Hybrid Dealiased Convolutions

by

Noel Murasko

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Applied Mathematics

Department of Mathematical and Statistical Sciences
University of Alberta

Abstract

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

Dedicated to Margaret King.

Acknowledgements

I would like to thank my supervisor Dr. John C. Bowman for his endless patience and support, both in this project and in my graduate studies as a whole. Without his guidance and collaboration, this project would not have been possible and I would not be the person I am today. I would like to thank Dr. Clement W. Bowman for the financial assistance provided by the Clement W. Bowman Mathematical Turbulence Scholarship.

Robert Joseph George has been an ongoing collaborator on the unequal case; almost everything I have to say about the matter is the result of our discussions.

I would also like to thank Emily Korfanty for managing to read my early drafts. As always, her comments have been invaluable.

Table of Contents

List of Figures

Chapter 1

Introduction

Convolutions are one of the most ubiquitous operations in mathematics, and so it should come as no surprise that discrete convolutions are found virtually everywhere in the computational sciences. There are two main types of discrete convolution: linear and circular. Linear convolutions are found in many applications such as the pseudospectral simulation of partial differential equations, signal processing, and machine learning. Efficiently computing linear convolutions is the main goal of this work.

Direct computation of the discrete convolution of arrays with length n requires $\mathcal{O}(n^2)$ operations. In practice, this is far too slow for large arrays. We can obtain a much faster algorithm by making use of the *discrete Fourier transform* (DFT), which can be computed using $\mathcal{O}(n \log n)$ operations.

Gauss discovered the first (known) fast algorithm for computing the DFT [Gau66]. This algorithm could be used on arrays of any composite integer length. Unfortunately, it went unpublished during Gauss's lifetime and wasn't widely recognized; while several fast algorithms were discovered for special cases, over a century passed before Gauss's algorithm was independently rediscovered [HJB85] by J. Cooley and J. Tukey [CT65]. The importance of their work should not be understated; it is rightfully regarded as one of the most important algorithms of the last century. We refer to any algorithm which computes the DFT with $\mathcal{O}(n \log n)$ operations as a *fast Fourier transform* (FFT).

We can use FFTs to compute fast convolutions by utilizing the convolution theorem. Informally, the convolution theorem states that the DFT of the convolution of two arrays is the product of their DFTs. Thus we can compute convolutions using $\mathcal{O}(n \log n)$ operations: we take the FFT of the inputs, multiply the results, and then compute the inverse FFT.

The caveat with using the DFT and the convolution theorem is that it results in a circular convolution. And so if we want a linear convolution, our results will contain errors, which are called *aliasing errors*. *Dealiasing* refers to any method which uses the convolution theorem to compute linear convolutions without such aliasing errors.

1.A Convolutions and dealiasing

We begin by providing rigorous definitions for the terms mentioned above. In this work, a one-dimensional *array* (or simply an array) refers to a finite-length sequence of complex numbers. A d-dimensional array is an ordered tuple of d one-dimensional arrays.

For our purposes, the linear and circular convolution are defined as binary operations on arrays:

Definition 1.1. Given two arrays $\boldsymbol{f} = \{f_j\}_{j=0}^{L_f-1}$ and $\boldsymbol{g} = \{g_j\}_{j=0}^{L_g-1}$, the *linear* *convolution* of $\boldsymbol{f}$ and $\boldsymbol{g}$ is the array defined by

$$(\boldsymbol{f} * \boldsymbol{g})_k \doteq \sum_{j=0}^{L_f-1} f_j g_{k-j}, \quad k \in \{0, \dots, L_f + L_g - 1\},$$

where it is understood that $g_{k-j} = 0$ when $k - j < 0$ or $k - j \geq L_g$ (we use $\doteq$ to denote definitions).

Definition 1.2. Given two arrays $\boldsymbol{f} = \{f_j\}_{j=0}^{L-1}$ and $\boldsymbol{g} = \{g_j\}_{j=0}^{L-1}$, their *circular convolution* is the array defined by

$$(\boldsymbol{f} \circledast \boldsymbol{g})_j \doteq \sum_{j=0}^{L-1} f_j g_{(k-j) \bmod L}, \quad k \in \{0, \dots, L-1\}.$$

While these definitions are for one-dimensional arrays, there is a natural extension to higher dimensions. If $\boldsymbol{f}$ and $\boldsymbol{g}$ are d-dimensional arrays of sizes $L_1 \times \cdots \times L_d$ and $\tilde{L}_1 \times \cdots \times \tilde{L}_d$ respectively, their linear convolution is the d-dimensional array defined by

$$(\boldsymbol{f} * \boldsymbol{g})_{k_1,\dots,k_d} \doteq \sum_{j_1=0}^{L_1} \cdots \sum_{j_d=0}^{L_d} f_{j_1,\dots,j_d} g_{k_1-j_1,\dots,k_d-j_d},$$

$$k_i \in \{0, \dots, L_i + \tilde{L}_i - 1\},\ i \in \{1, \dots, d\}.$$

Thus, in multiple dimensions, the linear convolution amounts to a one-dimensional linear convolution over each dimension. The multidimensional circular convolution is defined similarly. We can therefore focus our attention on the one-dimensional case.

The circular convolution is deeply related to the *discrete Fourier transform* (DFT). Let $\zeta_N \doteq e^{2\pi i/N}$ denote the $N\mathrm{th}$ primitive root of unity. Recall that the DFT of an array $\boldsymbol{f} = \{f_j\}_{j=0}^{L-1}$ is defined by:[1]

$$F_k = \mathrm{DFT}[\boldsymbol{f}]_k \doteq \sum_{j=0}^{L-1} \zeta_L^{kj} f_j, \quad k \in \{0, \dots, L-1\}.$$

Similarly, the *inverse discrete Fourier transform* is given by

$$f_j = \mathrm{DFT}^{-1}[\boldsymbol{F}]_j \doteq \frac{1}{L} \sum_{k=0}^{L-1} \zeta_L^{-jk} F_k, \quad j \in \{0, \dots, L-1\}.$$

The relationship between the circular convolution and the DFT is exemplified by the convolution theorem:

Theorem 1.3 (Discrete Convolution Theorem)**.** *Let $\boldsymbol{f}$ and $\boldsymbol{g}$ be arrays of the same length. Then we have:*

$$\mathrm{DFT}(\boldsymbol{f} \circledast \boldsymbol{g}) = \mathrm{DFT}(\boldsymbol{f}) \odot \mathrm{DFT}(\boldsymbol{g}),$$

where $\odot$ denotes Hadamard (element-wise) multiplication.

Proof. Let $L \in \mathbb{N}$ be the length of $\boldsymbol{f}$ and $\boldsymbol{g}$. We compute:

$$\begin{aligned}
\mathrm{DFT}[\boldsymbol{f} \circledast \boldsymbol{g}]_k &= \sum_{j=0}^{L-1} \zeta_L^{jk} (\boldsymbol{f} \circledast \boldsymbol{g})_j = \sum_{j=0}^{L-1} \zeta_L^{jk} \sum_{\ell=0}^{L-1} f_\ell g_{(j-\ell) \bmod L} \\
&= \sum_{\ell=0}^{L-1} f_\ell \sum_{j=0}^{L-1} \zeta_L^{jk} g_{(j-\ell) \bmod L} = \sum_{\ell=0}^{L-1} f_\ell \sum_{j=-\ell}^{L-\ell-1} \zeta_L^{(j+\ell)k} g_{j \bmod L} \\
&= \sum_{\ell=0}^{L-1} \zeta_L^{\ell k} f_\ell \sum_{j=-\ell}^{L-\ell-1} \zeta_L^{jk} g_{j \bmod L} = \sum_{\ell=0}^{L-1} \zeta_L^{\ell k} f_\ell \sum_{j=0}^{L-1} \zeta_L^{jk} g_j, \\
&= \mathrm{DFT}[\boldsymbol{f}]_k \, \mathrm{DFT}[\boldsymbol{g}]_k.
\end{aligned}$$

□

[1] In this work, we use lowercase letters to denote input arrays and uppercase letters to denote the corresponding DFT.

Note that in contrast to the linear convolution, the circular convolution and the DFT assume their inputs are periodic. Given an array $\{f_j\}_{j=0}^{L-1}$, both operations take $f_{j+nL} \doteq f_j$ for all $n \in \mathbb{Z}, j \in \{0, \ldots, L-1\}$. For all $n \neq 0$, we say that f_{j+nL} are *aliases* of f_j.

The standard method of dealiasing is to pad the input arrays explicitly with zeros before computing the DFT, removing the effect of periodicity. We refer to this practice as *explicit dealiasing*. For practical sizes, explicit dealiasing is much faster than direct computation, as shown[2] in figure 1.1.

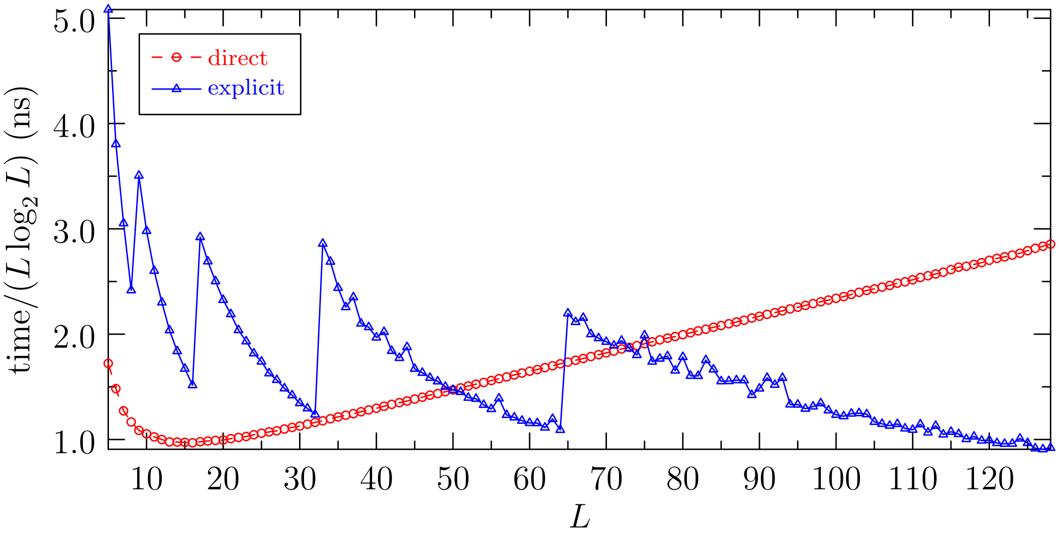

Figure 1.1: Normalized times for in-place 1D complex convolutions of two inputs of length L on 1 thread. Here, explicit dealiasing uses the common technique of zero-padding up to the next power of two.

While explicit dealiasing successfully avoids aliasing errors, it requires reading and multiplying values that are known *a priori* to be zero. For a dealiased convolution of two one-dimensional arrays of the same length, approximately $1/2$ of the inputs must be zero. While this is already inefficient, in higher dimensions the situation is

[2] The details of how we collected the timing data for the plots in this work can be found in chapter 7.

much worse: a d dimensional convolution of two arrays of the same size requires approximately $(2^d - 1)/2^d$ of the inputs to be zero. Not only does this result in wasted computation, but it also requires substantially more memory to store the input arrays.

Implicit dealiasing [BR11; RB18] provides an alternative to explicit dealiasing. Here, padded/unpadded FFTs are formulated to take account of the known zero values implicitly, avoiding the need for explicit zero padding. In many important cases, implicit dealiasing is more efficient than explicit dealiasing, with the most significant gains found in multidimensional convolutions.

1.B Limitations of implicit dealiasing

Even though implicit dealiasing can outperform explicit dealiasing, it has several limitations. To understand these limitations, it is useful to consider two illustrative examples:

- A key application of dealiased convolutions are pseudospectral simulations of partial differential equations [PO71; GO77]. To illustrate how pseudospectral simulations work, consider the three-dimensional incompressible Navier–Stokes equation

$$\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}t} + \boldsymbol{u} \cdot \boldsymbol{\nabla} \boldsymbol{u} + \boldsymbol{\nabla} P = \nu \Delta \boldsymbol{u} + \boldsymbol{F},$$

$$\nabla \cdot \boldsymbol{u} = 0.$$

  Here, $\boldsymbol{u}$ is the spacial velocity, P is the pressure, ν is the kinematic viscosity, and $\boldsymbol{F}$ is an external force. We assume periodic boundary conditions.

In a pseudospectral simulation, the equation is evolved in Fourier space (approximating the Fourier transform using the DFT). The advantage of this is that a spatial derivative can be computed via multiplication by a wave number. However, the nonlinear advection term, $\boldsymbol{u} \cdot \nabla \boldsymbol{u}$, becomes a linear convolution in Fourier space, which requires dealiasing.[3]

- In digital signal processing, convolutions are used to apply filters to data in order to analyze it. For example, consider the Sobel operator which is used for edge detection in image processing [Dud73; Sob14]. We have two operators:

$$S_x \doteq \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}, \quad S_y \doteq \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}.$$

By convolving each of these with an image A, we obtain approximations of the gradient in the x and y directions, respectively:

$$G_x \doteq S_x * A, \quad G_y \doteq S_y * A,$$

with magnitude

$$G \doteq \sqrt{G_x \odot G_x + G_y \odot G_y}.$$

The magnitude of the gradient is greatest where there are large changes in image intensity, i.e. at an edge. Thus, the image G resembles the image A with its edges emphasized. This is illustrated in figure 1.2.

[3] In the history of applying pseudospectral methods to turbulence, whether or not dealiasing matters has been controversial, although now it is generally agreed that it does. Such a discussion is outside the scope of this work, so we refer the interested reader to §3.10 of Canuto et al. [Can+06] for a brief overview of the topic.

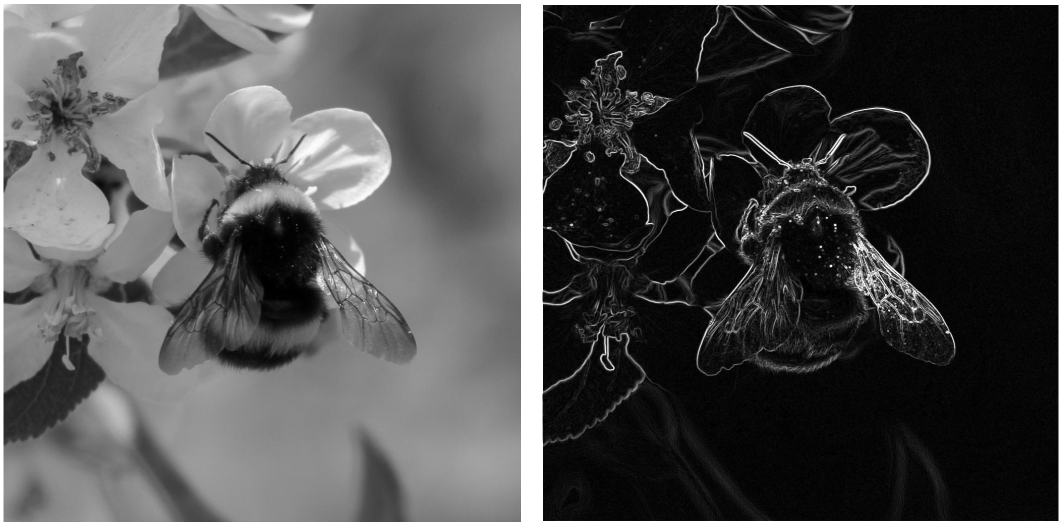

Figure 1.2: A grey-scale image (originally 1024×1024 pixels) transformed with the Sobel operator. Any pixel intensity greater than 255 is set to 255.

Related to image processing are convolutional neural networks (CNNs) [Lec+98]. Such networks operate much the same way, convolving a small kernel with an input to extract information, except now the elements of the kernel are learned parameters. Often such convolutions are computed directly (which is efficient for small problems), but the use of FFTs to accelerate CNNs is of increasing interest [MHL14; HR16; LY19; Chi+20].

Implicit dealiasing was developed specifically to improve the efficiency of pseudospectral simulations, and so the specific needs of that application dictated much of its design. Other applications, such as signal processing, may benefit from the techniques of implicit dealiasing, but cannot in its current state. We identify four key limitations of implicit dealiasing:

1.B.1 Arbitrary input sizes

The efficiency of implicit dealiasing is highly dependent on the length of the inputs. The most efficient FFTs, that we are aware of, are for products of powers of small prime radices such as 2, 3, 5, and 7 [FJ05]. For example, computing the convolution of two one-dimensional complex arrays of length L using implicit dealiasing requires computing FFTs of length L. If L is an inefficient length, the resulting convolution will be slow. This issue was not addressed by implicit dealiasing, because in pseudospectral simulations one can choose the size of the inputs. However this is not possible in all applications; for example, images can come in any size.

1.B.2 Arbitrary padding requirements

For complex (uncentered) inputs, implicit dealiasing is formulated to use $1/2$ padding[4] to dealias binary convolutions, as defined in definition 1.1. For centered or Hermitian symmetric inputs, implicit dealiasing is formulated to use $2/3$ padding. The $2/3$ padding ratio is used in pseudospectral simulations; only the center indices are retained, so these are the only indices that need to be dealiased [Ors71]. Thus, implicit dealiasing does not allow for arbitrary padding,[5] greatly limiting its generality.

This issue with arbitrary padding also affects pseudospectral simulations. For example, computing the cascade direction of nth-order Casimir invariants of two-dimensional turbulence, such as $\sum_j \omega^n(x_j)$, where ω is the scalar vorticity, requires a padding ratio of $2/(n+1)$ (where $n \geq 2$) [Bow13].

[4] This fraction is the ratio of input data to padded data.

[5] It should be noted that formulas for padded/unpadded FFTs with arbitrary padding ratios are briefly considered by Bowman and Roberts [BR11] in the complex uncentered case; however, algorithms for efficiently computing these FFTs were not developed. That being said, these equations are fundamental to the algorithms presented in this work.

1.B.3 Real valued inputs

Hermitian symmetric arrays are arrays $\{f_j\}_{j=0}^{L-1}$ with the symmetry $f_{L-j} = \overline{f_j}$ for all $j \in \{0, \ldots, L-1\}$. An important property of the DFT is that an array is Hermitian symmetric if and only if it is the DFT of a real array.[6] This is exploited by complex-to-real and real-to-complex FFTs, which efficiently compute the DFTs of Hermitian symmetric and real-valued arrays respectively.

Explicit dealiasing can use these efficient complex-to-real/real-to-complex FFTs to compute both Hermitian symmetric and real convolutions. Implicit dealiasing can efficiently compute Hermitian symmetric convolutions, also by using complex-to-real/real-to-complex FFTs and by taking the symmetries of the data into account. However, implicit dealiasing does not have any such methods for efficiently computing real-valued convolutions. Any application which uses real-valued data (such as signal/image processing) cannot take advantage of techniques of implicit dealiasing.

1.B.4 Unequal input sizes

Finally, implicit dealiasing assumes that the inputs are the same size. While this is true for pseudospectral simulations, in many applications this is not the case. For example, in signal processing one typically convolves input data with a relatively small kernel.

Note that if one of the arrays is very small compared to the other, aliases are not the main problem as the possible wrap-around is limited. Thus, one might think that the techniques of implicit dealiasing are of limited use. However, the circular convolution requires the inputs to be the same size, and so explicit methods require

[6] For this reason, Hermitian symmetric arrays are common in applications. For example, the arrays used in pseudospectral simulations of the Navier–Stokes equation are Hermitian symmetric because they are Fourier transforms of the real velocity field.

that the smaller array is zero-padded to at least the same size as the larger array. This can easily lead to an absurd amount of zero-padding on the smaller array.[7]

1.C Generalizing implicit dealiasing

Both explicit and implicit dealiasing transform the inputs in such a way that the resulting circular convolution is equivalent to the desired linear convolution. While explicit dealiasing does this by modifying the inputs, implicit dealiasing modifies the transforms.

In this work, we generalize implicit dealiasing with a technique that we call *hybrid dealiasing*. Hybrid dealiasing allows for a combination of explicit and implicit zero padding, modifying both the inputs and the transforms. In hybrid dealiasing, explicit dealiasing and implicit dealiasing are now special cases.

In chapters 2 and 3, we develop hybrid dealiasing for complex, centered, and Hermitian symmetric inputs, which are the cases considered by implicit dealiasing. For problems particularly well suited to either explicit dealiasing or implicit dealiasing, hybrid dealiasing matches or exceeds the performance of each. For problems not well suited to existing dealiasing methods (where input arrays have inefficient sizes), hybrid dealiasing exceeds the performance of explicit and implicit dealiasing. Hybrid dealiasing is also designed to allow arbitrary padding requirements. This solves the problems described in sections 1.B.1 and 1.B.2.

In chapter 4, we go beyond implicit dealiasing by developing hybrid dealiasing for real-valued arrays. We do this by exploiting conjugate symmetries in the transformed data, providing a solution to the problem in section 1.B.3.

[7] Things are a bit more complicated than this, as there exist techniques such as *overlap-add* and *overlap-save* to greatly improve the efficiency of such convolutions.

In chapter 5, we discuss the implementation of these ideas for higher dimensional arrays. Following implicit dealiasing, we compute multidimensional convolutions by decomposing them into lower dimensional convolutions, resulting in further improvements over conventional methods. Chapter 6 describes several numerical optimizations used in our implementation of hybrid padding in the open-source library `FFTW++` [BRM23]. Numerical results comparing hybrid dealiasing with the other dealiasing techniques are shown in chapter 7.

We conclude in chapter 8 and discuss future work for hybrid dealiasing; in particular, we discuss the problem of inputs with unequal size described in section 1.B.4, as well as a possible alternative method for computing the convolution of real inputs.

1.C.1 General dealiased convolutions

To conclude this introduction, we point out the generality of the algorithms presented in this work. Let X denote the space of finite arrays equipped with addition[8] and linear convolution. It is easy to show that this map forms a commutative ring.[9]

The motivation of this work is to compute linear convolutions as defined by definition 1.1; indeed this is likely the most valuable use of hybrid dealiasing. However, our algorithms are capable of computing much more than linear convolutions. In each algorithm, we compute a padded FFT of the inputs, and then transform the results using an arbitrary multiplication routine. The only assumption on this routine is that it performs operations element-wise. We then compute the inverse unpadded FFT of the resulting arrays.

[8] To add two arrays of different lengths, we treat them as infinite sequences with finite support.

[9] This ring is often identified with the ring of polynomials with complex coefficients: if $\{f_j\}_{j=0}^{L-1}$ is an array, we can identify it with a polynomial of degree $L-1$ via the ring isomorphism:

$$\{f_j\}_{j=0}^{L-1} \mapsto f_0 + f_1 x + f_2 x^2 + \ldots + f_{L-1} x^{L-1}.$$

Thus, all of the algorithms presented in this work can be used to compute general convolutions defined as follows:

Definition 1.4. Let $A, B \in \mathbb{N}$. A function $\mathcal{C} : X^A \mapsto X^B$ is a *general convolution* if there exists $M \in \mathbb{N}$ and an element-wise function $\mathcal{M} : X^A \mapsto X^B$ which satisfies

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

where $\boldsymbol{f}_1, \ldots, \boldsymbol{f}_A$ have been zero-padded to length M.

It should be noted that the only reason we include this definition is for completeness. The only practical examples of general convolutions that the author is presently aware of take the form of multivariate polynomials in X. For example if $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w} \in X$, then $\boldsymbol{u} * \boldsymbol{u} - \boldsymbol{v} * \boldsymbol{v}$ and $\boldsymbol{v} * \boldsymbol{v} - \boldsymbol{w} * \boldsymbol{w}$ are general convolutions.[10]

[10] These convolutions were used by C. Basdevant for the pseudospectral simulation of 3D incompressible turbulence, and reduce the necessary FFTs (each time step) from 9 to 8 [Bas83].

Chapter 2

Complex convolutions in one dimension

In this chapter, we review the algorithm for computing convolutions of complex arrays using implicit dealiasing [BR11; RB18]. We then introduce hybrid dealiasing for complex arrays, which combines explicit and implicit dealiasing.

2.A Implicitly dealiased convolutions

Suppose that, for some convolution, we have input data $\{a_j\}_{j=0}^{L-1}$ that needs to be padded with zeros to length M. We construct a buffer $\boldsymbol{f} \doteq \{f_j\}_{j=0}^{M-1}$ where $f_j = a_j$ for $j < L$ and $f_j = 0$ for $j \geq L$. The DFT of $\boldsymbol{f}$ can be written as

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

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

For now, assume that L and M share a common factor m, so that $L = pm$ and

$M = qm$, where $m, p, q \in \mathbb{N}$, with $q \geq p$. We can now reindex j and k as

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

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

This allows us to decompose the DFT via the Cooley–Tukey algorithm [CT65]. Following Bowman and Roberts [BR11]:

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

Computing the DFT of $\boldsymbol{f}$ then amounts to preprocessing $\boldsymbol{f}$ for each value of r, and computing q DFTs of size m; no explicit zero padding is needed. The inverse transform is similar [BR11]:

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

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

2.B Hybrid dealiasing

An issue with the above formulation is the assumption that L and M must share a common factor m. Furthermore, even if L and M do share a common factor, the resulting convolution might be inefficient (as mentioned in section 1.B.1).

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

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

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

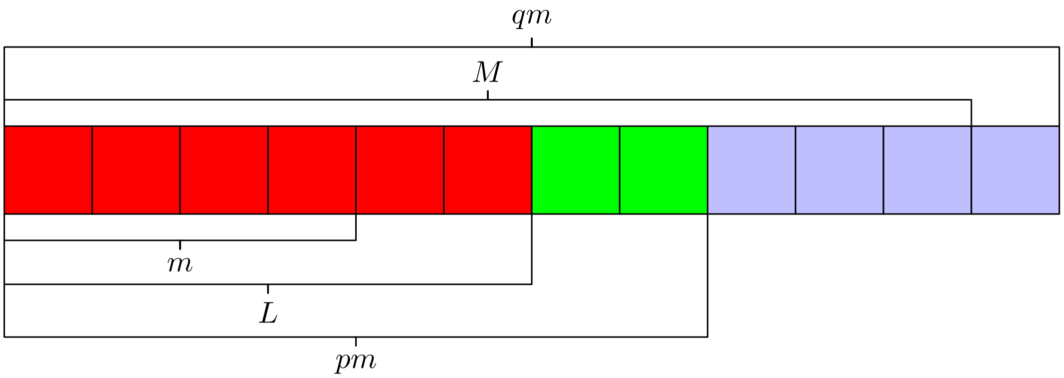

Figure 2.1: An illustration of hybrid padding for a one-dimensional array with $L = 6$ and $M = 11$. Choosing $m = 4$, we have $p = 2$ and $q = 3$. We explicitly pad our data of length L to length $pm = 8$, and then implicitly pad our data to length $qm = 12$.

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

2.C Convolutions one residue at a time

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

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

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

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

This formulation can be advantageous for large problems as it allows reuse of the memory needed to store $\boldsymbol{F}_r$, illustrated for a binary convolution in figure 2.2.

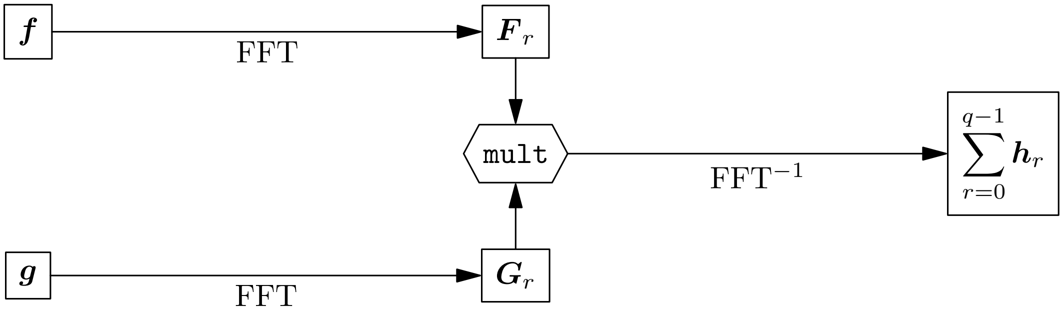

Figure 2.2: Accumulation of residue contributions to a convolution.

Of course, any number of residues can be computed at a time. When using the conjugate symmetry optimization (described in section 6.B), it is natural to compute two residues at a time, namely r and $-r \bmod q$. The $r = 0$ and $r = q/2$ (if q is even) residues must be treated with care. If two residues are being computed at a time, it is convenient to store the $r = 0$ and $r = q/2$ residues together.

Pseudocode for an in-place convolution[1] is given in algorithm 1. Pseudocode for the forward and backward transforms is shown in algorithms 3 and 4 for $p = 1$ and algorithms 5 and 6 for $p = 2$. For $p > 2$, it is more efficient to replace the summations in eqs. (2.1) and (2.2) by an inner DFT, which is discussed in section 6.A.1. For simplicity, the pseudocode illustrates only the case where one residue is computed at a time.

2.D Summary of Chapter 2

In this chapter, we introduced a new dealiasing algorithm called hybrid dealiasing and have shown how to apply it to complex one-dimensional arrays. In hybrid dealiasing, we consider inputs of length L that need to be zero padded to at least length M. For a given value of m, we explicitly zero pad from length L to length pm and then use implicit dealiasing to zero pad to length qm. This requires q FFTs of length m. The values of p and q are chosen to be the smallest integers such that $L \leq pm$ and $M \leq qm$.

By allowing for any value of L and M, we solve the first limitation of implicit dealiasing (described in section 1.B). Furthermore, because we can choose any value of m, we can ensure that we use efficient FFTs, solving the second limitation of implicit dealiasing.

[1] What we are computing is the output of the convolution, truncated to L values. To obtain the full output of a convolution, one cannot compute it in-place as the output array is larger than any of the inputs (see definition 1.1).

Algorithm 1 convolve is a one-dimensional standard convolution. There are A inputs (each of length L, to be padded to at least length M) and B outputs. The convolution uses the multiplication operator mult.

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

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

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

Algorithm 3 forward1 is the complex forward transform for residue r when $p = 1$.

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

Algorithm 4 backward1 is the complex backward transform for residue r when $p = 1$.

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

Algorithm 5 `forward2` is the complex forward transform for residue r when $p = 2$.

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

Algorithm 6 `backward2` is the complex backward transform for residue r when $p = 2$.

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

Chapter 3

Centered and Hermitian symmetric convolutions in one dimension

In this chapter, we build on chapter 2 and develop the ideas of hybrid dealiasing for centered arrays. These centered transforms are then used to construct hybrid dealiased convolutions for Hermitian symmetric data.

3.A Centered convolutions

In certain applications, it is convenient to center the data within the input array. While it is possible to multiply the output of the uncentered transform derived in chapter 2 by a primitive root of unity to obtain a centered transform, this is inefficient in practice. To handle the centered case, we build the shift directly into the transforms. Let $p, m \in \mathbb{N}$, with p even, and let $\boldsymbol{f} = \{f_j\}_{j=-pm/2}^{pm/2-1}$ be a centered array (obtained by symmetrically padding an array of length L to length pm, if needed). Implicit padding to length qm can be accomplished with the transform

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

Separating this sum and shifting the indices, we obtain

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

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

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

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

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

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

where

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

Because the transformed data is not centered, the inverse transform is identical to (2.2); one must only be careful to store the output values of the inverse transform in the correct locations.

Pseudocode for a centered convolution is given in algorithm 2. Pseudocode for the centered case when $p = 2$ is given in algorithms 7 and 8. For $p > 2$, see section 6.A.2.

Algorithm 7 `forward2C` is the centered complex forward transform for residue r when $p = 2$.

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

Algorithm 8 `backward2C` is the centered complex backward transform for residue r when $p = 2$.

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

3.B Hermitian convolutions

An array $\boldsymbol{g} = \{f_j\}_{j=0}^{L-1}$ is Hermitian symmetric if $g_j = \overline{g_{L-j}}$ (where the bar denotes complex conjugation) for all $j \in \{0, \dots, L-1\}$. Note that this symmetry implies $g_0 \in \mathbb{R}$, which is the so-called DC mode[1]. Likewise, if L is even, Hermitian symmetry implies that $g_{L/2} \in \mathbb{R}$, which is the so-called Nyquist mode.

With Hermitian symmetric data, one only has to store approximately half of the input values, as the rest of the data can be computed (by taking the conjugate) when needed. An array is Hermitian symmetric if and only if its DFT is real valued. Because of this, Hermitian symmetric data occurs naturally in many applications, including pseudospectral methods for partial differential equations.

Consider a centered array $\boldsymbol{f} = \{f_j\}_{j=-pm/2+1}^{pm/2-1}$ with Hermitian symmetry:

$$f_j = \overline{f_{-j}}, \quad j \in \{-\frac{pm}{2} + 1, \dots, \frac{pm}{2} - 1\}.$$

One can use the centered transforms from section 3.A to develop Hermitian trans-

[1] In signal processing, DC is an initialism for direct current.

forms. The forward transform is given by (3.2), where

$$w_{r,s} \doteq \zeta_{qm}^{rs} \left(\sum_{t=0}^{p/2-1} \zeta_q^{rt} \left[f_{tm+s} + \zeta_{2q}^{-rp} \overline{f_{pm/2-tm-s}} \right] \right). \tag{3.4}$$

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

The inverse transform is once again given by (2.1). Here, the key difference is that because the input is real, the output is Hermitian symmetric, so that real-to-complex DFTs can be used. Pseudocode for a Hermitian convolution is given in algorithm 2. Pseudocode for the Hermitian transforms when $p = 2$ is given in algorithms 9 and 10. For $p > 2$, see section 6.A.2.

Algorithm 9 `forward2H` is the Hermitian forward transform for residue r when $p = 2$.

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

Algorithm 10 `backward2H` is the Hermitian backward transform for residue r when $p = 2$.

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

3.C Summary of Chapter 3

In this chapter, we generalized the ideas of hybrid dealiasing to one-dimensional centered arrays. These algorithms build the centering into the transforms and do not require multiplication by a complex factor to achieve the shift. Just as in the uncentered case, our transforms require q FFTs of length m.

We then used these centered algorithms to develop hybrid dealiasing for Hermitian symmetric inputs. In this case, the q FFTs can each be computed using complex-to-real/real-to-complex FFTs.

Chapter 4

Real convolutions in one dimension

In this chapter, we go beyond the original scope of implicit dealiasing by developing hybrid dealiasing for real-valued inputs. One might suppose that the real case will be analogous to the Hermitian symmetric case, but there is a key difference between the two. In all cases up to this point, each problem reduces to computing FFTs of size m. The efficiency of the Hermitian symmetric algorithm is reliant on the fact that we can use complex-to-real and real-to-complex FFTs.

Unfortunately, the real case is not as simple: even though the inputs to the convolution are real, the inputs to the FFTs are generally complex due to the preprocessing. Thus we cannot rely on real-to-complex/ complex-to-real FFTs to compute the residue contributions efficiently.[1]

[1] There is a simple argument as to why we can use complex-to-real/real-to-complex FFTs in the Hermitian symmetric case, but not the real case:
If $\boldsymbol{f} = \{f_j\}_{j=0}^{qm-1}$ is a Hermitian symmetric array, then $\boldsymbol{F} = \{F_k\}_{k=0}^{qm-1}$ is a real array. For any $r \in \{0, \dots, q-1\}$, the residue contribution $\{F_{q\ell+r}\}_{\ell=0}^{m-1}$ is also real, which means it must be the DFT of a Hermitian symmetric array.
But if $\boldsymbol{f} = \{f_j\}_{j=0}^{qm-1}$ is a real array, then $\boldsymbol{F} = \{F_k\}_{k=0}^{qm-1}$ is a Hermitian symmetric array. Then one can see that for any $r \in \{1, \dots, q-1\}$, the residue contribution $\{F_{q\ell+r}\}_{\ell=0}^{m-1}$ is not, in general, Hermitian symmetric (when $\ell = 0$, the DC mode F_r need not be real). Thus, it cannot be the DFT of a real array.

4.A Real convolutions via conjugate symmetries

Instead of depending on real-to-complex/ complex-to-real FFTs, we compute convolutions of real inputs by directly exploiting conjugate symmetries in the transformed data. Recall that the forward transform (2.1) is given by

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

with inverse (2.2) given by

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

For each residue $r \in \{0, \ldots, q-1\}$, define the two-dimensional array $\boldsymbol{h}_r$ of size $m \times p$ via

$$h_{r,s,t} \doteq \zeta_q^{-tr} \zeta_{qm}^{-sr} \sum_{\ell=0}^{m-1} \zeta_m^{-s\ell} F_{q\ell+r}, \tag{4.1}$$

so that we can write

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

Now note that by Hermitian symmetry, the complex conjugate of $F_{q\ell+r}$ satisfies

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

and so we have

$$\begin{aligned} \overline{h_{r,s,t}} &= \zeta_q^{tr} \zeta_{qm}^{sr} \sum_{\ell=0}^{m-1} \zeta_m^{s\ell} \overline{F_{q\ell+r}} = \zeta_q^{tr} \zeta_{qm}^{sr} \sum_{\ell=0}^{m-1} \zeta_m^{s\ell} F_{q(m-\ell)-r} \\ &= \zeta_q^{-t(q-r)} \zeta_{qm}^{-s(q-r)} \sum_{\ell=0}^{m-1} \zeta_m^{-s\ell} F_{q\ell+q-r} = h_{q-r,s,t}. \end{aligned} \tag{4.2}$$

For convenience, let $\boldsymbol{h}_{q/2} \equiv \boldsymbol{0}$ if q is odd. Using (4.2), we can write

$$\begin{aligned} qmf_{tm+s} = \sum_{r=0}^{q-1} h_{r,s,t} &= h_{0,s,t} + \sum_{r=1}^{\lceil q/2 \rceil - 1} (h_{r,s,t} + h_{q-r,s,t}) + h_{q/2,s,t} \\ &= h_{0,s,t} + 2 \sum_{r=1}^{\lceil q/2 \rceil - 1} \operatorname{Re} h_{r,s,t} + h_{q/2,s,t}. \end{aligned}$$

Thus for $r \in \{1, \ldots, \lceil q/2 \rceil - 1\}$, we can use complex FFTs to compute $\boldsymbol{h}_r$ as we essentially get $\boldsymbol{h}_{q-r}$ for free. Therefore, it only remains to consider the cases when $r = 0$ and $r = q/2$.

Residue $r = 0$:

If $r = 0$, then forward transform is given by

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

This is just the DFT of a real array and so a real-to-complex FFT can be used. Similarly, (4.1) becomes

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

which can be computed using a complex-to-real FFT.

Residue $r = \frac{q}{2}$:

This case is only relevant when q is even. The forward transform is given by

$$F_{q\ell+q/2} = \sum_{s=0}^{m-1} \zeta_m^{\ell s} \zeta_{2m}^{s} \sum_{t=0}^{p-1} (-1)^t f_{tm+s}, \tag{4.3}$$

and (4.1) is given by

$$h_{q/2,s,t} \doteq (-1)^t \zeta_{2m}^{-s} \sum_{\ell=0}^{m-1} \zeta_m^{-s\ell} F_{q\ell+q/2}. \tag{4.4}$$

Note that we cannot use real-to-complex/complex-to-real FFTs as the preprocessed inputs are complex. The key insight is that even though this residue contribution is not strictly Hermitian symmetric, a Hermitian symmetry can still be exploited.

By the Hermitian symmetry of $\boldsymbol{F}$ we have

$$\overline{F_{q\ell+q/2}} = F_{qm-(q\ell+q/2)} = F_{q(m-1-\ell)+q/2}. \tag{4.5}$$

Thus, the $q/2$ residue contribution is closed under conjugation, and we only need to compute approximately half of the values of $\left\{F_{q\ell+q/2}\right\}_{\ell=0}^{m-1}$ in order to compute $\boldsymbol{h}_{q/2}$.

We require that m be even. If we let $e \doteq m/2$, we can reindex s and ℓ as follows:

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

$$\ell = 2c + d,\;\; c \in \{0,\dots,e-1\},\; d \in \{0,1\}.$$

Now following the Cooley–Tukey algorithm, (4.3) becomes

$$\begin{aligned} F_{q(2c+d)+q/2} &= \sum_{b=0}^{e-1} \sum_{a=0}^{1} \zeta_{2e}^{(2c+d)(ae+b)} \zeta_{2m}^{ae+b} \sum_{t=0}^{p-1} (-1)^t f_{tm+ae+b}, \\ &= \sum_{b=0}^{e-1} \zeta_e^{cb} \zeta_{2m}^{(2d+1)b} \sum_{t=0}^{p-1} (-1)^t \left(f_{tm+b} + i(-1)^d f_{tm+e+b}\right). \end{aligned}$$

Thus, the full forward transform can be computed using two DFTs of size e, corresponding to $d = 0$ and $d = 1$; however, we only need to compute this array for a single value of d. Choosing[2] $d = 0$, we have

$$F_{q(2c)+q/2} = \sum_{b=0}^{e-1} \zeta_e^{cb} \zeta_{2m}^{b} \sum_{t=0}^{p-1} (-1)^t \left(f_{tm+b} + i f_{tm+e+b} \right), \tag{4.6}$$

which can be computed using a complex FFT of size e.

Writing (4.4) in terms of our new indices gives us:

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

Next, define the arrays $\boldsymbol{w}_d = \{w_{d,b}\}_{b=0}^{e-1}$ via

$$w_{d,b} \doteq \zeta_{2m}^{-b} \sum_{c=0}^{e-1} \zeta_e^{-bc} F_{q(2c+d)+q/2}. \tag{4.7}$$

Note that $\boldsymbol{w}_d$ can be computed using an inverse DFT of size e. Equation (4.7) allows us to write $\boldsymbol{h}_{q/2}$ as

$$h_{q/2,ae+b,t} = (-1)^t i^{-a} \left[w_{0,b} + (-1)^a \zeta_m^{-b} w_{1,b} \right]. \tag{4.8}$$

Using eqs. (4.5) and (4.7)

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

[2] This choice is arbitrary. We could obtain the same results using $d = 1$.

So we can compute $w_{1,b}$ from $w_{0,b}$, and (4.8) becomes

$$h_{q/2,ae+b,t} = (-1)^t i^{-a} \left[w_{0,b} + (-1)^a \overline{w_{0,b}} \right],$$

so that

$$h_{q/2,b,t} = (-1)^t 2 \operatorname{Re} w_{0,b}, \quad h_{q/2,e+b,t} = (-1)^t 2 \operatorname{Im} w_{0,b}. \tag{4.9}$$

To summarize our computation of $\boldsymbol{h}_{q/2}$, we first compute half of the residue contributions with a complex FFT of size e using (4.6). If we were computing a convolution, we would then apply our multiplication routine. Then we compute $\boldsymbol{w}_0$ using (3.3) and an inverse complex FFT of size e, and obtain $\boldsymbol{h}_{q/2}$ using (4.9).

Pseudocode for a real convolution[3] is given by algorithm 11. Pseudocode for the forward and backward transforms are given by algorithms 12 and 13 (for $p = 1$), and algorithms 14 and 15 (for $p = 2$).

4.B Summary of chapter 4

In this chapter, we extended hybrid dealiasing to handle real-valued data; this provides a solution to the third limitation of implicit dealiasing described in section 1.B. Unlike the Hermitian case in section 3.B, we cannot rely on real-to-complex/complex-to-real FFTs for each residue $r \in \{0, \ldots, q-1\}$. Instead, the algorithm does the following:

- When $r = 0$, we can use real-to-complex/complex-to-real FFTs of length m.
- When $r = q/2$ (which only happens when q is even), we require that m be even, and can use complex FFTs of length $m/2$.

[3] Note that each residue contribution is not the same size in this algorithm, which the multiplication routine must account for.

- When $r \in \{1, \ldots, \lceil q/2 \rceil - 1\}$, we use complex FFTs of length m, simultaneously yielding the contributions for residues r and $q - r$.

Algorithm 11 `convolveR` is a one-dimensional real convolution. There are A inputs (each of length L, to be padded to at least length M) and B outputs. The convolution uses the multiplication operator `mult`.

```
Input: $\{\boldsymbol{f}_a\}_{a=0}^{A-1}$, $L$, $M$, $m$, $A$, $B$
  $p \leftarrow \lceil L/m \rceil$
  if $p = 1$ then
      $n \leftarrow \lceil M/m \rceil$
      $q \leftarrow n$
      Forward ←forward1R
      Backward ←backward1R
  else if $p = 2$ then
      $n \leftarrow \lceil M/m \rceil$
      $q \leftarrow n$
      Forward ←forward2R
      Backward ←backward2R
  else
      $n \leftarrow \lceil M/(pm) \rceil$
      if $n$ is even then
          Assert: $p$ is even
      $q \leftarrow pn$
      Forward ←forwardInnerR
      Backward ←backwardInnerR
  if $q$ is even then
      Assert: $m$ is even
  for $b = 0, \ldots, B-1$ do
      $\boldsymbol{h}_b \leftarrow \{0\}_{j=0}^{L-1}$
  for $r = 0, \ldots, \lceil (n+1)/2 \rceil - 1$ do
      for $a = 0, \ldots, A-1$ do
          $\boldsymbol{F}_a \leftarrow$ Forward$(\boldsymbol{f}_a, L, m, q, r)$
      $\{\boldsymbol{F}_b\}_{b=0}^{B-1} \leftarrow$ mult$(\{\boldsymbol{F}_a\}_{a=0}^{A-1})$
      for $b = 0, \ldots, B-1$ do
          $\boldsymbol{h}_b \leftarrow \boldsymbol{h}_b +$ Backward$(\boldsymbol{F}_b, L, m, q, r)$
  for $b = 0, \ldots, B-1$ do
      $\boldsymbol{f}_b \leftarrow \boldsymbol{h}_b/(qm)$
  return $\{\boldsymbol{f}_b\}_{b=0}^{B-1}$
```

Algorithm 12 `forward1R` is the real forward transform for residue r when $p = 1$.

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

Algorithm 13 `backward1R` is the real backward transform for residue r when $p = 1$.

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

Algorithm 14 forward2R is the real forward transform for residue r when $p = 2$.

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

Algorithm 15 backward2R is the real backward transform for residue r when $p = 2$.

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

Chapter 5

Multidimensional convolutions

An n-dimensional convolution is conventionally computed by performing an FFT of size $N_1 \times \ldots \times N_n$, applying the specified multiplication operator on the transformed data, and then performing an inverse FFT back to the original space. However, as described in [BR11] and [RB18], a better alternative is to decompose the n-dimensional convolution recursively into $\prod_{i=2}^{n} N_i$ FFTs in the first dimension, followed by N_1 convolutions of dimension $n-1$, and finally $\prod_{i=2}^{n} N_i$ inverse FFTs in the first dimension. This is illustrated in figure 5.1. At the innermost level, a recursive multidimensional convolution thus reduces to a one-dimensional convolution.

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

For example, the memory management for a single-threaded 2D padded complex convolution for $A = 2$ and $B = 1$ is shown in figure 5.2. For each $r_x \in \{0, 1, \ldots, q_x - 1\}$, the residue contribution to the padded x FFT of the input buffers is stored in the square boxes. A padded FFT of each input is then

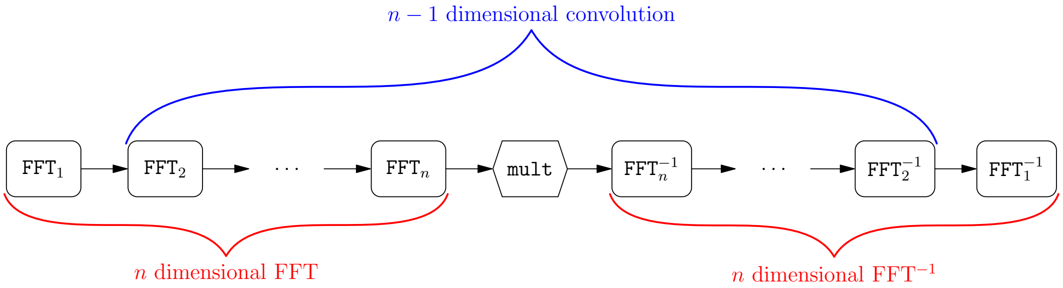

Figure 5.1: Recursive computation of an n-dimensional convolution.

performed in the y direction, column-by-column, using a one-dimensional work buffer, to produce a single column of the Fourier-transformed image, depicted in yellow. The Fourier transformed columns of two inputs F and G are then multiplied pointwise and stored back into the F column. At this point, the inverse y transform can then be performed, with the truncated result stored in the lower half of the column, next to the previously processed data shown in red. This process is repeated on the remaining columns, shifting and reusing the work buffers. Once all the columns have been processed, an inverse transform in the x direction produces the final r_x contribution to the convolution.

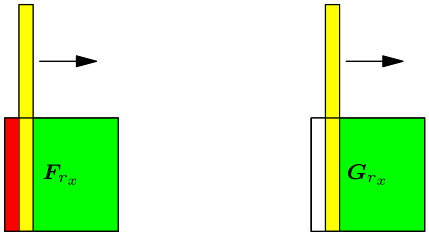

Figure 5.2: The reuse of memory to compute the contribution of a single x residue to a 2D binary convolution with two inputs and one output: a 1D padded y FFT is applied to columns of $\boldsymbol{F}_{r_x}$ and $\boldsymbol{G}_{r_x}$ to produce the two stacked yellow columns that are fed to the multiplication operator, producing one stacked column to be inverse y transformed into a single column (like the red one shown on the left). The upper column is then reused for processing subsequent columns.

The reuse of subconvolution work memory allows the convolution to be computed using less total memory: for a d-dimensional p/q padded convolution, the work memory requirement is $(A+B)pmL^{d-1}$ complex words, where $p=\lceil L/m\rceil$ (not counting the storage requirements for the input data). In contrast, explicit padding requires a typically much larger buffer of size $E \doteq C(q/p)^d L^d$, where $C = \max(A,B)$. For example, computing a d-dimensional dealiased convolution implicitly for $A=2$ and $B=1$ with padding ratio $p/q=1/2$ and $m=L\gg 1$ asymptotically requires a storage of $3E/2^{d+1}$. In particular, in one dimension, the general formulation of implicit padding requires $3/4$ of the work memory required by explicit padding. For a $2/3$ padding ratio, implicit padding requires $(2/3)^{d-1}\,E$. In addition to having reduced memory requirements, a dealiased multidimensional convolution decomposed in this way is significantly faster than a conventional implementation due to better data locality, optimal FFT sizes, and the elimination of transforms of data known *a priori* to be zero.

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

The FFTs in multithreaded convolutions can be parallelized by dividing the $\prod_{i=2}^{n} N_i$ one-dimensional FFTs between the T threads. Similarly, the N_1 subconvolutions can be parallelized over $T\leq N_1$ threads using T work buffers [RB18].

Chapter 6

Numerical implementation

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

6.A Inner loop optimization

While the algorithms presented so far allow FFTs of any size $m \in \mathbb{N}$, there is an issue of efficiency when m is much smaller than the input arrays. As m decreases, p and q increase, which increases the amount of preprocessing that our algorithms require. The inner loop optimization is a modification that allows one to compute the preprocessing and post-processing using FFTs of size p.

6.A.1 Inner loop for complex arrays

Consider the complex case described in chapter 2. One may note that the preprocessing done in (2.1) is itself a padded FFT from size p to size q. Similarly, the post-processing in (2.2) is an unpadded FFT from size q to size p. Thus, if p and q share a common factor, one can use these equations recursively.

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

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

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

$$F_{q\ell+un+v} = \sum_{s=0}^{m-1} \zeta_m^{\ell s} \zeta_{qm}^{(un+v)s} \sum_{t=0}^{p-1} \zeta_p^{ut} \left(\zeta_q^{vt} f_{tm+s} \right), \tag{6.1}$$

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

$$f_{tm+s} = \frac{1}{qm} \sum_{v=0}^{n-1} \zeta_{np}^{-tv} \sum_{u=0}^{p-1} \zeta_p^{-tu} \zeta_{qm}^{-s(un+v)} \sum_{\ell=0}^{m-1} \zeta_m^{-s\ell} F_{q\ell+un+v}, \tag{6.2}$$

requires the sum of n DFTs of size p. In this optimization we consider each v to be a residue; the full transform then has n residue contributions of size pm. Note that the sum over v is *not* a DFT as the input depends on both v and t. Pseudocode for the inner loop is given in algorithms 16 and 17.

6.A.2 Inner loop for centered and Hermitian arrays

Just as we did in the standard case, we can apply the same recursive techniques to the centered case described in section 3.A.

The summation in (3.3) is itself a padded DFT from size $p/2$ to size q. Therefore, if q shares a factor with $p/2$, we can use the transforms in section 6.A.1 to preform the preprocessing.

In our implementation, we consider two cases. If $p = 2$, we directly sum the

Algorithm 16 `forwardInner` is the complex forward transform for residue v when $p > 2$.

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

Algorithm 17 `backwardInner` is the complex backward transform for residue v when $p > 2$.

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

two terms in $w_{r,s}$. If $p > 2$ (with p assumed to be even), we define

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

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

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

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

$$F_{q\ell+un+v} = \sum_{s=0}^{m-1} \zeta_m^{\ell s} w_{un+v,s}. \tag{6.3}$$

The inverse transform is the same as (6.2). Pseudocode for the centered case is given in algorithms 18 and 19.

Algorithm 18 `forwardInnerC` is the centered complex forward transform for residue v when $p > 2$. Here δ_t is the Kronecker $\delta_{t,0}$.

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

Algorithm 19 `backwardInnerC` is the centered complex backward transform for residue v when $p > 2$. Here δ_t is the Kronecker $\delta_{t,0}$.

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

These equations also apply to the Hermitian case (section 3.B), using $f_j = \overline{f_{-j}}$ whenever $j < 0$; however, unlike the outer FFTs of length m, the preprocessing and postprocessing stages use complex FFTs. Pseudocode for the centered case is given in algorithms 18 and 19. Pseudocode for the Hermitian case is given in algorithms 20 and 21.

6.A.3 Inner loop for real arrays

The inner loop can also be developed for the real case described in chapter 4. The parameters for the real inner loop are defined in the same way as the complex

Algorithm 20 `forwardInnerH` is the Hermitian forward transform for residue v when $p > 2$. Here δ_t is the Kronecker $\delta_{t,0}$.

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

Algorithm 21 `backwardInnerH` is the Hermitian backward transform for residue v when $p > 2$. Here δ_t is the Kronecker $\delta_{t,0}$.

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

inner loop (section 6.A.1), and the forward and backward transforms are given by equations eqs. (6.1) and (6.2).

For each $v \in \{0, \ldots, n-1\}$, we define the two-dimensional array $\boldsymbol{h}_v$ of size $m \times p$ via

$$h_{v,s,t} \doteq \zeta_q^{-tv} \sum_{u=0}^{p-1} \zeta_p^{-tu} \zeta_{qm}^{-s(un+v)} \sum_{\ell=0}^{m-1} \zeta_m^{-s\ell} F_{q\ell+un+v}, \tag{6.4}$$

so that we can write

$$qmf_{tm+s} = \sum_{v=0}^{n-1} h_{v,s,t}.$$

Just as in chapter 4, we have the Hermitian symmetry

$$h_{v,s,t} = \overline{h_{n-v,s,t}},$$

which gives us

$$qmf_{tm+s} = h_{0,s,t} + 2 \sum_{v=1}^{\lceil n/2 \rceil - 1} \operatorname{Re}\{h_{v,s,t}\} + h_{n/2,s,t},$$

where we take $\boldsymbol{h}_{n/2} \equiv \mathbf{0}$ if n is odd.

Thus, if $v \in \{1, \ldots \lceil n/2 \rceil - 1\}$, we get the contributions of $\boldsymbol{h}_v$ and $\boldsymbol{h}_{n-v}$ simultaneously. Therefore, we can essentially use the algorithm for the complex inner loop of section 6.A.1 for these residues. There are two other residues to consider.

Residue $v = 0$:

In this case, the forward transform (6.1) reduces to

$$F_{q\ell+un} = \sum_{s=0}^{m-1} \zeta_m^{\ell s} \zeta_{qm}^{uns} \sum_{t=0}^{p-1} \zeta_p^{ut} f_{tm+s}. \tag{6.5}$$

and (6.4) becomes

$$h_{0,s,t} \doteq \sum_{u=0}^{p-1} \zeta_p^{-tu} \zeta_{qm}^{-sun} \sum_{\ell=0}^{m-1} \zeta_m^{-s\ell} F_{q\ell+un}. \tag{6.6}$$

Note that DFTs of size p in (6.6) can be computed using real-to-complex/complex-to-real FFTs. So if we compute the contribution for $u \in \{1, \ldots, \lceil p/2 \rceil - 1\}$, we can obtain the contribution from $p - u$ via its complex conjugate.[1] Thus, we don't sacrifice efficiency by using complex FFTs of size m for these values of u.

It remains to handle the cases $u = 0$ and $2u = p$. For convenience, let $\tilde{\boldsymbol{f}}$ be the output of the DFT of size p in (6.5), with elements

$$\tilde{f}_{um+s} = \sum_{t=0}^{p-1} \zeta_p^{ut} f_{tm+s}.$$

Note that $\tilde{\boldsymbol{f}}$ is Hermitian symmetric in u, for each $s \in \{0, \ldots, m-1\}$.

First, consider the case when $u = 0$. By Hermitian symmetry, $\tilde{f}_s \in \mathbb{R}$ for each $s \in \{0, \ldots, m-1\}$ (as these are the DC modes). The forward transform becomes

$$F_{q\ell} = \sum_{s=0}^{m-1} \zeta_m^{\ell s} \tilde{f}_s,$$

which can be computed using a real-to-complex FFT. Similarly, the inverse can be computed using a complex-to-real FFT.[2]

[1] We don't need to compute these conjugates explicitly, as everything is handled by the real-to-complex/complex-to-real FFTs of size p.

[2] Using real-to-complex/complex-to-real FFTs for this case actually has some disadvantages. One such disadvantage (pertaining to multidimensional convolutions) is that, in practice, computing many in-place real-to-complex/complex-to-real FFTs is often less efficient than computing the same number of in-place complex FFTs of the same size. We also don't save any memory by using real-to-complex/complex-to-real FFTs for this part of the residue, as we need this memory when $v > 0$. In contrast, the algorithms described in chapter 4 can save memory (specifically the important case when $q = 2$) when using real-to-complex/complex-to-real FFTs for residue 0. Another issue is that the residue contribution will have a gap in memory, as real-to-complex FFTs utilize Hermitian symmetry and only half of the modes will be computed. This must be accounted for when applying

Now consider the case when $u = p/2$. Recall that $q = np$, so $r = un = pn/2 = q/2$. Furthermore, $\tilde{f}_{pm/2+s} \in \mathbb{R}$ for all $0 \leq s < m$ (as these are the Nyquist modes).

Thus we see that this is same as the $r = q/2$ case of section 4.A, and we can handle it in the same way: we require that m be even and compute this contribution using complex FFTs of size $m/2$.

Residue $v = n/2$:

Note that this case is only relevant when n is even. The forward transform is now

$$F_{q\ell+un+n/2} = \sum_{s=0}^{m-1} \zeta_m^{\ell s} \zeta_{qm}^{(un+n/2)s} \sum_{t=0}^{p-1} \zeta_p^{ut} \zeta_{2p}^{t} f_{tm+s}, \tag{6.7}$$

and (6.4) becomes

$$h_{n/2,s,t} \doteq \zeta_{2p}^{-t} \sum_{u=0}^{p-1} \zeta_p^{-tu} \zeta_{qm}^{-s(un+n/2)} \sum_{\ell=0}^{m-1} \zeta_m^{-s\ell} F_{q\ell+un+n/2}. \tag{6.8}$$

Consider just the DFT of size p in (6.7)

$$\sum_{t=0}^{p-1} \zeta_p^{ut} \zeta_{2p}^{t} f_{tm+s}. \tag{6.9}$$

Note that because $\boldsymbol{f}$ is real, (6.9) is of the exact same form as the DFT in (4.3). Thus, if we require p to be even, then we can use the algorithm for $r = q/2$ in section 4.A to compute (6.9) using complex FFTs of size $p/2$.

the multiplication routine. Of course, there is less multiplication to perform if one uses real-to-complex/complex-to-real FFTs, but the inner loop is typically only useful for relatively small values of m (so the savings are minimal). For these reasons, our implementation of the real inner loop in `FFTW++` [BRM23] uses a complex FFT to compute this part of the residue. This is also what is done in the pseudocode for the real inner loop, which is given in algorithms 22 and 23.

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

$$t = \alpha\varphi + \beta, \ \ \alpha \in \{0, 1\}, \ \beta \in \{0, \ldots, \varphi - 1\},$$

$$u = 2\gamma + \delta, \ \ \gamma \in \{0, \ldots, \varphi - 1\}, \ \delta \in \{0, 1\}.$$

Writing (6.9) in terms of the new indices gives us

$$\sum_{t=0}^{p-1} \zeta_p^{ut} \zeta_{2p}^{t} f_{tm+s} = \sum_{\beta=0}^{\varphi-1} \zeta_\varphi^{\gamma\beta} \zeta_{2p}^{\beta(2\delta+1)} \left[f_{\beta m+s} + (-1)^{\delta} i f_{(\varphi+\beta)m+s} \right].$$

If we set $\delta = 0$, the forward transform becomes

$$F_{q\ell+2\gamma n+n/2} = \sum_{s=0}^{m-1} \zeta_m^{\ell s} \zeta_{qm}^{(2\gamma n+n/2)s} \sum_{\beta=0}^{\varphi-1} \zeta_\varphi^{\gamma\beta} \zeta_{2p}^{\beta} \left[f_{\beta m+s} + i f_{(\varphi+\beta)m+s} \right]. \tag{6.10}$$

Next, we write (6.8) in terms of the new indices

$$\begin{aligned} h_{n/2,s,\alpha\varphi+\beta} = i^{-\alpha} \zeta_{2p}^{-\beta} \Bigg[&\sum_{\gamma=0}^{\varphi-1} \zeta_\varphi^{-\beta\gamma} \zeta_{qm}^{-s(2\gamma n+n/2)} \sum_{\ell=0}^{m-1} \zeta_m^{-s\ell} F_{q\ell+2\gamma n+n/2} + \\ &(-1)^{\alpha} \zeta_p^{-\beta} \sum_{\gamma=0}^{\varphi-1} \zeta_\varphi^{-\beta\gamma} \zeta_{qm}^{-s((2\gamma+1)n+n/2)} \sum_{\ell=0}^{m-1} \zeta_m^{-s\ell} F_{q\ell+(2\gamma+1)n+n/2} \Bigg]. \end{aligned} \tag{6.11}$$

Similarly to (4.7), we define

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

so that (6.11) becomes

$$h_{n/2,s,\beta} = 2 \operatorname{Re}\left(w_{0,\beta,s} \right), \quad h_{n/2,s,\varphi+\beta} = 2 \operatorname{Im}\left(w_{0,\beta,s} \right).$$

This algorithm allows us to compute the $v = n/2$ residue using only $p/2$ DFTs of size m (instead of p DFTs of size m). Therefore, we can use complex FFTs of size m in (6.10) without sacrificing efficiency.

Pseudocode for the real inner loop is given in algorithms 22 and 23.

6.B Conjugate symmetry optimization

In the complex and Hermitian symmetric cases, we can exploit conjugate symmetries in the primitive roots.[3] This has been used in previous implementations of implicit dealiasing [BR11] for centered convolutions, as discussed in section 3.A. Here we extend the technique to more general situations, including $p > 2$.

First, we consider the standard case (chapter 2). We use the inner-loop formulation from section 6.A.1 for generality. Assuming that $v \notin \{0, n/2\}$, define

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

Then we have

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

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

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

A similar optimization in the centered and Hermitian cases (described in sec-

[3] This optimization does not apply to the real case, as using conjugate symmetries is already central to those algorithms.

Algorithm 22 `forwardInnerR` is the real forward transform for residue v when $p > 2$.

```
Input: ${f_j}_{j=0}^{L-1}$, $L, m, q, v$
  $\varphi \leftarrow \lceil p/2 \rceil$
  $h \leftarrow \lfloor m/2 \rfloor$
  if $v < n/2$ then
    for $t = 0, \dots, p-2$ do
      for $s = 0, \dots, m-1$ do
        $W_{tm+s} \leftarrow \zeta_q^{vt} f_{tm+s}$
    for $s = 0, \dots, L-(p-1)m-1$ do
      $W_{tm+s} \leftarrow \zeta_q^{v(p-1)} f_{tm+s}$
    for $s = L-(p-1)m, \dots, m-1$ do
      $W_{tm+s} \leftarrow 0$
    if $v = 0$ then
      for $s = 0, \dots, m-1$ do
        $\{W_{um+s}\}_{u=0}^{\lfloor p/2 \rfloor} \leftarrow \texttt{rcfft}(\{W_{tm+s}\}_{t=0}^{p-1})$
      for $u = 0, \dots, \varphi-1$ do
        for $s = 0, \dots, m-1$ do
          $W_{um+s} \leftarrow \zeta_{qm}^{uns} W_{um+s}$
        $\{V_{um+\ell}\}_{\ell=0}^{m-1} \leftarrow \texttt{fft}(\{W_{um+s}\}_{s=0}^{m-1})$
      if $p$ even then
        for $b = 0, \dots, h-1$ do
          $W_{\varphi m+b} \leftarrow \zeta_{2m}^{b} W_{\varphi m+b} +$
                  $i\zeta_{2m}^{b} W_{\varphi m+h+b}$
        $\{V_{\varphi m+c}\}_{c=0}^{h-1} \leftarrow \texttt{fft}(\{W_{\varphi m+b}\}_{b=0}^{h-1})$
        return $\{V_k\}_{k=0}^{\varphi m+h-1}$
      else
        return $\{V_k\}_{k=0}^{\varphi m-1}$
    else
      for $s = 0, \dots, m-1$ do
        $\{W_{um+s}\}_{u=0}^{p-1} \leftarrow \texttt{fft}(\{W_{tm+s}\}_{t=0}^{p-1})$
      for $u = 0, \dots, p-1$ do
        for $s = 0, \dots, m-1$ do
          $W_{um+s} \leftarrow \zeta_{qm}^{(un+v)s} W_{um+s}$
        $\{V_{um+\ell}\}_{\ell=0}^{m-1} \leftarrow \texttt{fft}(\{W_{um+s}\}_{s=0}^{m-1})$
    return $\{V_k\}_{k=0}^{pm-1}$
  else if $v = n/2$ then
    for $\beta = 0, \dots, \varphi-2$ do
      for $s = 0, \dots, m-1$ do
        $W_{\beta m+s} \leftarrow \zeta_{2p}^{\beta}(W_{\beta m+s} + iW_{(\varphi+\beta)m+s})$
    for $s = 0, \dots, L-(p-1)m-1$ do
      $W_{(\varphi-1)m+s} \leftarrow \zeta_{2p}^{\varphi-1} W_{(\varphi-1)m+s} +$
                $i\zeta_{2p}^{\varphi-1} W_{(\varphi-1+\beta)m+s}$
    for $s = L-(p-1)m, \dots, m-1$ do
      $W_{(\varphi-1)m+s} \leftarrow \zeta_{2p}^{\varphi-1} W_{(\varphi-1)m+s}$
    for $s = 0, \dots, m-1$ do
      $\{W_{\gamma m+s}\}_{\gamma=0}^{\varphi-1} \leftarrow \texttt{fft}(\{W_{\beta m+s}\}_{\beta=0}^{\varphi-1})$
    for $\gamma = 0 \dots, \varphi-1$ do
      for $s = 0 \dots, m-1$ do
        $W_{\gamma m+s} \leftarrow \zeta_{qm}^{(2\gamma n+n/2)s} W_{\gamma m+s}$
      $\{V_{\gamma m+\ell}\}_{\ell=0}^{m-1} \leftarrow \texttt{fft}(\{W_{\gamma m+s}\}_{s=0}^{m-1})$
    return $\{V_k\}_{k=0}^{\varphi m-1}$
```

Algorithm 23 `backwardInnerR` is the real backward transform for residue v when $p > 2$.

```
Input: $\{F_k\}_{k=0}^{pm-1}$, $L, m, q, v$
  $\varphi \leftarrow \lceil p/2 \rceil$
  $h \leftarrow \lfloor m/2 \rfloor$
  if $v = 0$ then
    for $u = 0, \dots, \varphi-1$ do
      $\{W_{um+s}\}_{s=0}^{m-1} \leftarrow \texttt{ifft}(\{V_{um+\ell}\}_{\ell=0}^{m-1})$
      for $s = 0, \dots, m-1$ do
        $W_{um+s} \leftarrow \zeta_{qm}^{-uns} W_{um+s}$
    if $p$ even then
      $\{W_{\varphi m+b}\}_{b=0}^{h-1} \leftarrow \texttt{ifft}(\{V_{\varphi m+c}\}_{c=0}^{h-1})$
      for $b = 0 \dots, h-1$ do
        $W_{\varphi m+h+b} \leftarrow 2\,\mathrm{Im}\Big\{\zeta_{2m}^{-b} W_{\varphi m+b}\Big\}$
        $W_{\varphi m+b} \leftarrow 2\,\mathrm{Re}\Big\{\zeta_{2m}^{-b} W_{\varphi m+b}\Big\}$
    for $s = 0, \dots, m-1$ do
      $\{W_{tm+s}\}_{t=0}^{p-1} \leftarrow \texttt{crfft}(\{W_{um+s}\}_{u=0}^{\lfloor p/2 \rfloor})$
  else if $v < n/2$ then
    for $u = 0, \dots, p-1$ do
      $\{W_{um+s}\}_{s=0}^{m-1} \leftarrow \texttt{ifft}(\{V_{um+\ell}\}_{\ell=0}^{m-1})$
      for $s = 0, \dots, m-1$ do
        $W_{um+s} \leftarrow \zeta_{qm}^{-s(un+v)} W_{um+s}$
    for $s = 0, \dots, m-1$ do
      $\{W_{tm+s}\}_{t=0}^{p-1} \leftarrow \texttt{ifft}(\{W_{um+s}\}_{u=0}^{p-1})$
    for $t = 0, \dots, p-2$ do
      for $s = 0, \dots, m-1$ do
        $W_{tm+s} \leftarrow 2\,\mathrm{Re}\Big\{\zeta_q^{-tv} W_{tm+s}\Big\}$
    for $s = 0, \dots, L-(p-1)m-1$ do
      $W_{(p-1)m+s} \leftarrow 2\,\mathrm{Re}\Big\{\zeta_q^{-(p-1)v} W_{(p-1)m+s}\Big\}$
  else if $v = n/2$ then
    for $\gamma = 0 \dots, \varphi-1$ do
      $\{W_{\gamma m+s}\}_{s=0}^{m-1} \leftarrow \texttt{ifft}(\{V_{\gamma m+\ell}\}_{\ell=0}^{m-1})$
      for $s = 0 \dots, m-1$ do
        $W_{\gamma m+s} \leftarrow \zeta_{qm}^{-s(2\gamma n+n/2)} W_{\gamma m+s}$
    for $s = 0, \dots, m-1$ do
      $\{W_{\beta m+s}\}_{\beta=0}^{\varphi-1} \leftarrow \texttt{ifft}(\{W_{\gamma m+s}\}_{\gamma=0}^{\varphi-1})$
    for $\beta = 0, \dots, \varphi-2$ do
      for $s = 0, \dots, m-1$ do
        $W_{(\varphi+\beta)m+s} \leftarrow 2\,\mathrm{Im}\Big\{\zeta_{2p}^{-\beta} W_{\beta m+s}\Big\}$
        $W_{\beta m+s} \leftarrow 2\,\mathrm{Re}\Big\{\zeta_{2p}^{-\beta} W_{\beta m+s}\Big\}$
    for $s = 0, \dots, L-(p-1)m-1$ do
      $W_{(p-1)m+s} \leftarrow 2\,\mathrm{Im}\Big\{\zeta_{2p}^{-(\varphi-1)} W_{(\varphi-1)m+s}\Big\}$
      $W_{(\varphi-1)m+s} \leftarrow 2\,\mathrm{Re}\Big\{\zeta_{2p}^{-(\varphi-1)} W_{(\varphi-1)m+s}\Big\}$
    for $s = L-(p-1)m, \dots, m-1$ do
      $W_{(\varphi-1)m+s} \leftarrow 2\,\mathrm{Re}\Big\{\zeta_{2p}^{-(\varphi-1)} W_{(\varphi-1)m+s}\Big\}$
  return $\{W_j\}_{j=0}^{L-1}$
```

tions 3.A and 3.B) is obtained with

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

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

which allows us to compute (6.3) using

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

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

6.C Overwrite optimization

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

The overwrite optimization is particularly advantageous in the standard case (chapter 2) when $M \leq 2L$ and $m = L$, so that $p = 1$ and $q = 2$. In this case, the input buffer already contains the preprocessed data for $r = 0$ and the preprocessed data $\zeta_{qm}^s f_s$ for $r = 1$ is written to the output buffer. The input and output buffers are then individually Fourier transformed to obtain the required residues. The backwards transform does the reverse operation: it first performs inverse Fourier transforms on the input and output buffers, then adds products of the output buffer and roots of unity to the input buffer.

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

6.D Loop optimizations

If the overwrite optimization is not applicable, other data flow improvements may be possible. Suppose that we compute a block of D residues at a time as described in section 2.C. Normally, the contribution to the inverse padded Fourier transform from the block containing residue 0 is stored in an accumulation buffer; the contributions from the remaining residues are then added to this buffer by iterating over the other residue blocks. If there are no other residue blocks, a separate accumulation buffer is not needed; one can accumulate the residues entirely within the input buffer.

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

Chapter 7

Numerical results

This work presents several algorithms for computing padded/unpadded FFTs. Which algorithm is optimal for a given problem depends on the number of inputs A and outputs B, the multiplication operator `mult`, the input data length L, the padded length M, the number of copies C of the transform to be computed simultaneously, and the stride S between successive data elements of each copy. For simplicity, we only consider the efficiently packed case, where the distance in memory between the first element of each copy is one. Vectorized and parallelized C++ versions of these algorithms have been implemented in the open-source library `FFTW++` [BRM23].

We determine the fastest algorithm for a given problem empirically, scanning over the underlying FFT size m, the number D of residues to be computed at a time (as described in section 2.C), and whether or not to use in-place or out-of-place FFTs. These parameters then determine the values of p and q to use in our padded FFT algorithms.

Our optimizer measures the time required to compute a one-dimensional in-place dealiased convolution for a particular set of parameters, using the given multiplication routine. As described in chapter 5, multidimensional convolutions are decomposed into a sequence of padded FFTs, a one-dimensional convolution, and then a sequence of inverse padded FFTs. We assume that the padded/unpadded FFT pairs can be optimized independently in each dimension. This is accomplished by performing one-dimensional convolutions using each padded FFT pair, without calling the multiplication routine. In practice, this decoupling works well and leads to efficient optimization of multidimensional convolutions. If the outermost pair FFT_1 and FFT_1^{-1} in figure 5.1 are to be multithreaded over T threads, optimization of the remaining FFTs should be performed over T concurrent copies, to simulate the execution environment.

Timings were done using the `chrono::nanoseconds` clock from the C++ standard library. For each size, we recorded the time to compute an unnormalized convolution[1] of two inputs. This was repeated for at least 5 seconds and until we had at least 20 samples. We plot the median times from each of these tests.

We benchmarked our algorithms with a liquid-cooled Intel i9-12900K processor (5.2GHz, 8 performance cores) on an ASUS ROG Strix Z690-F motherboard with 128GB of DDR5 memory (5GHz), using version 12.2.1 of the `GCC` compiler with the optimizations `-Ofast -fomit-frame-pointer -fstrict-aliasing -ffast-math`. The underlying FFTs were computed with version 3.3.10 of the adaptive `FFTW` [FJ; FJ05] library under the Fedora 37 operating system. Multithreading was implemented with the `OpenMP` library.

7.A Complex convolution benchmarks

First, we benchmark the median execution times for the complex convolution algorithms described in chapter 2. We do so in one, two, and three dimensions, each over

[1] By unnormalized convolution, we mean that we do not normalize the inverse FFTs.

one thread and eight threads. These algorithms are compared to implicit dealiasing [RB18] and explicit dealiasing. In one dimension, we compare to both in-place (IP) and out-of-place (OP) FFTs with explicit dealiasing. For multidimensional convolutions, only in-place FFTs are used for the explicit routines. The convolutions use $1/2$ padding, which is the necessary padding required to fully dealias a binary convolution.

7.A.1 One-dimensional complex convolutions

In figure 7.1 we plot the normalized times for one-dimensional in-place convolutions of L complex words over one thread. We see that hybrid dealiasing is much faster than both explicit and implicit dealiasing for large sizes and is generally competitive with optimized out-of-place explicit algorithms (to which it reduces) for small sizes. In figure 7.2 we plot the normalized times for the same one-dimensional convolutions parallelized over 8 threads. In these plots, only power-of-two sizes are benchmarked since these are optimal FFT sizes.

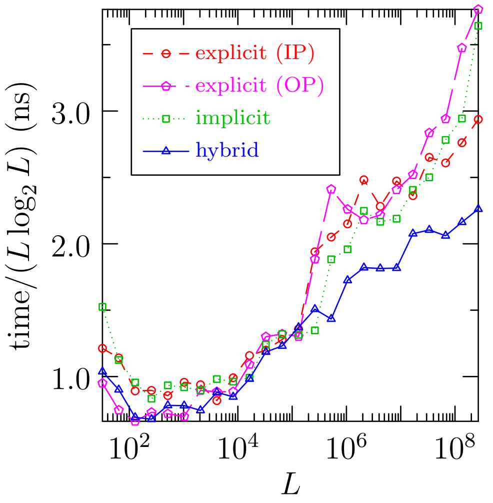

Figure 7.1: In-place 1D complex convolutions of length L with $A = 2$ and $B = 1$ on 1 thread.

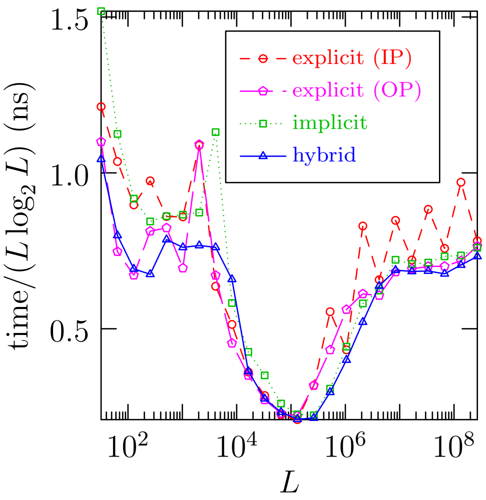

Figure 7.2: In-place 1D complex convolutions of length L with $A = 2$ and $B = 1$ on 8 threads.

In order to evaluate the normalized L^2 error for our algorithms, we define input arrays $f_j = \alpha e^{ij}$ and $g_j = \beta e^{ij}$ with $\alpha = \sqrt{3} + i\sqrt{7}$ and $\beta = \sqrt{5} + i\sqrt{11}$ for $j = 0, \ldots, L-1$, following Bowman and Roberts [BR11]. Then in figure 7.3, we display the error in the convolution of $\{f_k\}_{k=0}^{L-1}$ and $\{g_k\}_{k=0}^{L-1}$ computed using hybrid dealiasing relative to the exact solution $\sum_{p=0}^{j} f_p g_{j-p} = \alpha\beta(j+1)e^{ij}$.

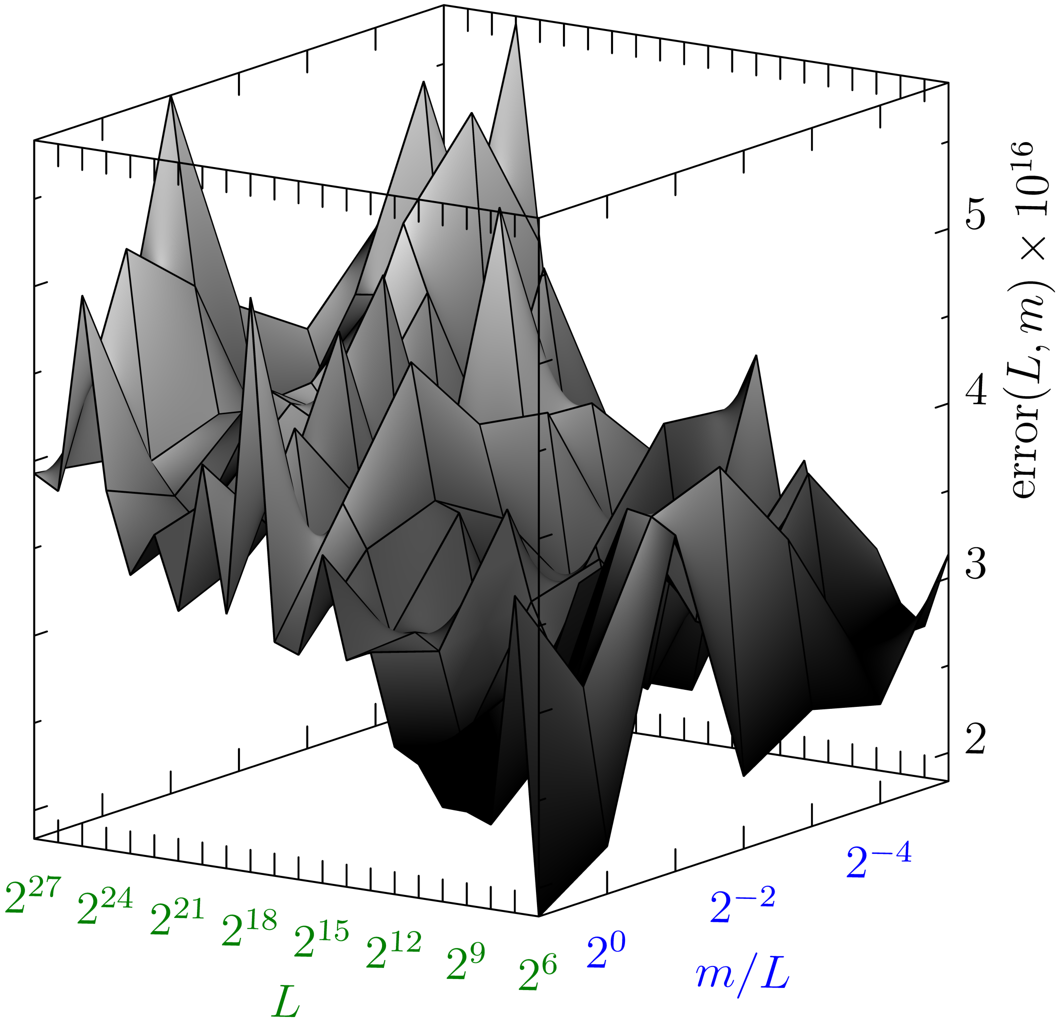

Figure 7.3: Normalized L^2 error for in-place 1D complex convolutions of size L for different m values. Here $D = 1$ and the FFTs are in place.

7.A.2 Two-dimensional complex convolutions

In figures 7.4 and 7.5 we plot the normalized times for two-dimensional complex convolutions of size $L \times L$. On a single thread, these benchmarks show that at all sizes, hybrid dealiasing is faster than implicit dealiasing and much faster than explicit dealiasing. On 8 threads, hybrid dealiasing outperforms both methods except at $L = 64$. In both cases, an x stride of $L + 2$ was used.

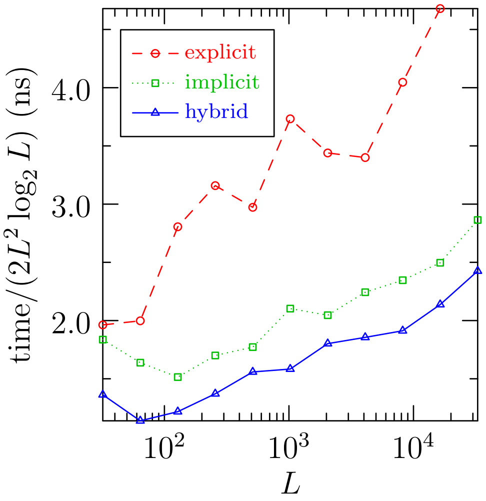

Figure 7.4: In-place 2D complex convolutions of size $L \times L$ with $A = 2$ and $B = 1$ on 1 thread.

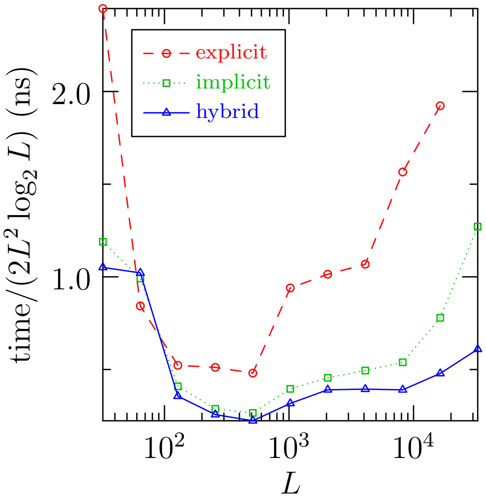

Figure 7.5: In-place 2D complex convolutions of size $L \times L$ with $A = 2$ and $B = 1$ on 8 threads.

7.A.3 Three-dimensional complex convolutions

Figures 7.6 and 7.7 show the normalized times for three-dimensional complex convolutions. In the single-threaded case, we used the y stride $S_y = L + 2$ and the x stride $S_y L + 2$; in the multithreaded-threaded case, we used the y stride $S_y = L$ and the x stride $S_y L + 4$. Again, we observe that hybrid dealiasing outperforms the other two methods.

In figure 7.8, we emphasize that, unlike explicit and implicit dealiasing, hybrid

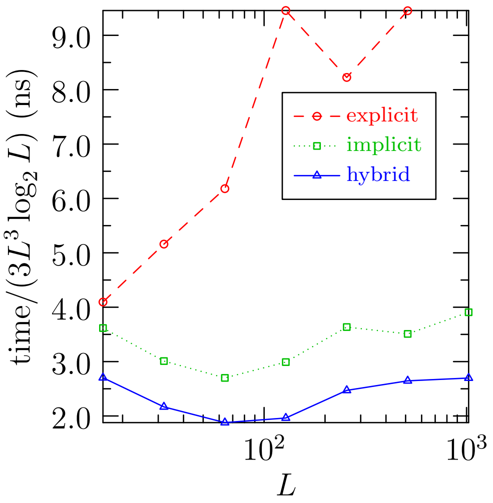

Figure 7.6: In-place 3D complex convolutions of size $L \times L \times L$ with $A = 2$ and $B = 1$ on 1 thread.

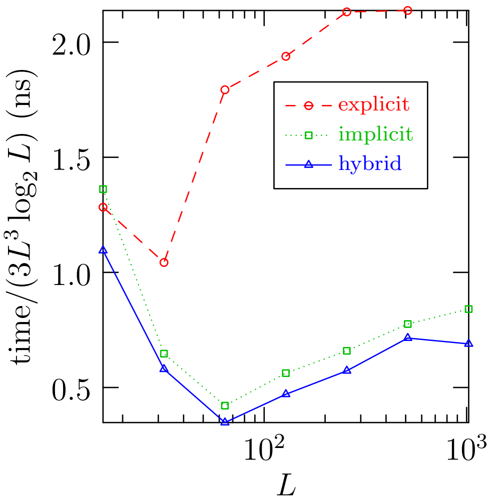

Figure 7.7: In-place 3D complex convolutions of size $L \times L \times L$ with $A = 2$ and $B = 1$ on 8 threads.

dealiasing performs exceptionally well over a range of arbitrary sizes.

7.B Hermitian convolution benchmarks

Next, we benchmark the median execution times for the Hermitian convolution algorithms (described in chapter 3). We do so in one, two, and three dimensions, and over one and eight threads. We perform the same comparisons that we did in the complex benchmarks (section 7.A). For one-dimensional Hermitian convolutions, L refers to the theoretical length of the input array, as we only store $\lfloor L/2 \rfloor + 1$ complex words in memory.

We also note that when decomposing multidimensional Hermitian convolutions (as in chapter 5), the outer FFTs are all complex and centered; only the inner one-dimensional convolutions are Hermitian.

These convolutions use $2/3$ padding (as is done in pseudospectral simulations). In the interest of making each routine as fast as possible, we use different input sizes

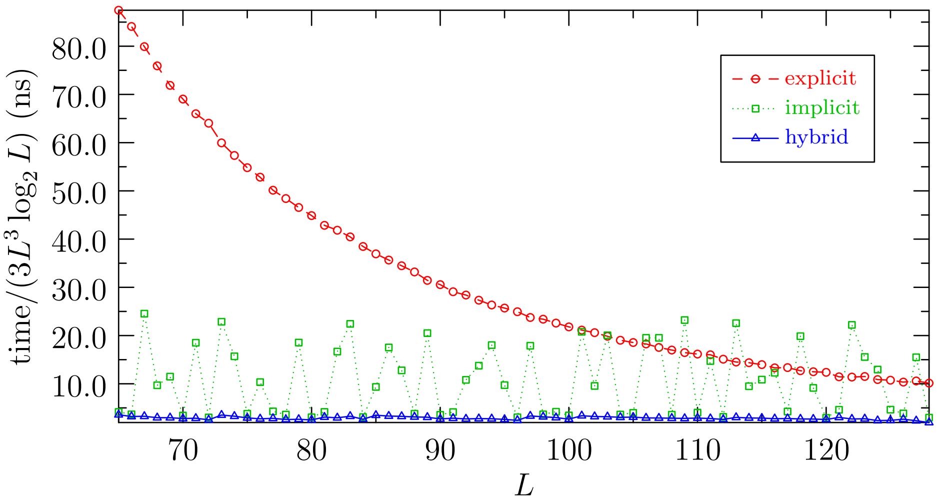

Figure 7.8: In-place 3D complex convolutions of incremental sizes $L \times L \times L$ with $A = 2$ and $B = 1$ on 1 thread. As is conventional, the explicit dealiasing algorithm pads past $2L$, up to the next power of two (in this case 256).

for the explicit and implicit routines and compare hybrid dealiasing to both. For the implicit dealiasing algorithms the optimal sizes are one less than a power of two. In our implementation of hybrid dealiasing, we normally adjust these sizes to exact powers of two to allow us to use the overwrite optimization. For explicit dealiasing, the optimal values of L are $2\left\lfloor \frac{2^n+2}{3} \right\rfloor - 1$ for positive integers n.

7.B.1 One-dimensional Hermitian convolutions

In figures 7.9 and 7.10 we plot the normalized times for one-dimensional Hermitian convolutions over one thread and eight threads, respectively. We observe that hybrid dealiasing outperforms implicit dealiasing at optimal implicit sizes, and performs about as well as explicit dealiasing at optimal explicit sizes.

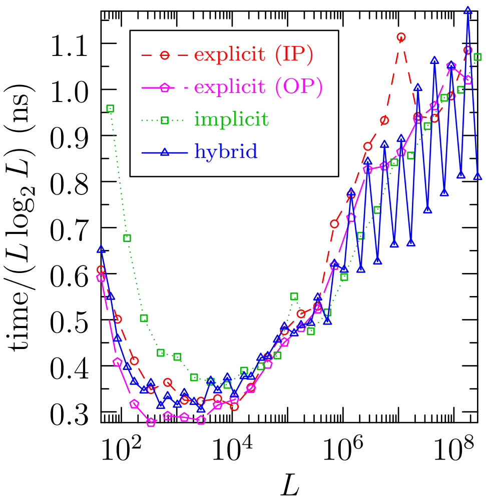

Figure 7.9: In-place 1D Hermitian convolutions of length L with $A = 2$ and $B = 1$ on 1 thread.

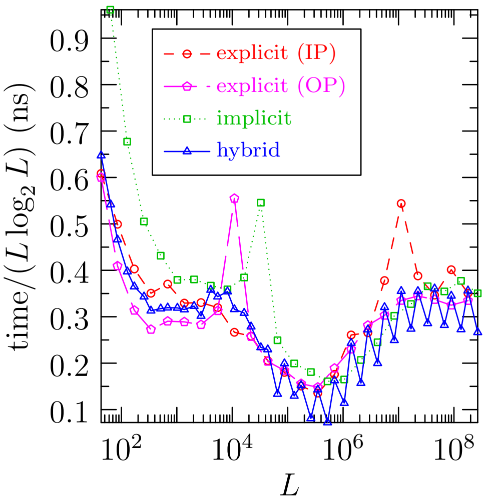

Figure 7.10: In-place 1D Hermitian convolutions of length L with $A = 2$ and $B = 1$ on 8 threads.

7.B.2 Two-dimensional Hermitian convolutions

In figures 7.11 and 7.12, we see that hybrid dealiasing outperforms both implicit and explicit dealiasing for all sizes. In both cases we used an x stride of $\lceil \frac{L}{2} \rceil + 2$.

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

7.B.3 Three-dimensional Hermitian convolutions

In figure 7.13 we observe that for three-dimensional Hermitian convolutions on a single thread, hybrid dealiasing performs better than implicit dealiasing and much better than explicit dealiasing, except at $L = 21$. As seen in figure 7.14, when run on 8 threads, hybrid dealiasing is much faster than the other methods for all

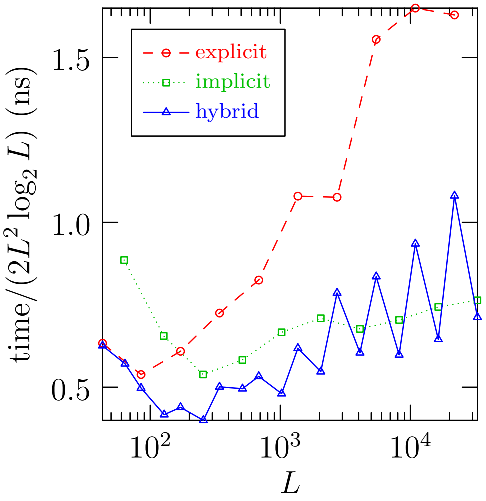

Figure 7.11: In-place 2D Hermitian convolutions of size $L \times L$ with $A = 2$ and $B = 1$ on 1 thread.

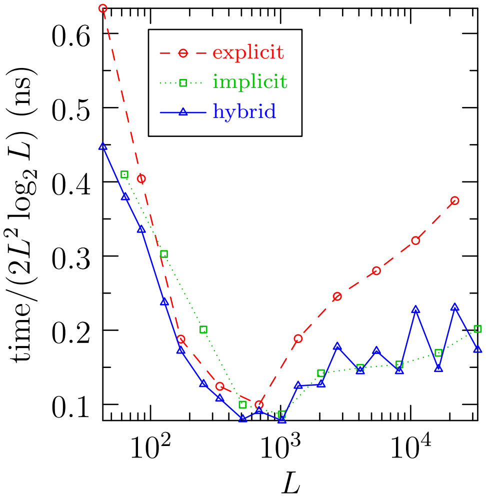

Figure 7.12: In-place 2D Hermitian convolutions of size $L \times L$ with $A = 2$ and $B = 1$ on 8 threads.

sizes. In the single-threaded case, we used the y stride $S_y = \lceil \frac{L}{2} \rceil + 2$ and the x stride $S_y L + 2$; in the multithreaded-threaded case, we used the y stride $S_y = L$ and the x stride $S_y L + 4$.

7.C Real convolution benchmarks

Finally, we benchmark the median execution times for the real convolution algorithms (described in chapter 4) in one and two dimensions, each over one thread.[2] We perform the explicit dealiasing comparisons that we did in the complex benchmarks (section 7.A).[3] Just like the complex benchmarks, the real convolutions also use $1/2$ padding.

Note that when decomposing multidimensional real convolutions (as in chapter 5), the outermost padded/unpadded FFTs are real, and all inner FFTs and

[2] Multithreading is available for real convolutions in `FFTW++` , but their efficiency is still lacking and is currently under development.

[3] We cannot compare to implicit dealiasing, as it was not developed for real-valued inputs.

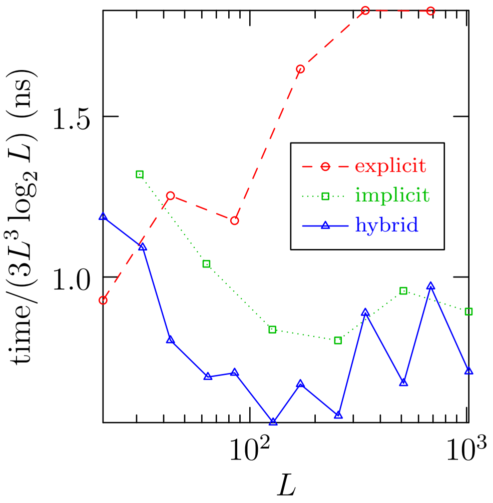

Figure 7.13: In-place 3D Hermitian convolutions of size $L \times L \times L$ with $A = 2$ and $B = 1$ on 1 thread.

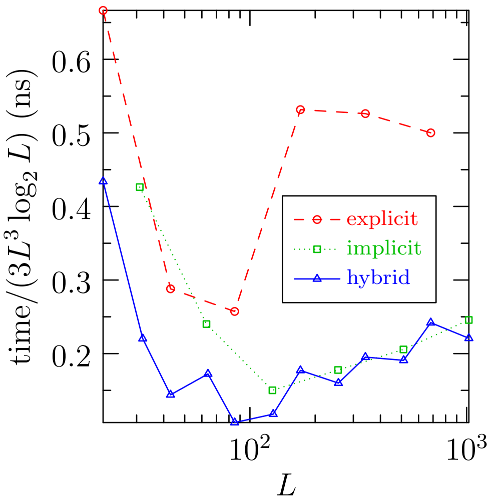

Figure 7.14: In-place 3D Hermitian convolutions of size $L \times L \times L$ with $A = 2$ and $B = 1$ on 8 threads.

inner one-dimensional convolutions are complex (and use the algorithms from chapter 2).

7.C.1 One-dimensional real convolutions

In figure 7.15 shows the normalized times for one-dimensional real convolutions over one thread. We see that hybrid dealiasing is competitive or comparable to explicit dealiasing, with some efficiency concerns at very large sizes. While these results may seem underwhelming, the algorithms used in these benchmarks form the basis of the 2D convolutions in the next section.

7.C.2 Two-dimensional real convolutions

In figure 7.16, we plot the normalized times for two-dimensional real convolutions. We see that hybrid dealiasing outperforms explicit dealiasing for all sizes.

We should also note a unique behavior regarding the real convolutions. In

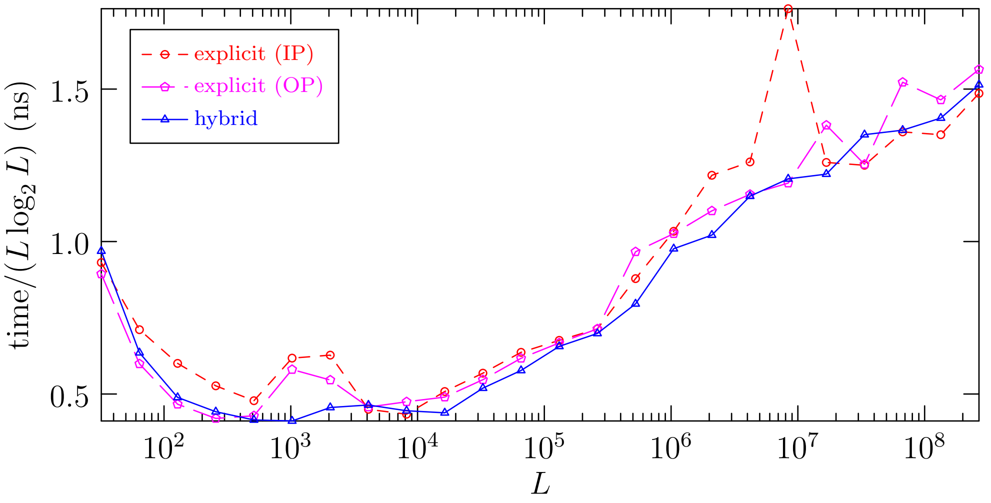

Figure 7.15: In-place 1D real convolutions of length L with $A = 2$ and $B = 1$ on 1 thread.

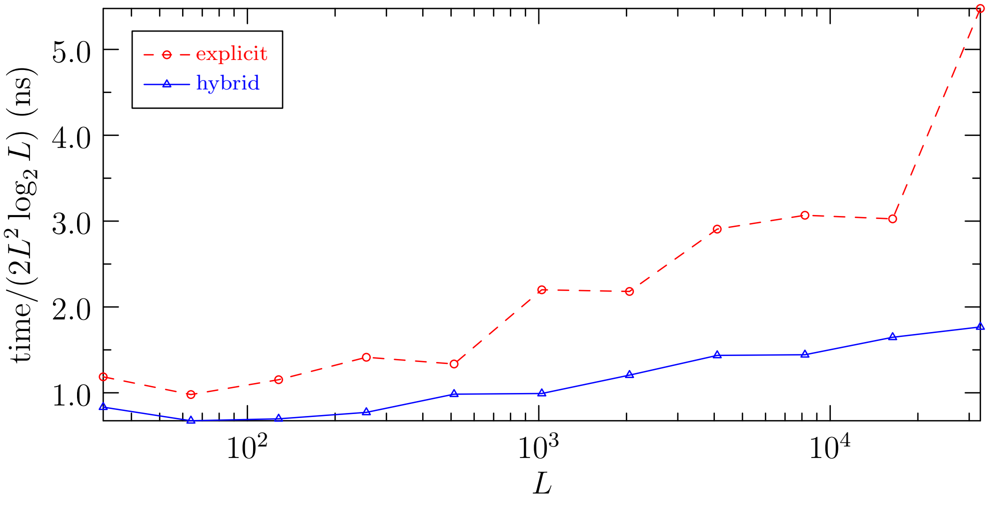

Figure 7.16: In-place 2D real convolutions of size $L \times L$ with $A = 2$ and $B = 1$ on 1 thread.

the complex and Hermitian cases, the high performance of multidimensional convolutions can be largely attributed to the memory savings obtained by using the decomposition described in chapter 5. While using implicit zero-padding is still optimal, one can use this decomposition with explicitly zero-padding, and still perform much better than the traditional explicit dealiasing methods. This is not true for the real convolutions; computing several in-place real-to-complex/complex-to-real FFTs at once is inefficient (compared to computing several complex FFTs). Because our real algorithms don't solely on real-to-complex/complex-to-real FFTs, they are much more efficient for computing the outer FFTs.[4]

7.D Summary of Chapter 7

In this chapter, we have benchmarked hybrid dealiasing and compared it to both explicit and implicit dealiasing. In one dimension, we saw that hybrid dealiasing either outperformed or was comparable to both explicit and implicit dealiasing. In two and three dimensions, hybrid dealiasing consistently outperformed implicit dealiasing and greatly outperformed explicit dealiasing.

The improved performance in two and three dimensions can largely be attributed to the memory-saving decomposition described in chapter 5. In the real case specifically, hybrid dealiasing's improved performance over explicit dealiasing is also due to the fact that the FFTs are not all real-to-complex/complex-to-real.

Most of the comparisons in this chapter were done using arrays with sizes optimal for explicit and implicit dealiasing. However, hybrid dealiasing is designed to be efficient for any input size. This was exemplified in figure 7.8, which shows that

[4] We don't have the same behavior with the Hermitian convolutions because in that case, the outer FFTs are all complex.

the performance of hybrid dealiasing is much less susceptible to the input size than both explicit and implicit dealiasing.

Chapter 8

Conclusion and future work

In section 1.B, we listed four problems with implicit dealiasing [BR11; RB18]. This list can be summarized as follows:

1. Implicit dealiasing is not efficient for inputs of all sizes.

2. Implicit dealiasing does not allow for arbitrary padding requirements.

3. Implicit dealiasing cannot efficiently convolve real-valued inputs.

4. Implicit dealiasing cannot convolve arrays of unequal sizes.

In this work, we have provided solutions to problems (1), (2), and (3). In the following section, we discuss an alternative approach to problem (3). In the section after that, we discuss some of the issues with problem (4) and outline a potential solution.

8.A Real convolutions via complex packing

In chapter 4, we developed a hybrid dealiasing algorithm which computed compute convolutions with real-valued inputs. This algorithm makes direct use of conjugate symmetries in the transformed data; however, there is another way we could have approached this problem.

Complex packing is a technique for computing the DFT of real data using complex FFTs [Pre+07]. Let $N \in \mathbb{N}$ be even[1] and let $\tilde{N} \doteq N/2$. Let $\{f_j\}_{j=0}^{N-1}$ be a real array and define $\tilde{\boldsymbol{f}} = \left\{\tilde{f}_j\right\}_{j=0}^{\tilde{N}-1}$ via

$$\tilde{f}_j = f_{2j} + i f_{2j+1}. \tag{8.1}$$

That is, the even and odd indices of $\boldsymbol{f}$ are the real and imaginary parts of $\tilde{\boldsymbol{f}}$. After taking the DFT $\tilde{\boldsymbol{F}}$, one can use symmetries to recover the $\boldsymbol{F}$. To *unpack* $\tilde{\boldsymbol{F}}$ in Fourier space, we have

$$F_k = \frac{1}{2}\left(\tilde{F}_k + \overline{\tilde{F}_{\tilde{N}-k}}\right) - \frac{i}{2}\left(\tilde{F}_k - \overline{\tilde{F}_{\tilde{N}-k}}\right)\zeta_{2\tilde{N}}^{-k}, \quad k \in \{0, \dots, \tilde{N}\}, \tag{8.2}$$

which has inverse[2]

$$\tilde{F}_k = \frac{1}{2}\left(F_k + \overline{F_{\tilde{N}-k}}\right) - \frac{i}{2}\left(F_k - \overline{F_{\tilde{N}-k}}\right)\zeta_N^{k}, \quad k \in \{0, \dots, \tilde{N}-1\}. \tag{8.3}$$

Thus, one can compute real convolutions using only complex DFTs, and the dealiasing can be done using the transforms described in chapter 2.

Even though one can use complex FFTs (which are, in practice, more efficient compared to real-to-complex/complex-to-real FFTs), packing and unpacking in Fourier space is costly. Indeed, it has been shown that packing algorithms for computing the DFT of real inputs “take several N more additions than a specialized algorithm for real input data” [Sor+87]. However, it is important to remember that

[1] These methods can also be used when N is odd. We only make this assumption for simplicity.

[2] Details of these calculations can be found in Press et al. [Pre+07].

we are not trying to compute the DFT; we are interested in convolutions. Thus, a possible solution to this problem is to build unpacking and packing into the multiplication operator.

For example, suppose we want to compute the binary convolution of two real arrays $\boldsymbol{f}$ and $\boldsymbol{g}$. We pack them to obtain $\tilde{\boldsymbol{f}}$ and $\tilde{\boldsymbol{g}}$ and compute $\tilde{\boldsymbol{F}}$ and $\tilde{\boldsymbol{G}}$. We then need to compute $\boldsymbol{F}$ and $\boldsymbol{G}$ using (8.2), compute the element wise Hadamard product $\boldsymbol{H} \doteq \boldsymbol{F} \odot \boldsymbol{G}$, and then compute $\tilde{\boldsymbol{H}}$ using (8.3). However, one can show (see appendix A for details) that $\tilde{\boldsymbol{H}}$ can be obtained directly from $\tilde{\boldsymbol{F}}$ and $\tilde{\boldsymbol{G}}$:

$$\tilde{\boldsymbol{H}} = \tilde{\boldsymbol{F}} \boxdot \tilde{\boldsymbol{G}},$$

where

$$(\tilde{\boldsymbol{F}} \boxdot \tilde{\boldsymbol{G}})_k \doteq \tilde{F}_k \tilde{G}_k - \frac{1}{4}\left(\tilde{F}_k - \overline{\tilde{F}_{(\tilde{N}-k) \bmod \tilde{N}}}\right)\left(\tilde{G}_k - \overline{\tilde{G}_{(\tilde{N}-k) \bmod \tilde{N}}}\right)\left(1 + \zeta_{\tilde{N}}^{-k}\right), \qquad k \in \{0, \dots, \tilde{N}-1\}.$$

Initially, it might seem that we can use our complex convolution algorithms exactly as they are, and simply change the multiplication operator. Unfortunately, things are not so simple: note that $\boxdot$ is not an element-wise operator[3] so convolutions cannot be computed one residue at a time (as described in section 2.C). This is could be a major performance issue, especially for multidimensional convolutions.

Despite the potential issues, it may still be desirable to investigate the use of complex packing further. Real-to-complex FFTs are complicated, and one can find examples in which they are less efficient than the corresponding complex FFTs. We also saw in section 7.C.1 that the real convolution algorithm of chapter 4 has

[3] Such convolutions are also not general convolutions in the sense of definition 1.4.

issues with efficiency at large sizes. It might be better to compute some of these convolutions with a complex packing algorithm.

8.B Convolutions of unequal sizes

The only limitation of implicit dealiasing that we have not treated is the problem of unequal-sized input arrays. Such convolutions are used in applications like signal processing and machine learning, so solving this problem is a high priority.

There are a few issues that must be addressed in order to generalize our approach to this unequal-sized inputs. When the arrays are of different sizes, it is likely inefficient to use FFTs of the same size for each. But regardless of their relative sizes, each of the inputs must be padded (implicitly or explicitly) to the same size. Thus our standard formulation in terms of m, p, and q (described in figure 2.1) must be modified.

We outline a possible solution to this problem in the case of two one-dimensional inputs, $\boldsymbol{f}$ and $\boldsymbol{g}$, of length L_f and L_g respectively. Suppose that we need to pad to a length M. Let $m, \Lambda \in \mathbb{N}$, and consider $m_f \doteq m$ and $m_g \doteq \Lambda m$. Then define

$$p_f \doteq \left\lceil \frac{L_f}{m} \right\rceil, \quad p_g \doteq \left\lceil \frac{L_g}{\Lambda m} \right\rceil, \quad q_g \doteq \left\lceil \frac{M}{\Lambda m} \right\rceil, \quad q_f \doteq \frac{q_g \Lambda m}{m} = \Lambda q_g,$$

so that $q_f m_f = q_g m_g$. The idea then is to use the algorithms presented in this work, using m_f, p_f, and q_f for $\boldsymbol{f}$ and m_g, p_g, and q_g for $\boldsymbol{g}$.

However, this leads to another issue: if $q_g \neq q_f$, then there are different numbers of residues in each computation. This is not a problem if all residues are computed at once, but it is a problem if one wants to compute one residue at a time (as in section 2.C).

To solve this, we note that every residue of $\boldsymbol{g}$ (which is of size $m_g = \Lambda m$), corresponds to Λ residues of $\boldsymbol{f}$ (each of size $m_f = m$). Let $r_g \in [0, q_g - 1]$ and $\ell_g \in [0, m_g - 1]$. Define

$$\ell_f \doteq \left\lfloor \frac{\ell_g}{\Lambda} \right\rfloor, \quad \lambda \doteq \ell_g \bmod \Lambda, \quad r_f \doteq q_g \lambda + r_g,$$

so that

$$q_g \ell_g + r_g = q_g \left(\Lambda \ell_f + \lambda\right) + r_g = \Lambda q_g \ell_f + \lambda q_g + r_g = q_f \ell_f + r_f.$$

In other words, if we compute the residue contribution $\left\{G_{q_g \ell_g + r_g}\right\}_{\ell_g = 0}^{m_g - 1}$, then we only need to compute $\left\{F_{q_f \ell_f + r_f}\right\}_{\ell_f = 0}^{m_f - 1}$ for $r_f \in \left\{q_g \lambda + r_g\right\}_{\lambda = 0}^{\Lambda - 1}$ in order to perform the multiplication.

While this provides a sketch of a possible solution to the problem, there is still much more work to be done. The algorithm needs to generalize to n input arrays and incorporate our numerical optimizations, specifically the inner loop (described in section 6.A).

There are also other algorithms for computing convolutions of unequal-sized arrays that we must now consider. In particular, the overwrite-add and overwrite-save algorithms are used for computing convolutions in which one array is much larger than the other. Informally, both of these algorithms convolve small sections of the larger array with the smaller array. The results of these small convolutions are then combined using linearity to compute the full convolution.[4] Hybrid dealiasing will likely need to incorporate these algorithms in order to be competitive for such applications

[4] The details of overwrite-add and overwrite-save can be found in most standard references on digital signal processing, such as Oppenheim, Schafer, and Buck [OSB99].

8.C Concluding remarks

Although implicit dealiasing offers great improvements to the efficiency of computing linear convolutions, its applications are limited. To remedy this, we have introduced hybrid dealiasing, which also avoids the need to explicitly pad the input data with zeros; however, unlike implicit dealiasing, hybrid dealiasing allows for explicit zero-padding when it is in the interest of performance.

In developing hybrid dealiasing, we observe that if a convolution is all that is needed, one can often achieve better performance by localizing the computations; one can compute the contribution to the convolution one residue at a time. Furthermore, multidimensional convolutions can be done more efficiently by decomposing them into an outer FFT, a lower-dimensional convolution, and an inverse FFT. For complex and Hermitian convolutions, the possibility of reusing work memory in this recursive formulation is responsible for most of the dramatic performance gains that are observed in two and three dimensions.

In this work hybrid dealiasing was also been developed to compute convolutions of real-valued inputs. This was not handled by implicit dealiasing, and it serves as an important step to increasing the number of applications that might benefit from these techniques.

There are still limitations of hybrid dealiasing, the foremost being its inability to compute convolutions of arrays with unequal sizes. Furthermore, future applications may have requirements that are not considered in this work. However, in contrast to implicit dealiasing, hybrid dealiasing serves as a more general framework for computing convolutions. It is therefore much more readily modified for new applications.

References

[Bas83] C Basdevant. “Technical improvements for direct numerical simulation of homogeneous three-dimensional turbulence”. In: *Journal of Computational Physics* 50.2 (1983), pp. 209–214.

[Bow13] John C. Bowman. “Casimir Cascades in Two-Dimensional Turbulence”. In: *J. Fluid Mech* 729 (2013), pp. 364–376.

[BR11] John C. Bowman and Malcolm Roberts. “Efficient Dealiased Convolutions without Padding”. In: *SIAM J. Sci. Comput.* 33.1 (2011), pp. 386–406.

[BRM23] John C. Bowman, Malcolm Roberts, and Noel Murasko. `FFTW++`: *A fast Fourier transform* C^{++} *header class for the* `FFTW3` *library*. `http://fftwpp.sourceforge.net`. 2023.

[Can+06] C. Canuto et al. *Spectral Methods: Fundamentals in Single Domains*. Springer, 2006.

[Chi+20] Kamran Chitsaz et al. *Acceleration of Convolutional Neural Network Using FFT-Based Split Convolutions*. 2020. arXiv: `2003.12621 [cs.CV]`.

[CT65] James W. Cooley and John W. Tukey. “An Algorithm for the Machine Calculation of Complex Fourier Series”. In: *Mathematics of Computation* 19.90 (Apr. 1965), pp. 297–301.

[Dud73] Richard O. Duda. *Pattern classification and scene analysis.* A Wiley-interscience publication. Wiley, 1973.

[FJ] Matteo Frigo and Steven G. Johnson. `FFTW`. http://www.fftw.org.

[FJ05] Matteo Frigo and Steven G. Johnson. "The design and implementation of `FFTW3`". In: *Proceedings of the IEEE* 93.2 (2005), pp. 216–231.

[Gau66] Carl Friedrich Gauss. "Nachlass: Theoria Interpolationis Methodo Nova Tractata". In: *Carl Friedrich Gauss Werke*. Vol. 3. Göttingen: Königliche Gesellschaft der Wissenschaften, 1866, pp. 265–327.

[GO77] David Gottlieb and Steven A. Orszag. *Numerical Analysis of Spectral Methods: Theory and Applications*. CBMS-NSF Regional Conference Series in Applied Mathematics. Society for Industrial and Applied Mathematics, 1977.

[HJB85] Michael T. Heideman, Don H. Johnson, and C. Sidney Burrus. "Gauss and the history of the fast Fourier transform". In: *Archive for History of Exact Sciences* 34.3 (Sept. 1985), pp. 265–277.

[HR16] Tyler Highlander and Andres Rodriguez. *Very Efficient Training of Convolutional Neural Networks using Fast Fourier Transform and Overlap-and-Add*. 2016. arXiv: 1601.06815 [cs.NE].

[Lec+98] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[LY19] Jinhua Lin and Yu Yao. "A Fast Algorithm for Convolutional Neural Networks Using Tile-based Fast Fourier Transforms". In: *Neural Processing Letters* 50.2 (2019), pp. 1951–1967.

[MHL14] Michael Mathieu, Mikael Henaff, and Yann LeCun. *Fast Training of Convolutional Networks through FFTs*. 2014. arXiv: 1312.5851 [cs.CV].

[Ors71] Steven A. Orszag. “On the Elimination of Aliasing in Finite Difference Schemes by Filtering High-Wavenumber Components”. In: *Journal of the Atmospheric Sciences* (1971), p. 1074.

[OSB99] Alan V. Oppenheim, Ronald W. Schafer, and John R. Buck. *Discrete-time signal processing*. Prentice Hall, 1999.

[PO71] G. S. Patterson Jr. and Steven A. Orszag. “Spectral calculations of isotropic turbulence: Efficient removal of aliasing interactions”. In: *Physics of Fluids* 14 (1971), p. 2538.

[Pre+07] William H. Press et al. *Numerical recipes : the art of scientific computing*. Third. Cambridge University Press, 2007.

[RB18] Malcolm Roberts and John C. Bowman. “Multithreaded implicitly dealiased convolutions”. In: *J. Comput. Phys.* 356 (2018), pp. 98–114.

[Sob14] Irwin Sobel. “An Isotropic 3x3 Image Gradient Operator”. In: *Presentation at Stanford A.I. Project 1968* (Feb. 2014).

[Sor+87] Henrik V. Sorensen et al. “Real-valued fast Fourier transform algorithms”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 35.6 (1987), pp. 849–863.

Appendix A

Complex packing multiplication

Let $\boldsymbol{f}$ and $\boldsymbol{g}$ be real arrays of length $N = 2\tilde{N}$ and let $\tilde{\boldsymbol{f}}$ and $\tilde{\boldsymbol{g}}$ be their corresponding complex packings (defined in (8.1)). We would like to compute $\boldsymbol{H} \doteq \boldsymbol{F} \odot \boldsymbol{G}$ in terms of $\tilde{\boldsymbol{F}}$ and $\tilde{\boldsymbol{G}}$. In other words, we would like an operator $\boxdot$ satisfying

$$\tilde{\boldsymbol{H}} = \tilde{\boldsymbol{F}} \boxdot \tilde{\boldsymbol{G}}$$

We take $\tilde{\boldsymbol{F}}$ and $\tilde{\boldsymbol{G}}$ to be periodic in $\tilde{N}$.[1] Unpacking in Fourier space with (8.2) gives us

$$\begin{aligned}(\boldsymbol{F} \odot \boldsymbol{G})_k &= \frac{1}{4}\left[\left(\tilde{F}_k + \overline{\tilde{F}_{\tilde{N}-k}}\right) - i\left(\tilde{F}_k - \overline{\tilde{F}_{\tilde{N}-k}}\right)\zeta_{2\tilde{N}}^k\right]\left[\left(\tilde{G}_k + \overline{\tilde{G}_{\tilde{N}-k}}\right) - i\left(\tilde{G}_k - \overline{\tilde{G}_{\tilde{N}-k}}\right)\zeta_{2\tilde{N}}^k\right] \\ &= \frac{1}{4}\left[\tilde{F}_k\tilde{G}_k + \overline{\tilde{F}_{\tilde{N}-k}}\tilde{G}_k + \tilde{F}_k\overline{\tilde{G}_{\tilde{N}-k}} + \overline{\tilde{F}_{\tilde{N}-k}\tilde{G}_{\tilde{N}-k}} - 2i\left(\tilde{F}_k\tilde{G}_k - \overline{\tilde{F}_{\tilde{N}-k}\tilde{G}_{\tilde{N}-k}}\right)\zeta_{2\tilde{N}}^k - \right. \\ &\qquad \left.\left(\tilde{F}_k\tilde{G}_k - \overline{\tilde{F}_{\tilde{N}-k}}\tilde{G}_k - \tilde{F}_k\overline{\tilde{G}_{\tilde{N}-k}} + \overline{\tilde{F}_{\tilde{N}-k}\tilde{G}_{\tilde{N}-k}}\right)\zeta_{\tilde{N}}^k\right].\end{aligned}$$

[1] We do this so that we do not have the indices $\bmod \tilde{N}$. In practice, this is only used to identify $\tilde{F}_{\tilde{N}}$ with $\tilde{F}_0$ and $\tilde{G}_{\tilde{N}}$ with $\tilde{G}_0$.

Now for $k = 0, \dots, \tilde{N}$, define

$$\begin{aligned}
A_k &\doteq \tilde{F}_k \tilde{G}_k + \overline{\tilde{F}_{\tilde{N}-k}} \tilde{G}_k + \tilde{F}_k \overline{\tilde{G}_{\tilde{N}-k}} + \overline{\tilde{F}_{\tilde{N}-k} \tilde{G}_{\tilde{N}-k}}, \\
B_k &\doteq \tilde{F}_k \tilde{G}_k - \overline{\tilde{F}_{\tilde{N}-k} \tilde{G}_{\tilde{N}-k}}, \\
C_k &\doteq \tilde{F}_k \tilde{G}_k - \overline{\tilde{F}_{\tilde{N}-k}} \tilde{G}_k - \tilde{F}_k \overline{\tilde{G}_{\tilde{N}-k}} + \overline{\tilde{F}_{\tilde{N}-k} \tilde{G}_{\tilde{N}-k}}.
\end{aligned}$$

So we have

$$(\boldsymbol{F} \odot \boldsymbol{G})_k = \frac{1}{4} \left(A_k - 2iB_k \zeta_{2\tilde{N}}^k - C_k \zeta_{\tilde{N}}^k \right). \tag{A.1}$$

Note that we have the following symmetries:

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

as well as $\zeta_{2\tilde{N}}^{\pm \tilde{N}} = \zeta_2^{\pm 1} = -1$. Using these symmetries we compute

$$\overline{(\boldsymbol{F} \odot \boldsymbol{G})_{\tilde{N}-k}} = \frac{1}{4} \left(A_k + 2iB_k \zeta_{2\tilde{N}}^k - C_k \zeta_{\tilde{N}}^k \right) \tag{A.2}$$

By (A.1) and (A.2), we have

$$(\boldsymbol{F} \odot \boldsymbol{G})_k + \overline{(\boldsymbol{F} \odot \boldsymbol{G})_{\tilde{N}-k}} = \frac{1}{2} \left(A_k - C_k \zeta_{\tilde{N}}^k \right), \tag{A.3}$$

and

$$(\boldsymbol{F} \odot \boldsymbol{G})_k - \overline{(\boldsymbol{F} \odot \boldsymbol{G})_{\tilde{N}-k}} = -iB_k \zeta_{2\tilde{N}}^k. \tag{A.4}$$

Repacking using (8.3) along with (A.3) and (A.4), gives us

$$\begin{aligned}
(\tilde{\boldsymbol{F}} \boxdot \tilde{\boldsymbol{G}})_k &= \frac{1}{2} \left((\boldsymbol{F} \odot \boldsymbol{G})_k + \overline{(\boldsymbol{F} \odot \boldsymbol{G})_{\tilde{N}-k}} \right) + \frac{i}{2} \left((\boldsymbol{F} \odot \boldsymbol{G})_k - \overline{(\boldsymbol{F} \odot \boldsymbol{G})_{\tilde{N}-k}} \right) \zeta_{2\tilde{N}}^{-k} \\
&= \frac{1}{4} \left(A_k - C_k \zeta_{\tilde{N}}^k \right) + \frac{1}{2} \left(B_k \zeta_{2\tilde{N}}^k \right) \zeta_{2\tilde{N}}^{-k} = \frac{1}{4} A_k - \frac{1}{4} C_k \zeta_{\tilde{N}}^k + \frac{1}{2} B_k.
\end{aligned}$$

Substituting the definitions of A_k, B_k and C_k back into this expression and simplifying gives

$$\begin{aligned}
(\tilde{\boldsymbol{F}} \boxdot \tilde{\boldsymbol{G}})_k &= \frac{1}{4}\left(\tilde{F}_k\tilde{G}_k + \overline{\tilde{F}_{\tilde{N}-k}}\tilde{G}_k + \tilde{F}_k\overline{\tilde{G}_{\tilde{N}-k}} + \overline{\tilde{F}_{\tilde{N}-k}\tilde{G}_{\tilde{N}-k}}\right) - \\
&\frac{1}{4}\left(\tilde{F}_k\tilde{G}_k - \overline{\tilde{F}_{\tilde{N}-k}}\tilde{G}_k - \tilde{F}_k\overline{\tilde{G}_{\tilde{N}-k}} + \overline{\tilde{F}_{\tilde{N}-k}\tilde{G}_{\tilde{N}-k}}\right)\zeta_{\tilde{N}}^{-k} + \frac{1}{2}\left(\tilde{F}_k\tilde{G}_k - \overline{\tilde{F}_{\tilde{N}-k}\tilde{G}_{\tilde{N}-k}}\right) \\
&= \tilde{F}_k\tilde{G}_k - \frac{1}{4}\left(\tilde{F}_k - \overline{\tilde{F}_{\tilde{N}-k}}\right)\left(\tilde{G}_k - \overline{\tilde{G}_{\tilde{N}-k}}\right)\left(1 + \zeta_{\tilde{N}}^{-k}\right),
\end{aligned}$$

which is the desired result. Thus, one can use complex packing to compute real convolutions without unpacking and packing in Fourier space.