Implicitly Padded Convolutions on Hybrid Parallel Architectures

John C. Bowman
University of Alberta

June 8, 2015

www.math.ualberta.ca/~bowman/talks

Acknowledgements: Malcolm Roberts (Université de Strasbourg)
Discrete Cyclic Convolution

• The FFT provides an efficient tool for computing the *discrete cyclic convolution*

\[
\sum_{p=0}^{N-1} F_p G_{k-p},
\]

where the vectors \( F \) and \( G \) have period \( N \).

• Define the *\( N \)th primitive root of unity*:

\[
\zeta_N = \exp \left( \frac{2\pi i}{N} \right).
\]

• The fast Fourier transform method exploits the properties that

\[
\zeta_N^r = \zeta_{N/r} \quad \text{and} \quad \zeta_N^N = 1.
\]

• However, the pseudospectral method requires a *linear convolution*. 
The unnormalized *backwards discrete Fourier transform* of \( \{F_k : k = 0, \ldots, N\} \) is

\[
f_j = \sum_{k=0}^{N-1} \zeta_{N}^{jk} F_k \quad j = 0, \ldots, N - 1.
\]

The corresponding *forward transform is*

\[
F_k = \frac{1}{N} \sum_{j=0}^{N-1} \zeta_{N}^{-kj} f_j \quad k = 0, \ldots, N - 1.
\]

The orthogonality of this transform pair follows from

\[
\sum_{j=0}^{N-1} \zeta_{N}^{\ell j} = \begin{cases} 
N & \text{if } \ell = sN \text{ for } s \in \mathbb{Z}, \\
\frac{1}{1 - \zeta_{N}^{\ell}} & \text{if otherwise}.
\end{cases}
\]
Convolution Theorem

\[
\sum_{j=0}^{N-1} f_j g_j \zeta_N^{-jk} = \sum_{j=0}^{N-1} \zeta_N^{-jk} \left( \sum_{p=0}^{N-1} \zeta_N^{jp} F_p \right) \left( \sum_{q=0}^{N-1} \zeta_N^{jq} G_q \right) \\
= \sum_{p=0}^{N-1} \sum_{q=0}^{N-1} F_p G_q \sum_{j=0}^{N-1} \zeta_N^{(-k+p+q)j} \\
= N \sum_{s} \sum_{p=0}^{N-1} F_p G_{k-p+sN}.
\]

- The terms indexed by \( s \neq 0 \) are *aliases*; we need to remove them!

- If only the first \( m \) entries of the input vectors are nonzero, aliases can be avoided by *zero padding* input data vectors of length \( m \) to length \( N \geq 2m - 1 \).

- *Explicit zero padding* prevents mode \( m - 1 \) from beating with itself, wrapping around to contaminate mode \( N = 0 \mod N \).
Since FFT sizes with small prime factors in practice yield the most efficient implementations, the padding is normally extended to $N = 2^m$:

$$\{F_k\}_{k=0}^{m-1} \quad \{G_k\}_{k=0}^{m-1}$$
Since FFT sizes with small prime factors in practice yield the most efficient implementations, the padding is normally extended to $N = 2^m$:

\[
\{F_k\}_{k=0}^{m-1} \quad \{G_k\}_{k=0}^{m-1}
\]

\[
\{0\}_{k=0}^{m-1}
\]
• Since FFT sizes with small prime factors in practice yield the most efficient implementations, the padding is normally extended to $N = 2^m$:

\[
\begin{align*}
\{F_k\}_{k=0}^{m-1} & \quad \{0\}_{k=0}^{m-1} \\
\{F_k\}_{k=0}^{m-1} & \quad \{0\}_{k=0}^{m-1} \\
\{G_k\}_{k=0}^{m-1} & \quad \{0\}_{k=0}^{m-1} \\
\{G_k\}_{k=0}^{m-1} & \quad \{0\}_{k=0}^{m-1} \\
\{f_j\}_{j=0}^{2^{m-1}} & \quad \{g_j\}_{j=0}^{2^{m-1}}
\end{align*}
\]
Since FFT sizes with small prime factors in practice yield the most efficient implementations, the padding is normally extended to \( N = 2^m \):

\[
\begin{align*}
\{F_k\}_{k=0}^{m-1} & \quad \{G_k\}_{k=0}^{m-1} \\
\{F_k\}_{k=0}^{m-1} & \quad \{0\}_{k=0}^{m-1} \\
\{G_k\}_{k=0}^{m-1} & \quad \{0\}_{k=0}^{m-1}
\end{align*}
\]
• Since FFT sizes with small prime factors in practice yield the most efficient implementations, the padding is normally extended to \( N = 2^m \):
Since FFT sizes with small prime factors in practice yield the most efficient implementations, the padding is normally extended to $N = 2^m$: 

\[ \{F_k\}_{k=0}^{m-1} \]

\[ \{0\}_{k=0}^{m-1} \]

\[ \{G_k\}_{k=0}^{m-1} \]

\[ \{0\}_{k=0}^{m-1} \]

\[ \{f_j\}_{j=0}^{2^m-1} \]

\[ \{g_j\}_{j=0}^{2^m-1} \]

\[ \{f_jg_j\}_{j=0}^{2^m-1} \]

\[ \{F \ast G\}_{k=0}^{m-1} \]

\[ F \ast G \]
Implicit Padding

• Let $N = 2m$. For $j = 0, \ldots, 2m - 1$ we want to compute

$$f_j = \sum_{k=0}^{2m-1} \zeta_{2m}^{jk} F_k.$$ 

• If $F_k = 0$ for $k \geq m$, one can easily avoid looping over the unwanted zero Fourier modes by decimating in wavenumber:

$$f_{2\ell} = \sum_{k=0}^{m-1} \zeta_{2m}^{2\ell k} F_k = \sum_{k=0}^{m-1} \zeta_m^{\ell k} F_k,$$

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

• This requires computing two subtransforms, each of size $m$, for an overall computational scaling of order $2m \log_2 m = N \log_2 m$. 
Odd and even terms of the convolution can then be computed separately, multiplied term-by-term, and transformed again to Fourier space:

\[ 2mF_k = 2m^{m-1} \sum_{j=0}^{2m-1} \zeta_{2m}^{-kj} f_j \]

\[ = \sum_{\ell=0}^{m-1} \zeta_{2m}^{-k(2\ell+1)} f_{2\ell+1} + \sum_{\ell=0}^{m-1} \zeta_{2m}^{-k(2\ell+1)} f_{2\ell+1} \]

\[ = \sum_{\ell=0}^{m-1} \zeta_{m}^{-k\ell} f_{2\ell} + \zeta_{2m}^{-k} \sum_{\ell=0}^{m-1} \zeta_{m}^{-k\ell} f_{2\ell+1} \quad k = 0, \ldots, m - 1. \]

No bit reversal is required at the highest level.

A 1D implicitly padded convolution is implemented in our FFTW++ library.

This in-place convolution was written to use six out-of-place transforms, thereby avoiding bit reversal at all levels.
- The computational complexity is $6Km \log_2 m$.
- The numerical error is similar to explicit padding and the memory usage is identical.

\[ \{F_k\}_{k=0}^{m-1} \quad \{G_k\}_{k=0}^{m-1} \]
• The computational complexity is $6Km \log_2 m$.

• The numerical error is similar to explicit padding and the memory usage is identical.
• The computational complexity is $6Km \log_2 m$.

• The numerical error is similar to explicit padding and the memory usage is identical.
- The computational complexity is $6Km \log_2 m$.
- The numerical error is similar to explicit padding and the memory usage is identical.
Input: vector $f$, vector $g$
Output: vector $f$

$u \leftarrow \text{fft}^{-1}(f);$  
$v \leftarrow \text{fft}^{-1}(g);$  
$u \leftarrow u \ast v;$  
for $k = 0$ to $m - 1$ do  
| $f[k] \leftarrow \zeta_{2m}^k f[k];$  
| $g[k] \leftarrow \zeta_{2m}^k g[k];$  
end  
$v \leftarrow \text{fft}^{-1}(f);$  
$f \leftarrow \text{fft}^{-1}(g);$  
$v \leftarrow v \ast f;$  
$f \leftarrow \text{fft}(u);$  
$u \leftarrow \text{fft}(v);$  
for $k = 0$ to $m - 1$ do  
| $f[k] \leftarrow f[k] + \zeta_{2m}^{-k} u[k];$  
end  
return $f/(2m);$
Implicit Padding in 1D

\[ \text{time}/(N \log_2 N) \text{ (ns)} \]

- \(\circ\) explicit
- \(\triangle\) implicit

\(N\): 
- \(10^2\)
- \(10^3\)
- \(10^4\)
- \(10^5\)
- \(10^6\)
Convolutions in Higher Dimensions

- An explicitly padded convolution in 2 dimensions requires 12 padded FFTs, and 4 times the memory of a cyclic convolution.
Convolutions in Higher Dimensions

• An explicitly padded convolution in 2 dimensions requires 12 padded FFTs, and 4 times the memory of a cyclic convolution.
Convolutions in Higher Dimensions

• An explicitly padded convolution in 2 dimensions requires 12 padded FFTs, and 4 times the memory of a cyclic convolution.
Convolutions in Higher Dimensions

- An explicitly padded convolution in 2 dimensions requires 12 padded FFTs, and 4 times the memory of a cyclic convolution.
Convolutions in Higher Dimensions

- An explicitly padded convolution in 2 dimensions requires 12 padded FFTs, and 4 times the memory of a cyclic convolution.
Convolutions in Higher Dimensions

- An explicitly padded convolution in 2 dimensions requires 12 padded FFTs, and 4 times the memory of a cyclic convolution.
Recursive Convolution

- Naive way to compute a multiple-dimensional convolution:

\[ \mathcal{F}_{N_1, \ldots, N_d} \rightarrow \text{multiply} \rightarrow \mathcal{F}^{-1}_{N_1, \ldots, N_d} \]

- The technique of *recursive convolution* allows one to avoid computing and storing the entire Fourier image of the data:

\[ \mathcal{F}_{N_d} \rightarrow N_d \times \text{convolve}_{N_1, \ldots, N_{d-1}} \rightarrow \mathcal{F}^{-1}_{N_d} \]
Implicit Padding in 2D

- Extra work memory need not be contiguous with the data.
Implicit Padding in 2D

- Extra work memory need not be contiguous with the data.
Implicit Padding in 2D

- Extra work memory need not be contiguous with the data.
Implicit Padding in 2D

• Extra work memory need not be contiguous with the data.
Implicit Padding in 2D

- Extra work memory need not be contiguous with the data.
Implicit Padding in 2D

- Extra work memory need not be contiguous with the data.

\[
\begin{align*}
\text{FFT}^{-1}\{F \ast G\} \\
&\quad \text{if } n_z \text{ even} \\
\text{FFT}^{-1}\{F \ast G\} \\
&\quad \text{if } n_z \text{ odd}
\end{align*}
\]
Implicit Padding in 2D

• Extra work memory need not be contiguous with the data.
Implicit Padding in 2D

\[
\text{time}/(N^2 \log_2 N^2) \ (\text{ns})
\]

- explicit
- \(y\)-pruned
- implicit

\[
N \\quad 10^2 \quad 10^3
\]

\[
7.5 \quad 10 \quad 12.5 \quad 15
\]
Implicit Padding in 3D

time/$\left( m^3 \log_2 m^3 \right) \text{ (ns)}$

- **explicit**
- **xz-pruned**
- **implicit**

$m$ range: $10^1$ to $10^2$
Hermitian Convolutions

- **Hermitian convolutions** arise when the input vectors are Fourier transforms of real data:

\[ f_{N-k} = \overline{f_k}. \]
Centered Convolutions

• For a centered convolution, the Fourier origin \( k = 0 \) is centered in the domain:

\[
\sum_{p=k-m+1}^{m-1} f_p g_{k-p}
\]

• Need to pad to \( N \geq 3m-2 \) to prevent mode \( m-1 \) from beating with itself to contaminate the most negative (first) mode, at wavenumber \(-m+1\).

• The ratio of the number of physical to total modes, \( (2m-1)/(3m-2) \) is asymptotic to \( 2/3 \) for large \( m \).

• The Hermiticity condition then appears as

\[
f_{-k} = \overline{f_k}.
\]
Parallelization

• Our implicit and explicit convolution routines have been multithreaded for shared-memory architectures.

• Parallel generalized slab/pencil model implementations have recently been developed for distributed-memory architectures (available in svn repository and upcoming 1.14 release).

• The key bottleneck is the distributed matrix transpose.

• We have compared several distributed matrix transpose algorithms, both blocking and nonblocking, under both pure MPI and hybrid MPI/OpenMP architectures.

• Local transposition is not required within a single MPI node.

• Hybrid MPI/OpenMP offers a larger communication block size than pure MPI for matrix transposition.
• Hybrid MPI/OpenMP is sometimes more efficient (by a factor of 2) than pure MPI for computing distributed matrix transposes [Bowman & Roberts 2013].

• We have developed an adaptive algorithm, dynamically tuned to choose the optimal block size and number of threads.
$8 \times 8$ Block Transpose over 8 processors
8 × 8 Block Transpose over 8 processors
8 × 8 Block Transpose over 8 processors
8 × 8 Block Transpose over 8 processors
8 × 8 Block Transpose over 8 processors
$8 \times 8$ Block Transpose over 8 processors
$8 \times 8$ Block Transpose over 8 processors
Matrix Transpose: Optimal Number of Threads

![Graph showing the optimal number of threads for matrix transpose. The graph plots time (µs) against nodes × threads. Two lines are shown: FFTW: 1024² (dashed blue) and hybrid: 1024² (solid red). The graph reaches a minimum at 512 × 2 nodes × threads.]
Advantages of Hybrid MPI/OpenMP

• Smaller problems sizes to be distributed over a large number of processors;

• More slab-like than pencil-like model; this reduces the size of or even eliminates the need for the second transpose.

• Overlapping computation with communication can yield a 10% speedup for 3D implicitly dealiased convolutions, where a natural parallelism exists between communication and computation.
Pure MPI Scaling of 2D Implicit Convolutions

Strong scaling: cconv2

![Graph showing speedup vs. number of cores for various data point sizes.](image)
Pure MPI Scaling of 3D Implicit Convolution

Strong scaling: cconv3

Number of cores: 10

Speedup:
- 32³
- 64³
- 128³
- 256³
- 512³
- 1024³
- 2048³

Number of cores: 1k, 2k, 4k, 8k
Multithreaded Hermitian Convolution

• The backwards implicitly padded centered Hermitian transform appears as

\[ u_{3\ell+r} = \sum_{k=0}^{m-1} \zeta_{m}^{\ell k} w_{k,r}, \]

where

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

• We exploit the Hermitian symmetry \( w_{k,r} = \overline{w_{m-k,r}} \) to reduce the problem to three complex-to-real Fourier transforms of the first \( c + 1 \) components of \( w_{k,r} \) (one for each \( r = -1, 0, 1 \)), where \( c = \lfloor m/2 \rfloor \) zeros.
• To facilitate an in-place implementation, in our original paper (SIAM, 2011), we stored the transformed values for \( r = 1 \) in reverse order in the upper half of the input vector.

• However, loop dependencies in the resulting algorithm prevented the top level of the 1D transforms from being multithreaded.
Multithreaded Hermitian Convolution

- Unrolling the loop to process four inputs and outputs simultaneously allows loop independence to be achieved, significantly improving performance in both the serial and parallel contexts.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>⋮</th>
<th>$c - 2$</th>
<th>$c - 1$</th>
<th>$c$</th>
<th>$c + 1$</th>
<th>$c + 2$</th>
<th>⋮</th>
<th>$2c - 3$</th>
<th>$2c - 2$</th>
<th>$2c - 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>r=0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r=1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Multithreaded Hermitian Convolution

- Unrolling the loop to process four inputs and outputs simultaneously allows loop independence to be achieved, significantly improving performance in both the serial and parallel contexts.

- As a result, even in 1D, implicit dealiasing of pseudospectral convolutions is now significantly faster than explicit zero padding.
1D Implicit Hermitian Convolution

\[ \text{time} / (N \log_2 N) \quad (\text{ns}) \]

- **explicit**
- **implicit**

![Graph showing the comparison between explicit and implicit methods for 1D Implicit Hermitian Convolution. The x-axis represents the size of the input data \( N \), ranging from \( 10^2 \) to \( 10^6 \). The y-axis represents the time in nanoseconds, ranging from 4 to 6. The graph demonstrates the efficiency of the implicit method as \( N \) increases.]
2D Pseudospectral Collocation [1 thread]

time/\(m \log_2 m\)^2 (s)

Explicit

Implicit

\[10^{-6}\]

\[10^2\]

\[10^3\]

\(m\)
2D Pseudospectral Collocation [4 threads]

\[ \text{time} \left( \frac{m \log_2 m}{m} \right)^2 (s) \]

- **Explicit**
- **Implicit**

\[ 10^2 \quad 10^3 \]

\[ m \]
Conclusions

- Memory savings: in $d$ dimensions implicit padding asymptotically uses $1/2^{d-1}$ [for centered convolutions $(2/3)^{d-1}$] of the memory required by conventional explicit padding.

- The factor of 2 speedup with implicit dealiasing is largely due to increased data locality.

- Highly optimized and parallelized implicit dealiasing routines have been implemented as a software layer FFTW++ on top of the FFTW library and released under the Lesser GNU Public License: [http://fftwpp.sourceforge.net/](http://fftwpp.sourceforge.net/)

- Writing a high-performance dealiased pseudospectral code is now a relatively straightforward exercise!

- Implicit dealiasing has been extended to handle nested convolutions and autocorrelations.

- Implicit dealiasing can also be applied to signal denoising and image filtering.
References
