r=-1}^{1}\zeta_{3m}^{-rk}\sum_{\ell=0}^{m-1}\zeta_m^{-\ell k}f_{3\ell+r},\qquad k=-m+1,\ldots,m-1, \tag{5}$$

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. (4) and (5). The number of complex multiplies needed to evaluate Eq. (4) for $r=\pm1$ can be reduced by computing the intermediate complex quantities $A_k\doteq\zeta_{3m}^{k}\big(\operatorname{Re}F_k+\zeta_3^{-1}\operatorname{Re}F_{k-m}\big)$ and $B_k\doteq i\zeta_{3m}^{k}\big(\operatorname{Im}F_k+\zeta_3^{-1}\operatorname{Im}F_{k-m}\big)$, 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 noncompact backward transform is given in Procedure `fft1padBackward`; its inverse is given in Procedure `fft1padForward`. The compact versions of these routines are Procedure `fft0padBackward` and it inverse, Procedure `fftpad0Forward` from Ref. [4].

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: $F_{N/2}=\overline{F_{-N/2}}=\overline{F_{N/2}}$. Letting $m=\lfloor N/2\rfloor+1$, in the case where N is even, the $2m$ modes can therefore

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

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

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.

be indexed as $\{F_{-m+1}, \ldots, F_m\}$ where F_0 and F_m are real. An odd number $2m-1$ of modes is indexed as $\{F_{-m+1}, \ldots, F_{m-1}\}$.

Hermitian symmetry can be used to reduce the computational complexity and storage requirements of real-to-complex and complex-to-real Fourier transforms by a factor of about two. A one-dimensional convolution of Hermitian data only requires the data corresponding to non-negative wavenumbers. In the compact case, with modes in $\{-m+1, \ldots, m-1\}$, 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 $\{0, \ldots, m\}$. 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 Function `conv` produces identical results. For $m = 2$, the noncompact version yields the correct values $\{(59,0), (20,-18)\}$ for the first m elements only if the data $(3,12)$ for the Nyquist mode at $k = m$ is taken into account.

Let us now describe a noncompact implicitly padded Hermitian 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), but now with

$$w_{k,r} \doteq \begin{cases} F_0 + \operatorname{Re} \zeta_3^{-r} F_m & \text{if } k = 0, \\ \zeta_{3m}^{rk} \left(F_k + \zeta_3^{-r} \overline{F_{m-k}} \right) & \text{if } 1 \le k \le m-1. \end{cases} \tag{6}$$

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) 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

An in-place implicitly padded Hermitian convolution was previously described in Function `conv` of Ref. [4] for the case of $2M$ inputs and one output, where the multiplication operator was restricted to a dot product. However, that algorithm cannot be efficiently applied to the autoconvolution case (with just one input and one output), to pseudospectral simulations of 3D Navier–Stokes and magnetohydrodynamic flows, or to the reduced-FFT scheme of Basdevant for 2D turbulence (see Appendix A). Furthermore, interloop dependencies at the outermost level prevent it from being multithreaded. Moreover, 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 nature of the real-space multiplication operator. By unrolling the outer loop of the in-place Hermitian 1D convolution, these deficiencies can be eliminated, resulting in the fully multithreaded implementation in Function `conv`, generalized to handle A inputs and B outputs and an arbitrary quadratic multiplication operator `mult` $: \mathbb{R}^A \to \mathbb{R}^B$.

```
**Input:** vectors $\{f_a\}_{a=0}^{A-1}$
**Output:** vectors $\{f_b\}_{b=0}^{B-1}$

**for** $a = 0$ **to** $A-1$ **do**
    pretransform($f_a, u_{A-1}$)
    $f_a^0 \leftarrow$ fft$^{-1}$($f_a^0$)
    $f_a^1 \leftarrow$ fft$^{-1}$($f_a^1$)
    $u_a \leftarrow$ fft$^{-1}$($u_{A-1}$)

$\{f_b^0\}_{b=0}^{B-1} \leftarrow$ mult($\{f_a^0\}_{a=0}^{A-1}$)
$\{f_b^1\}_{b=0}^{B-1} \leftarrow$ mult($\{f_a^1\}_{a=0}^{A-1}$)
$\{u_b\}_{b=0}^{B-1} \leftarrow$ mult($\{u_a\}_{a=0}^{A-1}$)

**for** $b = 0$ **to** $B-1$ **do**
    $u_0 \leftarrow$ fft($u_b$)
    $f_b^0 \leftarrow$ fft($f_b^0$)
    $f_b^1 \leftarrow$ fft($f_b^1$)
    posttransform($\{f_b^0\}_{k=0}^{c} \cup$
      $\{f_b^1\}_{k=2c+2-m}^{c}, f_b^1[1], u_0$)

**return** $\{f_b/(3m)\}_{b=0}^{B-1}$
```

Function `conv` returns the implicitly dealiased 1D Hermitian convolution of length m $(m+1)$ in the compact (noncompact) format, using the multiplication operator `mult` : $\mathbb{R}^A \to \mathbb{R}^B$, with $2A+2B+2$ in-place and $A+B-2$ out-of-place FFTS.

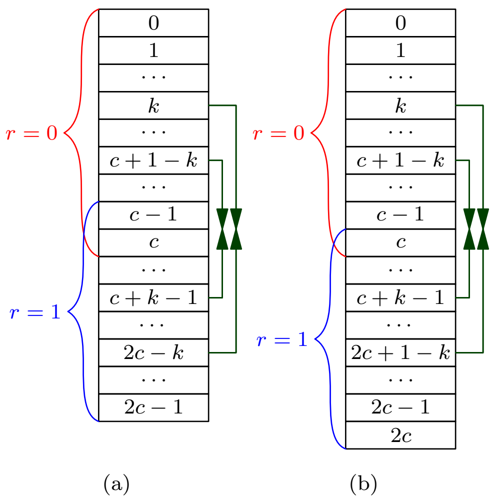


Figure 4 Loop unrolling for the Hermitian 1D convolution when (a) $m = 2c$ and (b) $m = 2c+1$.

To multithread Procedure `pretransform`, we unroll two iterations of the loop from Procedure `build` in Ref. [4] 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 when $m = 2c+1$ there is only one overlapping value, at index c. 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. A similar loop unrolling is used in a revised implementation of the post-processing phase (see Procedure `posttransform`) to allow for an arbitrary number of inputs A and outputs 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 when written to the array f, additional code is required to save and restore the overlapping elements in the actual in-place implementation.

In our general Function `conv`, only $A + B - 2$ of the $3A + 3B$ FFTs can be performed out of place. When $A = B$ this is the best that one can do: for example, for an autoconvolution ($A = B = 1$), there is no free buffer available that would enable the use of out-of-place transforms. Nevertheless when $A > B$ or $B > A$ the optimized Function `convA` or `convB`, respectively, performs one FFT in place and the remaining $3A + 3B - 1$ FFTs out of place. Even with one thread, for $A = 2$ and $B = 1$, Functions `conv` and `convA` both have slightly better performance than Function `conv` in Ref. [4], primarily due to the removal of loop interdependence. Most importantly, `conv`, `convA`, and `convB` have fully parallelized pre- and post-processing phases and use `FFTW`'s built-in parallel FFTs, which are typically much more efficient when the transforms are out of place. The new routines accept either compact or noncompact inputs and can therefore also benefit from the performance advantage of the noncompact data format discussed in Section 3.2.

```
Input: vector f
Output: vector f, vector 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 $\mathsf{F} \leftarrow \mathsf{f}[c]$
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 $c = \lfloor m/2 \rfloor$ and $d = \lfloor (c + 1)/2 \rfloor$. The Fourier origin corresponds to array position 0.

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, where $K = 34/9$ [4], in agreement with the leading-order scaling of an explicitly padded centered Hermitian convolution. In our new implementation, eight of the nine Fourier transforms can now 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.

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

Procedure `posttransform(f,w,u)` is called by Function `conv` to combine the contributions for $r = 0$, 1, and -1 into an implicitly-dealiased Hermitian convolution. The vector f has length m $(m + 1)$ in the compact (noncompact) format, and the auxiliary vector u has length $c+1$, where $c = \lfloor m/2 \rfloor$ and $d = \lfloor (c+1)/2 \rfloor$. When $m = 2c$, the scalar w contains the overlapped value for $r = 1$ and $k = 1$.

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. The optimal values of m for the explicit version are $\lfloor (2^p + 2)/3 \rfloor$ for natural numbers p, whereas for the implicit version the optimal values are powers of two, so direct comparison of the methods using optimal problem sizes is not possible. Instead, we compare the two methods using a linear interpolation (with respect to $\log m$) of the execution time rescaled by the computational complexity of the algorithm. With one thread, the implicit version runs between 5% and 23% faster for $m \geq 2048$; with four threads, the implicit version is between 12% and 105% faster for $m \geq 65536$, as shown in Fig. 5(a). 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 with four threads is between 45% and 68%, giving a speedup of a factor of 1.8 to 2.7.

```
Input: vectors $\{f_a\}_{a=0}^{A-1}$
Output: vectors $\{f_b\}_{b=0}^{B-1}$

for $a = 0$ to $A-1$ do
    pretransform($f_a, u_{A-1}$)
    $u_a \leftarrow$ fft$^{-1}$($u_{A-1}$)
$\{u_b\}_{b=0}^{B-1} \leftarrow$ mult($\{u_a\}_{a=0}^{A-1}$)

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

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

for $b = 0$ to $B-1$ do
    $u_{A-1} \leftarrow$ fft($u_b$)
    $f^0_b \leftarrow$ fft($f^0_{b+1}$)
    $f^1_b \leftarrow$ fft($f^1_{b+1}$)
    posttransform($\{f^0_b\}_{k=0}^{c} \cup$
      $\{f^1_b\}_{k=2c+2-m}^{c}, f^1_b[1], u_{A-1}$)

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

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

```
Input: vectors $\{f_a\}_{a=0}^{A-1}$
Output: vectors $\{f_b\}_{b=0}^{B-1}$

for $a = A-1$ to $0$ do
    pretransform($f_a, u_a$)
    $u_{a+1} \leftarrow$ fft$^{-1}$($u_a$)
    $f^0_{a+1} \leftarrow$ fft$^{-1}$($f^0_a$)
    $f^1_{a+1} \leftarrow$ fft$^{-1}$($f^1_a$)

$\{f^0_b\}_{b=1}^{B-1} \cup \{u_0\} \leftarrow$ mult($\{f^0_a\}_{a=1}^{A}$)
for $b = 0$ to $B-2$ do
    $f^0_b \leftarrow$ fft($f^0_{b+1}$)
$f^0_{B-1} \leftarrow$ fft($u_0$)

$\{f^1_b\}_{b=1}^{B-1} \cup \{u_0\} \leftarrow$ mult($\{f^1_a\}_{a=1}^{A}$)
for $b = 0$ to $B-2$ do
    $f^1_b \leftarrow$ fft($f^1_{b+1}$)
$f^1_{B-1} \leftarrow$ fft($u_0$)

$\{u_b\}_{b=1}^{B-1} \cup \{u_0\} \leftarrow$ mult($\{u_a\}_{a=1}^{A}$)
$u_0 \leftarrow$ fft($u_0$)
posttransform($\{f^0_{B-1}\}_{k=0}^{c} \cup$
  $\{f^1_{B-1}\}_{k=2c+2-m}^{c}, f^1_{B-1}[1], u_0$)
for $b = 0$ to $B-2$ do
    $u_0 \leftarrow$ fft($u_{b+1}$)
    posttransform($\{f^0_b\}_{k=0}^{c} \cup$
      $\{f^1_b\}_{k=2c+2-m}^{c}, f^1_b[1], u_0$)

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

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

4 Higher-dimensional convolutions

A d-dimensional convolution can be computed by performing an inverse FFT of size $m_1 \times \ldots \times m_d$, applying the appropriate multiplication on the transformed data, followed by an FFT back to the original space. Equivalently, one can perform $\prod_{i=2}^{d} m_i$ inverse FFTs in the first dimension, followed by m_1 convolutions of dimension $d-1$, and finally $\prod_{i=2}^{d} m_i$ FFTs in the first dimension. The innermost operation of a recursive multidimensional convolution thus reduces to a 1D convolution. Using this decomposition, one can reuse the work buffer for each implicitly dealiased subconvolution, thereby reducing the total memory demand relative to the explicit d-dimensional dealiasing requirement. For multithreaded implicitly dealiased convolutions, the initial inverse FFT can be parallelized by dividing the $\prod_{i=2}^{d} m_i$ 1D FFTs and m_1 subconvolutions between the T threads. Since each implicitly dealiased subconvolution requires a work buffer, the total memory requirement grows with the number of threads, but is still much lower

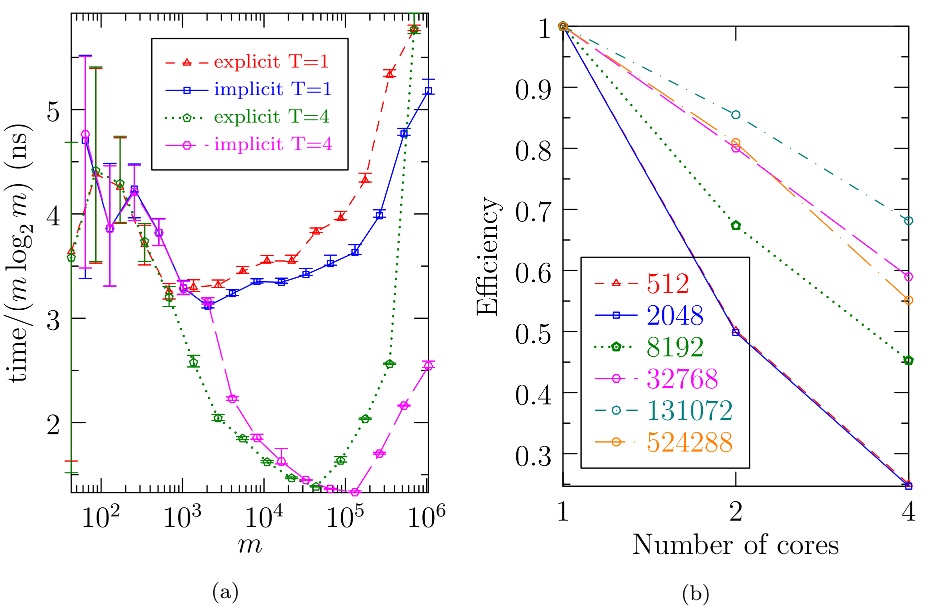


Figure 5 In-place 1D Hermitian convolutions of length m: (a) comparison of computation times for explicit and implicit dealiasing using $T = 1$ thread and $T = 4$ threads; (b) parallel efficiency of implicit dealiasing versus number of threads. For implicit convolutions, m is chosen to be a power of two, while for explicit convolutions $m = \lfloor (N + 2)/3 \rfloor$, where N is a power of two.

than that required for explicit multidimensional convolutions when $T \ll m_1$. When $T \leq m_1$, we compute T subconvolutions at a time, using one inner thread per subconvolution to avoid over-subscription. Otherwise, if $T > m_1$, we parallelize only the inner subconvolution (over all T threads).

A single-threaded 2D implicitly 1/2-padded complex convolution is shown in Figure 6. Each input buffer is implicitly padded and inverse Fourier transformed in the x direction to produce the data shown in the square 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 column of the Fourier transformed image, depicted in yellow. The Fourier transformed columns of two inputs F and G are then multiplied pointwise and stored back into the F column. At this point, the y forward-padded FFT can then be performed, with the result stored in the lower-half of the column, next to the previously processed data shown in red. The process is repeated on the remaining columns, shifting and reusing the work buffer. Once all the columns have been processed, a forward-padded FFT in the x direction produces the final convolution in the left-hand half of the F buffer.

The reuse of subconvolution work memory allows the convolution to be computed using less total memory: for 1/2 padded convolutions, the memory requirement per input is only about twice what would be required if dealiasing was outright ignored. 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 multidimensional 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.

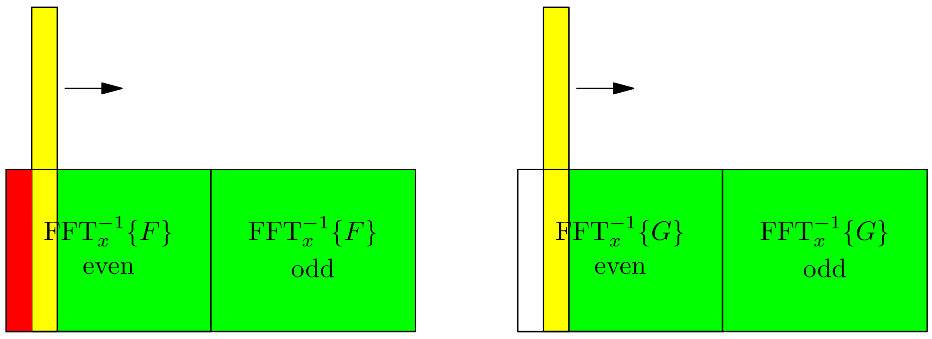

$\mathrm{FFT}_x^{-1}\{F\}$
even
$\mathrm{FFT}_x^{-1}\{F\}$
odd
$\mathrm{FFT}_x^{-1}\{G\}$
even
$\mathrm{FFT}_x^{-1}\{G\}$
odd

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

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. [4] 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 multidimensional and the required FFT is one-dimensional, we use the `FFTW` multiple 1D FFT routine. A multithreaded version of this routine is available in the `FFTW` library, but we found that its 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 automatically test for the possibility that this decomposition is faster than `FFTW`'s parallel multiple FFT and use whichever algorithm runs faster. This yielded 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 2.6, with a maximum speedup factor of 4.5. 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 58% to 92% with four threads, for a speedup of 2.3 to 3.7 relative to one thread. The explicit routine has a parallel efficiency between 25% and

90%. Notably, the 2D explicit version has poor parallel performance for problem sizes of 512^2 and above using FFTW's built-in multithreading.

Because the same temporary arrays u and v are used for each column of the convolution, the memory requirement is $2Cm_xm_y + TCm_y$ complex words using T threads, where $C = \max\{A, B\}$. Assuming that $T < 2m_x$, this is far less than the $4Cm_xm_y$ complex words needed for an explicitly padded convolution.

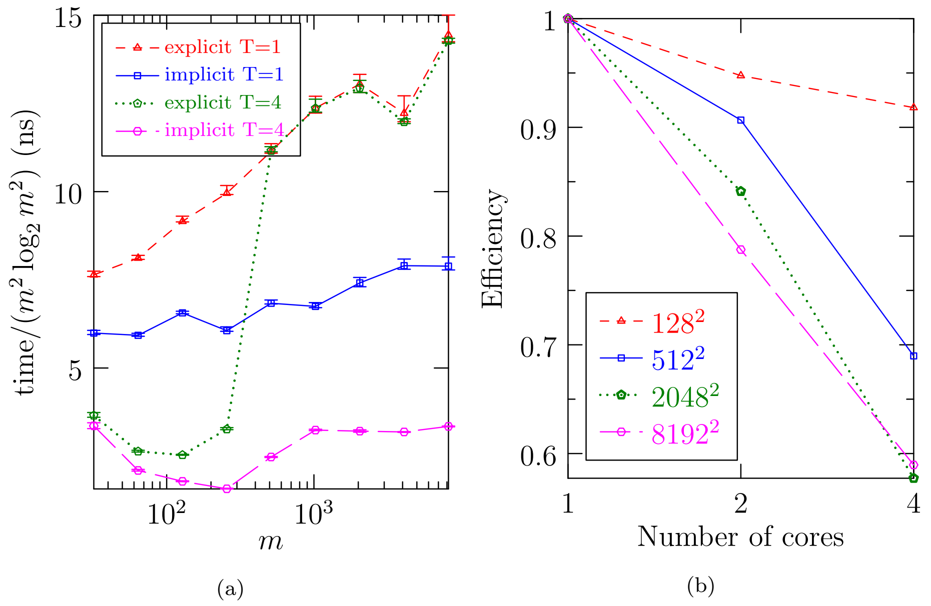


Figure 7 In-place 2D complex convolutions of size $m \times m$: (a) comparison of computation times for explicit and implicit dealiasing using $T = 1$ thread and $T = 4$ threads; (b) parallel efficiency of implicit dealiasing versus number of threads. Here m is chosen to be a power of two.

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 $\{-m_x+1, \ldots, m_x-1\} \times \{0, \ldots, m_y-1\}$, whereas the noncompact version has dimensions $\{-m_x, \ldots, m_x-1\} \times \{0, \ldots, m_y\}$. 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 $(2m_x - 2) \times (2m_y - 2)$ 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 case is typically slightly faster than the compact case when using one thread and 25% faster on average when using four threads.

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

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

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

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

The explicit version requires storage for $9Cm_x(m_y+1)/2$ complex words, where $C = \max\{A, B\}$. For the noncompact case, the implicit version using T threads requires storage for $3Cm_x(m_y + 1) + TC(\lfloor m_y/2 \rfloor + 1)$ complex words, which is much less than the explicit case when $m_x \geq T$. As shown in Fig. 8(a), implicit padding again yields a dramatic improvement in speed: the implicit version is on average 1.36 times faster than the explicit version when using one thread, and 2.93 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 73% and 87% efficiency when using four threads, giving a speedup of a factor of 2.9 to 3.5. As in the 2D complex case, the explicit version does not parallelize well for large problem sizes.

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 Func-

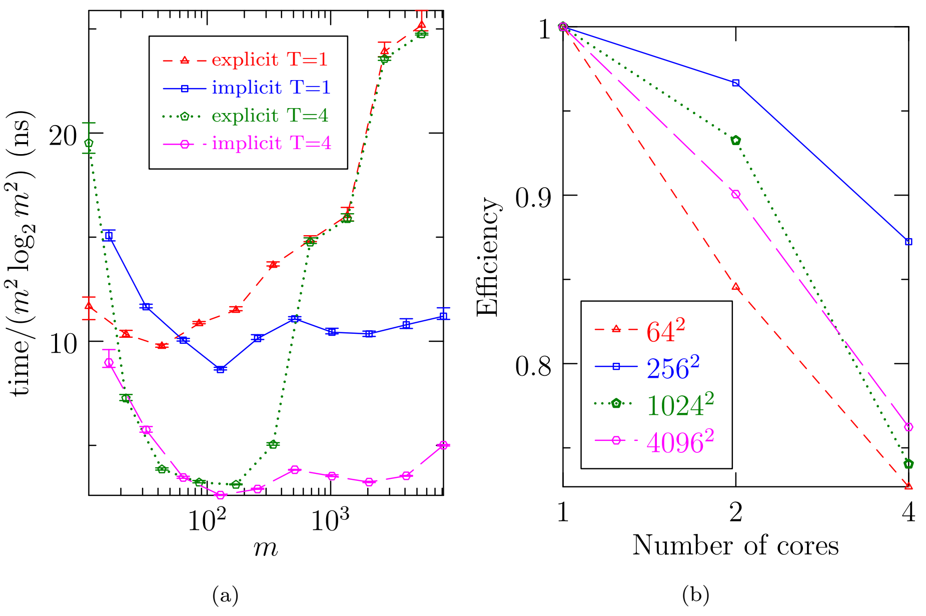


Figure 8 In-place 2D Hermitian convolutions of size $2m \times (m+1)$: (a) comparison of computation times for explicit and implicit dealiasing using $T = 1$ thread and $T = 4$ threads; (b) parallel efficiency of implicit dealiasing versus number of threads. For implicit convolutions, m is chosen to be a power of two, while for explicit convolutions $m = \lfloor (N+2)/3 \rfloor$, where N is a power of two.

tion `cconv3`. For A inputs with dimensions $m_x \times m_y \times m_z$ and B outputs, the explicit version requires $8Cm_xm_ym_z$ complex words, where $C = \max\{A, B\}$. In contrast, the implicit version with T threads requires $2Cm_xm_ym_z + TCm_ym_z + TCm_z$ complex words, 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 consequently much faster than their explicit counterparts. For a single thread, the implicit version is on average 1.9 times as fast as the explicit version and 4.3 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 65% and 86% efficient when using four threads, giving a speedup of a factor of 2.6 to 3.4 over one thread. The explicit version has reasonable parallel efficiency for small problem sizes, but this drops to roughly 25% on four threads for problem size $m \geq 64$.

4.4 Centered Hermitian 3D convolution

As with the 1D and 2D cases, we offer compact and noncompact versions of a 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 dimension 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.

```
Input: ${\mathsf{f}_a}_{a=0}^{A-1}$
Output: ${\mathsf{f}_b}_{b=0}^{B-1}$
$\mathsf{R} \leftarrow \{0,\ldots,m_y-1\}\times\{0,\ldots,m_z-1\}$
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: ${\mathsf{f}_b}_{b=0}^{B-1}$
$\mathsf{R} \leftarrow \{0,\ldots,2m_y\}\times\{0,\ldots,m_z\}$
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
```

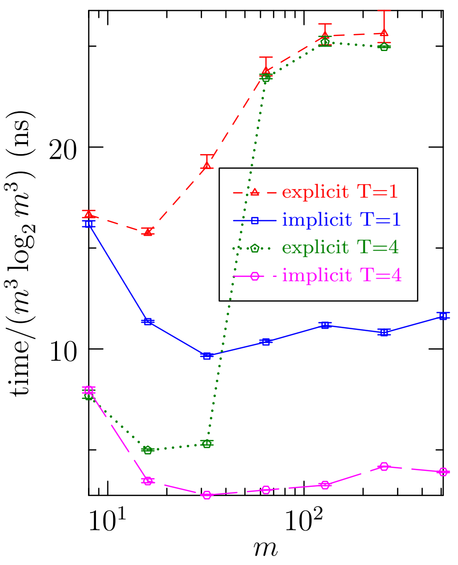
Function conv3 returns the in-place implicitly dealiased Hermitian convolution of $2m_x \times 2m_y \times (m_z+1)$ matrices $\{\mathsf{f_a}\}_{\mathsf{a=0}}^{\mathsf{A-1}}$, using A temporary $m_x \times m_y \times (m_z+1)$ matrices $\{\mathsf{U_a}\}_{\mathsf{a=0}}^{\mathsf{A-1}}$.

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 on a single thread, and the noncompact case offering between a 1% and 10% performance advantage when parallelized over four threads.

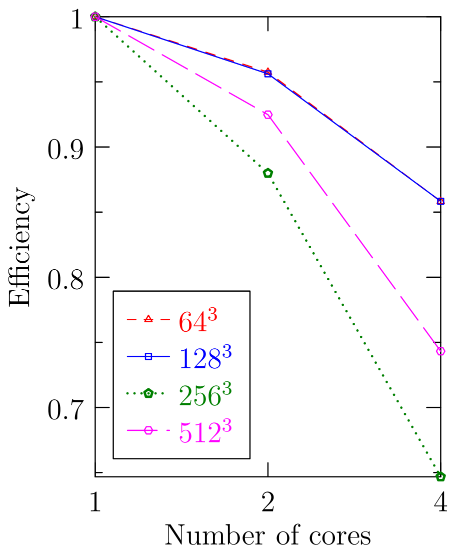
In the noncompact format, the memory requirements for an explicit 3D Hermitian convolution with A inputs and B outputs is $\frac{27}{2}Cm_xm_y(m_z+1)$ complex words, whereas the implicit version requires only $6Cm_xm_y(m_z+1)+TCm_y(m_z+1)+TC(\lfloor m_z/2\rfloor+1)$ complex words using T threads, where $C=\max\{A,B\}$. 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 65% and 92%, which translates to a speedup of a factor of 2.6 to 3.7 using four threads instead of one.

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, allowing for the efficient computation of autoconvolutions, correlations (which are identical to convolutions for Hermitian-symmetric data), and general nonlinearities in pseudospectral simulations.

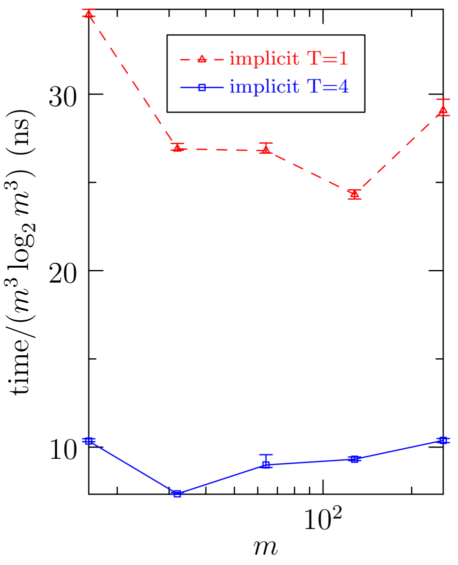


(a)

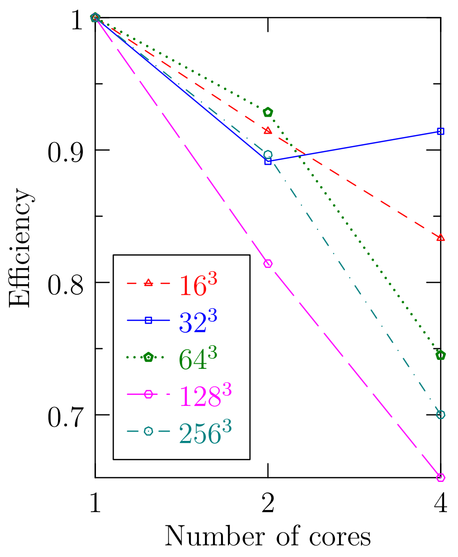


(b)

Figure 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. Here m is chosen to be a power of two.



(a)



(b)

Figure 10 Implicitly dealiased in-place 3D Hermitian convolutions of size $(2m - 1) \times (2m - 1) \times (m + 1)$ for $T = 1$ and $2m \times (2m - 1) \times (m + 1)$ for $T = 4$: (a) computation times using $T = 1$ thread and $T = 4$ threads; (b) parallel efficiency versus number of threads. Here m is chosen to be a power of two.

Implicitly dealiased convolutions require less memory, are faster, and have greater parallel efficiency than their explicitly dealiased counterparts. Specifically, in d dimensions the memory savings for 1/2 padding is a factor of 2^{d-1}; for 2/3 padding the savings factor is $(3/2)^{d-1}$. 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 four times faster than the explicit method when both are parallelized over 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.

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 of time series. We are also exploring novel applications of implicitly dealiased convolutions for computing sparse Fourier transforms [9], fractional phase Fourier (chirp-z) transforms [1], and partial Fourier transforms [13, 3].

A Basdevant formulation

A.1 3D incompressible Navier–Stokes equation

A naive implementation of the pseudospectral method for the 3D incompressible Navier–Stokes equation,

$$\frac{\partial u_i}{\partial t} + \frac{\partial D_{ij}}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \nu \frac{\partial^2 u_i}{\partial x_j^2} + F_i, \tag{7}$$

where $D_{ij} = u_i u_j$, requires three backward FFTs to compute the velocity components from their spectral representations and six forward FFTs of the independent components of the symmetric tensor D_{ij}, for a total of nine FFTs per integration stage. However Basdevant [2] showed that this number can be reduced to eight, by subtracting the divergence of the symmetric matrix $S_{ij} = \delta_{ij} \operatorname{tr} D/3$ from both sides of Eq. (7):

$$\frac{\partial u_i}{\partial t} + \frac{\partial (D_{ij} - S_{ij})}{\partial x_j} = -\frac{\partial (p\delta_{ij} + S_{ij})}{\partial x_j} + \nu \frac{\partial^2 u_i}{\partial x_j^2} + F_i. \tag{8}$$

Since the symmetric matrix $D_{ij} - S_{ij}$ is traceless, it has just five independent components. Together with the three backward FFTs required for the velocity components u_i, we see that only eight FFTs are required per integration stage. The effective pressure $p\delta_{ij} + S_{ij}$ is solved as usual from the inverse Laplacian of the force minus the nonlinearity.

A.2 2D incompressible Navier–Stokes equation

The vorticity $\boldsymbol{w} = \boldsymbol{\nabla}\times\boldsymbol{u}$ evolves according to

$$\frac{\partial \boldsymbol{w}}{\partial t} + (\boldsymbol{u}\cdot\boldsymbol{\nabla})\boldsymbol{w} = (\boldsymbol{w}\cdot\boldsymbol{\nabla})\boldsymbol{u} + \nu\nabla^2\boldsymbol{w} + \boldsymbol{\nabla}\times\boldsymbol{F},$$

where in two dimensions the vortex stretching term $(\boldsymbol{w}\cdot\boldsymbol{\nabla})\boldsymbol{u}$ vanishes and $\boldsymbol{w}$ is normal to the plane of motion. For C^2 velocity fields, the curl of the nonlinearity can be written in terms of $\widetilde{D}_{ij} \doteq D_{ij} - S_{ij}$:

$$\frac{\partial}{\partial x_1}\frac{\partial}{\partial x_j}\widetilde{D}_{2j} - \frac{\partial}{\partial x_2}\frac{\partial}{\partial x_j}\widetilde{D}_{1j} = \left(\frac{\partial^2}{\partial x_1^2} - \frac{\partial^2}{\partial x_2^2}\right)D_{12} + \frac{\partial}{\partial x_1}\frac{\partial}{\partial x_2}(D_{22} - D_{11}),$$

on recalling that S is diagonal and $S_{11} = S_{22}$. The scalar vorticity ω thus evolves as

$$\frac{\partial \omega}{\partial t} + \left(\frac{\partial^2}{\partial x_1^2} - \frac{\partial^2}{\partial x_2^2}\right)(u_1 u_2) + \frac{\partial^2}{\partial x_1 \partial x_2}\left(u_2^2 - u_1^2\right) = \nu\nabla^2\omega + \frac{\partial F_2}{\partial x_1} - \frac{\partial F_1}{\partial x_2}.$$

Two backward FFTs are required to compute u_1 and u_2 in physical space, from which the quantities $u_1 u_2$ and $u_2^2 - u_1^2$ can be calculated and then transformed to Fourier space with two additional forward FFTs. The advective term in 2D can thus be calculated with just four FFTs.

References

1. Bailey, D.H., Swarztrauber, P.N.: The fractional Fourier transform and applications. SIAM review **33**(3), 389–404 (1991)
2. Basdevant, C.: Technical improvements for direct numerical simulation of homogeneous three-dimensional turbulence. Journal of Computational Physics **50**(2), 209–214 (1983)
3. Bowman, J.C., Ghoggali, Z.: The partial fast Fourier transform. Submitted to J. Sci. Comput. (2017)
4. Bowman, J.C., Roberts, M.: Efficient dealiased convolutions without padding. SIAM J. Sci. Comput. **33**(1), 386–406 (2011)
5. Bowman, J.C., Roberts, M.: `FFTW++`: A fast Fourier transform C^{++} header class for the `FFTW3` library. `http://fftwpp.sourceforge.net` (May 6, 2010)
6. Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of complex Fourier series. Mathematics of Computation **19**(90), 297–301 (1965)
7. Gauss, C.F.: Nachlass: Theoria interpolationis methodo nova tractata. In: Carl Friedrich Gauss Werke, vol. 3, pp. 265–327. Königliche Gesellschaft der Wissenschaften, Göttingen (1866)
8. Gottlieb, D., Orszag, S.A.: Numerical Analysis of Spectral Methods: Theory and Applications. Society for Industrial and Applied Mathematics, Philadelphia (1977)
9. Hassanieh, H., Indyk, P., Katabi, D., Price, E.: Simple and practical algorithm for sparse Fourier transform. In: Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1183–1194. SIAM (2012)
10. Orszag, S.A.: Elimination of aliasing in finite-difference schemes by filtering high-wavenumber components. Journal of the Atmospheric Sciences **28**, 1074 (1971)
11. Roberts, M.: Multispectral reduction of two-dimensional turbulence. Ph.D. thesis, University of Alberta, Edmonton, AB, Canada (2011). `http://www.math.ualberta.ca/~bowman/group/roberts_phd.pdf`
12. Roberts, M., Leroy, M., Morales, J., Bos, W., Schneider, K.: Self-organization of helically forced MHD flow in confined cylindrical geometries. Fluid Dynamics Research **46**(6), 061,422 (2014). URL `http://stacks.iop.org/1873-7005/46/i=6/a=061422`
13. Ying, L., Fomel, S.: Fast computation of partial Fourier transforms. Multiscale Modeling and Simulation **8**(1), 110–124 (2009)