Multithreaded Implicitly Dealiased Pseudospectral Convolutions

Malcolm Roberts and John C. Bowman

Department of Mathematical and Statistical Sciences,
The University of Alberta, Edmonton, AB, T6G 2G1, Canada

Email: *mroberts@math.ualberta.ca*

ABSTRACT

Convolutions are used in data and image analysis and form the crux of the pseudospectral method for direct numerical simulations of fluids. Their calculation is a computationally expensive task that is facilitated by the use of FFTs at the expense of increased memory, which is required for the removal of aliased terms. Here, we present a multithreaded version of the method of implicit dealiasing (Bowman and Roberts, SIAM J. Sci. Comput. 33, 2011). Implicit dealiasing requires less memory than conventional explicit zero padding without increasing computational complexity or communication cost.

1 INTRODUCTION

Discrete linear convolutions arise in correlation analysis, digital signal processing, and spectral simulations of nonlinear partial differential equations such as the Navier–Stokes equations. The convolution of two infinite-length vectors F and G is

$$(F * G)_k \doteq \sum_{\ell\in\mathbb{Z}} F_\ell G_{k-\ell}, \quad k \in \mathbb{Z}. \tag{1}$$

If only N contiguous elements of the input vectors are nonzero, calculating equation (1) directly requires $O(N^2)$ operations and results in significant numerical error. One can avoid these difficulties by making use of the convolution theorem, which states that the Fourier transform of a convolution is a componentwise multiplication. Since fast Fourier transforms require $O(N\log N)$ operations [5, 3] and the multiplication requires $O(N)$ operations, FFT-based convolutions can be performed much more quickly than computing the convolution as a direct sum. However, FFTs treat arrays as periodic, so the indices in equation (1) are considered mod N, introducing extra terms called *aliases* in the summation.

Zero padding dealiases FFT-based convolutions by concatenating zero data to the input vectors. For non-centered input data $\{F_k\}_{k=0}^{N-1}$ (as is used in correlation analysis and digital signal processing), the data must be padded from length N to length $2N$, which is referred to as "$1/2$ padding". In the case of spectral simulations of nonlinear differential equations, the input data $\{F_k\}_{k=-N+1}^{N-1}$ is centered, and the data is padded from length $2N-1$ to $3N$ [6], which is referred to as "$2/3$ padding".

We have developed implicitly padded FFT-based convolution routines that in n dimensions reduce the memory requirements by a factor of 2^{n-1} for non-centered convolutions and $(3/2)^{n-1}$ for centered convolutions, while increasing speed by a factor of approximately two in both cases [2]. These algorithms were originally developed for serial computation; here, we present a multithreaded version of the algorithm.

2 IMPLICITLY PADDED FFTS

Implicitly padded convolution routines are based on the one-dimensional implicitly padded Fourier transform. Suppose that the input vectors F and G are of length N, to which we would like to apply $1/2$ padding, i.e. we pad the vector with zeroes so that the total length of each input is $2N$. The inverse discrete Fourier transform (DFT) of this data is equal to

$$f_x = \sum_{k=0}^{2N-1} \zeta_{2N}^{xk} F_k = \sum_{k=0}^{N-1} \zeta_{2N}^{xk} F_k, \quad x = 0,\ldots,2N-1 \tag{2}$$

since $F_k = 0$ for $k \geq N$. Here, ζ_{2N} is the $2N^{\text{th}}$ root of unity. Notice that the right-hand side of equation (2) is not directly amenable to computation via FFT. How-

ever, if we instead consider

$$f_{2j} = \sum_{k=0}^{N-1} \zeta_{2N}^{2jk} F_k = \sum_{k=0}^{N-1} \zeta_N^{jk} F_k, \quad (3)$$

$$f_{2j+1} = \sum_{k=0}^{N-1} \zeta_{2N}^{(2j+1)k} F_k = \sum_{k=0}^{N-1} \zeta_N^{jk} (\zeta_{2N}^k F_k) \quad (4)$$

for $j = 0, \ldots, N-1$, then equation (3) can be computed by performing a FFT on $\{F_k\}_{k=0}^{N-1}$ and equation (4) can be computed by performing an FFT on $\{\zeta_{2N}^k F_k\}_{k=0}^{N-1}$. Inverting this transformation involves calculating

$$2NF_k = \sum_{x=0}^{2N-1} \zeta_{2N}^{-kx} f_x = \sum_{j=0}^{N-1} \zeta_N^{-kj} f_{2j} + \zeta_{2N}^{-k} \sum_{j=0}^{N-1} \zeta_N^{-kj} f_{2j+1} \quad (5)$$

for $k = 0, \ldots, N-1$.

Implicitly padded FFTs are performed on $2/3$ padded data in an analogous fashion, with the output consisting of three vectors each of length N. Taking advantage of Hermitian symmetry, i.e. that $F_{-k} = F_k^*$, with $*$ denoting complex conjugation, the inverse implicitly $2/3$ padded transform is

$$f_{3\ell+r} = \sum_{k=-m+1}^{m-1} \zeta_m^{\ell k} \zeta_{3m}^{rk} F_k = \sum_{k=0}^{m-1} \zeta_m^{\ell k} w_{k,r}, \quad (6)$$

where $\ell = 0, \ldots, N-1$, $r \in \{-1, 0, 1\}$, and

$$w_{k,r} \doteq \begin{cases} F_0 & \text{if } k = 0, \\ \zeta_{3m}^{rk} (F_k + \zeta_3^{-r} F_{k-m}) & \text{if } 1 \le k \le m-1. \end{cases} \quad (7)$$

The forwards transform is accomplished in a fashion analogous to equation (5).

The ternary convolution of three input vectors F, G, and H is denoted $*(F,G,H)$ and given by

$$*(F,G,H)_k \doteq \sum_{a,b,c} F_{k_1} G_{k_2} H_{k_3} \delta_{k,k_1+k_2+k_3} \quad (8)$$

where δ is the Kronecker delta function. For non-centered data, one can use the fact that $*(F,G,H) = F*(G*H)$, but this does not hold for centered data [7], in which case one pads each vector from length $2N-1$ to length $4N$ and performs the computation all at once. Ternary convolutions can arise in pseudospectral simulations of compressible fluids or when considering the evolution of higher-order moments of incompressible turbulence [1].

In this paper, we consider complex non-centered convolutions in one, two, and three dimensions; centered Hermitian-symmetric convolutions in one, two, and three dimensions; and centered Hermitian symmetric ternary convolutions in one and two dimensions.

3 Multithreaded 1D convolutions

Parallelizing one-dimensional FFT-based convolutions is typically advantageous only for problems involving more than a few thousands data points. A comparison of parallel and serial timings for implicit one-dimensional complex non-centered convolutions is shown in Figure 1. Error bars indicate one-sided standard deviations. The multi-threaded version is faster starting at problems of size $N = 2048$ with 4 cores, yielding an asymptotic speed-up of a factor of approximately 3 (only one thread is used for $N < 2048$). The execution times for threaded implementations of implicit and explicit convolutions are given in Figure 2. In one dimension, the explicit and implicit algorithms are observed to have similar speeds.

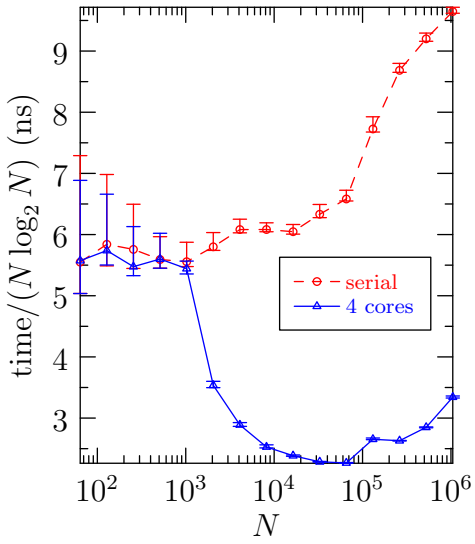

Figure 1: Computation times for non-centered complex 1D implicitly-padded convolutions on data of length N using one core vs. up to four cores.

Timing results for Hermitian centered one-dimensional implicit convolutions are shown in Figure 3 and compared with multithreaded explicitly padded convolutions in Figure 4. The multithreaded implicitly padded algorithm is faster than the serial implicitly padded algorithm for $N \ge 2048$ with four cores. The mutithreaded versions of the implicit and explicit methods perform similarly. The maximum speedup with four cores is only around a factor of two in this case.

Timing results for a ternary version of the Hermitian centered one-dimensional convolutions shown in Figure 5 and compared with an explicit method in Fig-

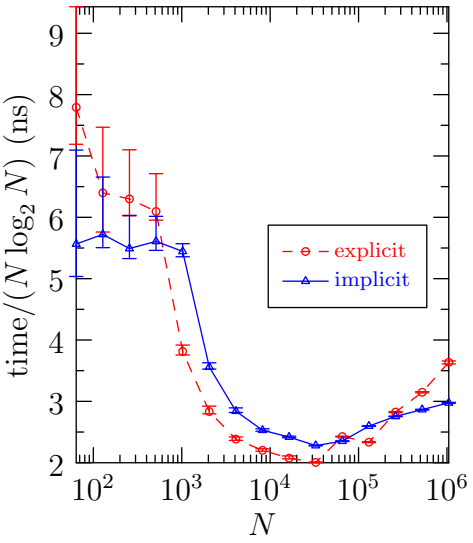

Figure 2: Computation times for explicit and implicit non-centered complex 1D convolutions of size N using up to four cores.

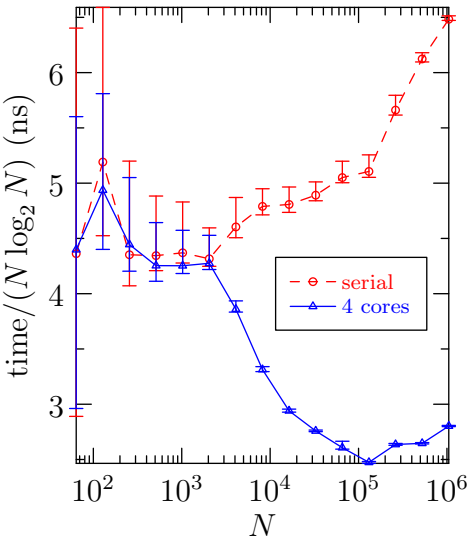

Figure 3: Computation times for centered Hermitian 1D implicitly padded convolutions on data of length N using one core vs. up to four cores.

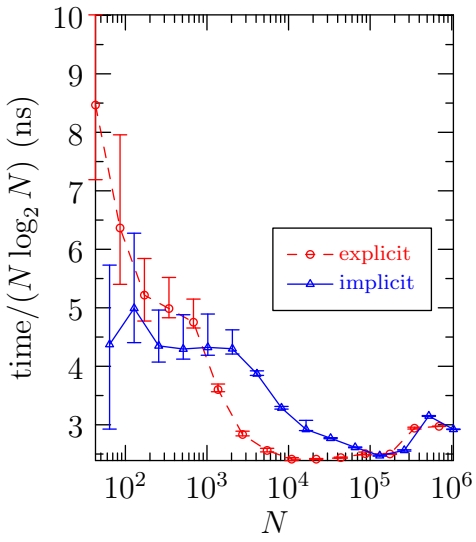

Figure 4: Computation times for explicit and implicit centered Hermitian 1D convolutions of size N using up to four cores.

ure 6. The version with four cores is faster than the serial version for $N \geq 2048$. The maximum speedup is around a factor of two when using four cores. The implicit and explicit methods exhibit similar performance with multiple cores.

4 Multithreaded 2D Convolutions

Implicitly dealiased convolutions are performed on two-dimensional data by first performing an implicitly padded FFT in the x direction, performing implicitly padded one-dimensional convolutions in the y direction, and then inverting the FFT in the x direction.

Since the zero-padding must be done in all dimensions, some of the x transforms in the explicit method can be skipped as they are performed on data that is known a priori to be zero. This optimization, known as *pruning*, reduces the computational complexity of the problem but does not reduce the memory requirements. Implicitly dealiased convolutions automatically skip FFTs on zero-data while also reducing memory requirements.

For non-centered data with an input array of size $N_x \times N_y$, the x FFT produces output in two arrays of size $N_x \times N_y$, each which can be done in parallel using a standard parallelized FFT. Then, each thread uses its own copy of a single-threaded y convolution, each of which requires two work arrays of size N_x. If there are P threads in total, then the total work memory used

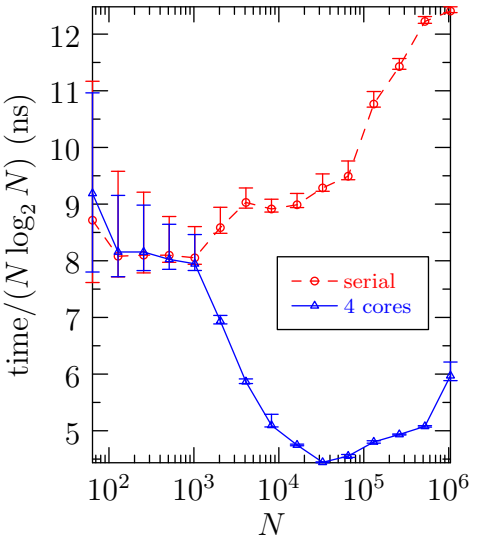

Figure 5: Computation times for centered Hermitian 1D implicitly padded ternary convolutions on data of size N using one core vs. up to four cores.

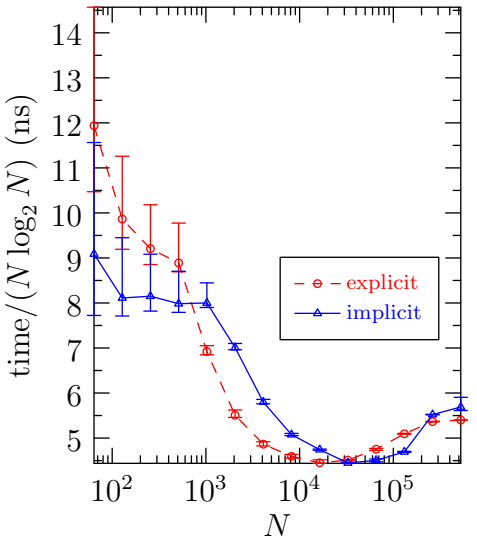

Figure 6: Computation times between explicit and implicit centered Hermitian 1D ternary convolutions of size N using up to four cores.

is $2(N_x+P)N_y$, whereas the explicit method requires $6N_xN_y$ words of extra memory. The timing results for this algorithm are shown in Figure 7. A comparison with the explicit method is shown in Figure 8.

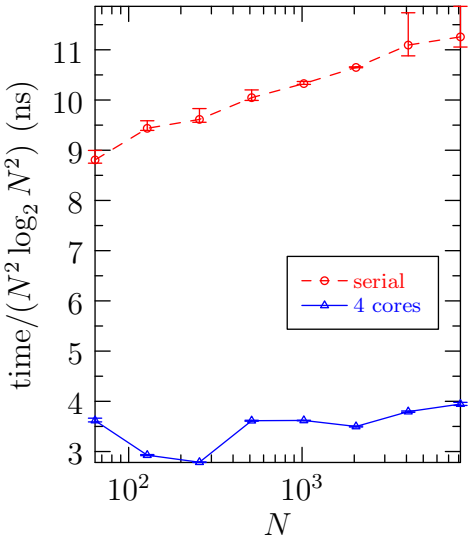

Figure 7: Computation times for non-centered implicitly padded complex 2D convolutions on data of size $N \times N$ using one core vs. four cores.

Centered data with Hermitian symmetry is stored in an array of dimensions $(2N_x-1)\times N_y$. The implicitly padded transform in the x direction requires an additional work array of size $(N_x+1)\times N_y$. For simplicity we describe only the case where N_y is even. The $3N_x$ convolutions in the y direction are divided between threads, with each thread requiring two work arrays of size $N_y/2+1$. Thus, the implicit method requires work arrays of size $2(N_x+1)N_y+P(N_y+2)$, whereas the explicit method requires $(5N_x-4)N_y$ words of extra memory. The results of our timing tests are shown in Figure 9, and a comparison with the explicit version is shown in Figure 10.

Ternary centered Hermitian convolutions operate on data of size $(2N_x-1)N_y$. The total work memory for the implicit convolution is then $3(2N_xN_y+4N_x+N_y-1)+P(3N_y+6)$, whereas the explicit method uses $18N_xN_y+15N_y$. The results of our timing tests are shown in Figure 11. A comparison with the explicitly padded version is shown in Figure 12.

The multithreaded implicit algorithm is significantly faster for all tested two-dimensional problem sizes when using four cores and produces a speedup factor between 3 and 3.5. As for the serial case, the implicit version is faster than explicit methods.

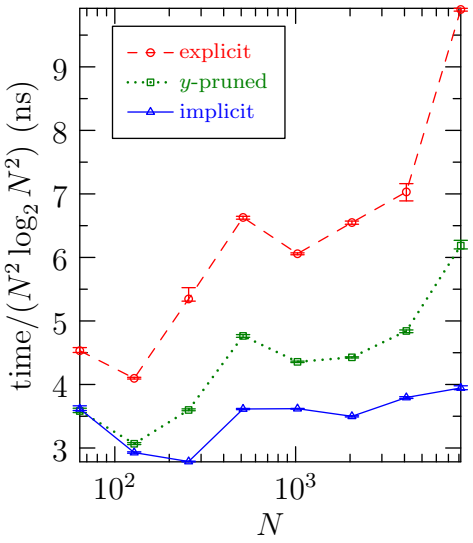

Figure 8: Computation times for explicit and implicit non-centered complex 2D convolution size $(2N-1)\times N$ using four cores.

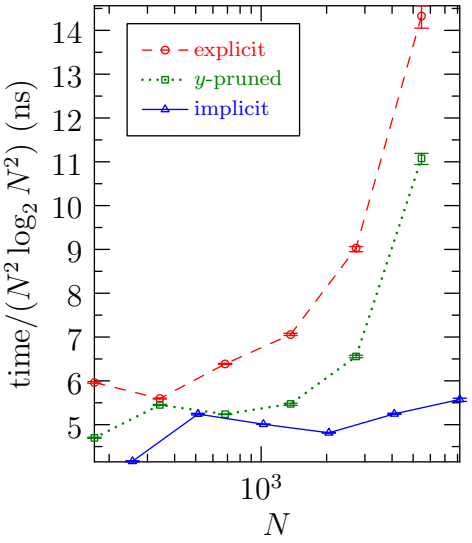

Figure 10: Computation times for explicit and implicit centered Hermitian 2D convolution of size $(2N-1)\times N$ using four cores.

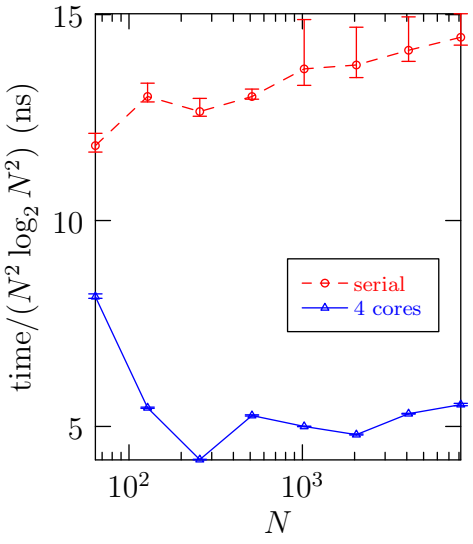

Figure 9: Computation times for centered Hermitian 2D implicitly padded convolutions on data of size $(2N-1)\times N$ using one core vs. four cores.

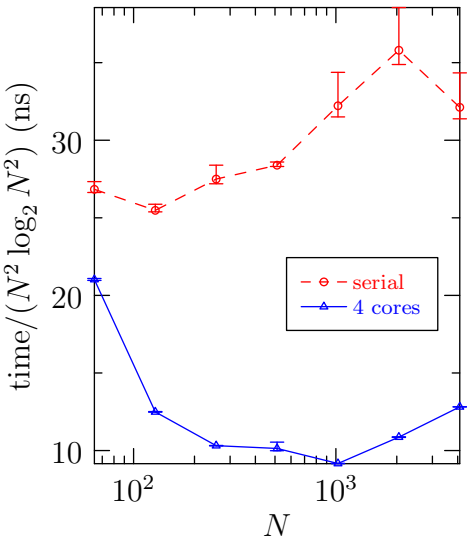

Figure 11: Timing results for centered Hermitian 2D ternary convolutions on data of size $(2N-1)\times N$ using one core vs. four cores.

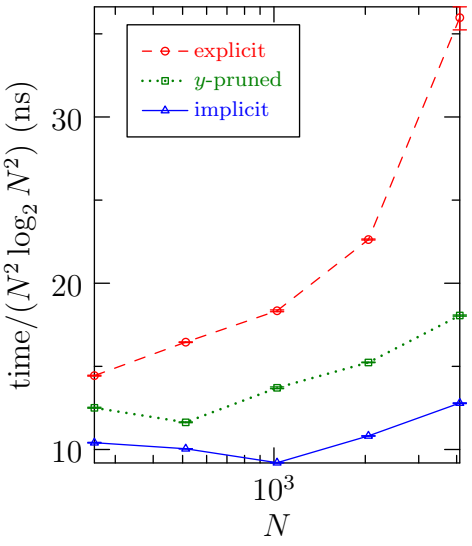

Figure 12: Computation times for explicit and implicit centered Hermitian 2D ternary convolutions of size $(2N-1) \times N$ using four cores.

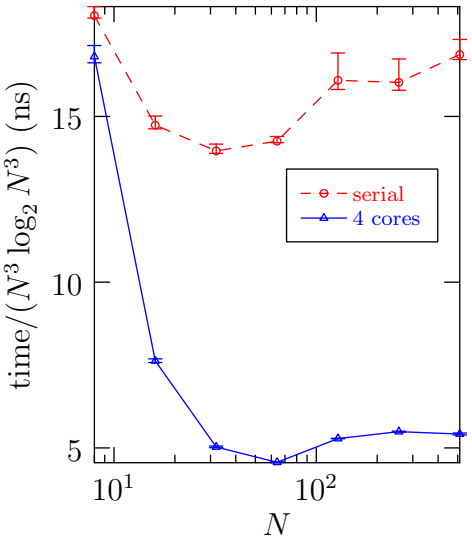

Figure 13: Computation times for non-centered complex 3D implicitly padded convolutions on data of size $N \times N \times N$ using one core vs. four cores.

5 Multithreaded 3D convolutions

Three-dimensional convolutions are calculated by Fourier transforming in the x direction, performing two-dimensional convolutions in the yz plane, and then transforming back in the x direction. This idea can be extended to general n-dimensional convolutions in a straightforward fashion by Fourier transforming in one dimension, performing several $(n-1)$-dimensional convolutions, and then inverting the initial Fourier transform.

Three-dimensional non-centered convolutions have input data of size $N_x \times N_y \times N_z$. The x direction implicitly padded Fourier transform uses the input memory and an additional work array of the same size, whereas sub-convolutions each require a work array of size $N_y N_z$. The total work memory required is then $2N_xN_yN_z + 2P(N_y+1)N_z$ for the implicit method, compared with $14N_xN_yN_z$ for the explicit method. Timing results are shown in Figure 13. A comparison with the explicit method is shown in Figure 14.

We also implemented a three-dimensional non-centered Hermitian binary convolution, which requires a work array of size $4N_xN_yN_z - 2N_xN_y - 2N_xN_z + 6N_yN_z - 3N_y - N_z + P(2N_xN_y + 3N_y + 2)$, while the explicit method requires $(19N_xN_y + 4N_x + 4N_y - 2)N_z$ extra words. Timing results are shown in Figure 15.

As with the two-dimensional case, multiple threads

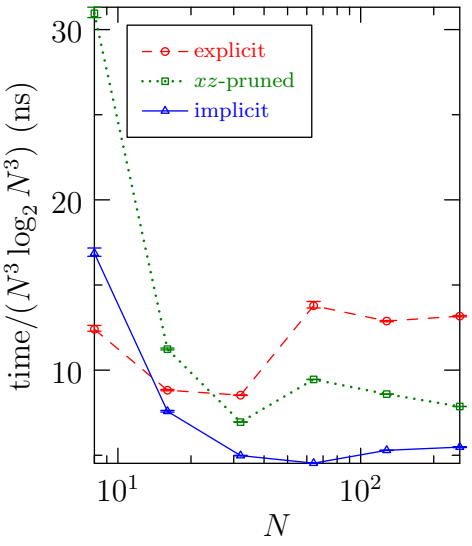

Figure 14: Computation times for explicit and implicit non-centered complex 3D convolution size $N \times N \times N$ using four cores.

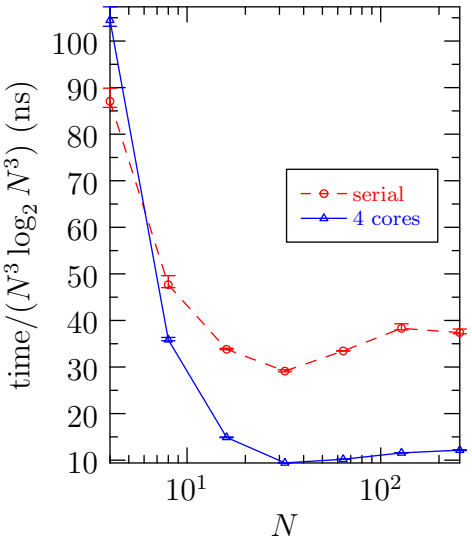

Figure 15: Timing results for centered Hermitian 3D implicitly padded convolutions on data of size $(2N-1)\times(2N-1)\times N$ using one core vs. four cores.

and implicit padding offer significant performance benefits in terms of both processing time and memory requirements. Using an implicit method with four cores led to an increase in speed by a factor between 3 and 3.6. The implicit method was significantly faster than the explicit method, even when transforms on zero data were omitted.

6 Implementation and Use

The convolution routines described above are available as open-source libraries under the LGPL in the project `FFTW++`[4], at `fftwpp.sourceforge.net`. The code is written it `C++` and makes use of `SIMD` extensions for vectorization and `OpenMP` for multithreading. Fourier transforms were performed using FFTW.

In addition to offering high-performance convolution routines, `FFTW++` offers wrappers for FFTW, taking care of much of the house-keeping automatically. These implicit convolutions are also designed to be easy to use and are illustrated with online examples.

7 Future Work

In this paper, we have demonstrated the benefit of using multiple cores to compute convolutions using the method of implicit dealiasing. One of the most common questions we have received when discussing implicit padding is whether the method is adaptable to distributed-memory architectures. While this work is on-going, the theoretical groundwork has been laid, and we expect that the method of implicit dealiasing will offer significant improvements in grid computing environments in terms of memory required, computational complexity, and communication costs.

8 Conclusion

The method of implicit padding is an efficient method for computing linear convolutions. In one dimension, it performs similarly to conventional methods. However, in multiple dimensions, it requires significantly less memory and is approximately twice as fast as conventional methods. It is also amenable to parallel computation on shared-memory architectures. While implicit dealiasing is more complex to implement, the availability of high-performance open-source libraries makes it easy for programmers to use these algorithms. We therefore expect implicit dealiasing to become a standard method for performing convolutions.

References

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

[2] J. C. Bowman and M. Roberts. Efficient dealiased convolutions without padding. *SIAM J. Sci. Comput.*, 33(1):386–406, 2011.

[3] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19(90):297–301, April 1965.

[4] M. Frigo and S. G. Johnson. The design and implementation of `FFTW3`. *Proceedings of the IEEE*, 93(2):216–231, 2005.

[5] C. F. Gauss. Nachlass: Theoria interpolationis methodo nova tractata. In *Carl Friedrich Gauss Werke*, volume 3, pages 265–327. Königliche Gesellschaft der Wissenschaften, Göttingen, 1866.

[6] S. A. Orszag. Elimination of aliasing in finite-difference schemes by filtering high-wavenumber components. *Journal of the Atmospheric Sciences*, 28:1074, 1971.

[7] M. Roberts and J. C. Bowman. Dealiased convolutions for pseudospectral simulations. *Journal of Physics: Conference Series*, 318(7):072037, 2011.