

DEALIASED CONVOLUTIONS FOR PSEUDOSPECTRAL SIMULATIONS

Malcolm Roberts & John C. Bowman

Department of Mathematical and Statistical Sciences, University of Alberta, Canada

Summary Efficient algorithms have recently been developed for calculating dealias 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 memory savings is achieved by computing the in-place Fourier transform of the data in blocks, rather than all at once. The technique also allows one to dealias the n -ary convolutions that arise on Fourier transforming cubic and higher powers. Implicitly dealias convolutions can be built on top of state-of-the-art adaptive fast Fourier transform libraries like FFTW. Vectorized multidimensional implementations for the complex and centered Hermitian (pseudospectral) cases have already been implemented in the open-source software FFTW++. With the advent of this library, writing a high-performance dealias pseudospectral code for solving nonlinear partial differential equations has now become a relatively straightforward exercise. New theoretical estimates of computational complexity and memory use are provided, including corrected timing results for 3D pruned convolutions and further consideration of higher-order convolutions.

ONE-DIMENSIONAL CONVOLUTIONS

A discrete non-centered convolution is a binary operation on two sequences F_k and G_k , $k = 0, \dots, m-1$, and is defined as

$$(F * G)_k \doteq \sum_{p=0}^{m-1} F_p G_{k-p}, \quad k = 0, \dots, m-1.$$

Non-centered convolutions appear in many areas, notably image processing, statistical analysis, and primality tests.

Computing a convolution as a direct sum requires $\mathcal{O}(m^2)$ operations. This can be reduced to $3Km \log_2 m$ operations (with $K = 34/9$ [6, 7]) by using fast Fourier transforms (FFTs). However, FFTs transform periodic data to periodic data; an FFT-based convolution is therefore a discrete cyclic convolution, where all the indices are considered equivalent mod m . The difference between linear and cyclic convolutions is called the *aliasing error*.

A centered convolution, which takes input arrays F_k, G_k , $k = -m+1, \dots, m-1$, and has output

$$(F * G)_k \doteq \sum_{p=k-m+1}^{m-1} F_p G_{k-p}, \quad k = -m+1, \dots, m-1,$$

appears in pseudospectral simulations of the Navier–Stokes equations and other non-linear differential equations.

Explicit zero-padding and phase-shift dealiasing

One way to eliminate the aliasing error (i.e. *dealias the convolution*) is to extend the input arrays by adding an appropriate number of zeroes. Arrays must be padded to twice their length for non-centered convolutions, but need only be extended by 50% [8] for centered convolutions. The resulting multiplications by zero remove the aliasing error. Explicit zero padding increases the memory footprint by 100% (or 50% for centered convolutions) over a cyclic convolution, and the computational effort increases to $6Km \log_2 m$ (or $\frac{9}{2}Km \log_2 m$ for the centered Hermitian case).

An alternative method is phase-shift dealiasing [9, 4], which allows one to calculate a linear convolution by calculating two cyclic convolutions with opposite aliasing error. This method is rarely used in one dimension, as phase-shift dealiasing increases the memory footprint by 100% and the computational cost is $6Km \log_2 m$ for both centered and non-centered convolutions.

Implicit zero padding

The key idea behind implicitly padding FFT-based convolutions is that the zero-padded input array can be transformed without having to store or process the zero padding [3]. For the non-centered case with 100% padding, the inverse Fourier transform $f_j = \sum_{k=0}^{2m-1} \zeta_{2m}^{kj} F_k$, where $\zeta_N = e^{2\pi i/N}$ and $F_k = 0$ if $k \geq m$, can be written as

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

Thus, data with even and odd indices can be stored in separate, non-contiguous arrays. These calculations can be done with existing FFT libraries such as FFTW [5]. After using the same algorithm to compute g_{2j} and g_{2j+1} , the (scaled) convolution is the forward transform of $f_{2\ell} g_{2\ell}$ and $f_{2\ell+1} g_{2\ell+1}$:

$$2m(F * G)_k = \sum_{j=0}^{2m-1} \zeta_{2m}^{-kj} f_j g_j = \sum_{\ell=0}^{m-1} \zeta_m^{-k\ell} f_{2\ell} g_{2\ell} + \zeta_{2m}^{-k} \sum_{\ell=0}^{m-1} \zeta_m^{-k\ell} f_{2\ell+1} g_{2\ell+1}, \quad k = 0, \dots, m-1.$$

Method	Complexity	Memory Footprint
Explicit Zero Padding	$6 Km \log m$	$4m$
Implicit Zero Padding	$6 Km \log m$	$4m$

Table 1. Comparison of methods for dealiasing one-dimensional non-centered complex binary convolutions of length m .

Method	Complexity	Memory Footprint
Phase-Shift Dealiasing	$6 Km \log m$	$4m$
Explicit Zero Padding	$\frac{9}{2} Km \log m$	$3m$
Implicit Zero Padding	$\frac{9}{2} Km \log m$	$3m$

Table 2. Comparison of methods for dealiasing one-dimensional centered Hermitian convolutions of length m .

The centered case works similarly, except that we divide the output into three arrays of length m .

Comparison of dealiasing techniques for one-dimensional convolutions

Explicitly and implicitly padded convolutions have similar computational complexities and memory use, as shown in Tables 1–2, and similar run times, as shown in Figures 1–2.

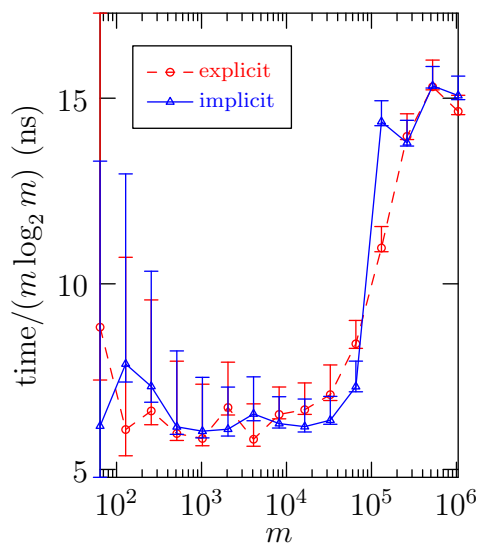


Figure 1. Comparison of computation times for explicitly and implicitly dealiasied non-centered complex in-place 1D convolutions of length m .

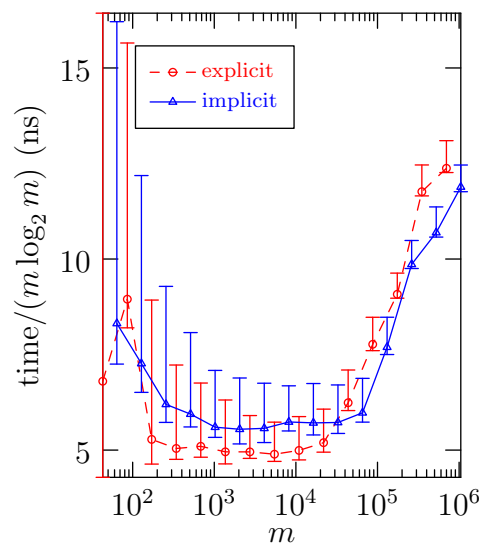


Figure 2. Comparison of computation times for explicitly and implicitly dealiasied centered Hermitian in-place 1D convolutions of length m .

While implicit zero padding allows one to perform convolutions without having to allocate memory contiguously, the memory use and computation time are effectively identical to those for the explicitly padded case. The true advantage of this method only becomes apparent when one considers convolutions in higher dimensions.

MULTI-DIMENSIONAL CONVOLUTIONS

Two-dimensional convolutions can be calculated by applying an inverse Fourier transform first in the x direction and then in the y direction, multiplying the output, and taking the forward transform in the y and x directions. Using explicit padding, this would require allocating a contiguous array of size $2m \times 2m$ for each input array f and g . Since a large part of this array is zero, one seemingly straightforward optimization is to skip or “prune” FFTs when the output is known *a priori* to be zero. While generally useful, this can be more time-consuming than taking transforms of the entire block because of the lack of FFT routines specifically optimized for pruned transforms.

To implicitly pad non-centered data, one first performs a discontinuous transform in the x direction, producing two

Method	Complexity	Memory Footprint
Explicit Zero Padding without Pruning	$3 \cdot 2^d d K m^d \log m$	$2^{d+1} m^d$
Explicit Zero Padding with Pruning	$6 (2^d - 1) K m^d \log m$	$2^{d+1} m^d$
Implicit Zero Padding	$6 (2^d - 1) K m^d \log m$	$4 m^d$

Table 3. Comparison of methods for dealiasing d -dimensional non-centered complex binary convolutions of size m^d .

discontiguous arrays of size $m \times m$ per input array. Then, one performs a convolution on the x -transformed arrays using a 1D implicit convolution, requiring only $2m$ words of extra storage. Since the x -transforms can be done one at a time, these $2m$ words of storage need only be allocated once. Consequently, an implicitly padded non-centered 2D convolution uses half of the memory required for an explicitly padded 2D convolution.

Dealiasing a convolution of centered data using implicit padding is performed similarly, except that the input arrays of length $2m - 1$ are discontinuously expanded to length $3m$. An implicitly padded centered 2D convolution uses two-thirds of the memory needed for an explicitly padded convolution.

Implicit padding can be extended to d dimensions, with an implicitly padded non-centered convolution requiring $1/2^{d-1}$ of the memory needed for its explicitly padded counterpart, and an implicitly padded centered convolution requiring $(2/3)^{d-1}$ of the memory needed for an explicitly padded version. The numerical error for implicit padding is similar to that for explicit padding. Skipping Fourier transforms on arrays of zeroes and decreased memory bandwidth requirements provides a speed-up by a factor of approximately two over explicit padding.

Comparison of dealiasing techniques for multi-dimensional convolutions

Implicitly padded multi-dimensional convolutions have the same computational complexity as explicit zero-padded convolutions, but require much less memory, as shown in Tables 3–4. Moreover, they are, in practice, faster than explicitly padded convolutions, as may be seen in Figures 3–5. If one requires all components of the convolution, then zero padding is superior to phase-shift dealiasing. If one does not require that all the modes be recovered, an alternative is to use a combination of phase-shift dealiasing and explicit zero padding, which allows one to recover approximately 44% of the modes for centered 3D convolutions. However, this technique still requires more memory and is computationally more complex than implicit zero padding. Unlike the corresponding figure in [3], Figure 4 depicts fully pruned convolutions.

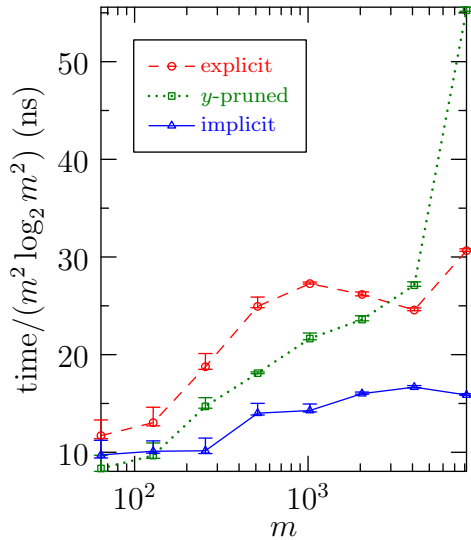


Figure 3. Comparison of computation times for explicitly and implicitly dealiasied non-centered complex in-place 2D convolutions of size m^2 .

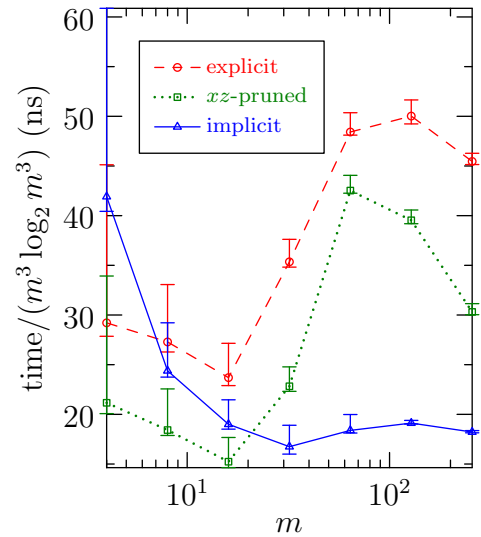


Figure 4. Comparison of computation times for explicitly and implicitly dealiasied non-centered complex in-place 3D convolutions of size m^3 .

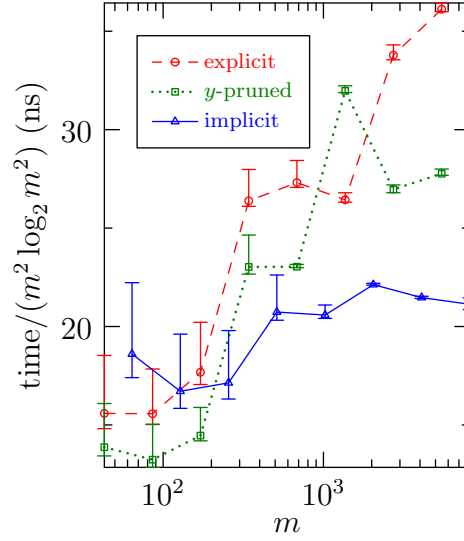


Figure 5. Comparison of computation times for explicitly and implicitly dealiased centered Hermitian in-place 2D convolutions of size $(2m - 1) \times m$.

Method	Complexity	Memory Footprint
Phase-Shift Dealiasing	$3 \cdot 2^{2d-1} d K m^d \log m$	$2^{2d} m^d$
Partial Phase-Shift Dealiasing	$3 \cdot 2^d d K m^d \log m$	$2^{d+1} m^d$
Explicit Zero Padding without Pruning	$\frac{3^{d+1}}{2} d K m^d \log m$	$3^d m^d$
Explicit Zero Padding with Pruning	$\frac{9}{2} (3^d - 2^d) K m^d \log m$	$3^d m^d$
Implicit Zero Padding	$\frac{9}{2} (3^d - 2^d) K m^d \log m$	$3 \cdot 2^{d-1} m^d$

Table 4. Comparison of methods for dealiasing d -dimensional centered Hermitian convolutions of size $(2m - 1)^{d-1} \times m$.

Method	Complexity	Memory Footprint
Explicit padding	$(n+1)n^d d K m^d \log nm$	$n^{d+1} m^d$
Explicit padding with pruning	$n(n+1) \frac{n^d-1}{n-1} K m^d \log nm$	$n^{d+1} m^d$
Implicit padding	$n(n+1) \frac{n^d-1}{n-1} K m^d \log nm$	$n^2 m^d$

Table 5. Comparison of methods for dealiasing d -dimensional n -ary non-centered complex convolutions on data of size m^d .

Method	Complexity	Memory Footprint
Explicit padding	$\frac{1}{2}(n+1)^{d+1} d K m^d \log nm$	$\frac{n(n+1)^d}{2} m^d$
Explicit padding with pruning	$\frac{1}{2}(n+1)^2 \frac{(n+1)^d-2^d}{n-1} K m^d \log nm$	$\frac{n(n+1)^d}{2} m^d$
Implicit padding	$\frac{1}{2}(n+1)^2 \frac{(n+1)^d-2^d}{n-1} K m^d \log nm$	$n(n+1)2^{d-2} m^d$

Table 6. Comparison of methods for dealiasing d -dimensional n -ary centered Hermitian convolutions on data of size $(2m-1)^{d-1} \times m$.

HIGHER-ORDER CONVOLUTIONS

The ℓ^{th} component of the non-centered n -ary convolution of n vectors f_k^1, \dots, f_k^n , for $k = 0, \dots, m-1$, is defined as

$$*(f^1, \dots, f^n)_\ell \doteq \sum_{\ell_1, \dots, \ell_n=0}^{m-1} f_{\ell_1}^1 \dots f_{\ell_n}^n \delta_{\ell_1+\dots+\ell_n, \ell}, \quad \ell = 0, \dots, m-1.$$

The ℓ^{th} component of the centered n -ary convolution of n vectors f_k^1, \dots, f_k^n , for $k = -m+1, \dots, m-1$, is defined as

$$*(f^1, \dots, f^n)_\ell \doteq \sum_{\ell_1, \dots, \ell_n=-m+1}^{m-1} f_{\ell_1}^1 \dots f_{\ell_n}^n \delta_{\ell_1+\dots+\ell_n, \ell}, \quad \ell = -m+1, \dots, m-1.$$

Such higher-order convolutions arise when performing simulations of the compressible Navier–Stokes equations or when considering high-order Casimir invariants in pseudo-spectral simulations [1].

If the input vectors are of infinite length, then an n -ary convolution can be computed via $n-1$ binary convolutions. However, it is important to note that this is not the case for convolutions of finite-length vectors of fixed length. If we consider the ternary convolution of f , g , and h , with data for each mode in the range $-m+1, \dots, m-1$, the $(m-1)^{\text{st}}$ component of the ternary convolution will include the term $f_{m-1}g_{m-1}h_{-m+1}$. However, if one computes $(f * g) * h$, one excludes this term, as $f_{m-1}g_{m-1}$ would not contribute to $f * g$.

In order to remove aliasing errors, one must zero pad ternary convolutions twice as much as binary convolutions, and quaternary convolutions three times as much, and so on. The computational complexity and memory requirements for implicitly padded n -ary multi-dimensional convolutions are given in Tables 5–6. Implicitly dealiased convolutions have the same computational complexity as pruned explicitly dealiased convolutions but require significantly less memory.

CONCLUSION

As dimension, order of convolution, and problem size increase, the advantage of implicit padding becomes more and more apparent. The algorithms described above are available in the open-source software package FFTW++ [2], including algorithms for non-centered, non-Hermitian data in one, two, and three dimensions; centered Hermitian data in one, two, and three dimensions; and centered, Hermitian ternary convolutions in one and two dimensions. These algorithms will allow researchers to create faster, less memory-intensive pseudo-spectral codes with minimal effort. We expect implicitly padded convolutions to become standard tools in computational fluid dynamics.

References

- [1] John C. Bowman. Casimir cascades in two-dimensional turbulence. *J. Fluid Mech.*, 2011. To be submitted.

- [2] John C. Bowman and Malcolm Roberts. FFTW++: A fast Fourier transform C++ header class for the FFTW3 library. <http://fftwpp.sourceforge.net>, 2010.
- [3] John C. Bowman and Malcolm Roberts. Efficient dealiased convolutions without padding. *SIAM J. Sci. Comput.*, 33(1):386–406, 2011.
- [4] C. Canuto, M.Y. Hussaini, A. Quarteroni, and T.A. Zang. *Spectral Methods: Fundamentals in Single Domains*. Scientific Computation. Springer, Berlin, 2006.
- [5] Matteo Frigo and Steven G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005.
- [6] S.G. Johnson and M. Frigo. A modified split-radix FFT with fewer arithmetic operations. *IEEE Transactions on Signal Processing*, 55(1):111, 2007.
- [7] T. Lundy and J. Van Buskirk. A new matrix approach to real FFTs and convolutions of length $2k$. *Computing*, 80(1):23–45, 2007.
- [8] Steven A. Orszag. Elimination of aliasing in finite-difference schemes by filtering high-wavenumber components. *Journal of the Atmospheric Sciences*, 28:1074, 1971.
- [9] G. S. Patterson Jr and Steven A. Orszag. Spectral calculations of isotropic turbulence: Efficient removal of aliasing interactions. *Physics of Fluids*, 14:2538, 1971.