EFFICIENT DEALIASED CONVOLUTIONS WITHOUT PADDING*

JOHN C. BOWMAN† AND MALCOLM ROBERTS†

SIAM J. Sci. Comput., **33:1**, 386-406 (2011)

Abstract. Algorithms are developed for calculating dealiased linear convolution sums without the expense of conventional zero-padding or phase-shift techniques. For one-dimensional in-place convolutions, the memory requirements are identical with the zero-padding technique, with the important distinction that the additional work memory need not be contiguous with the input data. This decoupling of data and work arrays dramatically reduces the memory and computation time required to evaluate higher-dimensional in-place convolutions. The technique also allows one to dealias the higher-order convolutions that arise from Fourier transforming cubic and higher powers. Implicitly dealiased convolutions can be built on top of state-of-the-art fast Fourier transform libraries: vectorized multidimensional implementations for the complex and centered Hermitian (pseudospectral) cases have been implemented in the open-source software `FFTW++`.

Key words. dealiasing, zero padding, convolution, ternary convolution, fast Fourier transform, bit reversal, implicit padding, pseudospectral method, in-place transform, 2/3 padding rule

AMS subject classifications. 65R99, 65T50

1. Introduction.

Discrete linear convolution sums based on the fast Fourier transform (FFT) algorithm [7, 4] have become important tools for image filtering, digital signal processing, and correlation analysis. They are also widely used in periodic domains to solve nonlinear partial differential equations, such as the Navier–Stokes equations. In some of these applications, such as direct numerical pseudospectral simulations of turbulent fluids, memory usage is a critical limiting factor, and self-sorting in-place multidimensional Fourier transforms [14] are typically used to reduce the memory footprint of the required spectral convolutions.

It is important to remove aliases from FFT-based convolutions applied to non-periodic (wavenumber-space) data because they assume cyclic input and produce cyclic output. Typically the input data arrays are extended by padding them with enough zeros so that wave beats of the positive frequencies cannot wrap around and contaminate the negative frequencies. A cyclic convolution $\sum_{p=0}^{N-1} f_p g_{k-p}$ is then performed using a correspondingly larger Fourier transform size N. If the cost of computing a complex Fourier transform of size N is asymptotic to $KN\log_2 N$ as $N\to\infty$ (the lowest bound currently achievable is $K = 34/9$ [8, 10]), the asymptotic cost of computing the convolution of two vectors of unpadded length m is $6Km\log_2 m$ (using three Fourier transforms with $N = 2m$).

Another important case in practice is the centered Hermitian 1D convolution, dealiased by the *2/3 padding rule* [11]. Since the computational cost of complex-to-real and real-to-complex Fourier transforms of size $N = 3m$ is asymptotic to $\frac{1}{2}KN\log_2 N$, the FFT-based Hermitian convolution $\sum_{p=k-m+1}^{m-1} f_p g_{k-p}$ requires three transforms and hence $\frac{9}{2}Km\log_2 m$ operations. Alternatively, phase shift dealiasing [12, 3] can be used to cancel out the aliasing errors between two convolutions with different phase shifts. However, this second technique is rarely used in practice since, in addition to doubling the memory requirements, it is computationally more expensive, requiring $6Km\log_2 m$ operations for a centered Hermitian convolution.

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

†Department of Mathematical and Statistical Sciences, University of Alberta, Edmonton, Alberta T6G 2G1, Canada.

An explicit application of the zero-padding technique involves the rather obvious inefficiency of summing over a large number of data values that are known *a priori* to be zero. However, it is worthwhile to consider the response provided by Steven G. Johnson to a frequently asked question about the possibility of avoiding this expense [5]:

> *The most common case where people seem to want a pruned FFT is for zero-padded convolutions, where roughly 50% of your inputs are zero (to get a linear convolution from an FFT-based cyclic convolution). Here, a pruned FFT is hardly worth thinking about, at least in one dimension. In higher dimensions, matters change (e.g. for a 3d zero-padded array about 1/8 of your inputs are non-zero, and one can fairly easily save a factor of two or so simply by skipping 1d sub-transforms that are zero).*

The reasoning behind the assertion that such one-dimensional pruned FFTs are not worth thinking about is that if only m of the N inputs are nonzero, the computational cost is reduced only from $N \log_2 N$ to $N \log_2 m$. For example, if $m = N/2$, the savings is a minuscule $1/\log_2 N$. Moreover, since a zero-padded Fourier transform of size N yields N (typically nonzero) output values, no storage savings appear possible in one dimension. Nevertheless, in this work we demonstrate that pruning the zero-padded elements of one-dimensional convolutions is still worth thinking about, primarily because this provides useful building blocks for constructing more efficient multidimensional convolutions.

The key observation is this: although the memory usage of our implicitly padded 1D convolution is identical to that of a conventional explicitly padded convolution, the additional temporary memory need not be contiguous with the user data. In a multidimensional context, this external work buffer can be reused for other low dimensional convolutions. As a result, in d dimensions an implicitly dealiased convolution asymptotically uses $1/2^{d-1}$ of the storage space needed for an explicitly padded convolution. When the Fourier origin is centered in the domain, memory usage is reduced to $(2/3)^{d-1}$ of the conventional amount. If saving memory alone were the goal, this reduction could also be achieved with explicit zero padding by copying the data for the innermost convolution to an external padded buffer, but such extra data communication would degrade overall performance. The fact that our one-dimensional convolution does not require this extra copying is the main feature that was exploited to obtain simultaneous improvements in memory usage and speed.

Nevertheless, the task of writing an efficient implicitly dealiased one-dimensional convolution is onerous, particularly if one tries to compete with a problem-dependent, architecture-adaptive FFT algorithm (like that provided by the award-winning FFTW [6] library, which empirically predetermines a near optimal butterfly graph at each subdivision). Effectively one wants to perform the outer FFT subdivision manually, dropping the zero terms and deferring the computation of the inner transforms to a standard library routine. But this also restricts the set of available platform-dependent algorithms that can be used at the highest level. Fortunately, several notable features of our algorithm help to offset this disadvantage. First, if the goal is to produce a convolution, bit reversal for the hand-optimized outermost subdivision is unnecessary: the scrambled Fourier subtransforms of the two input vectors can be multiplied together as they are produced (perhaps while still accessible in the cache). Second, the implicit method allows most of the subtransforms for an in-place convolution to be optionally computed as out-of-place transforms, which typically

execute faster than in-place transforms (cf. Figs. 1–6 of [6]) since they require no extra (pre-, post-, or interlaced) bit-reversal stage. These savings help keep our one-dimensional in-place implicit convolution competitive with an explicitly padded convolution based on the same highly optimized library.

In Section 2, we develop an algorithm for Fourier transforming a one-dimensional zero-padded vector without the need for explicit padding. We show how this algorithm can be used to calculate implicitly padded convolutions for both general and Hermitian inputs. We describe how implicit padding may be applied to compute the discrete Fourier transform of an input vector padded beyond an arbitrary fraction p/q of its length. Building on these one-dimensional algorithms, implicitly padded convolutions are implemented for two- and three-dimensional input in Section 3. Finally, in Section 4, we show for both general and Hermitian data how implicit padding may be used to dealias higher-order convolutions efficiently.

2. Implicitly dealiased 1D convolutions. In this section we describe the optimized 1D building blocks that are used in subsequent sections to construct higher-dimensional implicitly dealiased convolutions.

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

In terms of the Nth primitive root of unity, $\zeta_N \doteq \exp(2\pi i/N)$ (here $\doteq$ is used to emphasize a definition), the unnormalized backward discrete Fourier transform of a complex vector $\{U_k : k = 0, \ldots, N-1\}$ may be written as

$$u_j \doteq \sum_{k=0}^{N-1} \zeta_N^{jk} U_k, \qquad j = 0, \ldots, N-1.$$

The fast Fourier transform method exploits the properties that $\zeta_N^r = \zeta_{N/r}$ and $\zeta_N^N = 1$.

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

$$u_{2\ell} = \sum_{k=0}^{m-1} \zeta_{2m}^{2\ell k} U_k = \sum_{k=0}^{m-1} \zeta_m^{\ell k} U_k, \quad u_{2\ell+1} = \sum_{k=0}^{m-1} \zeta_{2m}^{(2\ell+1)k} U_k = \sum_{k=0}^{m-1} \zeta_m^{\ell k} \zeta_{2m}^{k} U_k. \tag{2.1}$$

This requires computing two subtransforms, each of size m, for an overall computational scaling of order $2m \log_2 m = N \log_2 m$.

The odd and even terms of the convolution can then be computed separately (without the need for a bit reversal stage), multiplied term by term, and finally transformed again to Fourier space using the (scaled) forward transform

$$\begin{aligned} 2mU_k &= \sum_{j=0}^{2m-1} \zeta_{2m}^{-kj} u_j = \sum_{\ell=0}^{m-1} \zeta_{2m}^{-k2\ell} u_{2\ell} + \sum_{\ell=0}^{m-1} \zeta_{2m}^{-k(2\ell+1)} u_{2\ell+1} \\ &= \sum_{\ell=0}^{m-1} \zeta_m^{-k\ell} u_{2\ell} + \zeta_{2m}^{-k} \sum_{\ell=0}^{m-1} \zeta_m^{-k\ell} u_{2\ell+1}, \qquad k = 0, \ldots, m-1. \end{aligned} \tag{2.2}$$

The implicitly padded transforms described by Eqs. (2.1) and (2.2) are implemented as Procedure `fftpadBackward` and `fftpadForward`. These algorithms are combined in Function `cconv` to compute a dealiased convolution of unpadded length m using two arrays of size m as input vectors instead of one array of size $2m$. This seemingly trivial distinction is the key to the improved efficiency and reduced storage requirements of the higher-dimensional implicit convolutions described in Section 3. Moreover, in Function `cconv` we see that implicit padding allows each of the six complex Fourier transforms of size m to be done out of place. In the listed pseudocode, an asterisk $(*)$ denotes an element-by-element (vector) multiply.

In principle, the stable trigonometric recursion described by Buneman [2], which requires two small precomputed tables, each of size $\log_2 N$, could be used to compute the required roots of unity ζ_N^k that appear in Eqs. (2.1) and (2.2).[1] While Buneman's recursion has the same average accuracy as an FFT itself [13], on modern hardware a factorization method that does not rely on successive table updates turns out to be more efficient [9], at the expense of somewhat higher memory usage. We instead calculate the ζ_N^k factors with a single complex multiply, using two short precomputed tables $H_a = \zeta_N^{as}$ and $L_b = \zeta_N^b$, where $k = as + b$ with $s = \lfloor\sqrt{m}\rfloor$, $a = 0, 1, \ldots, \lceil m/s\rceil - 1$, and $b = 0, 1, \ldots, s - 1$. Since these one-dimensional tables require only $\mathcal{O}(\sqrt{m})$ complex words of storage and our focus is on higher-dimensional convolutions anyway, we do not account for them in our storage estimates.

Referring to the computation times shown in Fig. 2.1, we see that the implicit padding algorithm described by Eqs. (2.1) and (2.2) can thus be implemented to be competitive with explicitly padded convolutions. The error bars indicate the lower and upper *one-sided standard deviations*

$$\sigma_L = \sqrt{\frac{1}{\frac{n}{2}-1}\sum_{\substack{i=1\\ t_i<T}}^{n}(t_i - T)^2}, \qquad \sigma_H = \sqrt{\frac{1}{\frac{n}{2}-1}\sum_{\substack{i=1\\ t_i>T}}^{n}(t_i - T)^2},$$

where T denotes the mean execution time of n samples. Both the FFTW-3.2.2 library and the convolution layer we built on top of it were compiled with the Intel C/C++ 11.0 20081105 compiler, using the optimization options `-ansi-alias -malign-double -fp-model fast=2` on a 64-bit 3GHz Intel E5450 Xenon processor with 6MB cache. Like the FFTW library, our algorithm was vectorized for this architecture with specialized single-instruction multiple-data (SIMD) code.

To compare the normalized error for the two methods, we considered the input vectors $f_k = Fe^{ik}$ and $g_k = Ge^{ik}$ for $k = 0, \ldots, m-1$, with $F = \sqrt{3} + i\sqrt{7}$ and $G = \sqrt{5} + i\sqrt{11}$. The Fourier transforms of these vectors have nonzero components for all transform sizes. In Fig. 2.2 we compare the normalized L^2 error $\sqrt{\sum_{k=0}^{m-1}|h_k - H_k|^2}/\sqrt{\sum_{k=0}^{m-1}|H_k|^2}$ for each of the computed solutions h_k relative to the exact solution $H_k = \sum_{p=0}^{k} f_p g_{k-p} = FG(k+1)e^{ik}$.

2.2. Implicitly dealiased centered Fourier transform. A basic building block for constructing multidimensional centered convolutions is an implicitly dealiased centered Fourier transform, where the input data length is odd, say $2m-1$, with the Fourier origin at index $m-1$. Here, one needs to pad to $N \geq 3m-2$ to

[1] We note that, in terms of the smallest positive number ϵ satisfying $1 + \epsilon > 1$ in a given machine representation, the singularity in Buneman's scheme can be removed by replacing $\sec \pi/2$ with $5/\epsilon$, $\sin 4\pi$ with $-\epsilon/2$, and $\sin 2\pi$ and $\sin \pi$ each with $-\epsilon/10$.

Input: vector f
Output: vector f, vector u
for $k = 0$ **to** $m - 1$ **do**
  $\mathsf{u}[k] \leftarrow \zeta_{2m}^{k}\mathsf{f}[k]$;
end
$\mathsf{f} \leftarrow \mathtt{fft}^{-1}(\mathsf{f})$;
$\mathsf{u} \leftarrow \mathtt{fft}^{-1}(\mathsf{u})$;

Procedure `fftpadBackward(f,u)` stores the scrambled $2m$-padded backward Fourier transform values of a vector f of length m in f and an auxiliary vector u of length m.

Input: vector f, vector u
Output: vector f
$\mathsf{f} \leftarrow \mathtt{fft}(\mathsf{f})$;
$\mathsf{u} \leftarrow \mathtt{fft}(\mathsf{u})$;
for $k = 0$ **to** $m - 1$ **do**
  $\mathsf{f}[k] \leftarrow \mathsf{f}[k] + \zeta_{2m}^{-k}\mathsf{u}[k]$;
end
return f/(2m);

Procedure `fftpadForward(f,u)` returns the inverse of `fftpadBackward`(f,u).

Input: vector f, vector g
Output: vector f
$\mathsf{u} \leftarrow \mathtt{fft}^{-1}(\mathsf{f})$;
$\mathsf{v} \leftarrow \mathtt{fft}^{-1}(\mathsf{g})$;
$\mathsf{u} \leftarrow \mathsf{u} * \mathsf{v}$;
for $k = 0$ **to** $m - 1$ **do**
  $\mathsf{f}[k] \leftarrow \zeta_{2m}^{k}\mathsf{f}[k]$;
  $\mathsf{g}[k] \leftarrow \zeta_{2m}^{k}\mathsf{g}[k]$;
end

$\mathsf{v} \leftarrow \mathtt{fft}^{-1}(\mathsf{f})$;
$\mathsf{f} \leftarrow \mathtt{fft}^{-1}(\mathsf{g})$;
$\mathsf{v} \leftarrow \mathsf{v} * \mathsf{f}$;

$\mathsf{f} \leftarrow \mathtt{fft}(\mathsf{u})$;
$\mathsf{u} \leftarrow \mathtt{fft}(\mathsf{v})$;

for $k = 0$ **to** $m - 1$ **do**
  $\mathsf{f}[k] \leftarrow \mathsf{f}[k] + \zeta_{2m}^{-k}\mathsf{u}[k]$;
end
return f/(2m);

Function `cconv(f,g,u,v)` computes an in-place implicitly dealiased convolution of two complex vectors f and g using two temporary vectors u and v, each of length m.

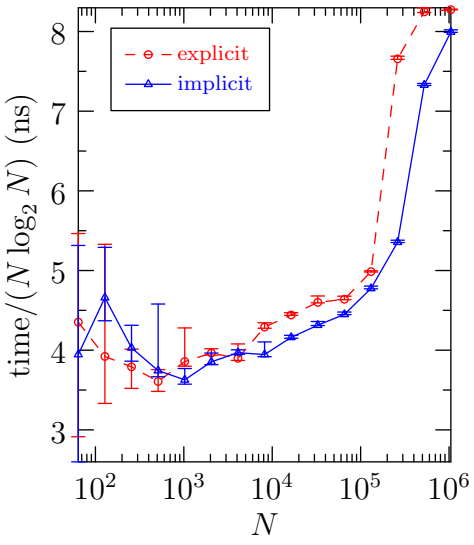

FIG. 2.1. *Comparison of computation times for explicitly and implicitly dealiased complex in-place 1D convolutions of length m. The storage requirements of the two algorithms are identical.*

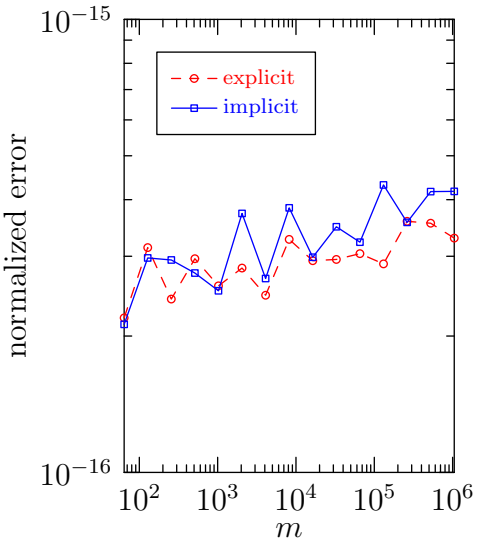

FIG. 2.2. *Normalized L^2 error for explicitly and implicitly dealiased complex in-place 1D convolutions of length m.*

prevent modes with wavenumber $m-1$ from beating together to contaminate the mode with wavenumber $-m+1$. The ratio of the number of physical to total modes, $(2m-1)/(3m-2)$, is asymptotic to $2/3$ for large m [11].

For explicit padding, one usually chooses the padded vector length N to be a power of 2, with $m = \lfloor (N+2)/3 \rfloor$. However, for implicit padding, it is advantageous to choose m itself to be a power of 2 since the algorithm reduces to computing FFTs of length m. Moreover, it is convenient to pad implicitly slightly beyond $3m-2$, to $N = 3m$, as this allows the use of a radix 3 subdivision at the outermost level, so that only two of the three subtransforms of length m need to be retained.

Suppose then that $U_k = 0$ for $k \geq m$. On decomposing $j = (3\ell + r) \bmod N$, where $r \in \{-1, 0, 1\}$, the substitution $k' = m + k$ allows us to write the backward transform as

$$u_{3\ell+r} = \sum_{k=-m+1}^{m-1} \zeta_m^{\ell k} \zeta_{3m}^{rk} U_k = \sum_{k'=1}^{m-1} \zeta_m^{\ell k'} \zeta_{3m}^{r(k'-m)} U_{k'-m} + \sum_{k=0}^{m-1} \zeta_m^{\ell k} \zeta_{3m}^{rk} U_k = \sum_{k=0}^{m-1} \zeta_m^{\ell k} w_{k,r}, \tag{2.3}$$

where

$$w_{k,r} \doteq \begin{cases} U_0 & \text{if } k = 0, \\ \zeta_{3m}^{rk} (U_k + \zeta_3^{-r} U_{k-m}) & \text{if } 1 \leq k \leq m-1. \end{cases} \tag{2.4}$$

The forward transform is then

$$3m U_k = \sum_{r=-1}^{1} \zeta_{3m}^{-rk} \sum_{\ell=0}^{m-1} \zeta_m^{-\ell k} u_{3\ell+r}, \qquad k = -m+1, \ldots, m-1. \tag{2.5}$$

The use of the remainder $r = -1$ instead of $r = 2$ allows us to exploit the optimization $\zeta_{3m}^{-k} = \overline{\zeta_{3m}^{k}}$ in Eqs. (2.4) and (2.5). The number of complex multiplies needed to evaluate Eq. (2.4) for $r = \pm 1$ can be reduced by computing the intermediate complex quantities

$$\begin{aligned} A_k &\doteq \zeta_{3m}^{k} \big(\operatorname{Re} U_k + \zeta_3^{-1} \operatorname{Re} U_{k-m} \big), \\ B_k &\doteq i \zeta_{3m}^{k} \big(\operatorname{Im} U_k + \zeta_3^{-1} \operatorname{Im} U_{k-m} \big), \end{aligned}$$

where $\zeta_3^{-1} = (-\frac{1}{2}, -\frac{\sqrt{3}}{2})$, so that for $k > 0$, $w_{k,1} = A_k + B_k$ and $w_{k,-1} = \overline{A_k - B_k}$. The resulting transforms, Procedures `fft0padBackward` and `fft0padForward`, each have an operation count asymptotic to $3Km \log_2 m$. We were able to implement strided multivector versions of these algorithms since they operate fully in place on their arguments, with no additional storage requirements.

2.3. Centered Hermitian convolution. In this frequently encountered case (relevant to the pseudospectral method), each input vector is the Fourier transform of real-valued data; that is, it satisfies the *Hermitian symmetry* $U_{-k} = \overline{U_k}$. While the physical data represented is of length $2m-1$, centered about the Fourier origin, the redundant modes (corresponding to negative wavenumbers) are not included in the input vectors. The input vectors are thus of length m, with the Fourier origin at index 0. Just as in Section 2.2, the unsymmetrized physical data needs to be padded with at least $m-1$ zeros. Hermitian symmetry then requires us to pad the m non-negative wavenumbers with at least $c \doteq \lfloor m/2 \rfloor$ zeros. The resulting $2/3$ padding ratio (for even m) turns out to work particularly well for developing implicitly dealiased

Input: vector f
Output: vector f, vector u
$\mathsf{u}[0] \leftarrow \mathsf{f}[m-1]$;
for $k = 1$ **to** $m-1$ **do**
  $\mathsf{A} \leftarrow \zeta_{3m}^{k}\Big[\mathrm{Re}\,\mathsf{f}[m-1+k] + \Big(-\frac{1}{2}, -\frac{\sqrt{3}}{2}\Big)\,\mathrm{Re}\,\mathsf{f}[0]\Big]$;
  $\mathsf{B} \leftarrow i\zeta_{3m}^{k}\Big[\mathrm{Im}\,\mathsf{f}[m-1+k] + \Big(-\frac{1}{2}, -\frac{\sqrt{3}}{2}\Big)\,\mathrm{Im}\,\mathsf{f}[0]\Big]$;
  $\mathsf{C} \leftarrow \mathsf{f}[m-1+k] + \mathsf{f}[0]$;
  $\mathsf{f}[0] \leftarrow \mathsf{f}[k]$;
  $\mathsf{f}[k] \leftarrow \mathsf{C}$;
  $\mathsf{f}[m-1+k] \leftarrow \mathsf{A} + \mathsf{B}$;
  $\mathsf{u}[k] \leftarrow \overline{\mathsf{A} - \mathsf{B}}$;
end
$\mathsf{f}[0, \ldots, m-1] \leftarrow \mathtt{fft}^{-1}(\mathsf{f}[0, \ldots, m-1])$;
$\mathsf{u}[m] \leftarrow \mathsf{f}[m-1]$;
$\mathsf{f}[m-1] \leftarrow \mathsf{u}[0]$;
$\mathsf{f}[m-1, \ldots, 2m-2] \leftarrow \mathtt{fft}^{-1}(\mathsf{f}[m-1, \ldots, 2m-2])$;
$\mathsf{u}[0, \ldots, m-1] \leftarrow \mathtt{fft}^{-1}(\mathsf{u}[0, \ldots, m-1])$;

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

Input: vector f, vector u
Output: vector f
$\mathsf{f}[m-1, \ldots, 2m-2] \leftarrow \mathtt{fft}(\mathsf{f}[m-1, \ldots, 2m-2])$;
$\mathsf{u}[m] \leftrightarrow \mathsf{f}[m-1]$;
$\mathsf{f}[0, \ldots, m-1] \leftarrow \mathtt{fft}(\mathsf{f}[0, \ldots, m-1])$;
$\mathsf{u}[0, \ldots, m-1] \leftarrow \mathtt{fft}(\mathsf{u}[0, \ldots, m-1])$;
$\mathsf{u}[m] \leftarrow \mathsf{f}[0] + \mathsf{u}[m] + \mathsf{u}[0]$;
for $k = 1$ **to** $m-1$ **do**
  $\mathsf{f}[k-1] \leftarrow \mathsf{f}[k] + \Big(-\frac{1}{2}, \frac{\sqrt{3}}{2}\Big)\zeta_{3m}^{-k}\mathsf{f}[m-1+k] + \Big(-\frac{1}{2}, -\frac{\sqrt{3}}{2}\Big)\zeta_{3m}^{k}\mathsf{u}[k]$;
  $\mathsf{f}[m-1+k] \leftarrow \mathsf{f}[k] + \zeta_{3m}^{-k}\mathsf{f}[m-1+k] + \zeta_{3m}^{k}\mathsf{u}[k]$;
end
$\mathsf{f}[m-1] \leftarrow \mathsf{u}[m]$;
return $\mathsf{f}/(3\mathrm{m})$;

Procedure `fft0padForward(f,u)` returns the inverse of `fft0padBackward`(f,u).

centered Hermitian convolutions. As in the centered case, we again choose the Fourier size to be $N = 3m$.

Given that $U_k = 0$ for $k \geq m$, the backward (complex-to-real) transform appears as Eq. (2.3), but now with

$$w_{k,r} \doteq \begin{cases} U_0 & \text{if } k = 0, \\ \zeta_{3m}^{rk}\big(U_k + \zeta_3^{-r}\overline{U_{m-k}}\big) & \text{if } 1 \leq k \leq m-1. \end{cases} \tag{2.6}$$

We note that $w_{k,r}$ obeys the Hermitian symmetry $w_{k,r} = \overline{w_{m-k,r}}$, so that the Fourier transform $\sum_{k=0}^{m-1} \zeta_m^{\ell k} w_{k,r}$ in Eq. (2.3) will indeed be real valued. This allows us to build

a backward implicitly dealiased centered Hermitian transform using three complex-to-real Fourier transforms of the first $c+1$ components of $w_{k,r}$ (one for each $r \in \{-1,0,1\}$). The forward transform is given by

$$3mU_k = \sum_{r=-1}^{1} \zeta_{3m}^{-rk} \sum_{\ell=0}^{m-1} \zeta_m^{-\ell k} u_{3\ell+r}, \qquad k = 0, \ldots, m-1. \tag{2.7}$$

Since $u_{3\ell+r}$ is real, a real-to-complex transform can be used to compute the first $c+1$ frequencies of $\sum_{\ell=0}^{m-1} \zeta_m^{-\ell k} u_{3\ell+r}$; the remaining $m-c-1$ frequencies needed in Eq. (2.7) are then computed using Hermitian symmetry.

Since there are two input vectors and one output vector, the complete convolution requires a total of nine Hermitian Fourier transforms of size m, for an overall computational scaling of $\frac{9}{2} Km \log_2 m$ operations, in agreement with the leading-order scaling of an explicitly padded centered Hermitian convolution. For simplicity, we document here only the practically important case $m = 2c$; minor changes are required to implement the case $m = 2c+1$. We see in Function `conv` that seven out of the nine Fourier transforms can be performed out of place using the same amount of memory, $2(\lfloor N/2 \rfloor + 1) = 6c + 2$ words, as would be used to compute a centered Hermitian convolution with explicit padding.

To facilitate an in-place implementation of the backward transform, we store the conjugate of the transformed values for $r = 1$ in reverse order in the upper half of the input vector, using the identity (for real u_j)

$$u_j = \overline{u_j} = \sum_{k=-c+1}^{c} \zeta_m^{-jk} \overline{U_k} = \sum_{k'=m-1}^{0} \zeta_m^{j(k'-c)} \overline{U_{c-k'}} = (-1)^j \sum_{k=0}^{m-1} \zeta_m^{jk} \overline{U_{c-k}}$$

obtained with the substitution $k' = c-k$. One can omit the factors of $(-1)^j$ here since they will cancel during the real term-by-term multiplication of the two transformed input vectors.

As seen in Fig. 2.3, the efficiency of the resulting implicitly dealiased centered Hermitian convolution is comparable to an explicit implementation. For each algorithm, we benchmark only those vector lengths that yield optimal performance.

To check the accuracy of our implementation, we used the test case $f_k = Fe^{ik}$ and $g_k = Ge^{ik}$ for $k = 0, \ldots, m-1$, with $F = \sqrt{3}$ and $G = \sqrt{5}$, noting that Hermitian symmetry requires that F and G be real. The exact solution is $H_k = FG \sum_{p=k-m+1}^{m-1} e^{ip} e^{i(k-p)} = FG(2m-1-k)e^{ik}$. The normalized L^2 errors for implicit and explicit padding are compared in Fig. 2.4.

2.4. General padding. Implicit padding corresponding to an arbitrary p/q rule is also possible. Suppose that pm data modes are zero padded to $N = qm$, where p and q are relatively prime. One decomposes $j = q\ell + r$, where $\ell = 0, \ldots, m-1$ and $r = 0, \ldots, q-1$. Similarly, one expresses $k = tm + s$, where $t = 0, \ldots, p-1$ and $s = 0, \ldots, m-1$:

$$u_{q\ell+r} = \sum_{k=0}^{pm-1} \zeta_{qm}^{(q\ell+r)k} U_k = \sum_{s=0}^{m-1} \sum_{t=0}^{p-1} \zeta_m^{\ell(tm+s)} \zeta_{qm}^{r(tm+s)} U_{tm+s} = \sum_{s=0}^{m-1} \zeta_m^{\ell s} \sum_{t=0}^{p-1} \zeta_{qm}^{r(tm+s)} U_{tm+s}.$$

Since there are q choices of r, the problem reduces to computing q Fourier transforms of length m, which requires $Kqm \log_2 m = KN \log_2(N/q)$ operations. Likewise, the

Input: vector f, vector g
Output: vector f
$\mathsf{F} \leftarrow \mathsf{f}[c]$;
`build`(f,u);
$\mathsf{C} \leftarrow \mathsf{f}[c]$;
$\mathsf{f}[c] \leftarrow 2\operatorname{Re}\mathsf{F}$;
$\mathsf{u}[c] \leftarrow \operatorname{Re}\mathsf{F} + \sqrt{3}\operatorname{Im}\mathsf{F}$;

$\mathsf{G} \leftarrow \mathsf{g}[c]$;
`build`(g,v);
$\mathsf{D} \leftarrow \mathsf{g}[c]$;
$\mathsf{g}[c] \leftarrow 2\operatorname{Re}\mathsf{G}$;
$\mathsf{v}[c] \leftarrow \operatorname{Re}\mathsf{G} + \sqrt{3}\operatorname{Im}\mathsf{G}$;

$\mathsf{u} \leftarrow$ `crfft`(u);
$\mathsf{v} \leftarrow$ `crfft`(v);
$\mathsf{v} \leftarrow \mathsf{v} * \mathsf{u}$;
$\mathsf{u} \leftarrow$ `rcfft`(v);

$\mathsf{v} \leftarrow$ `crfft`($\mathsf{f}[0,\ldots,c]$);
$\mathsf{f}[0,\ldots,c] \leftarrow$ `crfft`($\mathsf{g}[0,\ldots,c]$);
$\mathsf{v} \leftarrow \mathsf{v} * \mathsf{f}[0,\ldots,c]$;
$\mathsf{f}[0,\ldots,c] \leftarrow$ `rcfft`(v);

$\mathsf{S} \leftarrow \mathsf{f}[c-1]$;
$\mathsf{T} \leftarrow \mathsf{f}[c]$;
$\mathsf{f}[c-1] \leftarrow \operatorname{Re}\mathsf{F} - \sqrt{3}\operatorname{Im}\mathsf{F}$;
$\mathsf{f}[c] \leftarrow \mathsf{C}$;
$\mathsf{g}[c-1] \leftarrow \operatorname{Re}\mathsf{G} - \sqrt{3}\operatorname{Im}\mathsf{G}$;
$\mathsf{g}[c] \leftarrow \mathsf{D}$;

$\mathsf{v} \leftarrow$ `crfft`($\mathsf{g}[c-1,\ldots,2c-1]$);
$\mathsf{g}[c-1,\ldots,2c-1] \leftarrow$ `crfft`($\mathsf{f}[c-1,\ldots,2c-1]$);
$\mathsf{g}[c-1,\ldots,2c-1] \leftarrow \mathsf{g}[c-1,\ldots,2c-1] * \mathsf{v}$;
$\mathsf{v} \leftarrow$ `rcfft`($\mathsf{g}[c-1,\ldots,2c-1]$);

for $k = 1$ **to** $c-2$ **do**
  $\mathsf{f}[2c-k] \leftarrow \overline{\mathsf{f}[k]} + \left(-\frac{1}{2}, -\frac{\sqrt{3}}{2}\right)\zeta_{6c}^{k}\overline{\mathsf{v}[k]} + \left(-\frac{1}{2}, \frac{\sqrt{3}}{2}\right)\zeta_{6c}^{-k}\overline{\mathsf{u}[k]}$;
  $\mathsf{f}[k] \leftarrow \mathsf{f}[k] + \zeta_{6c}^{-k}\mathsf{v}[k] + \zeta_{6c}^{k}\mathsf{u}[k]$;
end
$\mathsf{f}[c-1] \leftarrow S + \zeta_{6c}^{1-c}\mathsf{v}[c-1] + \zeta_{6c}^{c-1}\mathsf{u}[c-1]$;
$\mathsf{f}[c] \leftarrow T - \left(-\frac{1}{2}, \frac{\sqrt{3}}{2}\right)\mathsf{v}[c] - \left(-\frac{1}{2}, -\frac{\sqrt{3}}{2}\right)\mathsf{u}[c]$;
if $c > 1$ **then**
  $\mathsf{f}[c+1] \leftarrow \overline{S} + \left(-\frac{1}{2}, -\frac{\sqrt{3}}{2}\right)\zeta_{6c}^{c-1}\overline{\mathsf{v}[c-1]} + \left(-\frac{1}{2}, \frac{\sqrt{3}}{2}\right)\zeta_{6c}^{1-c}\overline{\mathsf{u}[c-1]}$;
end
return f/(6c);

Function `conv(f,g,u,v)` uses Procedure `build` to compute an in-place implicitly dealiased convolution of centered Hermitian vectors f and g of length $2c$ using temporary vectors u and v of length $c+1$.

Input: vector f
Output: vector f, vector u
$\mathsf{u}[0] \leftarrow \mathsf{f}[0]$;
$\mathsf{F} \leftarrow \overline{f[2c-1]}$;
$\mathsf{f}[2c-1] \leftarrow \mathsf{f}[0]$;
for $k = 1$ **to** $c-1$ **do**
  $\mathsf{A} \leftarrow \zeta_{6c}^{k}\left[\operatorname{Re}\mathsf{f}[k] + \left(-\frac{1}{2}, \frac{\sqrt{3}}{2}\right)\operatorname{Re}\mathsf{F}\right]$;
  $\mathsf{B} \leftarrow -i\zeta_{6c}^{k}\left[\operatorname{Im}\mathsf{f}[k] + \left(-\frac{1}{2}, \frac{\sqrt{3}}{2}\right)\operatorname{Im}\mathsf{F}\right]$;
  $\mathsf{f}[k] \leftarrow \mathsf{f}[k] + \mathsf{F}$;
  $\mathsf{u}[k] \leftarrow \mathsf{A} - \mathsf{B}$;
  $\mathsf{F} \leftarrow \overline{\mathsf{f}[2c-1-k]}$;
  $\mathsf{f}[2c-1-k] \leftarrow \mathsf{A} + \mathsf{B}$;
end

Procedure `build(f,u)` builds the FFT arrays required for Function `conv` from an unpadded vector f of length $2c$ into f and an auxiliary vector u of length $c+1$.

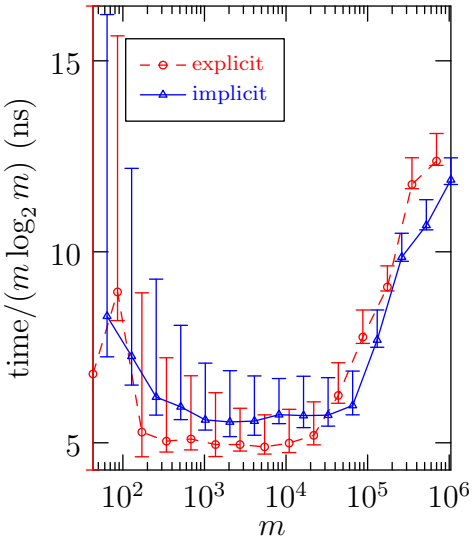

FIG. 2.3. *Comparison of computation times for explicitly and implicitly dealiased centered Hermitian in-place 1D convolutions of length* $2m-1$.

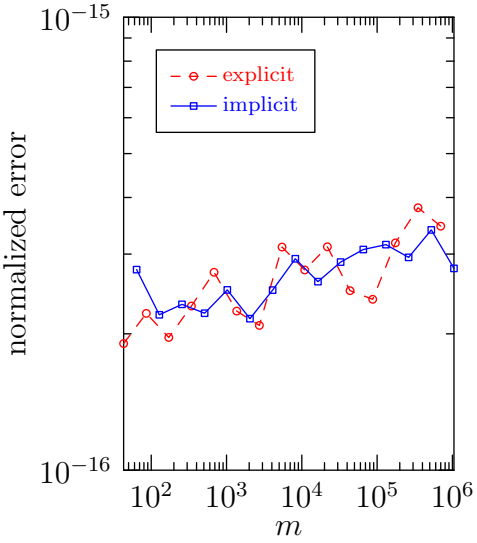

FIG. 2.4. *Normalized* L^2 *error for explicitly and implicitly dealiased centered Hermitian in-place 1D convolutions of length* m.

forward implicit transform

$$qmU_k = \sum_{\ell=0}^{m-1}\sum_{r=0}^{q-1}\zeta_{qm}^{-(q\ell+r)k}u_{q\ell+r} = \sum_{r=0}^{q-1}\zeta_{qm}^{-rk}\sum_{\ell=0}^{m-1}\zeta_{m}^{-k\ell}u_{q\ell+r}, \qquad k = 0,\ldots,pm-1$$

also requires q Fourier transforms of length m. Again, the computational savings for a one-dimensional transform is marginal.

3. Higher-dimensional convolutions. The algorithms developed in Section 2 can be used as building blocks to construct efficient implicitly padded higher-dimensional convolutions.

3.1. Complex 2D convolution. The implicitly padded 2D convolution shown in Function `cconv2` is designed for data stored with a stride of one in the y direction. Efficient multivector versions of Procedures `fft0padBackward` and `fft0padForward` are used for the transform in the x direction; this allows a single ζ_{3m}^k factor to be applied to a consecutive column of data at constant x. In principle, one could also develop a multivector version of Function `cconv` to perform simultaneous convolutions in the y direction, but our timing tests indicate that this would only slightly enhance the overall performance (since the data for constant y is not stored consecutively in memory) and would prevent the 1D convolution work arrays from being reused for the y convolution at each fixed x. The memory savings in our method comes precisely from this reuse of temporary storage, which in turn requires that the y convolutions be computed serially.

As shown in Fig. 3.1, the resulting implicit 2D algorithm dramatically outperforms the explicit version: a 1024^2 implicit complex convolution is 1.91 times faster.

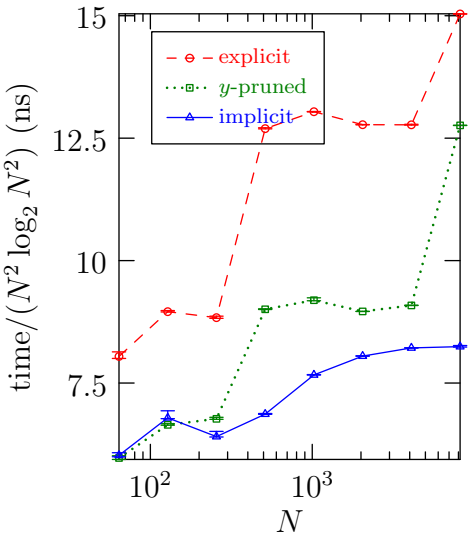

FIG. 3.1. *Comparison of computation times for explicitly and implicitly dealiased complex in-place 2D convolutions of size m^2.*

The third sentence of the quote from Steven G. Johnson on page 1 suggests a sensible optimization for explicitly padded 2D convolutions: one can omit for $m \le y < 2m$ the backward and forward Fourier transforms in the x direction. However potential data locality optimizations may be lost when a 2D convolution is expressed directly in terms of 1D transforms: as observed in Fig. 3.1, while a 1024^2 y-pruned explicit convolution is 1.26 times faster than a conventional explicit implementation, the pruned method becomes 1.80 times *slower* for the 8192^2 case. Our implicitly dealiased convolution is also subject to these same optimization losses, but the savings due to implicit padding, out-of-place subtransforms, the neglect of high-level bit reversal, and the immediate convolution of constant x columns (while still possibly in cache) outweigh these losses.

Because the same temporary arrays u and v are used for each column of the convolution, the memory requirement is $4m_xm_y + 2m_y$ complex words, far less than the $8m_xm_y$ complex words needed for an explicitly padded convolution.

3.2. Centered Hermitian 2D convolution. In two dimensions, the Fourier-centered Hermitian symmetry appears as $U_{-k,-\ell} = \overline{U_{k,\ell}}$. This symmetry is exploited in the centered Hermitian convolution algorithm described in Function `conv2`. As shown in Fig. 3.2, implicit padding again yields a dramatic improvement in speed.

When m_y is even, the memory usage for an implicitly dealiased $(2m_x-1)\times(2m_y-1)$ centered Hermitian convolution is $2(2m_x-1)m_y + 2(m_x+1)m_y + 2(m_y/2+1) = 6m_xm_y + m_y + 2$ complex words, compared with a minimum of $2(3m_x-2)(3m_y/2) = 9m_xm_y - 6m_y$ complex words required for an explicitly dealiased convolution.

```
Input: matrix f, matrix g
Output: matrix f
for j = 0 to m_y − 1 do
    fftpadBackward(f^T[j], U^T[j]);
    fftpadBackward(g^T[j], V^T[j]);
end
for i = 0 to m_x − 1 do
    cconv(f[i], g[i], u, v);
    cconv(U[i], V[i], u, v);
end
for j = 0 to m_y − 1 do
    fftpadForward(f^T[j], U^T[j]);
end
return f;
```

Function `cconv2(f,g,u,v,U,V)` returns an in-place implicitly dealiased convolution of $m_x \times m_y$ matrices `f` and `g` using temporary $m_x \times m_y$ matrices `U` and `V` and temporary vectors `u` and `v` of length m_y.

```
Input: matrix f, matrix g
Output: matrix f
for j = 0 to m_y − 1 do
    fft0padBackward(f^T[j], U^T[j]);
    fft0padBackward(g^T[j], V^T[j]);
end
for i = 0 to 2m_x − 2 do
    conv(f[i], g[i], u, v);
end
for i = 0 to m_x do
    conv(U[i], V[i], u, v);
end
for j = 0 to m_y − 1 do
    fft0padForward(f^T[j], U^T[j]);
end
return f;
```

Function `conv2(f,g,u,v,U,V)` returns an in-place implicitly dealiased centered Hermitian convolution of $(2m_x-1)\times m_y$ matrices `f` and `g` using temporary $(m_x+1)\times m_y$ matrices `U` and `V` and vectors `u` and `v` of length m_y.

3.3. 2D pseudospectral application. In our implementation, we allow the convolution inputs to be arrays of vectors, f_i and g_i $(i = 1,\ldots,M)$, interpreting in Functions `cconv`, `conv`, and `tconv`, the product $f * g$ as the element-by-element dot product $\sum_i f_i * g_i$. Convolving M input data blocks simultaneously like this enables, for example, the nonlinear term of the 2D incompressible Euler equation to be computed using five Fourier transforms (instead of three for each of the $M = 2$ input pairs). Specifically, the advective term $-\boldsymbol{u}\cdot\boldsymbol{\nabla}\omega = -(\hat{\boldsymbol{z}}\times\boldsymbol{\nabla}\nabla^{-2}\omega)\cdot\boldsymbol{\nabla}\omega$, which appears in Fourier space as

$$\sum_{\boldsymbol{p}} \frac{p_xk_y - p_yk_x}{|\boldsymbol{k}-\boldsymbol{p}|^2}\omega_{\boldsymbol{p}}\omega_{\boldsymbol{k}-\boldsymbol{p}},$$

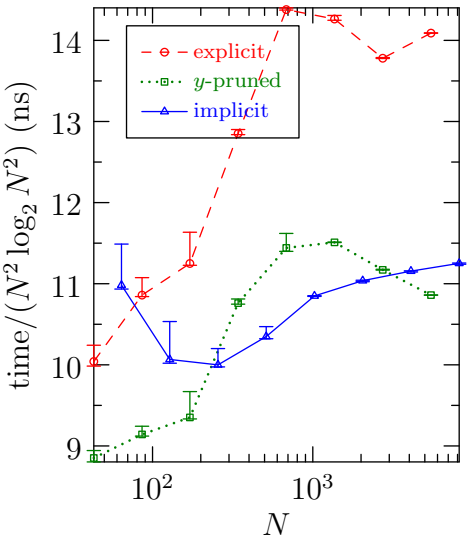

FIG. 3.2. *Comparison of computation times for explicitly and implicitly dealiased centered Hermitian in-place 2D convolutions of size* $(2m-1) \times m$.

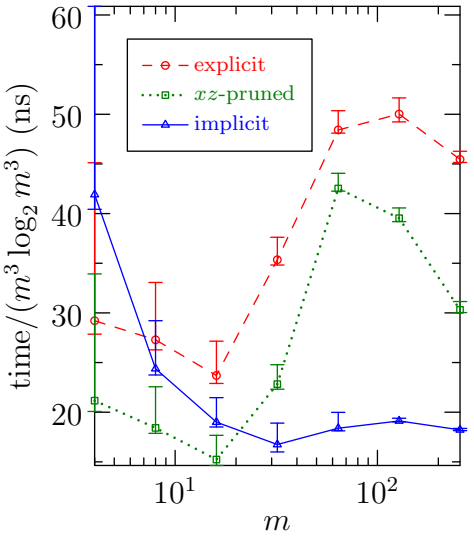

FIG. 3.3. *Comparison of computation times for explicitly and implicitly dealiased complex in-place 3D convolutions of size* m^3.

can be computed efficiently with the call $\texttt{conv2}(ik_x\omega, ik_y\omega, -ik_y\omega/k^2, ik_x\omega/k^2, \texttt{u}, \texttt{v})$, where `u` and `v` are work arrays.

3.4. Complex and centered Hermitian 3D convolutions. The decoupling of the 2D work arrays in Function `cconv2` facilitates the construction of an efficient 3D implicit complex convolution, as described in Function `cconv3`. As shown in Fig. 3.3, an implicit 256^3 convolution is 2.38 times faster than the explicit version, while an xz-pruned version is only 1.24 times faster. The memory usage of an implicitly padded 3D $m_x \times m_y \times m_z$ convolution is $4m_xm_ym_z + 2m_ym_z + 2m_z$ complex words, far less than the $16m_xm_ym_z$ complex words required by an explicit convolution based on power-of-two transforms.

A $(2m_x-1) \times (2m_y-1) \times (2m_z-1)$ implicit centered Hermitian 3D convolution was also implemented in an analogous manner. It required

$$6m_x(2m_y-1)m_z+2(m_y+1)m_z+2(m_z/2+1) = 12m_xm_ym_z-6m_xm_z+2m_ym_z+m_z+2$$

complex words, in comparison with the usual requirement of $27m_xm_ym_z$ complex words for explicit padding with power-of-two transforms.

4. Implicitly dealiased ternary convolutions. In this section, we show that implicit padding is well suited to dealiasing the centered ternary convolution

$$\sum_{p=-m+1}^{m-1} \sum_{q=-m+1}^{m-1} \sum_{r=-m+1}^{m-1} f_p g_q h_r \delta_{p+q+r,k},$$

which, for example, is required to compute the time evolution of the Casimir invariant $\int \omega^4 \, d\boldsymbol{x}$ associated with the nonlinearity of two-dimensional incompressible flow expressed in terms of the scalar vorticity ω. The basic building blocks for this problem are again the centered Fourier transform and Hermitian convolution.

Input: matrix f, matrix g
Output: matrix f
for $j = 0$ **to** $m_y - 1$ **do**
  for $k = 0$ **to** $m_z - 1$ **do**
    fftpadBackward($\mathsf{f}^T[k][j], \mathsf{U}^T[k][j]$);
    fftpadBackward($\mathsf{g}^T[k][j], \mathsf{V}^T[k][j]$);
  end
end
for $i = 0$ **to** $m_x - 1$ **do**
  cconv2($\mathsf{f}[i], \mathsf{g}[i], \mathsf{u}_1, \mathsf{v}_1, \mathsf{u}_2, \mathsf{v}_2$);
  cconv2($\mathsf{U}[i], \mathsf{V}[i], \mathsf{u}_1, \mathsf{v}_1, \mathsf{u}_2, \mathsf{v}_2$);
end
for $j = 0$ **to** $m_y - 1$ **do**
  for $k = 0$ **to** $m_z - 1$ **do**
    fftpadForward($\mathsf{f}^T[k][j], \mathsf{U}^T[k][j]$);
  end
end
return f;

Function cconv3(f,g) returns an in-place implicitly dealiased convolution of $m_x \times m_y \times m_z$ matrices f and g using temporary $m_x \times m_y \times m_z$ matrices U and V, $m_y \times m_z$ matrices u_2 and v_2, and vectors u_1 and v_1 of length m_z.

Input: vector f
Output: vector f, vector u
$\mathsf{u}[0] \leftarrow \mathsf{f}[0] \leftarrow 0$;
for $k = 1$ **to** $2m - 1$ **do**
  $\mathsf{u}[k] \leftarrow -i\zeta_{4m}^k \mathsf{f}[k]$;
end
$\mathsf{f} \leftarrow \mathtt{fft}^{-1}(\mathsf{f})$;
$\mathsf{u} \leftarrow \mathtt{fft}^{-1}(\mathsf{u})$;
return f;

Procedure fft0tpadBackward(f,u) stores the scrambled signed $4m$-padded centered backward Fourier transform values of a vector f of length $2m$ in f and an auxiliary vector u of length $2m$.

Input: vector f, vector u
Output: vector f
$\mathsf{f} \leftarrow \mathtt{fft}(\mathsf{f})$;
$\mathsf{u} \leftarrow \mathtt{fft}(\mathsf{u})$;
for $k = 1$ **to** $2m - 1$ **do**
  $\mathsf{f}[k] \leftarrow \mathsf{f}[k] + i\zeta_{4m}^{-k} \mathsf{u}[k]$;
end
return f/(4m);

Procedure fft0tpadForward(f,u) returns the inverse of Procedure fft0tpadBackward(f,u).

4.1. Implicit double-dealiased centered Fourier transform. Here the input data length is $2m - 1$, with the Fourier origin at index $m - 1$, so one needs to pad to $N \geq 4m - 3$ to prevent contamination due to wave beating.

Implicit padding is most efficiently implemented by padding the input vector with a single zero complex word at the beginning, to yield a vector of length $2m$, with the Fourier origin at index m. We choose m to be a power of 2 and $N = 4m$, with $U_k = 0$ for $k = -m$ and $k \geq m$.

On decomposing $j = 2\ell + r$, where $\ell = 0, \ldots, 2m - 1$ and $r \in \{0, 1\}$, we find on substituting $k' = k + m$ that

$$u_{2\ell+r} = \sum_{k=-m}^{m-1} \zeta_{2m}^{\ell k} \zeta_{4m}^{rk} U_k = \sum_{k'=0}^{2m-1} \zeta_{2m}^{\ell(k'-m)} \zeta_{4m}^{r(k'-m)} U_{k'-m} = (-1)^\ell i^{-r} \sum_{k=0}^{2m-1} \zeta_{2m}^{\ell k} \zeta_{4m}^{rk} U_{k-m}. \tag{4.1}$$

The forward transform is then given for $k = -m+1, \ldots, m-1$ by

$$\begin{aligned} 4mU_k &= \sum_{r=0}^{1} \zeta_{4m}^{-rk} \sum_{\ell=0}^{2m-1} \zeta_{2m}^{-\ell k} u_{2\ell+r} \\ &= \sum_{r=0}^{1} \zeta_{4m}^{-r(k'-m)} \sum_{\ell=0}^{2m-1} \zeta_{2m}^{-\ell(k'-m)} u_{2\ell+r} \\ &= \sum_{r=0}^{1} \zeta_{4m}^{-rk'} i^r \sum_{\ell=0}^{2m-1} \zeta_{2m}^{-\ell k'} (-1)^\ell u_{2\ell+r}, \qquad k' = 1, \ldots, 2m-1. \end{aligned} \tag{4.2}$$

For a ternary convolution, the product of the three factors $(-1)^\ell$ (one for each input vector) arising from Eq. (4.1) and the factor $(-1)^\ell$ in Eq. (4.2) cancel. Procedures `fft0tpadBackward` and `fft0tpadForward` each have an operation count asymptotic to $4Km \log_2 m$. As they operate fully in place on their arguments, with no additional storage requirements, it is straightforward to implement strided multivector versions of these algorithms.

4.2. Implicitly dealiased centered Hermitian 1D ternary convolution. Let us now consider a centered Hermitian ternary convolution with $N = 4m$, where m is a power of 2. For explicit padding, one needs to pad the m non-negative wavenumbers with $m+1$ zeros, for a total vector length of $2m+1$.

On decomposing $j = 2\ell + r$, where $\ell = 0, \ldots, 2m-1$ and $r \in \{0, 1\}$, the backward transform is given by

$$u_{2\ell+r} = \sum_{k=-m}^{m-1} \zeta_{2m}^{\ell k} \zeta_{4m}^{rk} U_k.$$

If we set $U_m = 0$, the real components $u_{2\ell+r}$ can thus be computed by taking a complex-to-real transform of $\{\zeta_{4m}^{rk} U_k : k = 0, \ldots, m\}$.

The forward transform is

$$4mU_k = \sum_{r=0}^{1} \zeta_{4m}^{-rk} \sum_{\ell=0}^{2m-1} \zeta_{2m}^{-\ell k} u_{2\ell+r}, \qquad k = -m+1, \ldots, m-1.$$

The resulting implicitly padded centered Hermitian ternary convolution, Function `tconv`, has an operation count of $8Km \log_2 m$. Five of the eight required Fourier transforms can be done out of place. In Fig. 4.1 we show that this algorithm is competitive with explicit padding. Function `tconv` requires $6(m+1)$ complex words of storage, slightly more than the $3(2m+1) = 6m+3$ complex words needed for explicit padding.

Just as for convolutions, the performance and memory benefits of dealiasing higher-order convolutions *via* implicit padding manifest themselves only in higher dimensions. For example, in Fig. 4.2, we observe for $m_x = m_y = 4096$ that the implicit $(2m_x - 1) \times m_y$ centered Hermitian ternary convolution computed with Function `tconv2` is 2.28 times faster than an explicit version. The memory usage for a $(2m_x - 1) \times m_y$ implicit centered Hermitian ternary convolution is $6 \cdot 2m_x(m_y + 1) + 3(m_y + 1) = 12m_x m_y + 12m_x + 3m_y + 3$ complex words, compared with $3 \cdot 4m_x(2m_y + 1) = 24m_x m_y + 12m_x$ complex words (using power-of-two transforms) for an explicit version. In contrast, the corresponding y-pruned convolution requires the same amount of storage as, but is 1.42 times faster than, an explicitly padded version.

Input: vector f, vector g, vector h
Output: vector f
for $k = 0$ **to** $m - 1$ **do**
  $\mathsf{u}[k] \leftarrow \zeta_{4m}^{k}\mathsf{f}[k]$;
  $\mathsf{v}[k] \leftarrow \zeta_{4m}^{k}\mathsf{g}[k]$;
  $\mathsf{w}[k] \leftarrow \zeta_{4m}^{k}\mathsf{h}[k]$;
end
$\mathsf{u}[m] \leftarrow \mathsf{v}[m] \leftarrow \mathsf{w}[m] \leftarrow 0$;
$\mathsf{u} \leftarrow \texttt{crfft}^{-1}(\mathsf{u})$;
$\mathsf{v} \leftarrow \texttt{crfft}^{-1}(\mathsf{v})$;
$\mathsf{w} \leftarrow \texttt{crfft}^{-1}(\mathsf{w})$;
$\mathsf{v} \leftarrow \mathsf{u} * \mathsf{v} * \mathsf{w}$;
$\mathsf{u} \leftarrow \texttt{rcfft}(\mathsf{v})$;
$\mathsf{f}[m] \leftarrow \mathsf{g}[m] \leftarrow \mathsf{h}[m] \leftarrow 0$;
$\mathsf{v} \leftarrow \texttt{crfft}^{-1}(\mathsf{f})$;
$\mathsf{w} \leftarrow \texttt{crfft}^{-1}(\mathsf{g})$;
$\mathsf{g} \leftarrow \texttt{crfft}^{-1}(\mathsf{h})$;
$\mathsf{v} \leftarrow \mathsf{v} * \mathsf{w} * \mathsf{g}$;
$\mathsf{f} \leftarrow \texttt{rcfft}(\mathsf{v})$;
for $k = 0$ **to** $m - 1$ **do**
  $\mathsf{f}[k] \leftarrow \mathsf{f}[k] + \zeta_{4m}^{-k}\mathsf{u}[k]$;
end
return f/(4m);

Function `tconv(f,g,h,u,v,w)` computes an in-place implicitly dealiased centered Hermitian ternary convolution of three centered Hermitian vectors f, g, and h, using three temporary vectors u, v, and w, each of length $m + 1$.

Input: matrix f, matrix g, matrix h
Output: matrix f
for $j = 0$ **to** $m_y - 1$ **do**
  `fft0tpadBackward`$(\mathsf{f}^T[j], \mathsf{U}^T[j])$;
  `fft0tpadBackward`$(\mathsf{g}^T[j], \mathsf{V}^T[j])$;
  `fft0tpadBackward`$(\mathsf{h}^T[j], \mathsf{W}^T[j])$;
end
for $i = 0$ **to** $2m_x - 1$ **do**
  `tconv`$(\mathsf{f}[i], \mathsf{g}[i], \mathsf{h}[i], \mathsf{u}, \mathsf{v}, \mathsf{w})$;
  `tconv`$(\mathsf{U}[i], \mathsf{V}[i], \mathsf{W}[i], \mathsf{u}, \mathsf{v}, \mathsf{w})$;
end
for $j = 0$ **to** $m_y - 1$ **do**
  `fft0tpadForward`$(\mathsf{f}^T[j], \mathsf{U}^T[j])$;
end
return f;

Function `tconv2(f,g,h)` returns an in-place implicitly dealiased centered Hermitian ternary convolution of $2m_x \times (m_y + 1)$ matrices f, g, and h using temporary $2m_x \times (m_y + 1)$ matrices U, V, and W and vectors u, v and w of length $m_y + 1$.

5. Concluding remarks. Explicitly padded Fourier transforms are frequently used to dealias convolutions in one or more dimensions. In this work we have developed an efficient method for avoiding explicit zero padding in multidimensional convolutions, thereby saving both memory and computation time. The key idea that was exploited was the decoupling of temporary storage and user data, which in higher dimensions allows the reuse of storage space. The resulting increased data locality significantly enhanced performance by as much as a factor of 2. The savings in memory use, obtained by computing the Fourier transformed data in blocks rather than all at once, was equally significant: asymptotically, as $m_x \to \infty$, an implicit complex convolution requires one-half of the memory needed for a zero-padded convolution in two dimensions and one-quarter in three dimensions. In the centered Hermitian case, the memory use in two dimensions is 2/3 of the amount used for an explicit convolution and 4/9 of the corresponding storage requirement in three dimensions.

Even in one dimension, where implicit padding can be implemented competitively with conventional methods, the method has notable advantages. For the intended

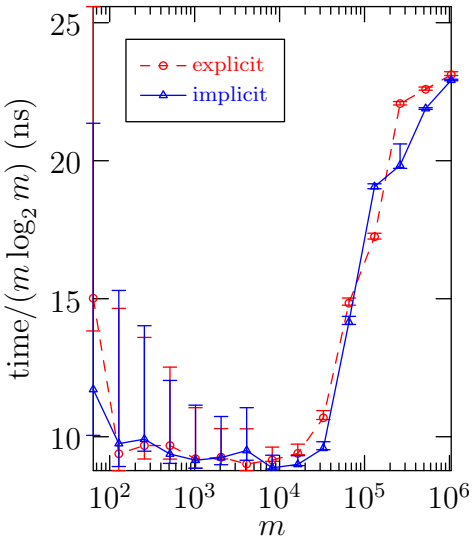

FIG. 4.1. *Comparison of computation times for explicitly and implicitly dealiased centered Hermitian in-place 1D ternary convolutions of length m.*

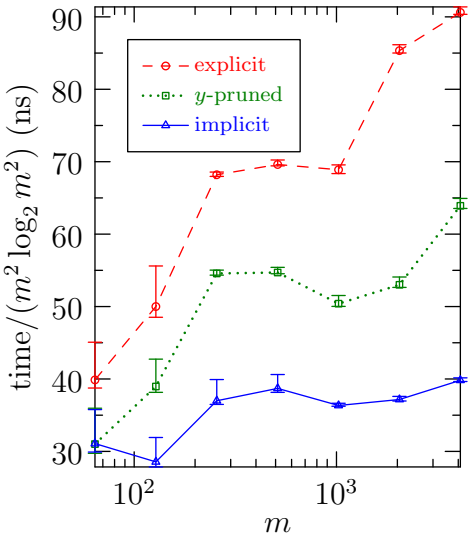

FIG. 4.2. *Comparison of computation times for explicitly and implicitly dealiased centered Hermitian in-place 2D ternary convolutions of size $(2m-1) \times m$.*

application to partial differential equations, there is flexibility in the choice of the exact convolution size. This is why we consider for each algorithm only those vector lengths that maximize performance. On the other hand, for those applications where the size of the convolution is dictated by external criteria, implicit padding effectively expands the available set of efficient convolution sizes to include integral powers of 2, a case of practical significance. Canuto *et al.* [3, p.136] point out that if only a power-of-two transform were available for a centered convolution, zero padding a vector of length $m = 2^k$ would require a transform size of $2m$, yielding an even slightly higher operation count, $6Km \log_2(2m)$, than the $6Km \log_2 m$ operations required for phase-shift dealiasing. The availability of implicitly dealiased convolutions now makes this argument moot.

Another advantage of implicit padding is the ability of the algorithm to work directly on raw unpadded user data without the inconvenience or extra storage requirements of a separate padding buffer. Having a prewritten, well-tested dealiased convolution that takes care of dealiasing internally is a major convenience for the average user. For 2D and 3D Hermitian convolutions, a prepackaged routine should also automatically enforce Hermitian symmetry of the data along the x axis or the xy plane, respectively. With the highly optimized implementations of the algorithms developed in this work made available in the open source package `FFTW++` [1], writing a pseudospectral code for solving nonlinear partial differential equations should now be a relatively straightforward exercise.

REFERENCES

[1] JOHN C. BOWMAN AND MALCOLM ROBERTS, `FFTW++`: *A fast Fourier transform* C^{++} *header class for the* `FFTW3` *library.* `http://fftwpp.sourceforge.net`, May 6, 2010.

[2] O. BUNEMAN, *Stable on-line creation of sines or cosines of successive angles*, Proceedings of

the IEEE, 75 (1987), pp. 1434–1435.

[3] C. Canuto, M.Y. Hussaini, A. Quarteroni, and T.A. Zang, *Spectral Methods: Fundamentals in Single Domains*, Scientific Computation, Springer, Berlin, 2006.

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

[5] Matteo Frigo and Steven G. Johnson, *Pruned FFTs and* `FFTW`. `http://www.fftw.org/pruned.html`.

[6] ———, *The design and implementation of* `FFTW3`, Proceedings of the IEEE, 93 (2005), pp. 216–231.

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

[8] S.G. Johnson and M. Frigo, *A modified split-radix FFT with fewer arithmetic operations*, IEEE Transactions on Signal Processing, 55 (2007), p. 111.

[9] Steven G. Johnson and Matteo Frigo, *Implementing FFTs in practice*. `http://cnx.org/content/m16336/1.14/`.

[10] T. Lundy and J. Van Buskirk, *A new matrix approach to real FFTs and convolutions of length* $2k$, Computing, 80 (2007), pp. 23–45.

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

[12] G. S. Patterson Jr. and Steven A. Orszag, *Spectral calculations of isotropic turbulence: Efficient removal of aliasing interactions*, Physics of Fluids, 14 (1971), p. 2538.

[13] M. Tasche and H. Zeuner, *Improved roundoff error analysis for precomputed twiddle factors*, Journal of Computational Analysis and Applications, 4 (2002), pp. 1–18.

[14] Clive Temperton, *Self-sorting in-place fast Fourier transforms*, SIAM Journal on Scientific and Statistical Computing, 12 (1991), pp. 808–823.