FFTW++: A Hybrid OpenMP/MPI Implementation of FFTs and Implicitly Dealiased Convolutions

John C. Bowman  
University of Alberta

Malcolm Roberts  
Advanced Micro Devices

Feb 14, 2020

www.math.ualberta.ca/~bowman/talks
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 \( \zeta_N^r = \zeta_{N/r} \) and \( \zeta_N^N = 1 \).

- However, the pseudospectral method requires a *linear convolution*. 
• The unnormalized \textit{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 \textit{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

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

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

\[ = \sum_{p=0}^{N-1} \sum_{q=0}^{N-1} F_p G_q \sum_{j=0}^{N-1} \zeta_N^{(-k+p+q)j} \]

\[ = N \sum_{s=0}^{N-1} \sum_{p=0}^{N-1} F_p G_{k-p+sN}. \]

- 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 and 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 $N = 2^m$:

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

\[
\begin{align*}
\{F_k\}_{k=0}^{m-1} & \quad \{G_k\}_{k=0}^{m-1} \\
\{0\}_{k=0}^{m-1} & \quad \{0\}_{k=0}^{m-1}
\end{align*}
\]
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 \( N = 2^m \):

\[
\{ F_k \}_{k=0}^{2^{m-1}} \quad \{ G_k \}_{k=0}^{2^{m-1}}
\]

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

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

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

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

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

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

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

- 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:

\[
2mF_k = \sum_{j=0}^{2m-1} \zeta_{2m}^{-kj} f_j \\
= \sum_{\ell=0}^{m-1} \zeta_{2m}^{-k2\ell} f_{2\ell} + \sum_{\ell=0}^{m-1} \zeta_{2m}^{-k(2\ell+1)} f_{2\ell+1} \\
= \sum_{\ell=0}^{m-1} \zeta_{m}^{-k\ell} f_{2\ell} + \zeta_{2m}^{-k} \sum_{\ell=0}^{m-1} \zeta_{m}^{-k\ell} f_{2\ell+1} \quad k = 0, \ldots, m-1.
\]

• No bit reversal is required at the highest level.

• A 1D implicitly padded convolution is implemented in our \texttt{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.

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

• The numerical error is similar to explicit padding and the memory usage is identical.

\[
\begin{align*}
\{F_k\}_{k=0}^{m-1} & \quad \{G_k\}_{k=0}^{m-1} \\
\{f_{2\ell}\}_{\ell=0}^{m-1} & \quad \{f_{2\ell+1}\}_{\ell=0}^{m-1} \\
\{g_{2\ell}\}_{\ell=0}^{m-1} & \quad \{g_{2\ell+1}\}_{\ell=0}^{m-1}
\end{align*}
\]
• The computational complexity is $6Kn \log_2 m$.

• The numerical error is similar to explicit padding and the memory usage is identical.

\[
\begin{align*}
\{F_k\}_{k=0}^{m-1} & \quad \{G_k\}_{k=0}^{m-1} \\
\{f_{2\ell}\}_{\ell=0}^{m-1} & \quad \{f_{2\ell+1}\}_{\ell=0}^{m-1} \\
\{g_{2\ell}\}_{\ell=0}^{m-1} & \quad \{g_{2\ell+1}\}_{\ell=0}^{m-1} \\
\{f_{2\ell}g_{2\ell}\}_{\ell=0}^{m-1} & \quad \{f_{2\ell+1}g_{2\ell+1}\}_{\ell=0}^{m-1}
\end{align*}
\]
• The computational complexity is $6Km \log_2 m$.

• The numerical error is similar to explicit padding and the memory usage is identical.

\[ \{F_k\}_{k=0}^{m-1} \quad \{G_k\}_{k=0}^{m-1} \]
\[ \{f_{2\ell}\}_{\ell=0}^{m-1} \quad \{f_{2\ell+1}\}_{\ell=0}^{m-1} \]
\[ \{g_{2\ell}\}_{\ell=0}^{m-1} \quad \{g_{2\ell+1}\}_{\ell=0}^{m-1} \]
\[ \{f_{2\ell}g_{2\ell}\}_{\ell=0}^{m-1} \quad \{f_{2\ell+1}g_{2\ell+1}\}_{\ell=0}^{m-1} \]
\[ \{(F \ast G)_k\}_{k=0}^{m-1} \]
Input: vector $f$, vector $g$
Output: vector $f$

$u \leftarrow \text{ffft}^{-1}(f)$;
$v \leftarrow \text{ffft}^{-1}(g)$;
$u \leftarrow u \ast v$;

for $k = 0$ to $m - 1$ do
| $f[k] \leftarrow \zeta_{2m}^k f[k]$;
| $g[k] \leftarrow \zeta_{2m}^k g[k]$;
end

$v \leftarrow \text{ffft}^{-1}(f)$;
$f \leftarrow \text{ffft}^{-1}(g)$;
$v \leftarrow v \ast f$;
$f \leftarrow \text{ffft}(u)$;
$u \leftarrow \text{ffft}(v)$;

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

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

- explicit $T=1$
- implicit $T=1$
- explicit $T=4$
- implicit $T=4$
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.

![Diagram](image.png)
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:

\[ \mathcal{F}_{N_1, \ldots, N_d} \xrightarrow{\text{multiply}} \mathcal{F}_{N_1, \ldots, N_d} \]

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

\[ \mathcal{F}_{N_d} \xrightarrow{N_d \times \text{convolve}_{N_1, \ldots, N_{d-1}}} \mathcal{F}_{N_d}^{-1} \]
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.

\[
\text{FFT}^{-1} \{F \ast G\} \quad n_x \text{ even} \\
\text{FFT}^{-1} \{F \ast G\} \quad n_x \text{ odd}
\]
Implicit Padding in 2D

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

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

- explicit $T=1$
- implicit $T=1$
- explicit $T=4$
- implicit $T=4$
Implicit Padding in 3D

\[ \text{time}/(m^3 \log_2 m^3) \ (\text{ns}) \]

- explicit $T=1$
- implicit $T=1$
- explicit $T=4$
- implicit $T=4$
Centered (Pseudospectral) 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 remove aliases.

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

- A Hermitian convolution arises since the input vectors are real:

\[
f_{-k} = \overline{f_k}.
\]
1D Implicit Hermitian Convolution

![Graph showing the comparison between explicit and implicit methods for different values of T.]
Distributed-Memory Parallelization

- The pseudospectral method uses a matrix transpose to localize the computation of the multi-dimensional FFTs onto individual processors.

- Parallel generalized slab/pencil decompositions have recently been developed for distributed-memory architectures.

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

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

- We have developed an adaptive algorithm, dynamically tuned to choose the optimal block size.
$8 \times 8$ 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
$8 \times 8$ Block Transpose over 8 processors
8 × 8 Block Transpose over 8 processors
8 × 8 Block Transpose over 8 processors
Advantages of Hybrid MPI/OpenMP

• Use hybrid OpenMP/MPI with the optimal number of threads:
  – yields larger communication block size;
  – local transposition is not required within a single MPI node;
  – allows smaller problems to be distributed over a large number of processors;
  – for 3D FFTs, allows for more slab-like than pencil-like models, reducing the size of or even eliminating the need for a second transpose;
  – sometimes more efficient (by a factor of 2) than pure MPI.

• The use of nonblocking MPI communications allows us to overlap computation with communication: this can yield up to an additional 32% performance gain for implicitly dealiased convolutions, for which a natural parallelism exists between communication and computation.
Pure MPI 2D Convolutions

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

- implicit P=24
- implicit P=192
- explicit P=24
- explicit P=192
Pure MPI 3D Convolutions

![Graph showing time/($m^3 \log_2 m^3$) (ns) vs. m for different P values](image-url)
Hybrid MPI 3D Adaptive Transpose Timing
Hybrid MPI 3D Adaptive Transpose Speedup
Communication Costs: Direct Transpose

• Suppose an $N \times N$ matrix is distributed over $P$ processes with $P \mid N$.

• Direct transposition involves $P - 1$ communications per process, each of size $N^2/P^2$, for a total per-process data transfer of

$$\frac{P - 1}{P^2}N^2.$$
Block Transpose

• Let $P = ab$. Subdivide $N \times M$ matrix into $a \times a$ blocks each of size $N/a \times M/a$.

• Inner: Over each team of $b$ processes, transpose the $a$ individual $N/a \times M/a$ matrices, grouping all $a$ communications with the same source and destination together.

• Outer: Over each team of $a$ processes, transpose the $a \times a$ matrix of $N/a \times M/a$ blocks.
Communication Costs

• Let $\tau_\ell$ be the typical latency of a message and $\tau_d$ be the time required to send each matrix element, so that the time to send a message consisting of $n$ matrix elements is

$$\tau_\ell + n\tau_d$$

• The time required to perform a direct transpose is

$$T_D = \tau_\ell(P - 1) + \tau_d\frac{P - 1}{P^2}NM = (P - 1)\left(\tau_\ell + \tau_d\frac{NM}{P^2}\right),$$

whereas a block transpose requires

$$T_B(a) = \tau_\ell\left(a + \frac{P}{a} - 2\right) + \tau_d\left(2P - a - \frac{P}{a}\right)\frac{NM}{P^2}.$$ 

• Let $L = \tau_\ell/\tau_d$ be the effective communication block length.
Direct vs. Block Transposes

• Since

\[ T_D - T_B = \tau_d \left( P + 1 - a - \frac{P}{a} \right) \left( L - \frac{NM}{P^2} \right), \]

we see that a direct transpose is preferred when \( NM \geq P^2L \), whereas a block transpose should be used when \( NM < P^2L \).

• To find the optimal value of \( a \) for a block transpose consider

\[ T'_B(a) = \tau_d \left( 1 - \frac{P}{a^2} \right) \left( L - \frac{NM}{P^2} \right). \]

• For \( NM < P^2L \), we see that \( T_B \) is convex, with a minimum at \( a = \sqrt{P} \).
Optimal Number of Threads

• The minimum value of $T_B$ is

$$T_B(\sqrt{P}) = 2\tau_d \left( \sqrt{P} - 1 \right) \left( L + \frac{NM}{P^{3/2}} \right)$$

$$\sim 2 \tau_d \sqrt{P} \left( L + \frac{NM}{P^{3/2}} \right), \quad P \gg 1.$$ 

• The global minimum of $T_B$ over both $a$ and $P$ occurs at

$$P \approx \left( \frac{2NM}{L} \right)^{2/3}.$$ 

• If the matrix dimensions satisfy $NM > L$, as is typically the case, this minimum occurs above the transition value $(NM/L)^{1/2}$. 
Conclusions

- For centered convolutions in $d$ dimensions implicit padding asymptotically uses $(2/3)^{d-1}$ of the conventional storage.

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

- Highly optimized and parallelized implicit dealiasing routines have been implemented as a software layer FFTW++ (v 2.05) on top of the FFTW library and released under the Lesser GNU Public License: [http://fftwpp.sourceforge.net/](http://fftwpp.sourceforge.net/)

- Hybrid MPI/OpenMP is often more efficient than pure MPI for distributed matrix transposes.

- The hybrid paradigm provides an optimal setting for nonlocal computationally intensive operations found in applications like the fast Fourier transform.

- The advent of implicit dealiasing of convolutions makes overlapping transposition with FFT computation feasible.
Writing of a high-performance dealiased pseudospectral code is now a relatively straightforward exercise. For example, see the protodns project at

http://github.com/dealias/dns
<table>
<thead>
<tr>
<th>Reference</th>
<th>Details</th>
</tr>
</thead>
</table>