SIAM J. SCI. COMPUT.
Vol. 46, No. 3, pp. B159–B178

HYBRID DEALIASING OF COMPLEX CONVOLUTIONS*

NOEL MURASKO† AND JOHN C. BOWMAN†

Abstract. Efficient algorithms based on the fast Fourier transform are developed for computing linear convolutions. A hybrid approach is described that combines the conventional practice of explicit dealiasing (explicitly padding the input data with zeros) and implicit dealiasing (mathematically accounting for these zero values). The new approach generalizes implicit dealiasing to arbitrary padding ratios and includes explicit dealiasing as a special case. Unlike existing implementations of implicit dealiasing, hybrid dealiasing tailors its subtransform sizes to the convolution geometry. Multidimensional convolutions are implemented with hybrid dealiasing by decomposing them into lower-dimensional convolutions. Convolutions of complex-valued and Hermitian inputs of equal length are illustrated with pseudocode and implemented in the open-source `FFTW++` library. Hybrid dealiasing is shown to outperform explicit dealiasing in one, two, and three dimensions.

Key words. dealiasing, hybrid padding, implicit padding, zero padding, convolution, discrete Fourier transform, fast Fourier transform, Hermitian symmetric data

MSC codes. 65R99, 65T50

DOI. 10.1137/23M1552073

See reproducibility of computational results at end of the article.

1. Introduction.

The convolution theorem provides a method for efficiently computing circular convolutions using fast Fourier transforms (FFTs). However, many applications require linear convolutions; these can be computed using the convolution theorem by padding the data with a sufficient number of zeros to avoid polluting the result with *aliases*: errors that arise from the lack of periodicity in the input. The standard practice is to pad the input arrays explicitly with zeros before taking the FFT. While this accomplishes dealiasing, it requires reading and multiplying values that are known a priori to be zero.

Implicit dealiasing [3, 15] provides an alternative to explicit dealiasing. The FFTs are formulated to take account of the known zero values implicitly, avoiding the need for explicit zero padding. In previous work, the focus was on developing implicit dealiasing for 1/2 and 2/3 padding ratios [13] (these fractions refer to the ratio of input data length to the zero-padded buffer size) for complex and Hermitian symmetric input data. While these cases are important for applications, such as signal processing and pseudospectral methods for solving partial differential equations [14, 9], many applications do not satisfy these requirements.

The goal of this work is to formulate and develop a systematic framework, which we call *hybrid dealiasing,* for the efficient dealiasing of complex-valued convolutions with arbitrary padding ratios. The decomposition of multidimensional dealiased convolutions into padded FFTs and one-dimensional subconvolutions [3, 15] is crucial for

*Submitted to the journal's Software, High-Performance Computing, and Computational Science and Engineering section February 8, 2023; accepted for publication (in revised form) December 12, 2023; published electronically May 2, 2024.
https://doi.org/10.1137/23M1552073
Funding: This work was supported by Natural Sciences and Engineering Research Council of Canada grants RES0043585 and RES0046040.

†Department of Mathematical and Statistical Sciences, University of Alberta, Edmonton, Alberta, T6G 2G1, Canada (murasko@ualberta.ca, bowman@ualberta.ca).

the enhanced performance afforded by hybrid dealiasing, even when the zero padding is explicit.

In section 2, we review the concept of implicit dealiasing and introduce hybrid padding. Sections 3 and 4 then extend hybrid padding to centered and Hermitian symmetric data. In section 5, we discuss how to compute multidimensional convolutions efficiently by decomposing them into lower-dimensional convolutions. An overview of numerical optimizations and results are given in sections 6 and 7. Finally, future applications of this work are discussed in section 8.

2. Complex convolutions in one dimension. Let X denote the space of sequences with bounded support equipped with addition and scalar multiplication.

Let $A, B \in \mathbb{N}$. An operator $\mathcal{C} : X^A \mapsto X^B$ is a *general convolution* if there exists $M \in \mathbb{N}$ and a pointwise operator $\mathcal{M} : X^A \mapsto X^B$ (called the *multiplication operator*), which satisfies

$$\mathrm{dft}[\mathcal{C}(\boldsymbol{f}_1, \ldots, \boldsymbol{f}_A)] = \mathcal{M}(\mathrm{dft}[\boldsymbol{f}_1], \ldots, \mathrm{dft}[\boldsymbol{f}_A]),$$

where $\boldsymbol{f}_1, \ldots, \boldsymbol{f}_A$ have been zero-padded to length M and dft denotes the discrete Fourier transform (DFT) of each component of its argument. An important special case is the convolution of two sequences $\boldsymbol{f} = \{f_j\}_{j\in\mathbb{Z}}$ and $\boldsymbol{g} = \{g_j\}_{j\in\mathbb{Z}}$ given by $(\boldsymbol{f} * \boldsymbol{g})_j \doteq \sum_{i\in\mathbb{Z}} f_i g_{j-i}$, where we use $\doteq$ to denote a definition. One can compute such convolutions efficiently by taking the DFT of each sequence, multiplying the results element by element, and then taking the inverse DFT. Due to the cyclic nature of the DFT, the resulting convolution is circular; for a linear convolution, the input data must be padded (*dealiased*) with zeros.

The amount of zero padding needed to compute the convolution using DFTs depends on the pointwise multiplication operator in the convolution and the length of the input data. For example, when $\mathcal{M}(\boldsymbol{f}_1 \ldots, \boldsymbol{f}_A)_j = (\boldsymbol{f}_1)_j \cdots (\boldsymbol{f}_A)_j$, one can obtain all components of a convolution of A inputs with lengths $L_1, \ldots, L_A$ by padding each input to size $M = \sum_{i=1}^{n} L_i - (A-1)$. However, in some applications not all components of the convolution are needed; this can reduce the padding requirements.

In this work, we only consider convolutions with A equal-size inputs and B outputs that result from an arbitrary element-by-element multiplication in the transformed domain. Given input data $\{a_j\}_{j=0}^{L-1}$ that needs to be padded with zeros to length M, we construct a buffer $\boldsymbol{f} \doteq \{f_j\}_{j=0}^{M-1}$, where $f_j = a_j$ for $j < L$ and $f_j = 0$ for $j \geq L$. The DFT of $\boldsymbol{f}$ can be written as

$$F_k = \sum_{j=0}^{M-1} \zeta_M^{kj} f_j = \sum_{j=0}^{L-1} \zeta_M^{kj} f_j, \quad k \in \{0, \ldots, M-1\},$$

where $\zeta_N \doteq \exp(2\pi i/N)$ is the Nth primitive root of unity.

For now, assume that L and M share a common factor m, so that $L = pm$ and $M = qm$, where $m, p, q \in \mathbb{N}$, with $q \geq p$. We can then reindex j and k as

$$j = tm + s, \quad t \in \{0, \ldots, p-1\}, \quad s \in \{0, \ldots, m-1\},$$

$$k = q\ell + r, \quad \ell \in \{0, \ldots, m-1\}, \quad r \in \{0, \ldots, q-1\}.$$

This allows us to decompose the DFT via the Cooley–Tukey algorithm [6] as in [3]:

$$F_{q\ell+r} = \sum_{s=0}^{m-1}\sum_{t=0}^{p-1} \zeta_{qm}^{(q\ell+r)(tm+s)} f_{tm+s} = \sum_{s=0}^{m-1} \zeta_m^{\ell s}\zeta_{qm}^{rs} \sum_{t=0}^{p-1} \zeta_q^{rt} f_{tm+s}. \tag{2.1}$$

Computing the DFT of $\boldsymbol{f}$ then amounts to preprocessing and computing q DFTs of size m; no explicit zero padding is needed. The inverse transform is similar [3]:

$$f_{tm+s} = \frac{1}{qm}\sum_{r=0}^{q-1} \zeta_q^{-tr}\zeta_{qm}^{-sr} \sum_{\ell=0}^{m-1} \zeta_m^{-s\ell} F_{q\ell+r}. \tag{2.2}$$

This transform requires q DFTs of size m, followed by postprocessing. We now demonstrate how these equations can be implemented to generalize implicit dealiasing to a wider class of convolutions.

2.1. Hybrid padding. An issue with the above formulation is the assumption that L and M must share a common factor m. Furthermore, even if L and M do share a common factor, the resulting convolution might be inefficient: the most efficient FFTs currently available are for products of powers of the radices 2, 3, 5, and 7 [8].

Our solution to this problem relies on the observation that M is the *minimum* size required to dealias a convolution: padding beyond M is fine (and perhaps desired if it increases efficiency). Given some $m \in \mathbb{N}$, we define

$$p \doteq \left\lceil \frac{L}{m} \right\rceil, \quad q \doteq \left\lceil \frac{M}{m} \right\rceil. \tag{2.3}$$

These are the smallest positive integers such that $pm \geq L$ and $qm \geq M$. To take the forward transform, we *explicitly* pad $\boldsymbol{f}$ with zeros to size pm and then use (2.1) to compute the padded transform of size qm. To take the inverse transform, we use (2.2), ignoring the last $pm - L$ elements. We refer to this combination of explicit and implicit padding, illustrated in Figure 1, as *hybrid padding*. If out-of-place FFTs are used, any explicit zero padding only needs to be written to the buffer once.

An advantage of hybrid padding is the ability to choose any m value, as the choice of m is independent of the size of the input array. For the remainder of this work, we exclusively refer to padding from size pm to qm, keeping in mind that we might have to use hybrid padding to achieve this.

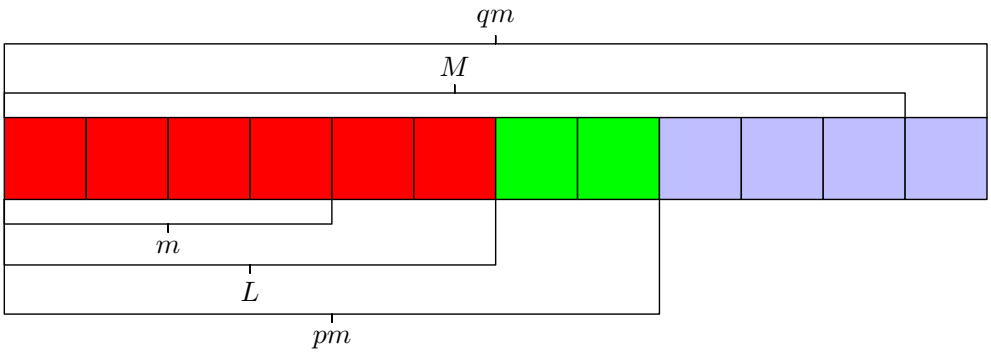

FIG. 1. *An illustration of hybrid padding for a one-dimensional array with $L = 6$ and $M = 11$. Choosing $m = 4$, we have $p = 2$ and $q = 3$. We explicitly pad our data of length L to length $pm = 8$ and then implicitly pad our data to length $qm = 12$.*

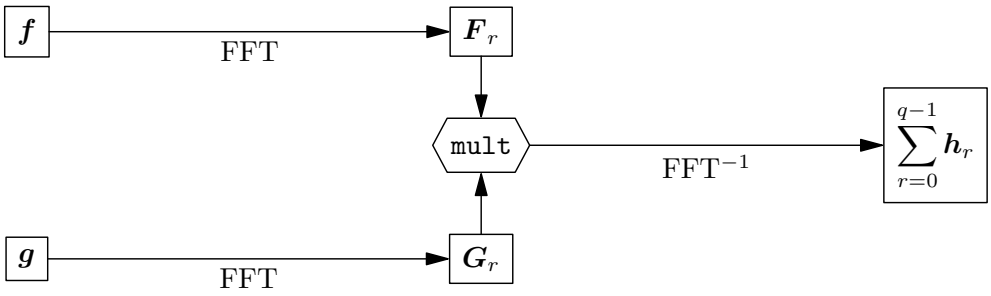

FIG. 2. *Accumulation of residue contributions to a convolution.*

2.2. Convolutions one residue at a time. For each $r \in \{0, \dots q-1\}$, we define the *residue contribution* $\boldsymbol{F}_r \doteq \{F_{q\ell+r}\}_{\ell=0}^{m-1}$. A key optimization that allows us to save memory is that we can compute contributions to the convolution one residue at a time. To find the inverse for that residue contribution, we compute

$$h_{r,s,t} \doteq \zeta_q^{-tr} \zeta_{qm}^{-sr} \sum_{\ell=0}^{m-1} \zeta_m^{-s\ell} F_{q\ell+r}.$$

Accumulating over $r \in \{0, \dots, q-1\}$, we obtain the inverse:

$$f_{tm+s} = \frac{1}{qm} \sum_{r=0}^{q-1} h_{r,s,t}.$$

This formulation can be advantageous for large problems as it allows reuse of the memory needed to store $\boldsymbol{F}_r$, illustrated for a binary convolution in Figure 2.

A general convolution applies a given multiplication operator to the DFT of A inputs to produce B outputs, which are then inverse transformed to produce the final result. The contributions to the forward transform from each residue can either be computed independently or grouped together. When using the conjugate symmetry optimization (described in subsection 6.3), it is convenient to compute the r and $-r \pmod q$ residues together, with the $r = 0$ and $r = q/2$ residues (if q is even) stored together.

Pseudocode for an in-place convolution is given in Algorithm 2.1. Pseudocode for the forward and backward transforms is shown in Algorithms 2.3 and 2.4 for $p = 1$ and Algorithms 2.5 and 2.6 for $p = 2$. For $p > 2$, it is more efficient to replace the summations in (2.1) and (2.2) by an inner DFT, which is discussed in subsection 6.1. The pseudocode illustrates only the case where one residue is computed at a time.

3. Centered convolutions in one dimension. In certain applications, it is convenient to center the data within the input array. While it is possible to multiply the output of the uncentered transform derived in section 2 pointwise by the primitive roots of unity $\{\zeta_{qm}^{-k\lfloor qm/2 \rfloor}\}_{k=0}^{qm-1}$ to obtain a centered transform, this is inefficient if qm is odd because it requires $qm-1$ extra complex multiplies. When qm is even, one could in principle account for these required signs within the multiplication routine (with no extra reading and writing of the Fourier transformed data); however, this would detract from the generality of our approach.

Algorithm 2.1. `convolve` is a one-dimensional complex convolution. There are A inputs (each of length L, to be padded to at least length M) and B outputs. The convolution uses the multiplication operator `mult`.

Input: $\{\boldsymbol{f}_a\}_{a=0}^{A-1}$, L, M, m, A, B
  $p \leftarrow \lceil L/m \rceil$
  if $p = 1$ **then**
    $n \leftarrow \lceil M/m \rceil$
    $q \leftarrow n$
    `Forward` ←`forward1`
    `Backward` ←`backward1`
  else if $p = 2$ **then**
    $n \leftarrow \lceil M/m \rceil$
    $q \leftarrow n$
    `Forward` ←`forward2`
    `Backward` ←`backward2`
  else
    $n \leftarrow \lceil M/m \rceil$
    $q \leftarrow np$
    `Forward` ←`forwardInner`
    `Backward` ←`backwardInner`
  for $b = 0, \ldots, B-1$ **do**
    $\boldsymbol{h}_b \leftarrow \{0\}_{j=0}^{L-1}$
  for $r = 0, \ldots, n-1$ **do**
    for $a = 0, \ldots, A-1$ **do**
      $\boldsymbol{F}_a \leftarrow \texttt{Forward}(\boldsymbol{f}_a, L, m, q, r)$
    $\{\boldsymbol{F}_b\}_{b=0}^{B-1} \leftarrow \texttt{mult}(\{\boldsymbol{F}_a\}_{a=0}^{A-1})$
    for $b = 0, \ldots, B-1$ **do**
      $\boldsymbol{h}_b \leftarrow \boldsymbol{h}_b + \texttt{Backward}(\boldsymbol{F}_b, L, m, q, r)$
  for $b = 0, \ldots, B-1$ **do**
    $\boldsymbol{f}_b \leftarrow \boldsymbol{h}_b/(qm)$
  return $\{\boldsymbol{f}_b\}_{b=0}^{B-1}$

Algorithm 2.2. `convolveX` is a one-dimensional centered or Hermitian convolution. There are A inputs (each of length L, to be padded to at least length M) and B outputs. The convolution uses the multiplication operator `mult`. In the centered version, `X` denotes `C`. In the Hermitian version, `X` denotes `H` and only $\lceil L/2 \rceil$ inputs are provided (corresponding to the nonnegative indices).

Input: $\{\boldsymbol{f}_a\}_{a=0}^{A-1}$, L, M, m, A, B
  $p \leftarrow 2\lceil L/(2m) \rceil$ {force even p}
  if $p = 2$ **then**
    `Forward` ←`forward2X`
    `Backward` ←`backward2X`
  else
    `Forward` ←`forwardInnerX`
    `Backward` ←`backwardInnerX`
  $n \leftarrow \lceil 2M/(pm) \rceil$
  $q \leftarrow np/2$
  for $b = 0, \ldots, B-1$ **do**
    $\boldsymbol{h}_b \leftarrow \{0\}_{j=0}^{L-1}$
  for $r = 0, \ldots, n-1$ **do**
    for $a = 0, \ldots, A-1$ **do**
      $\boldsymbol{F}_a \leftarrow \texttt{Forward}(\boldsymbol{f}_a, L, m, q, r)$
    $\{\boldsymbol{F}_b\}_{b=0}^{B-1} \leftarrow \texttt{mult}(\{\boldsymbol{F}_a\}_{a=0}^{A-1})$
    for $b = 0, \ldots, B-1$ **do**
      $\boldsymbol{h}_b \leftarrow \boldsymbol{h}_b + \texttt{Backward}(\boldsymbol{F}_b, L, m, q, r)$
  for $b = 0, \ldots, B-1$ **do**
    $\boldsymbol{f}_b \leftarrow \boldsymbol{h}_b/(qm)$
  return $\{\boldsymbol{f}_b\}_{b=0}^{B-1}$

Algorithm 2.3. `forward1` is the complex forward transform for residue r when $p = 1$.

Input: $\{f_j\}_{j=0}^{L-1}, L, m, q, r$
  for $s = 0, \ldots, L-1$ **do**
    $W_s \leftarrow \zeta_{qm}^{rs} f_s$
  for $s = L, \ldots, m-1$ **do**
    $W_s \leftarrow 0$
  $\{V_\ell\}_{\ell=0}^{m-1} \leftarrow \texttt{fft}(\{W_s\}_{s=0}^{m-1})$
  return $\{V_k\}_{k=0}^{m-1}$

Algorithm 2.4. `backward1` is the complex backward transform for residue r when $p = 1$.

Input: $\{F_k\}_{k=0}^{m-1}, L, m, q, r$
  $\{W_s\}_{s=0}^{m-1} \leftarrow \texttt{ifft}(\{F_\ell\}_{\ell=0}^{m-1})$
  for $s = 0, \ldots, L-1$ **do**
    $W_s \leftarrow \zeta_{qm}^{-rs} W_s$
  return $\{W_j\}_{j=0}^{L-1}$

Algorithm 2.5. `forward2` is the complex forward transform for residue r when $p=2$.

Input: $\{f_j\}_{j=0}^{L-1}, L, m, q, r$
  for $s=0,\dots,L-m-1$ **do**
    $W_s \leftarrow \zeta_{qm}^{rs} f_s + \zeta_{qm}^{r(m+s)} f_{m+s}$
  for $s=L-m,\dots,m-1$ **do**
    $W_s \leftarrow \zeta_{qm}^{rs} f_s$
  $\{V_\ell\}_{\ell=0}^{m-1} \leftarrow \texttt{fft}(\{W_s\}_{s=0}^{m-1})$
  return $\{V_k\}_{k=0}^{m-1}$

Algorithm 2.6. `backward2` is the complex backward transform for residue r when $p=2$.

Input: $\{F_k\}_{k=0}^{m-1}, L, m, q, r$
  $\{W_s\}_{s=0}^{m-1} \leftarrow \texttt{ifft}(\{F_\ell\}_{\ell=0}^{m-1})$
  for $s=0,\dots,m-1$ **do**
    $V_s \leftarrow \zeta_{qm}^{-sr} W_s$
  for $s=m,\dots,L-1$ **do**
    $V_s \leftarrow \zeta_{qm}^{-(s-m)r} W_{s-m}$
  return $\{V_j\}_{j=0}^{L-1}$

To handle the centered case, we build the shift directly into the transforms. Let $p, m \in \mathbb{N}$, with p even, and let $\boldsymbol{f} = \{f_j\}_{j=-pm/2}^{pm/2-1}$ be a centered array (obtained by symmetrically padding an array of length L to length pm if needed). Implicit padding to length qm can be accomplished with the transform

$$F_k = \sum_{j=-pm/2}^{pm/2-1} \zeta_{qm}^{kj} f_j, \quad k \in \{0,\dots,qm-1\}. \tag{3.1}$$

Separating this sum and shifting the indices, we obtain

$$F_k = \sum_{j=0}^{pm/2-1} \zeta_{qm}^{kj} f_j + \sum_{j=-pm/2}^{-1} \zeta_{qm}^{kj} f_j = \sum_{j=0}^{pm/2-1} \zeta_{qm}^{kj} f_j + \zeta_q^{-kp/2} \sum_{j=0}^{pm/2-1} \zeta_{qm}^{kj} f_{j-pm/2}.$$

Just as in section 2, we reindex our sum (using the fact that p is even):

$$j = tm+s, \quad t \in \left\{0,\dots,\frac{p}{2}-1\right\}, \quad s \in \{0,\dots,m-1\},$$

$$k = q\ell + r, \quad \ell \in \{0,\dots,m-1\}, \quad r \in \{0,\dots,q-1\}.$$

Then (3.1) can be computed with q DFTs of size m:

$$F_{q\ell+r} = \sum_{s=0}^{m-1} \zeta_m^{\ell s} w_{r,s}, \tag{3.2}$$

where

$$w_{r,s} \doteq \zeta_{qm}^{rs} \left(\sum_{t=0}^{p/2-1} \zeta_q^{rt} \left[f_{tm+s} + \zeta_{2q}^{-rp} f_{tm+s-pm/2} \right] \right). \tag{3.3}$$

Because the transformed data is not centered, the inverse transform is identical to (2.2); one must only be careful to store the output values of the inverse transform in the correct locations. Pseudocode for a centered convolution is given in Algorithm 2.2. Pseudocode for centered transforms when $p=2$ is given in Algorithms 3.1 and 3.2. For $p>2$, see subsection 6.2.

Algorithm 3.1. `forward2C` is the centered forward transform for residue r when $p = 2$.

Input: $\{f_j\}_{j=0}^{L-1}, L, m, q, r$
  $H \leftarrow \lfloor L/2 \rfloor$
  for $s = 0, \ldots, m - H - 1$ **do**
    $W_s \leftarrow \zeta_{qm}^{rs} f_{H+s}$
  for $s = m - H, \ldots, L - H - 1$ **do**
    $W_s \leftarrow \zeta_{qm}^{r(s-m)} f_{H+s-m} + \zeta_{qm}^{rs} f_{H+s}$
  for $s = L - H, \ldots, m - 1$ **do**
    $W_s \leftarrow \zeta_{qm}^{r(s-m)} f_{H+s-m}$
  $\{V_s\}_{s=0}^{m-1} \leftarrow \texttt{fft}(\{W_s\}_{s=0}^{m-1})$
  return $\{V_k\}_{k=0}^{m-1}$

Algorithm 3.2. `backward2C` is the centered backward transform for residue r when $p = 2$.

Input: $\{F_k\}_{k=0}^{2m-1}, L, m, q, r$
  $H \leftarrow \lfloor L/2 \rfloor$
  $\{W_s\}_{s=m}^{2m-1} \leftarrow \texttt{ifft}(\{F_s\}_{s=m}^{2m-1})$
  for $s = m - H, \ldots, m - 1$ **do**
    $V_{H+s-m} \leftarrow \zeta_{qm}^{-r(s-m)} W_s$
  for $s = 0, \ldots, L - H - 1$ **do**
    $V_{H+s} \leftarrow \zeta_{qm}^{-rs} W_s$
  return $\{V_j\}_{j=0}^{L-1}$

4. Hermitian symmetric convolutions in one dimension. A centered array $\boldsymbol{f} = \{f_j\}_{j=-pm/2+1}^{pm/2-1}$ is Hermitian symmetric if $f_j = \overline{f_{-j}}$ (where the bar denotes complex conjugation) for all j. Due to the symmetry of the data, one only has to store approximately half of the input values, as the rest of the data can be computed (by taking the conjugate) when needed. An array is Hermitian symmetric if and only if its DFT is real valued. Because of this, Hermitian symmetric data occurs naturally in many applications, including pseudospectral methods for partial differential equations.

One can use the centered transforms from section 3 to develop Hermitian transforms. The forward transform is given by (3.2), where

$$w_{r,s} \doteq \zeta_{qm}^{rs} \left(\sum_{t=0}^{p/2-1} \zeta_q^{rt} \left[f_{tm+s} + \zeta_{2q}^{-rp} \overline{f_{pm/2-tm-s}} \right] \right). \tag{4.1}$$

Note that we only require f_j for $j \in \{0, \ldots, pm/2\}$. Furthermore, we have the Hermitian symmetry $w_{r,s} = \overline{w_{r,-s}}$ (which holds since the DFT of $\{w_{r,s}\}_{s=0}^{m-1}$ produces real-valued output), so we only need to compute $w_{r,s}$ for $s \in \{0, \ldots, \lfloor m/2 \rfloor + 1\}$, using a complex-to-real DFT to compute each residue contribution.

The inverse transform is once again given by (2.1). Here, the key difference is that because the input is real, the output is Hermitian symmetric, so that real-to-complex DFTs can be used. Pseudocode for a Hermitian convolution is given in Algorithm 2.2. Pseudocode for the Hermitian transforms when $p = 2$ is given in Algorithms 4.1 and 4.2. For $p > 2$, see subsection 6.2.

5. Multidimensional convolutions. An n-dimensional convolution is conventionally computed by performing an FFT of size $N_1 \times \cdots \times N_n$, applying the specified multiplication operator on the transformed data and then performing an inverse FFT back to the original space. However, as described in [3, 15], a better alternative is to decompose the n-dimensional convolution recursively into $\prod_{i=2}^{n} N_i$ FFTs in the first dimension, followed by N_1 convolutions of dimension $n-1$, and finally $\prod_{i=2}^{n} N_i$ inverse FFTs in the first dimension. This is illustrated in Figure 3. At the innermost level, a recursive multidimensional convolution reduces to a one-dimensional convolution.

Algorithm 4.1. `forward2H` is the Hermitian forward transform for residue r when $p = 2$.

Input: $\{f_j\}_{j=0}^{\tilde{H}}, L, m, q, r$
  $\tilde{H} \leftarrow \lceil L/2 \rceil$
  $e = \lfloor m/2 \rfloor + 1$
  for $s = 0, \ldots, m - \tilde{H}$ **do**
    $W_s \leftarrow \zeta_{qm}^{rs} f_s$
  for $s = m - \tilde{H} + 1, \ldots, e - 1$ **do**
    $W_s \leftarrow \zeta_{qm}^{rs} f_s + \zeta_{qm}^{r(s-m)} \overline{f_{m-s}}$
  $\{V_\ell\}_{\ell=0}^{m-1} \leftarrow$ `crfft`$(\{W_s\}_{s=0}^{e-1})$
  return $\{V_k\}_{k=0}^{2m-1}$

Algorithm 4.2. `backward2H` is the Hermitian backward transform for residue r when $p = 2$.

Input: $\{F_k\}_{k=0}^{2m-1}, L, m, q, r$
  $\tilde{H} \leftarrow \lceil L/2 \rceil$
  $e \leftarrow \lfloor m/2 \rfloor + 1$
  $\{W_s\}_{s=0}^{e-1} \leftarrow$ `rcfft`$(\{F_s\}_{s=0}^{m-1})$
  for $s = 0, \ldots, m - e$ **do**
    $V_s \leftarrow \zeta_{qm}^{-rs} W_s$
  for $s = m - \tilde{H} + 1, \ldots, m - e$ **do**
    $V_{m-s} \leftarrow \zeta_{qm}^{-r(m-s)} \overline{W_s}$
  if m is even **then**
    $V_{e-1} \leftarrow \zeta_{2q}^{-r} W_{e-1}$
  return $\{V_j\}_{j=0}^{\tilde{H}}$

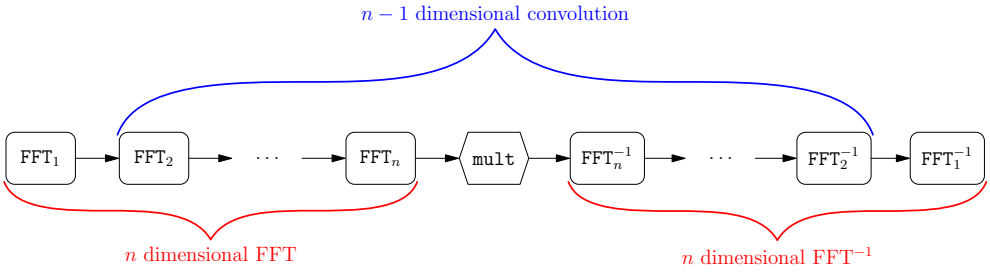

FIG. 3. *Recursive computation of an n-dimensional convolution.*

The most important advantage of decomposing a multidimensional convolution is that one can reuse the work buffer for each subconvolution, reducing the total memory footprint. These storage savings are attainable regardless of whether explicit or implicit dealiasing is used for the underlying padded FFTs.

For example, the memory management for a single-threaded two-dimensional padded complex convolution for $A = 2$ and $B = 1$ is shown in Figure 4. For each $r_x \in \{0, 1, \ldots, q_x - 1\}$, the residue contribution to the padded x FFT of the input buffers is stored in the square boxes. A padded FFT of each input 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 inverse y transform can then be performed, with the truncated result stored in the lower half of the column, next to the previously processed data shown in red. This process is repeated on the remaining columns, shifting and reusing the work buffers. Once all the columns have been processed, an inverse transform in the x direction produces the final r_x contribution to the convolution.

The reuse of subconvolution work memory allows the convolution to be computed using less total memory: for a d-dimensional p/q padded convolution, the work memory requirement is $(A + B)pmL^{d-1}$ complex words, where $p = \lceil L/m \rceil$ (not counting

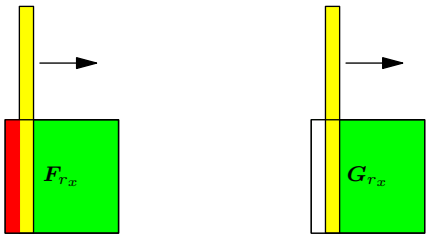

FIG. 4. *The reuse of memory to compute the contribution of a single x residue to a two-dimensional binary convolution with two inputs and one output: a one-dimensional padded y FFT is applied to columns of $\boldsymbol{F}_{r_x}$ and $\boldsymbol{G}_{r_x}$ to produce the two stacked yellow columns that are fed to the multiplication operator, producing one stacked column to be inverse y transformed into a single column (like the red one shown on the left). The upper column is then reused for processing subsequent columns.*

the storage requirements for the input data). In contrast, explicit padding requires a typically much larger buffer of size $E \doteq C(q/p)^d L^d$ words, where $C = \max(A, B)$. For example, computing a d-dimensional dealiased convolution implicitly for $A = 2$ and $B = 1$ with padding ratio $p/q = 1/2$ and $m = L \gg 1$ asymptotically requires a storage of $3E/2^{d+1}$. In particular, in one dimension, the general formulation of implicit padding requires 3/4 of the work memory required by explicit padding. For a 2/3 padding ratio, implicit padding requires a storage of $(2/3)^{d-1} E$. In addition to having reduced memory requirements, a dealiased multidimensional convolution decomposed in this way is significantly faster than a conventional implementation due to better data locality and the elimination of transforms of data known a priori to be zero.

For Hermitian-symmetric data, we assume that the origin is in the center of the unsymmetrized domain (only about half of which is retained). The outer convolutions are centered, while the innermost convolution is Hermitian. The input is assumed to be Hermitian symmetric on the hyperplane orthogonal to the innermost direction.

The FFTs in multithreaded convolutions can be parallelized by dividing the $\prod_{i=2}^{n} N_i$ one-dimensional FFTs among several threads. Similarly, the N_1 subconvolutions can be parallelized over $T \le N_1$ threads using T work buffers [15].

6. Numerical implementation. In this section, we describe several optimizations that significantly improve the performance of the underlying one-dimensional padded/unpadded FFTs.

6.1. Inner-loop optimization. Consider the complex case described in section 2. The preprocessing done in (2.1) is itself a padded FFT from size p to size q, and the postprocessing in (2.2) is an unpadded FFT from size q to size p. Thus, if p and q share a common factor, one can use these equations recursively.

We redefine q in (2.3) as the smallest positive multiple of p such that $qm \ge M$:

$$n \doteq \left\lceil \frac{M}{pm} \right\rceil, \quad q \doteq np.$$

To compute the preprocessing in (2.1), we let $r = un + v$, where $u \in \{0, \dots, p-1\}$, and $v \in \{0, \dots, n-1\}$. Then the forward transform becomes

$$F_{q\ell+un+v} = \sum_{s=0}^{m-1} \zeta_m^{\ell s} \zeta_{qm}^{(un+v)s} \sum_{t=0}^{p-1} \zeta_p^{ut} \left(\zeta_q^{vt} f_{tm+s} \right),$$

so the sum over t can be computed using n DFTs of size p. Similarly, the postprocessing in (2.2),

$$f_{tm+s} = \frac{1}{qm} \sum_{v=0}^{n-1} \zeta_{np}^{-tv} \sum_{u=0}^{p-1} \zeta_p^{-tu} \zeta_{qm}^{-s(un+v)} \sum_{\ell=0}^{m-1} \zeta_m^{-s\ell} F_{q\ell+un+v}, \tag{6.1}$$

requires the sum of n DFTs of size p. In this optimization, we consider each v to be a residue; the full transform then has n residue contributions of size pm. Note that the sum over v is *not* a DFT, as the input depends on both v and t. Pseudocode for the inner loop is given in Algorithms 6.1 and 6.2.

Algorithm 6.1. `forwardInner` is the complex forward transform for residue v when $p > 2$.

Input: $\{f_j\}_{j=0}^{L-1}, L, m, q, v$
  $p \leftarrow \lceil L/m \rceil$
  for $t = 0, \ldots, p-2$ **do**
    for $s = 0, \ldots, m-1$ **do**
      $W_{tm+s} \leftarrow \zeta_q^{vt} f_{tm+s}$
  $t_0 \leftarrow p-1$
  for $s = 0, \ldots, L - t_0 m - 1$ **do**
    $W_{t_0 m+s} \leftarrow \zeta_q^{vt_0} f_{t_0 m+s}$
  for $s = L - t_0 m, \ldots, m-1$ **do**
    $W_{t_0 m+s} \leftarrow 0$
  for $s = 0, \ldots, m-1$ **do**
    $\{W_{um+s}\}_{u=0}^{p-1} \leftarrow \texttt{fft}(\{W_{tm+s}\}_{t=0}^{p-1})$
  for $u = 0, \ldots, p-1$ **do**
    for $s = 0, \ldots, m-1$ **do**
      $W_{um+s} \leftarrow \zeta_{qm}^{(un+v)s} W_{um+s}$
  $\{F_{um+\ell}\}_{\ell=0}^{m-1} \leftarrow \texttt{fft}(\{W_{um+s}\}_{s=0}^{m-1})$
  return $\{F_k\}_{k=0}^{pm-1}$

Algorithm 6.2. `backwardInner` is the complex backward transform for residue v when $p > 2$.

Input: $\{F_k\}_{k=0}^{pm-1}, L, m, q, v$
  $p \leftarrow \lceil L/m \rceil$
  for $u = 0, \ldots, p-1$ **do**
    $\{W_{um+s}\}_{s=0}^{m-1} \leftarrow \texttt{ifft}(\{F_{um+\ell}\}_{\ell=0}^{m-1})$
  for $u = 0, \ldots, p-1$ **do**
    for $s = 0, \ldots, m-1$ **do**
      $W_{um+s} \leftarrow \zeta_{qm}^{-s(un+v)} W_{um+s}$
  for $s = 0, \ldots, m-1$ **do**
    $\{W_{tm+s}\}_{t=0}^{p-1} \leftarrow \texttt{ifft}(\{W_{um+s}\}_{u=0}^{p-1})$
  for $t = 0, \ldots, p-2$ **do**
    for $s = 0, \ldots, m-1$ **do**
      $V_{tm+s} \leftarrow \zeta_q^{-tv} W_{tm+s}$
  $t_0 \leftarrow p-1$
  for $s = 0, \ldots, L - mt_0 - 1$ **do**
    $V_{t_0 m+s} \leftarrow \zeta_q^{-t_0 v} W_{t_0 m+s}$
  return $\{V_j\}_{j=0}^{L-1}$

6.2. Inner loop for centered and Hermitian arrays. Just as we did for the complex (uncentered) case, we can apply the same recursive techniques to the centered case described in section 3. The summation in (3.3) is itself a padded DFT from size $p/2$ to size q. Therefore, if q shares a factor with $p/2$, we can use the transforms in subsection 6.1 to preform the preprocessing.

In our implementation, we consider two cases. If $p = 2$, we directly sum the two terms in $w_{r,s}$. If $p > 2$ (with p assumed to be even), we define

$$n \doteq \left\lceil \frac{2M}{pm} \right\rceil, \quad q \doteq n\frac{p}{2}.$$

Then, letting $r = un+v$, where $u \in \{0, \ldots, p/2-1\}$, and $v \in \{0, \ldots, n-1\}$, we compute

$$w_{un+v,s} = \zeta_{qm}^{(un+v)s} \sum_{t=0}^{p/2-1} \zeta_{p/2}^{ut} \left[\zeta_q^{vt} \left(f_{tm+s} + \zeta_n^{-v} f_{tm+s-pm/2} \right) \right].$$

For each value of v, each of these sums is a DFT of length $p/2$. Then, using (3.2), we compute

$$F_{q\ell+un+v} = \sum_{s=0}^{m-1} \zeta_m^{\ell s} w_{un+v,s}. \tag{6.2}$$

The inverse transform is the same as (6.1). Pseudocode for the centered case is given in Algorithms A.1 and A.2.

These equations also apply to the Hermitian case (section 4), using $f_j = \overline{f_{-j}}$ whenever $j < 0$; however, unlike the outer FFTs of length m, the preprocessing and postprocessing stages use complex-to-complex FFTs. Pseudocode for the Hermitian case is given in Algorithms A.3 and A.4.

6.3. Conjugate symmetry optimization. We can also exploit conjugate symmetries in the primitive roots. This has been used in previous implementations of implicit dealiasing [3] for centered convolutions, as discussed in section 3. Here, we extend the technique to more general situations, including $p > 2$.

First, we consider the complex case (section 2). We use the inner-loop formulation from subsection 6.1 for generality. Assuming that $v \notin \{0, n/2\}$, define

$$A_{s,t,v} \doteq \zeta_q^{vt} \mathrm{Re}\ f_{tm+s}, \quad B_{s,t,v} \doteq i\zeta_q^{vt} \mathrm{Im}\ f_{tm+s}.$$

Then we have

$$F_{q\ell+un+v} = \sum_{s=0}^{m-1} \zeta_m^{\ell s} \zeta_{qm}^{(un+v)s} \left[\sum_{t=0}^{p-1} \zeta_p^{ut} \left(A_{s,t,v} + B_{s,t,v} \right) \right],$$

$$F_{q\ell+un-v} = \sum_{s=0}^{m-1} \zeta_m^{\ell s} \zeta_{qm}^{(un-v)s} \left[\sum_{t=0}^{p-1} \zeta_p^{ut} \left(\overline{A_{s,t,v}} - \overline{B_{s,t,v}} \right) \right].$$

This allows us to compute $F_{q\ell+un+v}$ and $F_{q\ell+un-v}$ together efficiently.

A similar optimization in the centered and Hermitian cases (described in sections 3 and 4) is obtained with

$$A_{s,t,v} \doteq \zeta_q^{vt} \left(\mathrm{Re}\ f_{tm+s} + \zeta_n^{-v} \mathrm{Re}\ f_{tm+s-pm/2} \right),$$

$$B_{s,t,v} \doteq i\zeta_q^{vt} \left(\mathrm{Im}\ f_{tm+s} + \zeta_n^{-v} \mathrm{Im}\ f_{tm+s-pm/2} \right),$$

which allows us to compute (6.2) using

$$w_{un+v,s} = \zeta_{qm}^{(un+v)s} \left[\sum_{t=0}^{p/2-1} \zeta_{p/2}^{ut} \left(A_{s,t,v} + B_{s,t,v} \right) \right],$$

$$w_{un-v,s} = \zeta_{qm}^{(un-v)s} \left[\sum_{t=0}^{p/2-1} \zeta_{p/2}^{ut} \left(\overline{A_{s,t,v}} - \overline{B_{s,t,v}} \right) \right].$$

6.4. Overwrite optimization. For certain padded FFTs, where all residues are computed at once, the input array is large enough to hold all but one of the residue contributions. We have designed specialized algorithms for such cases, with one residue contribution written to the output buffer and the others stored in the input buffer.

The overwrite optimization is particularly advantageous in the complex case (section 2) when $M \leq 2L$ and $m = L$, so that $p = 1$ and $q = 2$. In this case, the input buffer already contains the preprocessed data for $r = 0$ and the preprocessed data $\zeta_{qm}^s f_s$ for $r = 1$ is written to the output buffer. The input and output buffers are then individually Fourier transformed to obtain the required residues. The backwards transform

does the reverse operation: it first performs inverse Fourier transforms on the input and output buffers and then adds products of the output buffer and roots of unity to the input buffer.

To use the overwrite optimization for computing a one-dimensional convolution, the number of inputs A must be at least as large as the number of outputs B. Under this same restriction, the overwrite optimization can be implemented for each dimension of a multidimensional convolution.

6.5. Loop optimizations. If the overwrite optimization is not applicable, other data flow improvements may be possible. Suppose that we compute a block of D residues at a time as suggested in subsection 2.2. Normally, the contribution to the inverse padded Fourier transform from the block containing residue 0 is stored in an accumulation buffer; the contributions from the remaining residues are then added to this buffer by iterating over the other residue blocks. If there are no other residue blocks, a separate accumulation buffer is not needed; one can accumulate the residues entirely within the input buffer.

Another optimization is possible when $A > B$ and there are exactly two residue blocks, which we label 0 and r. In this case, we compute all A forward-padded FFT contributions to residue block 0 in an output buffer F and apply the multiplication operator, freeing up the storage in F associated with $A-B$ inputs. We then transform the contributions to residue block r, $A-B$ inputs at a time, each time writing $A-B$ inverse-transformed contributions from residue block 0 to the input buffer. Once all A contributions to residue block r have been forward transformed, we apply the multiplication operator and accumulate the contributions from the inverse transform in the input buffer.

7. Numerical results. This work presents several algorithms for computing padded FFTs. Which algorithm is optimal for a given problem depends on the number of inputs A and outputs B, the multiplication operator `mult`, the input data length L, the padded length M, the number of copies C of the transform to be computed simultaneously, and the stride S between successive data elements of each copy. We only consider the efficiently packed case, where the distance in memory between the first elements of each copy is one. Vectorized and parallelized C++ versions of these algorithms have been implemented in the open-source library `FFTW++` [4].

We determine the fastest algorithm for a given problem empirically, scanning over the underlying FFT size m, the number D of residues to be computed at a time, and whether or not to use in-place or out-of-place FFTs. These parameters then determine the values of p and q to use in our padded FFT algorithms.

Our optimizer measures the time required to compute a one-dimensional in-place dealiased convolution for a particular set of parameters, using the given multiplication routine. As described in section 5, multidimensional convolutions are decomposed into a sequence of padded FFTs, a one-dimensional convolution, and then a sequence of inverse padded FFTs. We assume that the padded/unpadded FFT pairs can be optimized independently in each dimension. This is accomplished by performing one-dimensional convolutions using each padded FFT pair, without calling the multiplication routine. In practice, this decoupling works well and leads to efficient optimization of multidimensional convolutions. If the outermost pair FFT_1 and FFT_1^{-1} in Figure 3 are to be multithreaded over T threads, optimization of the remaining FFTs should be performed over T concurrent copies, to simulate the execution environment.

We benchmarked our algorithms with a liquid-cooled Intel i9-12900K processor (5.2 GHz, 8 performance cores) on an ASUS ROG Strix Z690-F motherboard with

128 GB of DDR5 memory (5 GHz), using version 12.2.1 of the `GCC` compiler with the optimizations `-Ofast -fomit-frame-pointer -fstrict-aliasing -ffast-math`. The underlying FFTs were computed with version 3.3.10 of the adaptive `FFTW` [7, 8] library under the Fedora 37 operating system. Multithreading was implemented with the `OpenMP` library.

7.1. One-dimensional convolutions. In Figure 5, we plot median execution times (normalized to $L\log_2 L$) for one-dimensional in-place convolutions of L complex words (with $M = 2L$) over one thread for explicit zero padding using in-place (IP) or out-of-place (OP) FFTs, implicit dealiasing [15], and hybrid dealiasing. We see that hybrid dealiasing is much faster than both explicit and implicit dealiasing for large sizes and is generally competitive with optimized out-of-place explicit algorithms (which it reduces to) for small sizes. In Figure 6, we plot the normalized times for the same one-dimensional convolutions parallelized over 8 threads. In these plots, only power-of-two sizes are benchmarked since these are optimal FFT sizes.

In Figures 7 and 8, we plot the normalized times for one-dimensional Hermitian convolutions of size L padded to $M = 3L/2$ over 1 thread and 8 threads, respectively. We benchmark hybrid dealiasing separately for those sizes that are optimal for explicit dealiasing and for implicit dealiasing. For the implicit dealiasing algorithms developed in [3, 15], the optimal sizes are one less than a power of two. In our implementation of hybrid dealiasing, we normally adjust these sizes to exact powers of two to allow us to use the overwrite optimization. For explicit dealiasing, the optimal values of L are $2\lfloor\frac{2^n+2}{3}\rfloor - 1$ for positive integers n. We observe that hybrid dealiasing outperforms implicit dealiasing at optimal implicit sizes and performs about as well as explicit dealiasing at optimal explicit sizes.

7.2. Two-dimensional convolutions. In Figures 9 and 10, we plot the normalized median times for two-dimensional complex convolutions of size $L \times L$. On a single thread, these benchmarks show that at all sizes, hybrid dealiasing is faster than implicit dealiasing and much faster than explicit dealiasing. On 8 threads, hybrid dealiasing outperforms both methods, except at $L = 64$. In both cases, an x stride of $L + 2$ was used.

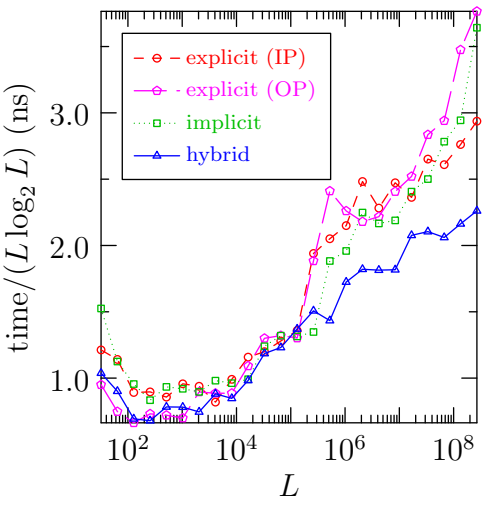

FIG. 5. *In-place one-dimensional complex convolutions of length L with $A = 2$ and $B = 1$ on 1 thread.*

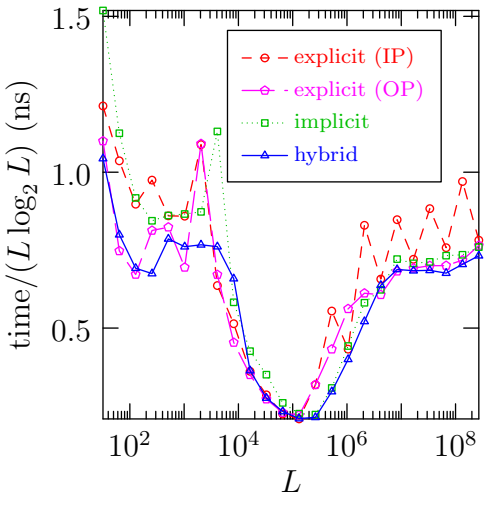

FIG. 6. *In-place one-dimensional complex convolutions of length L with $A = 2$ and $B = 1$ on 8 threads.*

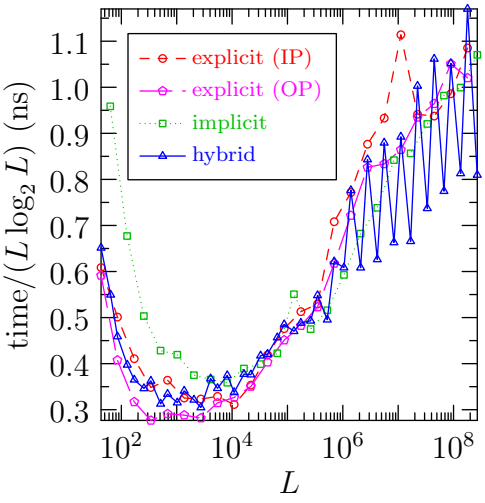

FIG. 7. *In-place one-dimensional Hermitian convolutions of length L with $A = 2$ and $B = 1$ on 1 thread.*

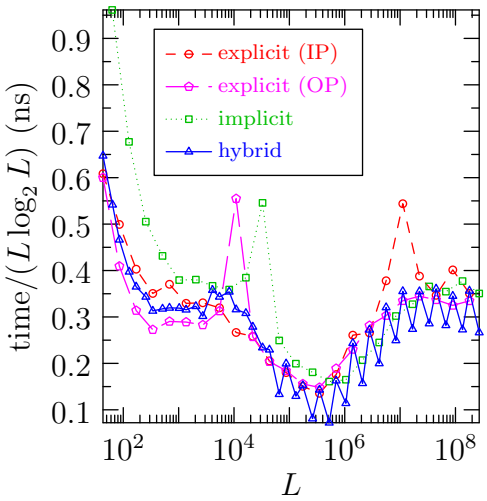

FIG. 8. *In-place one-dimensional Hermitian convolutions of length L with $A = 2$ and $B = 1$ on 8 threads.*

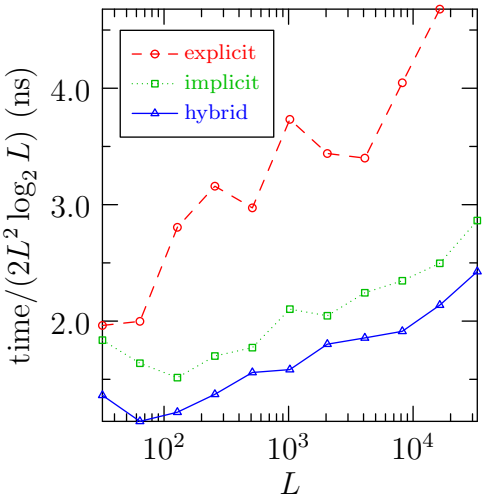

FIG. 9. *In-place two-dimensional complex convolutions of size $L \times L$ with $A = 2$ and $B = 1$ on 1 thread.*

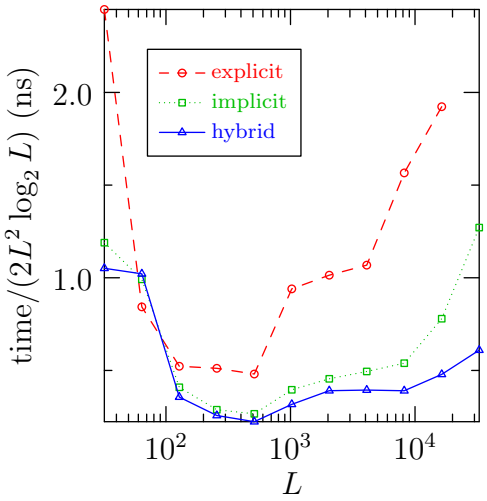

FIG. 10. *In-place two-dimensional complex convolutions of size $L \times L$ with $A = 2$ and $B = 1$ on 8 threads.*

In Figures 11 and 12, which record normalized median times for two-dimensional Hermitian convolutions of size $L \times L$, we see that hybrid dealiasing outperforms both implicit and explicit dealiasing for all sizes. In both cases we used an x stride of $\lceil \frac{L}{2} \rceil + 2$.

To illustrate the sometimes counterintuitive optimal parameters, in the single-threaded case with $L = 2048$, the optimizer chose the parameters $m = 16$, $p = 128$, $q = 192$, and $D = 1$ in the x direction and performed the transform in place (using the inner-loop optimization), whereas in the y direction, out-of-place explicit dealiasing ($m = 3072$ and $p = q = D = 1$) was optimal.

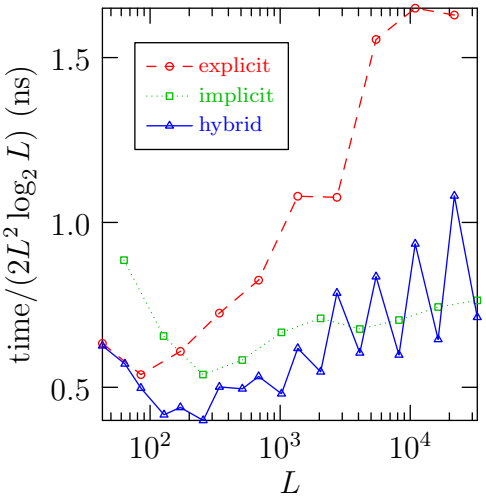

FIG. 11. *In-place two-dimensional Hermitian convolutions of size $L \times L$ with $A = 2$ and $B = 1$ on 1 thread.*

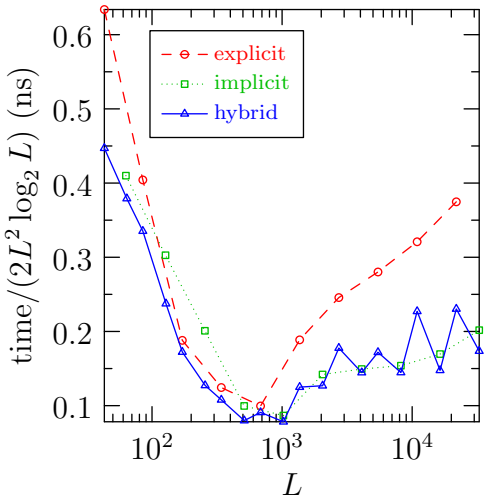

FIG. 12. *In-place two-dimensional Hermitian convolutions of size $L \times L$ with $A = 2$ and $B = 1$ on 8 threads.*

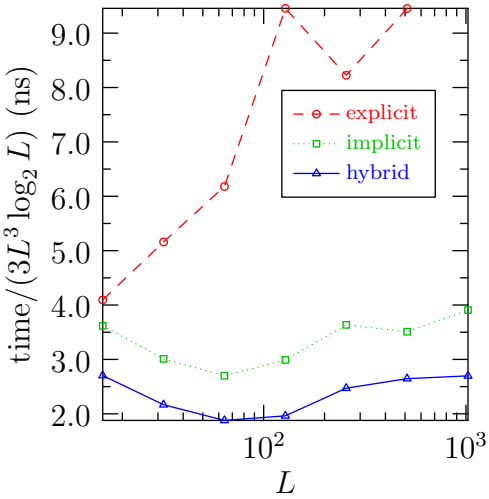

FIG. 13. *In-place three-dimensional complex convolutions of size $L \times L \times L$ with $A = 2$ and $B = 1$ on 1 thread.*

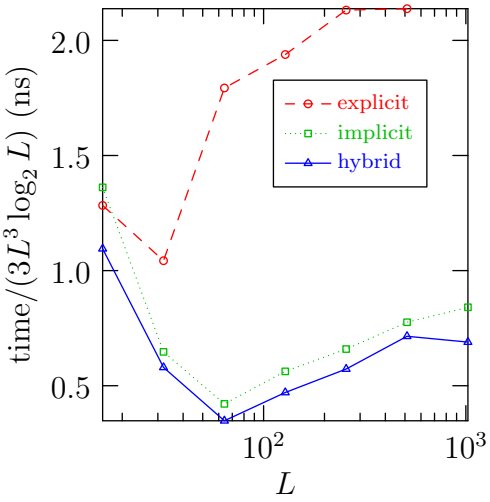

FIG. 14. *In-place three-dimensional complex convolutions of size $L \times L \times L$ with $A = 2$ and $B = 1$ on 8 threads.*

7.3. Three-dimensional convolutions. In Figures 13 and 14, we benchmark three-dimensional complex convolutions. In the single-threaded case, we used the y stride $S_y = L + 2$ and the x stride $S_y L + 2$; in the multithreaded-threaded case, we used the y stride $S_y = L$ and the x stride $S_y L + 4$. Again, we observe that hybrid dealiasing outperforms the other two methods.

In Figure 17, we emphasize that, unlike explicit and implicit dealiasing, hybrid dealiasing performs consistently well over a range of arbitrary sizes.

In Figure 15, we observe that for three-dimensional Hermitian convolutions on a single thread, hybrid dealiasing performs better than implicit dealiasing and much better than explicit dealiasing, except at $L = 21$. As seen in Figure 16, when run on

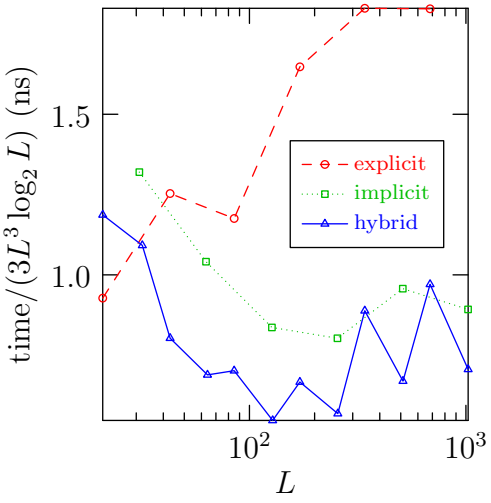

FIG. 15. *In-place three-dimensional Hermitian convolutions of size $L \times L \times L$ with $A = 2$ and $B = 1$ on 1 thread.*

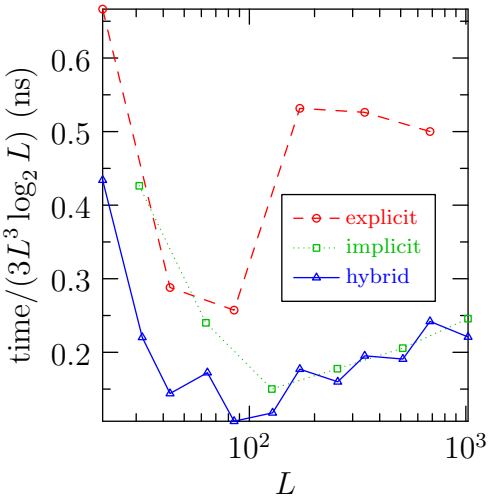

FIG. 16. *In-place three-dimensional Hermitian convolutions of size $L \times L \times L$ with $A = 2$ and $B = 1$ on 8 threads.*

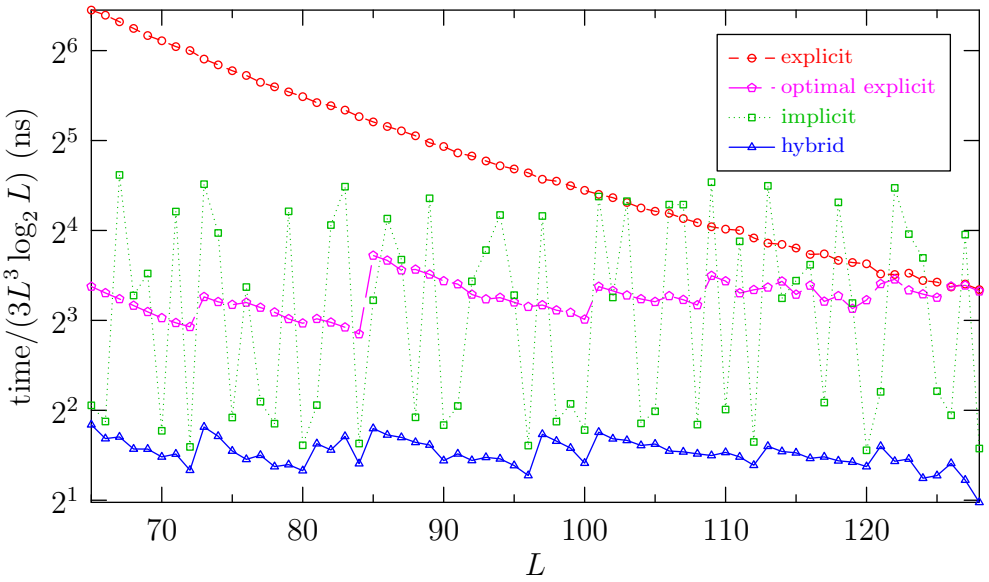

FIG. 17. *Semi-log plot of three-dimensional in-place complex convolution timings for incremental sizes $L \times L \times L$ with $A = 2$ and $B = 1$ on 1 thread. As is common practice, explicit dealiasing pads past $2L$ up to the next power of two (in this case 256). Optimal explicit dealiasing zero pads beyond $2L$ to the empirically determined optimal size.*

8 threads, hybrid dealiasing is much faster than the other methods for all sizes. In the single-threaded case, we used the y stride $S_y = \lceil \frac{L}{2} \rceil + 2$ and the x stride $S_y L + 2$; in the multithreaded-threaded case, we used the y stride $S_y = L$ and the x stride $S_y L + 4$.

8. Future work. Many applications rely on real convolutions: signal processing, image processing, and convolutional neural networks. FFT-based convolutions are widely used in signal and image processing and are beginning to be used to accelerate the training of convolutional neural networks [12, 10, 11, 5]. The algorithms described in this work operate on complex data and are not optimal for the real input data used in such applications. Moreover, many of these applications involve convolving input data with a relatively small kernel. Because kernel sizes are typically much smaller than the arrays with which they are convolved, the majority of the padded kernel is made up of zeros. In future work, we will specialize hybrid dealiasing to real arrays of different lengths and develop a general toolkit for diverse applications.

One of the strengths of our formulation for computing padded/unpadded DFTs is that we decompose them into full DFTs, facilitating the use of existing FFT algorithms. Hybrid dealiasing does not attempt to compete with, but rather harness, modern FFT libraries.

Unfortunately, the real-to-complex case is not as simple as the complex and Hermitian cases described in this work. For Hermitian symmetric data (section 4), each residue contribution is real, as it is part of a larger real array and can be computed using a complex-to-real DFT. For real data, the contribution of any residue other than zero is not, in general, Hermitian symmetric. Therefore, one cannot simply compute the residue contribution using a real-to-complex DFT. While one could pack the real data into a smaller complex array and use the algorithms described in this paper, such schemes are less efficient than algorithms designed with real data in mind [16].

9. Conclusion. This work combines several ideas to implement efficient dealiased convolutions for arbitrary padding ratios. Hybrid dealiasing combines the techniques of implicit and explicit dealiasing to exploit optimal transform sizes of the underlying FFT library. By exploiting smaller FFT sizes than used in [3], hybrid dealiasing can store the required roots of unity in short unfactorized tables. The recognition that the preprocessing in (2.1) is itself a padded FFT is crucial to an efficient implementation for general padding ratios.

Hybrid dealiasing is ideally suited for implementing pseudospectral simulations of partial differential equations. For example, the three-dimensional incompressible Navier–Stokes equations can be implemented with the Basdevant formulation using $A = 3$ and $B = 5$, while the two-dimensional formulation uses $A = B = 2$ [1]. Likewise, the incompressible three-dimensional magnetohydrodynamic equations can be implemented with $A = 6$ and $B = 8$.[1] The generality of the formulation described in this paper can also be applied to the cascade direction of nth-order Casimir invariants of two-dimensional turbulence, such as $\sum_j \omega^n(x_j)$, where ω is the scalar vorticity [2]; this requires a padding ratio of $2/(n+1)$.

The naive way of exploiting the convolution theorem is to take the Fourier transform of the entire input data, perform the necessary multiplication, and then take the inverse transform. However, if a convolution is all that is needed, we have seen that better performance can be achieved by localizing the computations; one can compute

[1] Here we correct an error in [15]; only $A + B = 14$ FFT calls are required.

the contribution to the convolution one residue at a time. Furthermore, multidimensional convolutions can be done more efficiently by decomposing them into an outer FFT, a lower-dimensional convolution, and an inverse FFT. The possibility of reusing work memory in this recursive formulation is responsible for most of the dramatic performance gains that are observed in two and three dimensions.

Appendix A. Centered and Hermitian inner-loop pseudocode. Pseudocode for computing centered padded/unpadded FFTs with the inner-loop optimization (subsection 6.2) is presented in Algorithms A.1 and A.2. The Hermitian case is documented in Algorithms A.3 and A.4.

Algorithm A.1. `forwardInnerC` is the centered complex forward transform for residue v when $p > 2$. Here, δ_t is the Kronecker $\delta_{t,0}$.

Input: $\{f_j\}_{j=0}^{L-1}, L, m, q, v$
  $H \leftarrow \lfloor L/2 \rfloor$
  $p_2 \leftarrow \lceil L/(2m) \rceil$
  $m_0 \leftarrow p_2 m - H$
  $m_1 \leftarrow L - H - (p_2 - 1)m$
  for $s = 0, \ldots, m_0 - 1$ **do**
    $W_s \leftarrow f_{H+s}$
  for $t = 0, \ldots, p_2 - 1$ **do**
    $m_2 \leftarrow (m_1 - m)\delta_{t-(p_2-1)} + m$
    for $s = m_0\delta_t, \ldots, m_2 - 1$ **do**
      $W_{tm+s} \leftarrow \zeta_q^{v(t-p_2)} f_{(t-p_2)m+H+s} + \zeta_q^{vt} f_{tm+H+s}$
  for $s = m_1, \ldots, m - 1$ **do**
    $W_{(p_2-1)m+s} \leftarrow \zeta_q^{-v} f_{-m+H+s}$
  for $s = 0, \ldots, m - 1$ **do**
    $\{W_{um+s}\}_{u=0}^{p_2-1} \leftarrow \texttt{fft}(\{W_{tm+s}\}_{t=0}^{p_2-1})$
  for $u = 0, \ldots, p_2 - 1$ **do**
    for $s = 1, \ldots, m - 1$ **do**
      $W_{um+s} \leftarrow \zeta_{qm}^{(un+v)s} W_{um+s}$
    $\{V_{um+\ell}\}_{\ell=0}^{m-1} \leftarrow \texttt{fft}(\{W_{um+s}\}_{s=0}^{m-1})$
  return $\{V_k\}_{k=0}^{p_2 m-1}$

Algorithm A.2. `backwardInnerC` is the centered complex backward transform for residue v when $p > 2$. Here, δ_t is the Kronecker $\delta_{t,0}$.

Input: $\{F_k\}_{k=0}^{pm/2-1}, L, m, q, v$
  $H \leftarrow \lfloor L/2 \rfloor$
  $p_2 \leftarrow \lceil L/(2m) \rceil$
  $m_0 \leftarrow p_2 m - H$
  $m_1 \leftarrow L - H - (p_2 - 1)m$
  for $u = 0, \ldots, p_2 - 1$ **do**
    $\{W_{um+s}\}_{s=0}^{m-1} \leftarrow \texttt{ifft}(\{F_{um+\ell}\}_{\ell=0}^{m-1})$
    for $s = 1, \ldots, m - 1$ **do**
      $W_{um+s} \leftarrow \zeta_{qm}^{-(un+v)s} W_{um+s}$
  for $s = 0, \ldots, m - 1$ **do**
    $\{W_{tm+s}\}_{t=0}^{p_2-1} \leftarrow \texttt{ifft}(\{W_{um+s}\}_{u=0}^{p_2-1})$
  for $s = 0, \ldots, m_0 - 1$ **do**
    $V_{H+s} \leftarrow W_s$
  for $t = 0, \ldots, p_2 - 1$ **do**
    $m_2 \leftarrow (m_1 - m)\delta_{t-(p_2-1)} + m$
    for $s = m_0\delta_t, \ldots, m_2 - 1$ **do**
      $V_{(t-p_2)m+H+s} \leftarrow \zeta_q^{-v(t-p_2)} W_{tm+s}$
      $V_{tm+H+s} \leftarrow \zeta_q^{-vt} W_{tm+s}$
  for $s = m_1, \ldots, m - 1$ **do**
    $V_{-m+H+s} \leftarrow \zeta_q^{v} W_{(p_2-1)m+s}$
  return $\{V_j\}_{j=0}^{L-1}$

Algorithm A.3. `forwardInnerH` is the Hermitian forward transform for residue v when $p > 2$. Here, δ_t is the Kronecker $\delta_{t,0}$.

Input: $\{f_j\}_{j=0}^{\tilde{H}}, L, m, q, v$
$\tilde{H} \leftarrow \lceil L/2 \rceil$
$e \leftarrow \lfloor m/2 \rfloor + 1$
$p_2 \leftarrow \lceil L/(2m) \rceil$
$n \leftarrow q/p_2$
$m_0 \leftarrow \min\left(p_2 m - \tilde{H} + 1, e\right)$
for $s = 0, \ldots, m_0 - 1$ **do**
  $W_s \leftarrow f_s$
for $t = 0, \ldots, p_2 - 1$ **do**
  for $s = m_0\delta_t, \ldots, e - 1$ **do**
    $W_{te+s} \leftarrow \zeta_q^{v(t-p_2)}\overline{f_{(p_2-t)m-s}} + \zeta_q^{vt} f_{tm+s}$
for $s = 0, \ldots, e - 1$ **do**
  $\{W_{ue+s}\}_{u=0}^{p_2-1} \leftarrow \texttt{fft}(\{W_{te+s}\}_{t=0}^{p_2-1})$
for $u = 0, \ldots, p_2 - 1$ **do**
  for $s = 1, \ldots, e - 1$ **do**
    $W_{ue+s} \leftarrow \zeta_{qm}^{(un+v)s} W_{ue+s}$
  $\{V_{um+\ell}\}_{\ell=0}^{m-1} \leftarrow \texttt{crfft}(\{W_{ue+s}\}_{s=0}^{e-1})$
return $\{V_k\}_{k=0}^{p_2 m-1}$

Algorithm A.4. `backwardInnerH` is the Hermitian backward transform for residue v when $p > 2$. Here, δ_t is the Kronecker $\delta_{t,0}$.

Input: $\{F_k\}_{k=0}^{pm/2-1}, L, m, q, v$
$\tilde{H} \leftarrow \lceil L/2 \rceil$
$e \leftarrow \lfloor m/2 \rfloor + 1$
$p_2 \leftarrow \lceil L/(2m) \rceil$
$n \leftarrow q/p_2$
$m_0 \leftarrow \min\left(p_2 m - \tilde{H} + 1, e\right)$
for $u = 0, \ldots, p_2 - 1$ **do**
  $\{W_{ue+s}\}_{s=0}^{e-1} \leftarrow \texttt{rcfft}(\{F_{um+\ell}\}_{\ell=0}^{m-1})$
  for $s = 1, \ldots, e - 1$ **do**
    $W_{ue+s} \leftarrow \zeta_{qm}^{-(un+v)s} W_{ue+s}$
for $s = 0, \ldots, e - 1$ **do**
  $\{W_{te+s}\}_{t=0}^{p_2-1} \leftarrow \texttt{ifft}(\{W_{ue+s}\}_{u=0}^{p_2-1})$
for $s = 1, \ldots, m_0 - 1$ **do**
  $V_s \leftarrow W_s$
  for $t = 0, \ldots, p_2 - 1$ **do**
    $V_{tm} \leftarrow \zeta_q^{-vt} W_{te}$
  for $s = (m_0 - 1)\delta_t + 1, \ldots, m - e$ **do**
    $V_{tm+s} \leftarrow \zeta_q^{-vt} W_{te+s}$
    $V_{(p_2-t)m-s} \leftarrow \zeta_q^{-v(p_2-t)}\overline{W_{te+s}}$
  if m is even **then**
    for $t = 0, \ldots, p_2$ **do**
      $V_{tm+e-1} \leftarrow \zeta_q^{-vt} W_{e(t+1)-1}$
return $\{V_j\}_{j=0}^{\tilde{H}}$

Reproducibility of computational results. This paper has been awarded the "SIAM Reproducibility Badge: Code and Data Available" as a recognition that the authors have followed reproducibility principles valued by SISC and the scientific computing community. Code and data that allow readers to reproduce the results in this paper are available from https://github.com/dealias/fftwpp and in the supplementary materials.

REFERENCES

[1] C. Basdevant, *Technical improvements for direct numerical simulation of homogeneous three-dimensional turbulence*, J. Comput. Phys., 50 (1983), pp. 209–214.
[2] J. C. Bowman, *Casimir cascades in two-dimensional turbulence*, J. Fluid Mech., 729 (2013), pp. 364–376.
[3] J. C. Bowman and M. Roberts, *Efficient dealiased convolutions without padding*, SIAM J. Sci. Comput., 33 (2011), pp. 386–406, https://doi.org/10.1137/100787933.
[4] J. C. Bowman, M. Roberts, and N. Murasko, *FFTW++: A Fast Fourier Transform C++ Header Class for the FFTW3 Library*, http://fftwpp.sourceforge.net, 2023.
[5] K. Chitsaz, M. Hajabdollahi, N. Karimi, S. Samavi, and S. Shirani, *Acceleration of Convolutional Neural Network Using FFT-Based Split Convolutions,* preprint, https://arxiv.org/abs/2003.12621, 2020.

[6] J. W. Cooley and J. W. Tukey, *An algorithm for the machine calculation of complex Fourier series*, Math. Comp., 19 (1965), pp. 297–301.
[7] M. Frigo and S. G. Johnson, *FFTW*, http://www.fftw.org.
[8] M. Frigo and S. G. Johnson, *The design and implementation of FFTW3*, Proc. IEEE, 93 (2005), pp. 216–231.
[9] D. Gottlieb and S. A. Orszag, *Numerical Analysis of Spectral Methods: Theory and Applications*, CBMS-NSF Regional Conf. Ser. in Appl. Math. 26, SIAM, Philadelphia, 1977, https://doi.org/10.1137/1.9781611970425.
[10] T. Highlander and A. Rodriguez, *Very efficient training of convolutional neural networks using fast Fourier transform and overlap-and-add*, in Proc. British Machine Vision Conference, X. H. Xie, M. W. Jones, and G. K. L. Tam, eds., 2015, pp. 160.1–160.9, https://doi.org/10.48550/arXiv.1601.06815.
[11] J. Lin and Y. Yao, *A fast algorithm for convolutional neural networks using tile-based fast Fourier transforms*, Neural Process. Lett., 50 (2019), pp. 1951–1967, https://doi.org/10.1007/s11063-019-09981-z.
[12] M. Mathieu, M. Henaff, and Y. LeCun, *Fast training of convolutional networks through FFTs*, in 2nd International Conference on Learning Representations, Y. Bengio and Y. LeCun, eds., Conference Track Proceedings, 2014, https://doi.org/10.48550/arXiv.1312.5851.
[13] S. A. Orszag, *On the elimination of aliasing in finite difference schemes by filtering high-wavenumber components*, J. Atmos. Sci., 28 (1971), 1074.
[14] G. S. Patterson, Jr., and S. A. Orszag, *Spectral calculations of isotropic turbulence: Efficient removal of aliasing interactions*, Phys. Fluids, 14 (1971), pp. 2538–2541.
[15] M. Roberts and J. C. Bowman, *Multithreaded implicitly dealiased convolutions*, J. Comput. Phys., 356 (2018), pp. 98–114.
[16] H. V. Sorensen, D. L. Jones, M. T. Heideman, and C. S. Burrus, *Real-valued fast Fourier transform algorithms*, IEEE Trans. Acoust. Speech Signal Process., 35 (1987), pp. 849–863.