

HYBRID DEALIASING AND IMPLICIT PACKING FOR REAL CONVOLUTIONS

NOEL MURASKO* AND JOHN C. BOWMAN*

Abstract. Hybrid dealiasing is an FFT-based method for computing linear convolutions of complex-valued data that reduces the cost of dealiasing by performing zero padding implicitly. We develop two new algorithms that extend hybrid dealiasing to real-valued convolutions.

The first algorithm exploits conjugate symmetries in the transformed data and computes each residue contribution directly. The second algorithm employs complex-valued hybrid dealiasing via a new *implicit packing* technique, which packs real-valued data into complex-valued arrays and performs all unpacking and packing operations implicitly in the transformed domain. Multidimensional versions of both algorithms are obtained by recursive decomposition into lower-dimensional convolutions.

Based on these algorithms, efficient routines to compute convolutions of equal-length input arrays are implemented in the open-source `FFTW++` library. These routines outperform the standard method of explicit zero padding in one, two, and three dimensions. In many cases, implicit packing without hybrid dealiasing is nearly as fast as the general hybrid dealiasing algorithm, particularly when multithreading.

Key words. dealiasing, hybrid dealiasing, implicit packing, real-valued convolution, discrete Fourier transform, implicit dealiasing, zero padding, fast Fourier transform

AMS subject classifications. 65T50, 65Y05

1. Introduction. The fast Fourier transform (FFT) can be used to compute discrete circular convolutions efficiently; however, many applications require linear convolutions. A naive application of FFTs to linear convolutions produces data polluted by aliasing errors arising from the assumed periodicity. The process of removing these aliases is called *dealiasing*.

The standard dealiasing method explicitly zero pads the input arrays; we call this *explicit dealiasing*. While effective, explicit dealiasing requires reading and multiplying values known *a priori* to be zero. This inefficiency worsens with dimension: explicitly dealiasing a d -dimensional convolution of two arrays of equal size requires approximately $(2^d - 1)/2^d$ of the inputs to be zero.

An alternative is *implicit dealiasing* [1, 14], which constructs padded and unpadded FFTs to account for known zeros implicitly, eliminating the need for explicit padding. Implicit dealiasing often outperforms explicit dealiasing, with the greatest gains in higher dimensions. However, the original formulation was limited, as it did not handle arbitrary padding ratios (the ratio of output FFT size to input size) or convolutions of real-valued data.

Hybrid dealiasing improves upon implicit dealiasing by combining explicit and implicit zero padding. In [12], we developed hybrid dealiasing algorithms for complex and Hermitian-symmetric arrays, generalizing implicit dealiasing to arbitrary padding ratios. These algorithms were implemented in version 3 of `FFTW++` [2] and were shown to match or outperform implicit dealiasing in one, two, and three dimensions. A summary is given in [section 2](#).

Here we extend hybrid dealiasing to efficiently compute convolutions of real-valued data via two new algorithms. The first ([section 3](#)) is analogous to the Hermitian-symmetric algorithm of [12]: the convolution is decomposed into *residues* (see [subsection 2.1](#)), and each residue contribution is computed by exploiting conjugate sym-

*University of Alberta (murasko@ualberta.ca, bowman@ualberta.ca)

metries in the Fourier transformed data.

The second algorithm (section 4) packs real-valued data into complex arrays of half the size, enabling use of the efficient complex hybrid dealiasing algorithms of [12]. Standard packing methods require an expensive unpacking and packing step in Fourier space. We avoid this via *implicit packing*, which performs these operations implicitly through the multiplication operator.

In section 5, we extend both algorithms to multidimensional convolutions using the decomposition of [12]. This decomposition computes a multidimensional convolution recursively, yielding substantial memory savings for large arrays. Extending the implicit packing algorithm is particularly involved, as the modified multiplication operator is not an element-wise operation.

Both algorithms are implemented in version 3 of FFTW++ in one, two, and three dimensions. We benchmark them in section 6 against explicitly dealiased convolutions using real-to-complex/complex-to-real FFTs. Both hybrid-dealiased algorithms outperform explicit dealiasing, with implicit packing typically being fastest. We also compare our algorithms to a routine using implicit packing with explicit dealiasing, which is competitive in some cases.

In section 7 we discuss potential future work, including application to convolutional neural networks.

2. A recap of complex hybrid dealiasing. In this section, we summarize the basic ideas of hybrid dealiasing in one dimension (see [12, §2] for details). Given a one-dimensional complex array $\mathbf{f} = \{f_j\}_{j=0}^{L-1}$ that must be zero padded to at least length $M \geq L$ before computing its DFT $\mathbf{F} \doteq \{F_k\}_{k=0}^M$ (we denote DFTs with upper-case letters and definitions by $\doteq$), we choose a parameter $m \in \mathbb{N}$ and set $p \doteq \lceil L/m \rceil$ and $q \doteq \lceil M/m \rceil$. We explicitly zero pad $\mathbf{f}$ from length L to pm , then compute the implicitly zero-padded DFT of length qm .

More specifically, we reindex $\mathbf{f}$ and $\mathbf{F}$ using

$$j = tm + s, \quad t = 0, \dots, p-1, \quad s = 0, \dots, m-1,$$

and

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

For $N \in \mathbb{N}$, let $\zeta_N \doteq e^{2\pi i/N}$. Then, following the Cooley–Tukey decomposition [4], the implicitly zero-padded DFT of $\mathbf{f}$ is given by

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

and the inverse is given by

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

These transforms require q FFTs of length m . We note that the parameter m can be optimized to minimize the total convolution cost.

2.1. One residue at a time. For each *residue* $r \in \{0, \dots, q-1\}$, define the *residue contribution* $\mathbf{F}_r \doteq \{F_{q\ell+r}\}_{\ell=0}^{m-1}$. When unambiguous, we use *residue* as shorthand for *residue contribution*; for example, computing residue r means computing $\mathbf{F}_r$.

Hybrid-dealiased convolutions can be computed one residue at a time [12, §2.2]. For complex-valued data this is primarily a memory-saving optimization; for the real algorithm of section 3 it is essential, since not all residue contributions are computed identically.

From (2.1), $\mathbf{F}_r$ is computed by applying an FFT of length m to

$$\zeta_{qm}^{rs} \sum_{t=0}^{p-1} \zeta_q^{rt} f_{tm+s}.$$

For the inverse, we apply another FFT of length m to obtain $\mathbf{h}_r = \{h_{r,j}\}_{j=0}^{pm-1}$ where

$$(2.3) \quad h_{r,tm+s} \doteq \zeta_q^{-tr} \zeta_{qm}^{-sr} \sum_{\ell=0}^{m-1} \zeta_m^{-s\ell} F_{q\ell+r}.$$

Following (2.2), we accumulate over residues to obtain the inverse

$$(2.4) \quad \mathbf{f} = \frac{1}{qm} \sum_{r=0}^{q-1} \mathbf{h}_r.$$

Each residue contribution to the implicitly-padded complex DFTs in (2.1) and (2.2) requires an FFT of length m , which can be computed by an existing library such as FFTW [5, 6].

2.2. Hermitian symmetric and real arrays. A periodic array $\mathbf{f} = \{f_j\}_{j=0}^{L-1}$ with period L is Hermitian symmetric if $f_j = \overline{f_{L-j}}$ (where the bar denotes complex conjugation) for all $j \in \{0, \dots, L-1\}$. This symmetry implies that the *DC mode* f_0 is real; if L is even, the *Nyquist mode* $f_{L/2}$ is also real.

It is easy to show that an array is Hermitian symmetric if and only if its DFT is real. Since real-valued data appears in many applications, considerable effort has been devoted to efficient real-to-complex and complex-to-real FFTs.

As in the complex case, hybrid-dealiased Hermitian-symmetric convolutions can be computed one residue at a time using complex-to-real/real-to-complex FFTs of length m [12, §4]. The argument is straightforward: if $\mathbf{f}$ is Hermitian symmetric, its DFT $\mathbf{F} = \{F_k\}_{k=0}^{qm-1}$ is real. For any $r \in \{0, \dots, q-1\}$, the residue contribution $\mathbf{F}_r = \{F_{q\ell+r}\}_{\ell=0}^{m-1}$ is also real (being a sub-array of $\mathbf{F}$), so $\mathbf{F}_r$ must be the DFT of a Hermitian symmetric array.

Unfortunately, this approach does not extend to implicitly-padded real transforms. If $\mathbf{g} = \{g_j\}_{j=0}^{qm-1}$ is real, then $\mathbf{G} = \{G_k\}_{k=0}^{qm-1}$ is Hermitian symmetric. For any $r \in \{1, \dots, q-1\}$, the residue contribution $\{G_{q\ell+r}\}_{\ell=0}^{m-1}$ is not generally Hermitian symmetric (at $\ell = 0$, the DC mode $\mathbf{G}_r$ need not be real), so it cannot be the DFT of a real array. Consequently, real hybrid dealiasing cannot rely on real-to-complex/complex-to-real FFTs.

3. Real hybrid dealiasing via conjugate symmetries. In this section, we assume that $\mathbf{f} = \{f_j\}_{j=0}^{pm-1}$ is real-valued. Unless stated otherwise, the notation and indices of section 2 apply throughout this section.

3.1. Exploiting conjugate symmetries. Although real-to-complex/complex-to-real FFTs cannot be used directly, we can exploit conjugate symmetries in the data. By Hermitian symmetry, the complex conjugate of $F_{q\ell+r}$ satisfies

$$\overline{F_{q\ell+r}} = F_{qm-(q\ell+r)} = F_{q(m-\ell)-r},$$

so (2.3) satisfies

$$\begin{aligned}
 \overline{h_{r,tm+s}} &= \zeta_q^{tr} \zeta_{qm}^{sr} \sum_{\ell=0}^{m-1} \zeta_m^{s\ell} \overline{F_{q\ell+r}} = \zeta_q^{tr} \zeta_{qm}^{sr} \sum_{\ell=0}^{m-1} \zeta_m^{s\ell} F_{q(m-\ell)-r} \\
 &= \zeta_q^{-t(q-r)} \zeta_{qm}^{-s(q-r)} \sum_{\ell=0}^{m-1} \zeta_m^{-s\ell} F_{q\ell+q-r} = h_{q-r,tm+s},
 \end{aligned}
 \tag{3.1}$$

where the penultimate equality follows from $\ell \mapsto m-1-\ell$. This yields

$$\mathbf{f} = \frac{1}{qm} \left(\mathbf{h}_0 + 2 \sum_{r=1}^{\lceil q/2 \rceil - 1} \operatorname{Re} \mathbf{h}_r + \mathbf{h}_{q/2} \right),
 \tag{3.2}$$

where we take $\mathbf{h}_{q/2} \doteq \mathbf{0}$ when q is odd. Equation (3.2) forms the basis of the real hybrid dealiasing algorithm. There are three cases to consider.

3.1.1. Case $\mathbf{h}_0$. This case is straightforward: since $\sum_{t=0}^{p-1} f_{tm+s}$ is real, the forward transform

$$F_{q\ell} = \sum_{s=0}^{m-1} \zeta_m^{\ell s} \sum_{t=0}^{p-1} f_{tm+s}$$

can be computed with a real-to-complex FFT of length m . Similarly, (2.3) becomes

$$h_{0,tm+s} = \sum_{\ell=0}^{m-1} \zeta_m^{-s\ell} F_{q\ell},$$

which requires a complex-to-real FFT of length m .

3.1.2. Case $\mathbf{h}_r$, $r \in \{1, \dots, \lceil q/2 \rceil - 1\}$. This case is also straightforward; since (3.2) does not require $\mathbf{h}_{q-r}$, this formulation uses complex FFTs of length m without sacrificing efficiency.

3.1.3. Case $\mathbf{h}_{q/2}$. This case requires more care and arises only when q is even. Simplifying (2.1), the residue $\mathbf{F}_{q/2}$ is given by

$$F_{q\ell+q/2} = \sum_{s=0}^{m-1} \zeta_m^{\ell s} \zeta_{2m}^s \sum_{t=0}^{p-1} (-1)^t f_{tm+s}.
 \tag{3.3}$$

Since $\zeta_{2m}^s \sum_{t=0}^{p-1} (-1)^t f_{tm+s}$ is not real, a real-to-complex FFT (as used for $r=0$) is inapplicable. A full complex FFT of length m would also be wasteful.

The key observation is that $\mathbf{F}_{q/2}$ is *almost*¹ Hermitian symmetric: the $q/2$ residue contribution is closed under conjugation:

$$\overline{F_{q\ell+q/2}} = F_{qm-(q\ell+q/2)} = F_{q(m-1-\ell)+q/2}.
 \tag{3.4}$$

That is, the conjugates of the even indices of $\mathbf{F}_{q/2}$ coincide with its odd indices. We restrict m to be even, let $e \doteq m/2$, and reindex s :

$$s = ae + b, \quad a \in \{0, 1\}, \quad b \in \{0, \dots, e-1\}.$$

¹True Hermitian symmetry requires $\overline{F_{q\ell+q/2}} = F_{q(m-\ell)+q/2}$.

Only the even² indices of $\mathbf{F}_{q/2}$ need to be computed. For $c = 0, \dots, e-1$, the even indices of (3.3) are

$$(3.5) \quad F_{2cq+q/2} = \sum_{b=0}^{e-1} \zeta_e^{cb} \zeta_{2m}^b \sum_{t=0}^{p-1} (-1)^t (f_{tm+b} + i f_{tm+e+b}).$$

To compute the inverse, define

$$(3.6) \quad w_b \doteq \zeta_{2m}^{-b} \sum_{c=0}^{e-1} \zeta_e^{-bc} F_{2cq+q/2}.$$

Ignoring the twiddle factors, $\mathbf{w}$ is essentially the inverse DFT of (3.5), and can be computed with an FFT of length e . To recover $\mathbf{h}_{q/2}$ from $\mathbf{w}$, apply (3.4) to the conjugate of (3.6):

$$\begin{aligned} \overline{w_b} &= \zeta_{2m}^b \sum_{c=0}^{e-1} \zeta_e^{bc} \overline{F_{2cq+q/2}} = \zeta_{2m}^b \sum_{c=0}^{e-1} \zeta_e^{bc} F_{(2e-1-2c)q+q/2} \\ &= \zeta_{2m}^b \sum_{c=0}^{e-1} \zeta_e^{-b(c+1)} F_{(2c+1)q+q/2} = \zeta_{2m}^{-3b} \sum_{c=0}^{e-1} \zeta_e^{-bc} F_{(2c+1)q+q/2}, \end{aligned}$$

where the penultimate equality follows from $c \mapsto e-1-c$. Then (2.3) becomes

$$\begin{aligned} h_{q/2, tm+ae+b} &= (-1)^t \zeta_{2m}^{-(ae+b)} \sum_{\ell=0}^{m-1} \zeta_m^{-(ae+b)\ell} F_{q\ell+q/2} \\ &= (-1)^t i^{-a} \zeta_{2m}^{-b} \left[\sum_{c=0}^{e-1} \zeta_e^{-bc} F_{2cq+q/2} + (-1)^a \zeta_m^{-b} \sum_{c=0}^{e-1} \zeta_e^{-bc} F_{(2c+1)q+q/2} \right] \\ &= (-1)^t i^{-a} [w_b + (-1)^a \overline{w_b}]. \end{aligned}$$

Considering $a = 0$ and $a = 1$ separately gives us

$$(3.7) \quad h_{q/2, tm+b} = (-1)^t 2 \operatorname{Re} w_b, \quad h_{q/2, tm+e+b} = (-1)^t 2 \operatorname{Im} w_b,$$

and so we can compute $\mathbf{h}_{q/2}$ using a complex FFT of length e .

3.2. Summary. The algorithm described in this section accumulates residue contributions via (3.2) as follows:

- When $r = 0$, use a real-to-complex/complex-to-real FFT of length m .
- When $r \in \{1, \dots, \lceil q/2 \rceil - 1\}$, use a complex FFT of length m , simultaneously yielding the contributions for residues r and $q-r$.
- When $r = q/2$, require that m be even and use a complex FFT of length $m/2$.

Pseudocode for the forward and backward transforms is given in algorithms 3.1 and 3.2 for the case $p = 1$. The case $p = 2$ appears in [11, Algorithms 14 and 15]. For $p > 2$, we apply an optimization that computes the preprocessing and postprocessing using FFTs of length p , analogous to [12, §6.1]; see appendix A for details.

²The choice of even indices is arbitrary; the odd indices yield identical results.

Algorithm 3.1 forward1R is the real forward transform for residue r when $p = 1$.

Input: $\{f_j\}_{j=0}^{L-1}, L, m, q, r$
 $e \leftarrow \lfloor m/2 \rfloor$
if $r < q/2$ **then**
 for $s = 0, \dots, L-1$ **do**
 $W_s \leftarrow \zeta_{qm}^{rs} f_s$
 for $s = L, \dots, m-1$ **do**
 $W_s \leftarrow 0$
 if $r = 0$ **then**
 $\{V_\ell\}_{\ell=0}^{e+1} \leftarrow \text{rcfft}(\{W_s\}_{s=0}^{m-1})$ **return**
 $\{V_k\}_{k=0}^{e+1}$
 else
 $\{V_\ell\}_{\ell=0}^{m-1} \leftarrow \text{fft}(\{W_s\}_{s=0}^{m-1})$ **return**
 $\{V_k\}_{k=0}^{m-1}$
else if $r = q/2$ **then**
 $B_1 \leftarrow \max(0, L-e)$
 $B_2 \leftarrow \min(e, L)$
 for $b = 0, \dots, B_1-1$ **do**
 $W_b \leftarrow \zeta_{2m}^b (f_b + i f_{e+b})$
 for $b = B_1, \dots, B_2-1$ **do**
 $W_b \leftarrow \zeta_{2m}^b f_b$
 for $b = L, \dots, e-1$ **do**
 $W_b \leftarrow 0$
 $\{V_c\}_{c=0}^{e-1} \leftarrow \text{fft}(\{W_b\}_{b=0}^{e-1})$
 return $\{V_k\}_{k=0}^{e-1}$

Algorithm 3.2 backward1R is the real backward transform for residue r when $p = 1$. The normalization factor $1/(qm)$ from (3.2) would be applied during the final accumulation step.

Input: $\{F_k\}_{k=0}^{2m-1}, L, m, q, r$
 $e \leftarrow \lfloor m/2 \rfloor$
if $r = 0$ **then**
 $\{W_s\}_{s=0}^{m-1} \leftarrow \text{crfft}(\{F_\ell\}_{\ell=0}^{e+1})$
else if $r < \lceil q/2 \rceil$ **then**
 $\{W_s\}_{s=0}^{m-1} \leftarrow \text{ifft}(\{F_\ell\}_{\ell=0}^{m-1})$
 for $s = 0, \dots, L-1$ **do**
 $W_s \leftarrow 2 \text{Re} \zeta_{qm}^{-rs} W_s$
else
 $\{W_b\}_{b=0}^{e-1} \leftarrow \text{ifft}(\{F_c\}_{c=0}^{e-1})$
 $B_1 \leftarrow \max(0, L-e)$
 $B_2 \leftarrow \min(e, L)$
 for $b = 0, \dots, B_1-1$ **do**
 $W_{e+b} \leftarrow 2 \text{Im} \zeta_{2m}^{-b} W_b$
 $W_b \leftarrow 2 \text{Re} \zeta_{2m}^{-b} W_b$
 for $b = B_1, \dots, B_2-1$ **do**
 $W_b \leftarrow 2 \text{Re} \zeta_{2m}^{-b} W_b$
 return $\{W_s\}_{s=0}^{L-1}$

4. Real hybrid dealiasing via implicit packing. Our second algorithm packs real-valued data into a complex array of half the size. Let $M \in \mathbb{N}$. For a real array $\mathbf{f} \doteq \{f_j\}_{j=0}^{2M-1}$, define its *complex packing* $\tilde{\mathbf{f}} = \{\tilde{f}_j\}_{j=0}^{M-1}$ by

$$(4.1) \quad \tilde{f}_j = f_{2j} + i f_{2j+1}.$$

That is, the even and odd indices of $\mathbf{f}$ form the real and imaginary parts of $\tilde{\mathbf{f}}$, respectively. Let $\tilde{\mathbf{F}}$ denote the complex DFT (of length M) of $\tilde{\mathbf{f}}$; this notation applies throughout the section. For convenience, we also define $\tilde{F}_M \doteq \tilde{F}_0$. After computing $\tilde{\mathbf{F}}$, one can use symmetries to recover $\mathbf{F}$ (the DFT of $\mathbf{f}$); indeed, the *unpacking* formula in Fourier space is

$$(4.2) \quad F_k = \frac{1}{2} \left(\tilde{F}_k + \overline{\tilde{F}_{M-k}} \right) - \frac{i}{2} \left(\tilde{F}_k - \overline{\tilde{F}_{M-k}} \right) \zeta_{2M}^k, \quad k \in \{0, \dots, M\},$$

with inverse

$$(4.3) \quad \tilde{F}_k = \frac{1}{2} (F_k + \overline{F_{M-k}}) + \frac{i}{2} (F_k - \overline{F_{M-k}}) \zeta_{2M}^{-k}, \quad k \in \{0, \dots, M-1\}.$$

Since $\mathbf{F}$ is Hermitian symmetric, values of F_k for $k \in \{M+1, \dots, 2M-1\}$ need not be computed. For details of the derivation of (4.2) and (4.3), see [13].

Complex packing enables the computation of real DFTs via complex DFTs of half the length. For hybrid dealiasing this would permit use of the complex convolution

algorithms of [12, §2]. However, despite the practical efficiency of complex FFTs over real-to-complex/complex-to-real variants, explicit packing and unpacking in Fourier space is costly: packing-based DFT algorithms for real inputs require “more additions than a specialized algorithm for real input data” [15]. However, since our objective is the computation of a convolution rather than a DFT, we can perform the packing implicitly via a custom multiplication operator.

Let $\mathbf{g} \doteq \{g_j\}_{j=0}^{2M-1}$ be another real array, $\tilde{\mathbf{g}}$ its complex packing, and $\tilde{\mathbf{G}}$ the DFT of $\tilde{\mathbf{g}}$. Using (4.2) and (4.3), define the *Fourier unpacking operator* $U : \tilde{\mathbf{F}} \mapsto \mathbf{F}$ and its inverse $U^{-1} : \mathbf{F} \mapsto \tilde{\mathbf{F}}$. We then define the real convolution multiplication (RCM) operator $\boxtimes$ to be the binary operation on $\mathbb{C}^M$ given by

$$(4.4) \quad \tilde{\mathbf{F}} \boxtimes \tilde{\mathbf{G}} \doteq U^{-1}(U(\tilde{\mathbf{F}}) \odot U(\tilde{\mathbf{G}})),$$

where $\odot$ denotes element-wise multiplication. By the convolution theorem, $\tilde{\mathbf{F}} \boxtimes \tilde{\mathbf{G}}$ is the DFT of the complex packing of $\mathbf{f} * \mathbf{g}$. Crucially, $\tilde{\mathbf{F}} \boxtimes \tilde{\mathbf{G}}$ can be computed without explicitly computing $\mathbf{F}$ and $\mathbf{G}$. Indeed, expanding and simplifying (4.4) gives

$$(4.5) \quad (\tilde{\mathbf{F}} \boxtimes \tilde{\mathbf{G}})_k \doteq \tilde{F}_k \tilde{G}_k - \frac{1}{4} \left(\tilde{F}_k - \overline{\tilde{F}_{M-k}} \right) \left(\tilde{G}_k - \overline{\tilde{G}_{M-k}} \right) (1 + \zeta_M^k),$$

for $k \in \{0, \dots, M-1\}$. The derivation of (4.5) is straightforward but lengthy; for completeness we include it in [appendix B](#).

The principal difficulty with (4.5) is that it is not an element-wise operator: computing $(\tilde{\mathbf{F}} \boxtimes \tilde{\mathbf{G}})_k$ requires $\tilde{F}_k, \tilde{G}_k, \tilde{F}_{M-k}$, and $\tilde{G}_{M-k}$. This imposes the constraint that at least two residues must be computed simultaneously. Using the framework of [section 2](#), our goal is to compute $(\tilde{\mathbf{F}} \boxtimes \tilde{\mathbf{G}})_{q\ell+r}$. Since $M-k = qm - q\ell - r = q(m-\ell-1) + q-r$, define

$$W_{\ell,r} \doteq \frac{1}{4} \left(\tilde{F}_{q\ell+r} - \overline{\tilde{F}_{q(m-\ell-1)+q-r}} \right) \left(\tilde{G}_{q\ell+r} - \overline{\tilde{G}_{q(m-\ell-1)+q-r}} \right) (1 + \zeta_{qm}^{q\ell+r}),$$

so that

$$(4.6) \quad (\tilde{\mathbf{F}} \boxtimes \tilde{\mathbf{G}})_{q\ell+r} = \tilde{F}_{q\ell+r} \tilde{G}_{q\ell+r} - W_{\ell,r}.$$

Computing $W_{\ell,r}$ requires residues r and $q-r$. The cases $r=0$ and $r=q/2$ are treated together, as they are self-conjugate.³ For $r \notin \{0, q/2\}$, one easily verifies that $\overline{W_{\ell,r}} = W_{m-\ell-1, q-r}$, so that on mapping $\ell \mapsto m-\ell-1$ and $r \mapsto q-r$, (4.6) becomes

$$(4.7) \quad (\tilde{\mathbf{F}} \boxtimes \tilde{\mathbf{G}})_{q(m-\ell-1)+q-r} = \tilde{F}_{q(m-\ell-1)+q-r} \tilde{G}_{q(m-\ell-1)+q-r} - \overline{W_{\ell,r}}.$$

Equations (4.6) and (4.7) allow simultaneous computation of residues r and $q-r$. This formulation also enables the multiplication to be performed in-place.

4.1. Implicitly packed convolutions of more than two arrays. Although binary convolution is the most common case, some applications require convolving more than two arrays. The implicitly-packed multiplication operator extends from (4.5). Let $\mathbf{f}^1, \dots, \mathbf{f}^n$ be real arrays of length $2M$. For $j = 1, \dots, n$, let $\tilde{\mathbf{f}}^j$ be the complex packing of $\mathbf{f}^j$ (as in (4.1)) and let $\tilde{\mathbf{F}}^j$ be the DFT of $\tilde{\mathbf{f}}^j$. We first see that the RCM operator is associative:

$$\begin{aligned} (\tilde{\mathbf{F}}^1 \boxtimes \tilde{\mathbf{F}}^2) \boxtimes \tilde{\mathbf{F}}^3 &= U^{-1}(U(\tilde{\mathbf{F}}^1) \odot U(\tilde{\mathbf{F}}^2)) \boxtimes \tilde{\mathbf{F}}^3 = U^{-1}(U(\tilde{\mathbf{F}}^1) \odot U(\tilde{\mathbf{F}}^2) \odot U(\tilde{\mathbf{F}}^3)) \\ &= \tilde{\mathbf{F}}^1 \boxtimes U^{-1}(U(\tilde{\mathbf{F}}^2) \odot U(\tilde{\mathbf{F}}^3)) = \tilde{\mathbf{F}}^1 \boxtimes (\tilde{\mathbf{F}}^2 \boxtimes \tilde{\mathbf{F}}^3). \end{aligned}$$

³That is, $(q-r) \bmod q = r$ when $r=0$ or $r=q/2$.

By the convolution theorem, the DFT of the complex packing of $\mathbf{f}^1 * \dots * \mathbf{f}^n$ is then

$$U^{-1}(U(\tilde{\mathbf{F}}^1) \odot \dots \odot U(\tilde{\mathbf{F}}^n)) = \tilde{\mathbf{F}}^1 \boxdot \dots \boxdot \tilde{\mathbf{F}}^n.$$

Efficient computation therefore reduces to evaluating $\tilde{\mathbf{F}}^1 \boxdot \dots \boxdot \tilde{\mathbf{F}}^n$.

This may be computed using a recursive formula. Define $\mathbf{R}^1 \doteq \tilde{\mathbf{F}}^1$, and for $j \in \{2, \dots, n\}$, define $\mathbf{R}^j \doteq \tilde{\mathbf{F}}^1 \boxdot \dots \boxdot \tilde{\mathbf{F}}^j$. One can compute $\mathbf{R}^j$ using $\mathbf{R}^{j-1}$ and $\tilde{\mathbf{F}}^j$. For $k = 0, \dots, M-1$, define

$$(4.8) \quad W_k^j \doteq \frac{1}{4} \left(R_k^{j-1} - \overline{R_{M-k}^{j-1}} \right) \left(\tilde{F}_k^j - \overline{\tilde{F}_{M-k}^j} \right) (1 + \zeta_M^k).$$

It is easy to check that $W_{M-k}^j = \overline{W_k^j}$. This allows simultaneous computation of

$$(4.9) \quad R_k^j = (\mathbf{R}^{j-1} \boxdot \tilde{\mathbf{F}}^j)_k = R_k^{j-1} \tilde{F}_k^j - W_k^j,$$

and

$$(4.10) \quad R_{M-k}^j = R_{M-k}^{j-1} \tilde{F}_{M-k}^j - \overline{W_{M-k}^j}.$$

4.2. Comparison with explicit packing. We compare the floating-point operation (flop) counts of implicit and explicit packing (using (4.2) and (4.3) directly). Consider convolving n arrays $\tilde{\mathbf{F}}^1, \dots, \tilde{\mathbf{F}}^n$, each of length M . For simplicity, we assume that M is even.

For explicit packing, let $k = 1, \dots, M/2 - 1$ (the indices $k = 0, M/2$ contribute only $O(n)$ terms). For each $j = 1, \dots, n$, compute

$$A_k^j \doteq \frac{1}{2} \left(\tilde{F}_k^j + \overline{\tilde{F}_{M-k}^j} \right), \quad B_k^j \doteq \frac{i}{2} \left(\tilde{F}_k^j - \overline{\tilde{F}_{M-k}^j} \right) \zeta_{2M}^k.$$

Computing A_k^j requires 1 complex addition (2 flops) and 1 real-complex multiplication (2 flops). With the constant factor $\frac{i}{2} \zeta_{2M}^k$ precomputed, B_k^j requires 1 complex addition (2 flops) and 1 complex multiplication (6 flops). From (4.2),

$$F_k^j = A_k^j - B_k^j, \quad F_{M-k}^j = \overline{A_k^j + B_k^j}.$$

Assuming conjugation is free, this adds two complex additions (4 flops). Computing F_k^j and F_{M-k}^j thus requires 16 flops per pair. With approximately $M/2$ pairs and n arrays, computing $\mathbf{F}^1, \dots, \mathbf{F}^n$ requires approximately $8nM$ flops. Multiplying these arrays requires $(n-1)M$ complex multiplications ($6(n-1)M$ flops). Applying (4.3) to the product requires another $8M$ flops. The total is approximately $2M(7n+1)$ flops.

For implicit packing, each step $j = 2, \dots, n$ computes W_k^j (as in (4.8)) using 2 complex multiplications (12 flops) and 2 complex additions (4 flops). Computing R_k^j and R_{M-k}^j via (4.9) and (4.10) adds 2 more complex multiplications (12 flops) and 2 more complex additions (4 flops), for 32 flops per pair. With approximately $M/2$ pairs processed $n-1$ times, the total is approximately $16M(n-1)$ flops.

For a binary convolution ($n = 2$), implicit packing requires approximately half as many flops as explicit packing. However, its flop count scales worse with n : for $n > 9$, it exceeds that of explicit packing. Of course, this analysis uses the recursive formula of subsection 4.1; it is possible that a more optimal formula may reduce the count for large n .

5. Multidimensional real convolutions. To compute multidimensional real convolutions, we use the decomposition of [1, 14].

An n -dimensional complex convolution is conventionally computed using complex FFTs of size $N_1 \times \dots \times N_n$. Instead, we 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 reduces a multidimensional convolution to one-dimensional convolutions at the innermost level, permitting buffer reuse across subconvolutions and substantially reducing memory usage. For more details on this algorithm, see [12, §5].

This decomposition applies directly to the algorithms of section 3. The only difference is that the outermost DFT must handle real values; all subconvolutions are complex⁴ and use the hybrid-dealiased convolutions of [12].

In contrast, applying the decomposition to the implicit packing algorithms of section 4 is much more involved. Let $M_1, \dots, M_n \in \mathbb{N}$. For this section we use indices $k_1, \dots, k_n$ with $0 \leq k_j < M_j$ for $j = 1, \dots, n-1$, and write $k_j^* \doteq (M_j - k_j) \bmod M_j$. The multi-index is $\mathbf{k} \doteq (k_1, \dots, k_n)$ with conjugate $\mathbf{k}^* \doteq (k_1^*, \dots, k_n^*)$.

Unless stated otherwise, all n -dimensional arrays are complex and of size $M_1 \times \dots \times M_n$. For any such array $\mathbf{A}$ and $0 \leq k_1 < M_1$, define the $(n-1)$ -dimensional slice $\mathbf{A}^{k_1}$ by

$$(5.1) \quad A_{(k_2, \dots, k_n)}^{k_1} \doteq A_{k_1, k_2, \dots, k_n}.$$

Let $\mathbf{f}$ and $\mathbf{g}$ be real n -dimensional arrays of size $M_1 \times \dots \times M_{n-1} \times 2M_n$. Their complex packings $\tilde{\mathbf{f}}$ and $\tilde{\mathbf{g}}$ are obtained by packing along the last dimension:

$$(5.2) \quad \tilde{f}_{j_1, \dots, j_n} \doteq f_{j_1, \dots, 2j_n} + i f_{j_1, \dots, 2j_n+1}, \quad \tilde{g}_{j_1, \dots, j_n} \doteq g_{j_1, \dots, 2j_n} + i g_{j_1, \dots, 2j_n+1},$$

where $0 \leq j_i < M_i$ for $i = 1, \dots, n$. Let $\tilde{\mathbf{F}}$ and $\tilde{\mathbf{G}}$ denote the n -dimensional DFTs of $\tilde{\mathbf{f}}$ and $\tilde{\mathbf{g}}$, respectively. It is straightforward to show that the RCM operator (4.5) generalizes to

$$(5.3) \quad (\tilde{\mathbf{F}} \square_n \tilde{\mathbf{G}})_{\mathbf{k}} \doteq \tilde{F}_{\mathbf{k}} \tilde{G}_{\mathbf{k}} - \frac{1}{4} (\tilde{F}_{\mathbf{k}} - \overline{\tilde{F}_{\mathbf{k}^*}}) (\tilde{G}_{\mathbf{k}} - \overline{\tilde{G}_{\mathbf{k}^*}}) (1 + \zeta_{M_n}^{k_n}).$$

Equation (5.3) shows why implicit packing complicates the multidimensional decomposition: computing $(\tilde{\mathbf{F}} \square_n \tilde{\mathbf{G}})_{\mathbf{k}}$ requires elements indexed by both $\mathbf{k}$ and $\mathbf{k}^*$. To handle this, we define a generalized operator. For any n -dimensional arrays $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{D}$, let $\mathcal{C}_n(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$ be the array with entries

$$(\mathcal{C}_n(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}))_{\mathbf{k}} = A_{\mathbf{k}} B_{\mathbf{k}} - \frac{1}{4} (A_{\mathbf{k}} - \overline{C_{\mathbf{k}^*}}) (B_{\mathbf{k}} - \overline{D_{\mathbf{k}^*}}) (1 + \zeta_{M_n}^{k_n}).$$

One easily verifies that

$$(\mathcal{C}_n(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}))_{(k_1, k_2, \dots, k_n)} = (\mathcal{C}_{n-1}(\mathbf{A}^{k_1}, \mathbf{B}^{k_1}, \mathbf{C}^{k_1^*}, \mathbf{D}^{k_1^*}))_{(k_2, \dots, k_n)}.$$

By construction, $(\tilde{\mathbf{F}} \square_n \tilde{\mathbf{G}}) = \mathcal{C}_n(\tilde{\mathbf{F}}, \tilde{\mathbf{G}}, \tilde{\mathbf{F}}, \tilde{\mathbf{G}})$. For $0 \leq k_1 \leq M_1/2$, we compute $\tilde{\mathbf{F}}^{k_1}$, $\tilde{\mathbf{F}}^{k_1^*}$, $\tilde{\mathbf{G}}^{k_1}$, and $\tilde{\mathbf{G}}^{k_1^*}$, which yields both

$$(\tilde{\mathbf{F}} \square_n \tilde{\mathbf{G}})_{(k_1, k_2, \dots, k_n)} = (\mathcal{C}_{n-1}(\tilde{\mathbf{F}}^{k_1}, \tilde{\mathbf{G}}^{k_1}, \tilde{\mathbf{F}}^{k_1^*}, \tilde{\mathbf{G}}^{k_1^*}))_{(k_2, \dots, k_n)},$$

⁴This contrasts with the Hermitian-symmetric case, where only the innermost FFTs need to be complex-to-real/real-to-complex.

278 and

$$279 \quad (\tilde{F} \boxtimes_n \tilde{G})_{(k_1^*, k_2, \dots, k_n)} = (\mathcal{C}_{n-1}(\tilde{F}^{k_1^*}, \tilde{G}^{k_1^*}, \tilde{F}^{k_1}, \tilde{G}^{k_1}))_{(k_2, \dots, k_n)}.$$

280 These contributions are handled together. Applying this recursively reduces n -dimensional
 281 implicitly packed convolutions to a sequence of 1-dimensional implicitly packed con-
 282 volutions.

283 Intuitively, the first two arguments of $\mathcal{C}_{n-1}$ correspond to the *bottom half* of the
 284 array and the last two to the *top half*. Figure 1 illustrates this in two dimensions.

285 Pseudocode for one- and two-dimensional implicitly packed convolutions is given
 286 in algorithms 5.1 and 5.3, both using the common multiplication routine algorithm 5.2.
 287 For simplicity, the pseudocode considers only explicit zero padding, but may be ex-
 288 tended to the hybrid dealiasing algorithms of [12].

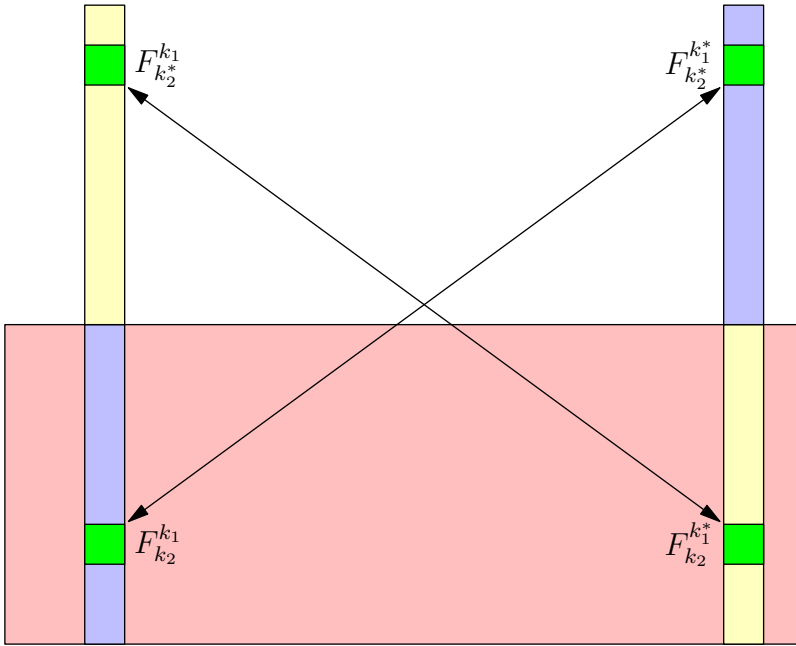


FIG. 1. Relative positions of conjugate modes in 2D implicit packing. The DFT is first computed along the horizontal dimension. For each k_1 , two 1D implicitly packed convolutions are computed along the vertical dimension: the blue array (bottom left and top right) serves as input to the first convolution and the yellow array (top left and bottom right) to the second.

289 **6. Numerical results.** To benchmark the above algorithms, we follow [12, §7],
 290 tuning the parameters empirically by scanning over the FFT size m , the number D
 291 of residues computed simultaneously, and the choice between in-place and out-of-
 292 place FFTs. For multidimensional convolutions we assume that the parameters for
 293 each dimension can be optimized independently. We consider only power-of-two sizes
 294 (optimal for FFTs), even though hybrid dealiasing performs well for arbitrary sizes
 295 (see [12, Fig. 17]).

296 The benchmarks use FFTW++ [2], which contains vectorized and parallelized C++
 297 implementations of these algorithms. We benchmark on an Intel i9-12900K proces-
 298 sor (5.2 GHz, 8 performance cores) with an ASUS ROG Strix Z690-F motherboard

Algorithm 5.1 `convolveRCM1` returns an explicitly dealiased and implicitly packed one-dimensional convolution of two real arrays $\mathbf{f}$ and $\mathbf{g}$. It makes use of routines `zeroPad`, which performs the necessary explicit zero padding, as well as `complexPack` and `complexUnPack`, which pack and unpack the real data as in (4.1). The multiplication is done using [algorithm 5.2](#).

Input: $\mathbf{f}, \mathbf{g}$

```

 $\tilde{\mathbf{f}} \leftarrow \text{complexPack}(\text{zeroPad}(\mathbf{f}))$ 
 $\tilde{\mathbf{g}} \leftarrow \text{complexPack}(\text{zeroPad}(\mathbf{g}))$ 
 $\tilde{\mathbf{F}} \leftarrow \text{fft}(\tilde{\mathbf{f}})$ 
 $\tilde{\mathbf{G}} \leftarrow \text{fft}(\tilde{\mathbf{g}})$ 
 $\text{multRCM}(\tilde{\mathbf{F}}, \tilde{\mathbf{G}}, \tilde{\mathbf{F}}, \tilde{\mathbf{G}}, \text{True})$ 
 $\tilde{\mathbf{h}} \leftarrow \text{ifft}(\tilde{\mathbf{F}})$ 
 $\mathbf{h} \leftarrow \text{complexUnPack}(\tilde{\mathbf{h}})$ 
return  $\mathbf{h}$ 

```

Algorithm 5.2 `multRCM` is an in-place implicit packing multiplication routine. It takes four length M one-dimensional complex arrays $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ (where we assume M is even), and a boolean value `first`. Output is written to $\mathbf{A}$ and $\mathbf{C}$. This routine makes use of pre-computed values $Z_k = (1 + \zeta_M^k)/4$ where $k = 1, \dots, M/2 - 1$.

Input: $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}, \text{first}$

```

if first then
   $\alpha \leftarrow \frac{1}{2} (A_0 - \overline{C_0}) (B_0 - \overline{D_0})$ 
   $A_0 \leftarrow A_0 B_0 - \alpha$ 
  if not( $\mathbf{A} = \mathbf{C}$  or  $\mathbf{B} = \mathbf{D}$ ) then
     $C_0 \leftarrow C_0 D_0 - \overline{\alpha}$ 
  for  $k = 1, \dots, M/2 - 1$  do
     $\alpha \leftarrow (A_k - \overline{C_{M-k}}) (B_k - \overline{D_{M-k}}) Z_k$ 
     $A_k \leftarrow A_k B_k - \alpha$ 
     $C_{M-k} \leftarrow C_{M-k} D_{M-k} - \overline{\alpha}$ 
 $A_{M/2} \leftarrow A_{M/2} B_{M/2}$ 

```

Algorithm 5.3 `convolveRCM2` returns an explicitly dealiased and implicitly packed two-dimensional convolution of two real arrays $\mathbf{f}$ and $\mathbf{g}$. It makes use of routines `zeroPad`, which explicitly zero pads the arrays to size $M_1 \times 2M_2$, as well as `complexPack` and `complexUnPack`, which pack and unpack the real data along the last dimension as in (5.2). The routine `fftx` computes FFTs along the first dimension. We use the notation defined in (5.1) for one-dimensional slices. The multiplication is done using [algorithm 5.2](#).

Input: $\mathbf{f}, \mathbf{g}$

```

 $\tilde{\mathbf{f}} \leftarrow \text{complexPack}(\text{zeroPad}(\mathbf{f}))$ 
 $\tilde{\mathbf{g}} \leftarrow \text{complexPack}(\text{zeroPad}(\mathbf{g}))$ 
 $\tilde{\mathbf{F}} \leftarrow \text{fftx}(\tilde{\mathbf{f}})$ 
 $\tilde{\mathbf{G}} \leftarrow \text{fftx}(\tilde{\mathbf{g}})$ 
for  $k_1 = 0, \dots, M_1/2 - 1$  do
   $k_1^* \leftarrow M_1 - k_1$ 
  if  $k_1 = 0$  then
     $k_1^* \leftarrow M_1/2$ 
     $\tilde{\mathbf{F}}^{k_1} \leftarrow \text{ffty}(\tilde{\mathbf{F}}^{k_1})$ 
     $\tilde{\mathbf{G}}^{k_1} \leftarrow \text{ffty}(\tilde{\mathbf{G}}^{k_1})$ 
     $\tilde{\mathbf{F}}^{k_1^*} \leftarrow \text{ffty}(\tilde{\mathbf{F}}^{k_1^*})$ 
     $\tilde{\mathbf{G}}^{k_1^*} \leftarrow \text{ffty}(\tilde{\mathbf{G}}^{k_1^*})$ 
  if  $k_1 = 0$  then
     $\text{multRCM}(\tilde{\mathbf{F}}^{k_1}, \tilde{\mathbf{G}}^{k_1}, \tilde{\mathbf{F}}^{k_1}, \tilde{\mathbf{G}}^{k_1}, \text{True})$ 
     $\text{multRCM}(\tilde{\mathbf{F}}^{k_1^*}, \tilde{\mathbf{G}}^{k_1^*}, \tilde{\mathbf{F}}^{k_1^*}, \tilde{\mathbf{G}}^{k_1^*}, \text{True})$ 
  else
     $\text{multRCM}(\tilde{\mathbf{F}}^{k_1}, \tilde{\mathbf{G}}^{k_1}, \tilde{\mathbf{F}}^{k_1^*}, \tilde{\mathbf{G}}^{k_1^*}, \text{True})$ 
     $\text{multRCM}(\tilde{\mathbf{F}}^{k_1^*}, \tilde{\mathbf{G}}^{k_1^*}, \tilde{\mathbf{F}}^{k_1}, \tilde{\mathbf{G}}^{k_1}, \text{False})$ 
   $\tilde{\mathbf{F}}^{k_1} \leftarrow \text{iffty}(\tilde{\mathbf{F}}^{k_1})$ 
   $\tilde{\mathbf{F}}^{k_1^*} \leftarrow \text{iffty}(\tilde{\mathbf{F}}^{k_1^*})$ 
 $\tilde{\mathbf{h}} \leftarrow \text{ifftx}(\tilde{\mathbf{F}})$ 
 $\mathbf{h} \leftarrow \text{complexUnPack}(\tilde{\mathbf{h}})$ 
return  $\mathbf{h}$ 

```

299 and 128 GB DDR5 memory (5 GHz), compiled with GCC 15.2.1 using flags `-Ofast`
 300 `-fomit-frame-pointer -fstrict-aliasing -ffast-math`. The underlying FFTs
 301 use FFTW 3.3.10 [5, 6] under Fedora 43, with multithreading via OpenMP.

302 We measure the time to compute an unnormalized⁵ in-place dealiased binary
 303 convolution using the `chrono::nanoseconds` clock from the C++ standard library.
 304 Each size is timed for at least 5 seconds and until at least 20 samples are collected.

⁵By unnormalized convolution, we mean that the inverse FFTs are not normalized.

We plot the median times from each of these tests. We compare four algorithms:

- **explicit** uses real-to-complex/complex-to-real FFTs and explicit padding (the standard approach). In one dimension, both in-place and out-of-place FFTs are benchmarked and the minimum time is plotted. In two and three dimensions, in-place multidimensional FFTs are used for all sizes.
- **hybridCS** uses the conjugate-symmetry algorithm of [section 3](#). In two and three dimensions, it employs the multidimensional decomposition of [section 5](#) with complex FFTs from [\[12\]](#) for the inner transforms.
- **hybridRCM** uses the implicit packing algorithm of [section 4](#) with complex hybrid dealiasing from [\[12\]](#). In two and three dimensions, it employs the multidimensional decomposition of [section 5](#).
- **explicitRCM** is a simplified variant of **hybridRCM**: it uses complex packing and the RCM operator but only explicit zero padding for dealiasing.

6.1. Convolution benchmarks in one dimension. [Figure 2](#) plots median execution times (normalized to $L \log_2 L$) for one-dimensional in-place convolutions of L real words ($M = 2L$) on a single thread. Both **hybridCS** and **explicitRCM** are, on average, slightly faster than **explicit**. On the other hand, the **hybridRCM** algorithm outperforms **explicit** at all sizes and is significantly faster for large convolutions, with a maximum speedup factor of approximately 1.4.

[Figure 3](#) plots normalized times for the same convolutions parallelized over eight threads. For large sizes, **explicit** is outperformed by all other algorithms. Again, the **hybridRCM** algorithm is the clear winner, with a maximum speedup factor of approximately 3.9 over **explicit**. Notably, **explicitRCM** performs nearly as well as **hybridRCM** for many large sizes, although its performance degrades for the largest convolutions.

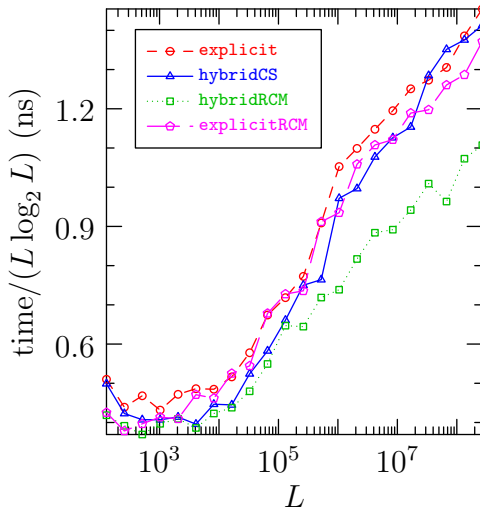


FIG. 2. In-place 1D real convolutions of two arrays of length L on 1 thread.

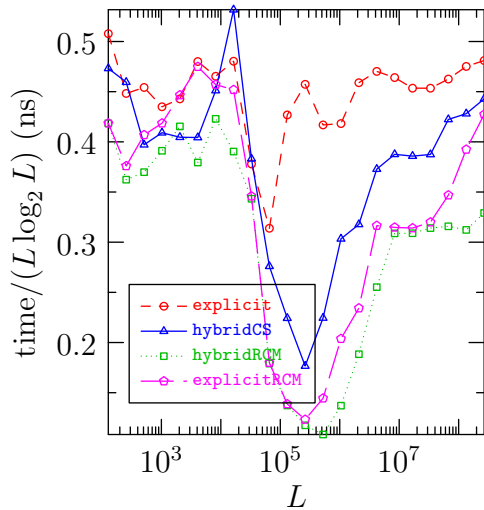


FIG. 3. In-place 1D real convolutions of two arrays of length L on 8 threads.

Although **hybridRCM** and **explicitRCM** often achieve similar timings, we note that they employ very different algorithms: for these large cases, the optimal parameters for **hybridRCM** always use the complex inner-loop optimization of [\[12, §6.1\]](#). Under multithreading, most of the speedup from implicit packing appears to stem from the

ability to use complex FFTs rather than real-to-complex/complex-to-real variants.

6.2. Convolution benchmarks in two and three dimensions. Figures 4 and 5 plot normalized times for two-dimensional real convolutions of size $L \times L$ padded to $2L \times 2L$, on one thread and eight threads respectively. As in [12], **explicit** is outperformed by all other algorithms. On both one and eight threads, **hybridRCM** matches or exceeds all others. On one thread, **hybridCS** now outperforms **explicitRCM** at all sizes; on eight threads, **explicitRCM** usually outperforms **hybridCS**. We also observe degraded performance of **explicitRCM** at the largest size, consistent with the one-dimensional eight-thread results in Figure 3.

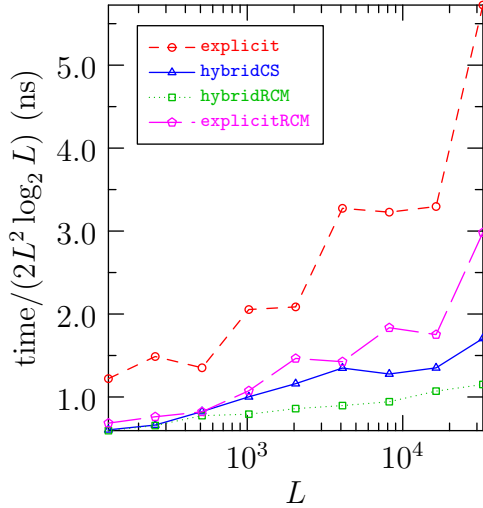


FIG. 4. In-place 2D real convolutions of two arrays of size $L \times L$ on 1 thread.

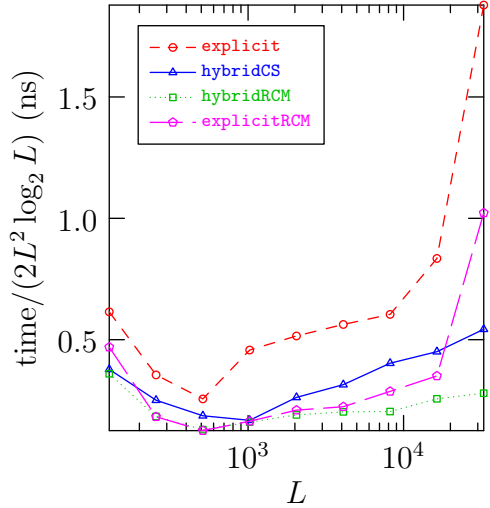


FIG. 5. In-place 2D real convolutions of two arrays of size $L \times L$ on 8 threads.

Figures 6 and 7 plot normalized times for three-dimensional real convolutions of size $L \times L \times L$ padded to $2L \times 2L \times 2L$, on one thread and eight threads respectively. Again, **explicit** is outperformed by all other algorithms, and **hybridRCM** is the clear winner on both one and eight threads (except at the smallest benchmark). Both **hybridCS** and **explicitRCM** are comparable here.

6.3. Discussion. The results of section 6 indicate that **hybridRCM** consistently outperforms all other tested algorithms. Nevertheless, both **explicitRCM** and **hybridCS** have distinct merits.

Benchmarking **explicitRCM** is motivated by the difficulty of implementing efficient hybrid-dealiased convolutions. While such routines are available in the high-performance C++ library FFTW++ [2], some applications cannot integrate this library (for instance, no GPU implementation of hybrid dealiasing currently exists). Implementing implicitly packed convolutions with explicit dealiasing and multidimensional decomposition is considerably simpler. Our benchmarks suggest this is worthwhile: in several cases **explicitRCM** performs nearly as well as **hybridRCM**, particularly under multithreading.

The **hybridCS** algorithm also offers advantages over **hybridRCM**. First, its multiplication operator generalizes easily. As discussed in subsection 4.1, the RCM operator (4.5) can be extended to convolutions of $n \geq 2$ real arrays, but deriving and implementing a multiplication operator for implicit packing may be difficult for more

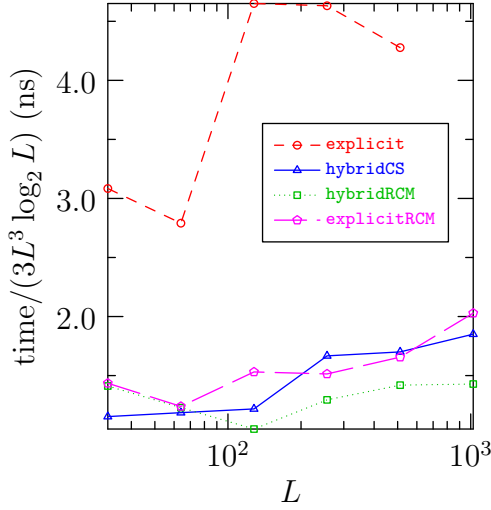


FIG. 6. In-place 3D real convolutions of two arrays of size $L \times L \times L$ on 1 thread.

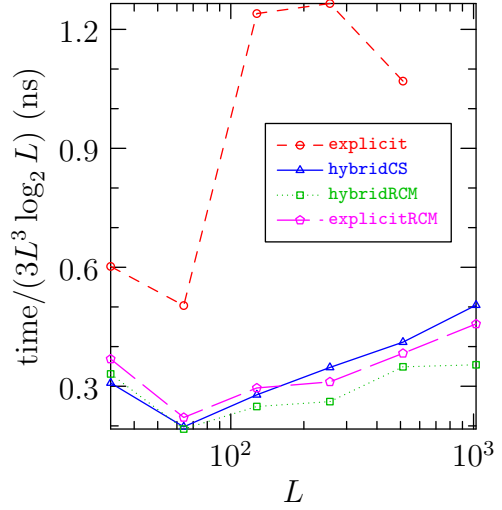


FIG. 7. In-place 3D real convolutions of two arrays of size $L \times L \times L$ on 8 threads.

complicated generalized convolutions. Moreover, even when derivable, an implicitly packed multiplier may not be efficient (as shown in subsection 4.2 when convolving more than 9 arrays at once). Second, some applications require the Fourier transforms of the inputs alongside the convolution; these are computed explicitly by `hybridCS` but not by `hybridRCM`.

An alternative approach that combines `hybridCS` and `hybridRCM` may be worth investigating: computing residue contributions is more involved in `hybridRCM` than in `hybridCS`. As discussed in section 4, for residues $r \notin \{0, q/2\}$ we must compute residues r and $q-r$ simultaneously, increasing the required work memory. Since this is particularly significant in higher dimensions, one could follow the `hybridCS` algorithm of section 3 but use implicit packing for the 0 and $q/2$ residue contributions. This would allow each residue to be computed independently.

7. Conclusion and future work. The original motivation for implicit dealiasing (the predecessor to hybrid dealiasing) was improving the efficiency of pseudospectral simulations of partial differential equations, which rely on convolutions of Hermitian-symmetric arrays. Extending hybrid dealiasing to real-valued inputs substantially broadens the range of applicable problems. In particular, these algorithms may accelerate convolutional neural networks (CNNs).

In a CNN, input data is convolved with a small kernel of learned parameters. These convolutions are typically computed directly or via the Winograd method [8], though FFT-based approaches have also been explored [3, 7, 9, 10]. FFT-based convolutions become competitive for large kernels, large inputs, and high-dimensional problems. For small kernels, aliasing errors affect only a narrow boundary region, and it is common to use “valid” padding, discarding outputs near the boundary after convolution. The primary purpose of zero padding is then to match the kernel size to the input arrays. Since kernels are typically much smaller than inputs, this entails substantial zero padding.

One direction for future work is modifying the hybrid dealiasing algorithms to handle arrays of different sizes, enabling implicit padding of the kernel. Following

the derivation of [section 2](#), one could allow the arrays to use different values of p . Specifically, to convolve an array $\mathbf{f}$ of length L_f with a kernel $\mathbf{g}$ of length $L_g \leq L_f$, both zero padded to length $M \geq L_f$: first find the optimal FFT size m for $\mathbf{f}$, then explicitly pad $\mathbf{f}$ and $\mathbf{g}$ to lengths $p_f m$ and $p_g m$ respectively, where $p_f \doteq \lceil L_f/m \rceil$ and $p_g \doteq \lceil L_g/m \rceil$. Both arrays are then implicitly padded and transformed to length qm with $q \doteq \lceil M/m \rceil$. Since both arrays share the same q and m , the convolution can still be computed one residue at a time. Given that the required zero padding scales exponentially with dimension, this approach may benefit high-dimensional convolutions.

However, implicit zero padding of the kernel may be unnecessary in some applications. In a CNN, zero padding and transforming the kernel is needed only when the kernel is updated. This does not occur during inference, and only occurs in training after each mini-batch. The padded, transformed kernel can therefore be precomputed and incorporated into the multiplication routine (though this must be done for each input size). If input size variability is low, the padding and transform cost will likely be amortized.

Future work could also extend hybrid dealiasing to block-based convolution algorithms such as overlap-add or overlap-save. These methods partition the input into overlapping blocks, convolve each with the kernel, and combine the results. They reduce FFT length (and hence padding/transform costs), particularly for very large inputs. The block size introduces an additional optimization parameter; it would be interesting to study how different block sizes affect the optimal hybrid dealiasing parameters.

Finally, a GPU implementation of hybrid dealiasing is a priority. To date, these algorithms have been implemented and benchmarked only on CPUs, whereas standard CNN training and inference pipelines are typically GPU-accelerated.

As noted in [subsection 6.3](#), a plausible starting point is implementing `explicitRCM` on the GPU, using implicit packing with explicit dealiasing. This would leverage existing GPU FFT-based convolution techniques. The RCM operator, though more complex than Hadamard multiplication, could be implemented as a custom kernel. The results of [section 6](#) suggest that even without hybrid dealiasing, a GPU version of `explicitRCM` may be enough to outperform standard convolution methods.

Appendix A. Inner loop for real arrays. The algorithms presented so far support any FFT size $m \in \mathbb{N}$, but efficiency degrades when m is much smaller than the input arrays. As m decreases, both p and q increase, making the preprocessing and postprocessing in [\(2.1\)](#) and [\(2.2\)](#) considerably more expensive. The inner loop optimization performs these steps using FFTs of size p . This was described for the complex case in [\[12, §6.1\]](#) (and is therefore used in the implicit packing algorithm of [section 4](#)). Here we develop it for the algorithm of [section 3](#). Pseudocode appears in [\[11, Algorithms 22 and 23\]](#).

We begin by redefining q : let $n \doteq \lceil M/(pm) \rceil$ and set $q \doteq np$. We reindex

$$r = un + v, \quad u = 0, \dots, p-1, \quad v = 0, \dots, n-1.$$

From [\(2.1\)](#), the forward transform is

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

and from (2.2), the inverse is

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

Following subsection 2.1, we consider one residue at a time (with v replacing r). Define

$\mathbf{h}_v \doteq \{h_{v,j}\}_{j=0}^{pm-1}$ where

$$(A.1) \quad h_{v,tm+s} \doteq \zeta_q^{-tv} \sum_{u=0}^{p-1} \zeta_p^{-tu} \zeta_{qm}^{-s(un+v)} \sum_{\ell=0}^{m-1} \zeta_m^{-s\ell} F_{q\ell+un+v},$$

so that $\mathbf{f} = \frac{1}{qm} \sum_{v=0}^{n-1} \mathbf{h}_v$. As in (3.1), the Hermitian symmetry $\overline{\mathbf{h}_v} = \mathbf{h}_{n-v}$ gives

$$(A.2) \quad \mathbf{f} = \frac{1}{qm} \left(\mathbf{h}_0 + 2 \sum_{v=1}^{\lceil n/2 \rceil - 1} \operatorname{Re} \mathbf{h}_v + \mathbf{h}_{n/2} \right),$$

where $\mathbf{h}_{n/2} \equiv \mathbf{0}$ if n is odd. As in subsection 3.1, there are three cases.

A.1. Case $\mathbf{h}_0$. This case reduces essentially to the algorithm of subsection 3.1.

When $v = 0$, the forward transform becomes

$$(A.3) \quad F_{q\ell+un} = \sum_{s=0}^{m-1} \zeta_m^{\ell s} \zeta_{qm}^{uns} \sum_{t=0}^{p-1} \zeta_p^{ut} f_{tm+s}.$$

The sum in t is computed with a real-to-complex FFT of length p . For the sum in s , note that (A.3) has the same form as (2.1) with $r = un$. We therefore apply the algorithm from subsection 3.1: when $u = 0$ we use a real-to-complex FFT of length m .⁶ When $u = 1, \dots, \lceil p/2 \rceil - 1$, we use complex FFTs of length m . Finally, when $u = p/2$ we use a complex FFT of length $m/2$ (requiring m even).

A.2. Case $\mathbf{h}_v$, $v \in \{1, \dots, \lceil n/2 \rceil - 1\}$. From (A.2), the contributions of $\mathbf{h}_v$ and $\mathbf{h}_{n-v}$ are obtained simultaneously. We therefore use complex FFTs without sacrificing efficiency, following the complex inner loop algorithm of [12, §6.1]: the forward transform uses m FFTs of length p followed by p complex FFTs of length m , and the inverse uses p complex FFTs of length m followed by m complex FFTs of length p .

A.3. Case $\mathbf{h}_{n/2}$. This case arises only when n is even. The forward transform becomes

$$(A.4) \quad F_{q\ell+un+n/2} = \sum_{s=0}^{m-1} \zeta_m^{\ell s} \zeta_{qm}^{(un+n/2)s} \sum_{t=0}^{p-1} \zeta_p^{ut} \zeta_{2p}^t f_{tm+s},$$

and (A.1) becomes

$$h_{n/2,tm+s} \doteq \zeta_{2p}^{-t} \sum_{u=0}^{p-1} \zeta_p^{-tu} \zeta_{qm}^{-s(un+n/2)} \sum_{\ell=0}^{m-1} \zeta_m^{-s\ell} F_{q\ell+un+n/2}.$$

⁶For simplicity, our implementation in FFTW++ [2] uses a complex FFT for this part. The inner loop is typically useful when p is large, so this does not significantly affect performance.

461 Consider the DFT of length p in (A.4):

$$462 \quad (\text{A.5}) \quad \sum_{t=0}^{p-1} \zeta_p^{ut} \zeta_{2p}^t f_{tm+s}.$$

463 Setting $p = 1$ in (3.3), the sum over s has the same form as the sum over t in (A.5).
 464 If p is even, we can therefore use the algorithm for $r = q/2$ from subsection 3.1.3 to
 465 compute (A.5) using complex FFTs of size $p/2$.

Let $\varphi \doteq p/2$, and reindex t and u as follows:

$$t = \alpha\varphi + \beta, \quad \alpha \in \{0, 1\}, \quad \beta \in \{0, \dots, \varphi - 1\}.$$

466 We need only compute (A.4) for even values of u . For $\gamma = 0, \dots, \varphi - 1$ we obtain

$$467 \quad (\text{A.6}) \quad F_{q\ell+2\gamma n+n/2} = \sum_{s=0}^{m-1} \zeta_m^{\ell s} \zeta_{qm}^{(2\gamma n+n/2)s} \sum_{\beta=0}^{\varphi-1} \zeta_\varphi^{\gamma\beta} \zeta_{2p}^\beta [f_{\beta m+s} + i f_{(\varphi+\beta)m+s}].$$

468 Next we handle the inverse transform. As in (3.6), define

$$469 \quad w_{s,\beta} \doteq \zeta_{2p}^{-\beta} \sum_{\gamma=0}^{\varphi-1} \zeta_\varphi^{-\beta\gamma} \zeta_{qm}^{-s(2\gamma n+n/2)} \sum_{\ell=0}^{m-1} \zeta_m^{-s\ell} F_{q\ell+2\gamma n+n/2}.$$

470 This array is computed using φ complex FFTs of length m , followed by m complex
 471 FFTs of length φ . We then have

$$472 \quad h_{n/2,\beta m+s} = 2 \operatorname{Re}(w_{s,\beta}), \quad h_{n/2,(\varphi+\beta)m+s} = 2 \operatorname{Im}(w_{s,\beta}),$$

473 by an argument analogous to that underlying (3.7).

474 Thus the $v = n/2$ residue requires only $p/2$ DFTs of length m (instead of p), so
 475 complex FFTs of length m in (A.6) incur no loss in efficiency.

476 **Appendix B. Derivation of the RCM operator.** Let $\mathbf{f}$ and $\mathbf{g}$ be real arrays
 477 of length $N = 2M$, and let $\tilde{\mathbf{f}}$ and $\tilde{\mathbf{g}}$ be their complex packings ((4.1)). We derive
 478 (4.5) from (4.4), identifying $\tilde{F}_M \doteq \tilde{F}_0$ and $\tilde{G}_M \doteq \tilde{G}_0$. Unpacking in Fourier space via
 479 (4.2),

$$\begin{aligned} 480 & (U(\tilde{\mathbf{F}}) \odot U(\tilde{\mathbf{G}}))_k \\ 481 &= \frac{1}{4} \left[\left(\tilde{F}_k + \overline{\tilde{F}_{M-k}} \right) - i \left(\tilde{F}_k - \overline{\tilde{F}_{M-k}} \right) \zeta_{2M}^k \right] \left[\left(\tilde{G}_k + \overline{\tilde{G}_{M-k}} \right) - i \left(\tilde{G}_k - \overline{\tilde{G}_{M-k}} \right) \zeta_{2M}^k \right] \\ 482 &= \frac{1}{4} \left[\tilde{F}_k \tilde{G}_k + \overline{\tilde{F}_{M-k}} \tilde{G}_k + \tilde{F}_k \overline{\tilde{G}_{M-k}} + \overline{\tilde{F}_{M-k}} \tilde{G}_{M-k} - 2i \left(\tilde{F}_k \tilde{G}_k - \overline{\tilde{F}_{M-k}} \tilde{G}_{M-k} \right) \zeta_{2M}^k - \right. \\ 483 & \quad \left. \left(\tilde{F}_k \tilde{G}_k - \overline{\tilde{F}_{M-k}} \tilde{G}_k - \tilde{F}_k \overline{\tilde{G}_{M-k}} + \overline{\tilde{F}_{M-k}} \tilde{G}_{M-k} \right) \zeta_M^k \right], \end{aligned}$$

484 for all $k = 0, \dots, M$. Define

$$\begin{aligned} 485 & A_k \doteq \tilde{F}_k \tilde{G}_k + \overline{\tilde{F}_{M-k}} \tilde{G}_k + \tilde{F}_k \overline{\tilde{G}_{M-k}} + \overline{\tilde{F}_{M-k}} \tilde{G}_{M-k}, \\ 486 & B_k \doteq \tilde{F}_k \tilde{G}_k - \overline{\tilde{F}_{M-k}} \tilde{G}_{M-k}, \\ 487 & C_k \doteq \tilde{F}_k \tilde{G}_k - \overline{\tilde{F}_{M-k}} \tilde{G}_k - \tilde{F}_k \overline{\tilde{G}_{M-k}} + \overline{\tilde{F}_{M-k}} \tilde{G}_{M-k}, \end{aligned}$$

so that

$$(U(\tilde{\mathbf{F}}) \odot U(\tilde{\mathbf{G}}))_k = \frac{1}{4} (A_k - 2iB_k\zeta_{2M}^k - C_k\zeta_M^k).$$

Note the symmetries

$$A_k = \overline{A_{M-k}}, \quad B_k = -\overline{B_{M-k}}, \quad C_k = \overline{C_{M-k}}.$$

Using these, we compute

$$\overline{(U(\tilde{\mathbf{F}}) \odot U(\tilde{\mathbf{G}}))_{M-k}} = \frac{1}{4} (A_k + 2iB_k\zeta_{2M}^k - C_k\zeta_M^k)$$

From (B.1) and (B.2),

$$(U(\tilde{\mathbf{F}}) \odot U(\tilde{\mathbf{G}}))_k + \overline{(U(\tilde{\mathbf{F}}) \odot U(\tilde{\mathbf{G}}))_{M-k}} = \frac{1}{2} (A_k - C_k\zeta_M^k),$$

and

$$(U(\tilde{\mathbf{F}}) \odot U(\tilde{\mathbf{G}}))_k - \overline{(U(\tilde{\mathbf{F}}) \odot U(\tilde{\mathbf{G}}))_{M-k}} = -iB_k\zeta_{2M}^k.$$

Repacking via (4.3) along with (B.3) and (B.4) gives

$$(\tilde{\mathbf{F}} \boxtimes \tilde{\mathbf{G}})_k = \frac{1}{4} (A_k - C_k\zeta_M^k) + \frac{1}{2} (B_k\zeta_{2M}^k) \zeta_{2M}^{-k} = \frac{1}{4} A_k - \frac{1}{4} C_k\zeta_M^k + \frac{1}{2} B_k.$$

Substituting A_k , B_k , and C_k and simplifying gives

$$\begin{aligned} (\tilde{\mathbf{F}} \boxtimes \tilde{\mathbf{G}})_k &= \frac{1}{4} \left(\tilde{F}_k \tilde{G}_k + \overline{\tilde{F}_{M-k}} \tilde{G}_k + \tilde{F}_k \overline{\tilde{G}_{M-k}} + \overline{\tilde{F}_{M-k}} \overline{\tilde{G}_{M-k}} \right) - \\ &\quad \frac{1}{4} \left(\tilde{F}_k \tilde{G}_k - \overline{\tilde{F}_{M-k}} \tilde{G}_k - \tilde{F}_k \overline{\tilde{G}_{M-k}} + \overline{\tilde{F}_{M-k}} \overline{\tilde{G}_{M-k}} \right) \zeta_M^k + \\ &\quad \frac{1}{2} \left(\tilde{F}_k \tilde{G}_k - \overline{\tilde{F}_{M-k}} \overline{\tilde{G}_{M-k}} \right) \\ &= \tilde{F}_k \tilde{G}_k - \frac{1}{4} \left(\tilde{F}_k - \overline{\tilde{F}_{M-k}} \right) \left(\tilde{G}_k - \overline{\tilde{G}_{M-k}} \right) (1 + \zeta_M^k), \end{aligned}$$

as required.

Acknowledgements. The authors thank Robert Joseph George for discussions regarding the extension of hybrid dealiasing to arrays of different sizes. The authors also acknowledge the use of ChatGPT and Qwen3.6 during the preparation of this manuscript to assist with code development, correct grammar and typographical errors, and improve the clarity and style of the text. The authors assume responsibility for all content.

Financial support for this work was provided by grants RES0043585 and RES0046040 from the Natural Sciences and Engineering Research Council of Canada.

References.

- [1] J. C. Bowman and M. Roberts. “Efficient Dealiasd Convolutions without Padding”. *SIAM J. Sci. Comput.* 33.1 (2011), 386–406.
- [2] J. C. Bowman, M. Roberts, and N. Murasko. **FFTW++: A fast Fourier transform C++ header class for the FFTW3 library.** <http://fftwpp.sourceforge.net>. 2024.

- [3] K. Chitsaz et al. *Acceleration of Convolutional Neural Network Using FFT-Based Split Convolutions*. 2020. arXiv: [2003.12621](https://arxiv.org/abs/2003.12621) [[cs.CV](#)].
- [4] J. W. Cooley and J. W. Tukey. “An Algorithm for the Machine Calculation of Complex Fourier Series”. *Mathematics of Computation* 19.90 (Apr. 1965), 297–301.
- [5] M. Frigo and S. G. Johnson. “The Design and Implementation of FFTW3”. *Proceedings of the IEEE* 93.2 (2005). Special issue on “Program Generation, Optimization, and Platform Adaptation”, 216–231.
- [6] M. Frigo and S. G. Johnson. FFTW. <http://www.fftw.org>.
- [7] T. Highlander and A. Rodriguez. “Very Efficient Training of Convolutional Neural Networks using Fast Fourier Transform and Overlap-and-Add”. In: *Proceedings of the British Machine Vision Conference*. Ed. by X. Xie, M. W. Jones, and G. K. L. Tam. BMVA Press, Sept. 2015, 160.1–160.9.
- [8] A. Lavin and S. Gray. “Fast Algorithms for Convolutional Neural Networks”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, 4013–4021.
- [9] J. Lin and Y. Yao. “A Fast Algorithm for Convolutional Neural Networks Using Tile-based Fast Fourier Transforms”. *Neural Processing Letters* 50.2 (Oct. 2019), 1951–1967.
- [10] M. Mathieu, M. Henaff, and Y. LeCun. “Fast Training of Convolutional Networks through FFTs”. In: *2nd International Conference on Learning Representations*. Ed. by Y. Bengio and Y. LeCun. Banff, AB, Canada: Conference Track Proceedings, Apr. 2014.
- [11] N. Murasko. “Hybrid Dealiasied Convolutions”. Master’s thesis. University of Alberta, 2024.
- [12] N. Murasko and J. C. Bowman. “Hybrid Dealiasing of Complex Convolutions”. *SIAM Journal on Scientific Computing* 46.3 (2024), B159–B178.
- [13] W. H. Press et al. *Numerical recipes : the art of scientific computing*. Third. Cambridge University Press, 2007.
- [14] M. Roberts and J. C. Bowman. “Multithreaded Implicitly Dealiasied Convolutions”. *Journal of Computational Physics* 356.1 (Mar. 2018), 98–114.
- [15] H. V. Sorensen et al. “Real-valued fast Fourier transform algorithms”. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 35.6 (1987), 849–863.