Implicitly Dealiased Convolutions: Parallelization of a New Algorithm for FFT-based Convolutions

Malcolm Roberts*,1, John C. Bowman2

1University of Strasbourg
2University of Alberta

Séminaire Equations aux dérivées partielles, IRMA,
Université de Strasbourg, 2016-05-03

*malcolm.i.w.roberts@gmail.com, www.malcolmiwroberts.com
Abstract

Implicitly Dealiiased Convolutions: Parallelization of a New Algorithm for FFT-based Convolutions

Convolutions are an important numerical tool with applications to, for example, signal processing, machine learning, and simulation of nonlinear PDEs. Convolutions can be efficiently computed using FFTs and the convolution theorem at the cost of having to perform extra work to remove aliased terms. The method of implicitly dealiased convolutions [Bowman and Roberts, SIAM J. Sci. Comput. 2011] reduces the cost of dealiasing convolutions by re-using memory when computing multi-dimensional convolutions. Here, we present the implementation of a hybrid OpenMP/MPI parallel version of the convolutions and a new recursive transpose algorithm designed for clusters of multi-core computers.
Outline

- FFT-based convolutions
  - Conventional dealiasing
  - Implicit dealiasing
- Shared-memory implementation
  - 1/2-padding performance results
  - 2/3-padding performance results
- Distributed-memory implementation
  - OpenMP/MPI parallelism
  - Hybrid distributed transpose
  - 1/2-padding performance results
  - 2/3-padding performance results
FFT-based convolutions

The convolution of \( \{ F_k \}_{k=0}^{m-1} \) and \( \{ G_k \}_{k=0}^{m-1} \) is

\[
(F \ast G)_k = \sum_{\ell=0}^{k} F_{\ell} G_{k-\ell}, \quad k=0, \ldots, m-1. \tag{1}
\]

For example, if \( F \) and \( G \) are:

Then \( F \ast G \) is:
FFT-based convolutions

Applications:
- Signal processing
- Machine learning: convolutional neural networks
- Image processing
- Particle-Image-Velocimetry
- Pseudospectral simulations of nonlinear PDEs

The convolution theorem:

\[ \mathcal{F}[F \ast G] = \mathcal{F}[F] \odot \mathcal{F}[G]. \]  

Using FFTs improves speed and accuracy.
Example: you have all computed convolutions

$$42 \times 13 = ?$$  \hfill (3)\\

$$42 = 2 \times 10^0 + 4 \times 10^1 + 0 \times 10^2$$ \hfill (4)\\
$$= (2, 4, 0)$$

$$13 = 3 \times 10^0 + 1 \times 10^1 + 0 \times 10^2$$ \hfill (5)\\
$$= (3, 1, 0)$$

$$(2, 4, 0) \ast (3, 1, 0) = (2 \times 3, 4 \times 3 + 2 \times 1, 0 \times 2 + 4 \times 1 + 0 \times 3)$$
$$= (6, 14, 4)$$
$$= 6 \times 10^0 + 14 \times 10^1 + 4 \times 10^2$$
$$= 6 \times 10^0 + 4 \times 10^1 + 5 \times 10^2$$
$$= 546$$  \hfill (6)
Example: detecting periodicity
Let $\zeta_m = \exp\left(\frac{2\pi i}{m}\right)$. Forward and backward Fourier transforms are given by:

$$f_j = \sum_{k=0}^{m-1} \zeta_m^{jk} F_k, \quad F_k = \frac{1}{m} \sum_{j=0}^{m-1} \zeta_m^{-kj} f_k,$$

(7)

We will use the identity

$$\sum_{j=0}^{m-1} \zeta_m^{\ell j} = \begin{cases} m & \text{if } \ell = sm \text{ for } s \in \mathbb{Z}, \\ \frac{1-\zeta_m^{\ell m}}{1-\zeta_m} & = 0 \text{ otherwise.} \end{cases}$$

(8)
FFT-based convolutions

The convolution theorem works because

\[ m-1 \sum_{j=0} f_j g_j \zeta_m^{-jk} = m \sum_{j=0} \zeta_m^{-jk} \left( m-1 \sum_{p=0} \zeta_m^j F_p \right) \left( m-1 \sum_{q=0} \zeta_m^q G_q \right) \]

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

\[ = m \sum_{s=0} \sum_{p=0} F_p G_{k-p+sm}. \]

The terms \( s \neq 0 \) are aliases; they are bad.
Conventional dealiasing: zero padding

Let
\[ \tilde{F} = \{ F_0, F_1, \ldots, F_{m-2}, F_{m-1}, 0, \ldots, 0 \}. \]  \hspace{1cm} (10)

Then,
\[
\left( \tilde{F} \star_{2^m} \tilde{G} \right)_k = \sum_{\ell=0}^{2m-1} \tilde{F}_{\ell \text{ mod } (2^m)} \tilde{G}_{(k-\ell) \text{ mod } (2^m)} \\
= \sum_{\ell=0}^{m-1} F_{\ell} \tilde{G}_{(k-\ell) \text{ mod } (2^m)} \\
= \sum_{\ell=0}^{k} F_{\ell} G_{k-\ell}. \hspace{1cm} (11)
\]
Explicit zero-padding

\[
\begin{align*}
\{F_k\}_{k=0}^{m-1} & \quad \{G_k\}_{k=0}^{m-1} \\
\{F_k\}_{k=0}^{m-1} & \quad \{0\}_{k=0}^{m-1} \\
\{0\}_{n=0}^{2m-1} & \quad \{0\}_{n=0}^{m-1} \\
\{f_n\}_{n=0}^{2m-1} & \quad \{g_n\}_{n=0}^{2m-1} \\
\{f_n g_n\}_{n=0}^{2m-1} & \quad \{(F \ast G)_k\}_{k=0}^{m-1} \\
\{(F \ast G)_k\}_{k=0}^{m-1} & \quad \{(F \ast G)_k\}_{k=0}^{m-1}
\end{align*}
\]
Dealiasing with conventional zero-padding

\[ F \rightarrow F \rightarrow f \rightarrow fg \rightarrow F \ast G \nonumber \]

\[ G \rightarrow G \rightarrow g \rightarrow F \ast G \nonumber \]
Dealiasing with implicit zero-padding

We modify the FFT to account for the zeros implicitly. Let \( \zeta_n = \exp(-i2\pi/n) \). The Fourier transform of \( \tilde{F} \) is

\[
\tilde{F}(x) = \sum_{k=0}^{2m-1} \zeta_{2m}^{xk} \tilde{F}_k = \sum_{k=0}^{m-1} \zeta_{2m}^{xk} \tilde{F}_k
\]  

We can compute this using two discontiguous buffers:

\[
f_{2x} = \sum_{k=0}^{m-1} \zeta_m^{xk} F_k \quad f_{2x+1} = \sum_{k=0}^{m-1} \zeta_m^{xk} (\zeta_{2m}^k F_k)
\]
Implicit zero-padding

\[
\{F_k\}_{k=0}^{m-1} \quad \{G_k\}_{k=0}^{m-1} \\
\{f_n\}_{n=0}^{m-1}, n \text{ even} \quad \{f_n\}_{n=0}^{m-1}, n \text{ odd} \\
\{f_n g_n\}_{n=0}^{N-1}, n \text{ even} \quad \{f_n g_n\}_{n=0}^{N-1}, n \text{ odd} \\
\{(F \ast G)_k\}_{k=0}^{N-1}
\]
Suppose we have $A$ inputs and $B$ outputs. Examples:

- **Binary convolution, $f \ast g$:**
  - $A = 2, B = 1$, multiplier: $(f_x, g_x) \rightarrow f_x g_x$

- **Autocorrelation, $f \ast f$:**
  - $A = 1, B = 1$, multiplier: $(f_x) \rightarrow f_x f_x^*$

- **2D Navier–Stokes vorticity formulation (2/3 padding):**
  - $A = 4, B = 1$, multiplier:
    \[
    \left( u_x, u_y, \frac{\partial \omega}{\partial x}, \frac{\partial \omega}{\partial y} \right) \rightarrow u_x \frac{\partial \omega}{\partial x} + u_y \frac{\partial \omega}{\partial y}
    \]

- **3D magneto-hydrodynamic flow:**
  - $A = 12, B = 6$, $(u, \omega, B, j) \rightarrow (u \times \omega + j \times B, u \times B)$

For $1/2$ padding with $A > B$, we can do FFTs out-of-place.
Shared-memory implementation
Shared-memory implementation
Shared-memory implementation

\[ F^{-1} \]
Shared-memory implementation
Shared-memory implementation

\[ F^{-1} \]
Shared-memory implementation
Shared-memory implementation

\[ F^{-1} \]
Shared-memory implementation
Shared-memory implementation

\[ F^{-1} \]
Shared-memory implementation
Shared-memory implementation
Shared-memory implementation
Shared-memory implementation
Shared-memory implementation

\[ \mathcal{F} \]
Shared-memory implementation
Shared-memory implementation
Dealiasing with implicit zero-padding
Dealiasing with implicit zero-padding

\begin{align*}
\text{FFT}^{-1}_x \{ F \} & \quad n_x \text{ even} \\
\text{FFT}^{-1}_x \{ F \} & \quad n_x \text{ odd} \\
\text{FFT}^{-1}_x \{ G \} & \quad n_x \text{ even} \\
\text{FFT}^{-1}_x \{ G \} & \quad n_x \text{ odd}
\end{align*}
Dealiasing with implicit zero-padding
Dealiasing with implicit zero-padding
Dealiasing with implicit zero-padding
Dealiasing with implicit zero-padding

\[
\text{FFT}_{x}^{-1}\{F \ast G\}
\]
for even \(n_x\)

\[
\text{FFT}_{x}^{-1}\{F \ast G\}
\]
for odd \(n_x\)
Dealiasing with implicit zero-padding

\[ F \star G \]
Shared-memory implementation

- Implicit dealiasing requires less memory.
- We avoid FFTs on zero-data.
- By using discontiguous buffers, we can use multiple NUMA nodes.
- SSE2 vectorization instructions.
- Additional threads requires additional sub-dimensional work buffers.
- We use strides instead of transposes because we need to multi-thread.
Multi-threaded performance: 1D

performance: \( m \log_2 m / \text{time, (ns)}^{-1} \)

- △ - Implicit T=1
- □ - Implicit T=4
- ○ - Explicit T=1
- ◊ - Explicit T=4
Multi-threaded performance: 2D

performance: \( m^2 \log_2 m^2 / \text{time, (ns)}^{-1} \)

- Implicit \( T=1 \)
- Implicit \( T=4 \)
- Explicit \( T=1 \)
- Explicit \( T=4 \)
Multi-threaded performance: 3D

performance: $m^3 \log_2 m^3 / \text{time, (ns)}^{-1}$

- ▲ - Implicit $T=1$
- ■ - Implicit $T=4$
- ● - Explicit $T=1$
- ○ - Explicit $T=4$

Malcolm Roberts
malcolmiwroberts.com
Multi-threaded speedup: 3D

![Graph showing multi-threaded speedup with relative speed plotted against m for T=1 and T=4.]

Malcolm Roberts
malcolm.iwroberts.com
The Fourier transform of $\{ f_x \in \mathbb{R} \}_{x=0}^{2m}$ is
\[
F = \{ F_k \in \mathbb{C}, F_{-k} = F^*_k \}_{k=-m}^{m-1}.
\] (14)

The convolution is
\[
(F \ast G)_k = \sum_{\ell = k-m}^{m-1} F_{\ell} G_{k-\ell}
\] (15)

One must pad from length $2m$ to length $3m$.

We do $2A + 3B$ out-of-place FFTs if $A \geq 2B$ ($A$ in-place).

The implicitly dealiased convolution routines can either include (non-compact format) or exclude (compact format) the Nyquist mode $F_{-m}$. 
Implicit vs explicit:

![Graph showing performance comparison between implicit and explicit methods for different T values. The x-axis represents the variable m, and the y-axis represents the performance in \( m \log_2 m/\text{time, (ns)}^{-1} \). The graph includes data points for Implicit T=1, Implicit T=4, Explicit T=1, and Explicit T=4, with different markers and line styles for each category.]
2/3 padding: 2D

Compact vs non-compact, $T=4$: 

![Graph showing performance of compact vs non-compact models with $T=4$. The x-axis represents $m$ with values from $10^2$ to $10^3$, and the y-axis represents performance as $m^2 \log_2 m^2$/time, (ns)$^{-1}$. The graph includes lines for Implicit X=0 Y=0, Implicit X=1 Y=0, Implicit X=0 Y=1, and Implicit X=1 Y=1.]

Malcolm Roberts  
malcolmwiwroberts.com  
23
Implicit vs explicit:

\[ \text{performance: } m^2 \log_2 m^2 / \text{time, (ns)}^{-1} \]

\begin{align*}
\text{Explicit } T=1 & \quad \text{Implicit } T=1 \\
\text{Explicit } T=4 & \quad \text{Implicit } T=4
\end{align*}
Compact vs non-compact, $T=1$:

Performance: $m^3 \log_2 m^3 / \text{time}, \ (\text{ns})^{-1}$

- $\triangle$ - Implicit $X=0 \ Y=0 \ Z=0$
- $\square$ - Implicit $X=1 \ Y=0 \ Z=0$
- $\diamond$ - Implicit $X=0 \ Y=1 \ Z=0$
- $\circ$ - Implicit $X=1 \ Y=1 \ Z=0$
- $\ast$ - Implicit $X=0 \ Y=0 \ Z=1$
- $\times$ - Implicit $X=1 \ Y=0 \ Z=1$
- $\times$ - Implicit $X=0 \ Y=1 \ Z=1$
- $\ast$ - Implicit $X=1 \ Y=1 \ Z=1$

Malcolm Roberts
malcolmroberts.com
Compact vs non-compact, $T=4$:

\[ \text{performance: } m^3 \log_2 m^3 / \text{time, (ns)}^{-1} \]

- $\text{Implicit } X=0, Y=0, Z=0$
- $\text{Implicit } X=1, Y=0, Z=0$
- $\text{Implicit } X=0, Y=1, Z=0$
- $\text{Implicit } X=1, Y=1, Z=0$
- $\text{Implicit } X=0, Y=0, Z=1$
- $\text{Implicit } X=1, Y=0, Z=1$
- $\text{Implicit } X=0, Y=1, Z=1$
- $\text{Implicit } X=1, Y=1, Z=1$
2/3 padding: 3D

Implicit:

performance: $m^3 \log_2 m^3 / \text{time, (ns)}^{-1}$

- ▲ - Implicit $T=1$
- ■ - Implicit $T=4$

Malcolm Roberts malcolmiwroberts.com
Distributed-memory implementation

We want to run on clusters of multi-core nodes.

- Implicit dealiasing requires less communication.
- By using discontiguous buffers, we can overlap communication and computation.
- We use a hybrid OpenMP/MPI parallelization for clusters of multi-core machines.
- 2D MPI data decomposition.
- We make use of the hybrid transpose algorithm.

Suppose that the nodes have $C$ cores each.

- We will use $P$ MPI processes with $T \leq C$ threads per process.
- We launch $C/T$ processes per node.
Matrix transpose is an essential primitive of high-performance computing.

They allow one to localize data on one process so that shared-memory algorithms can be applied.

I will discuss two algorithms for transposes:

- Direct transpose.
- Recursive transpose.

We combine these into a hybrid transpose.
Direct (AlltoAll) Transpose

- Efficient for $P \ll m$ (large messages).
- Most direct method.
- Many small messages when $P \approx m$.

Implementations:
- MPI_Alltoall
- MPI_Send, MPI_Recv
Direct (AlltoAll) Transpose
Direct (AlltoAll) Transpose
Direct (AlltoAll) Transpose
Recursive Transpose

- Efficient for $P \gg m$ (large messages).
- Recursively subdivides transpose into smaller block transposes.
- $\log m$ phases.
- Communications are grouped to reduce latency.
- Requires intermediate communication.

Implementations:
- FFTW
Recursive Transpose
Recursive Transpose
Recursive Transpose
Recursive Transpose

Process 0
1
2
3
4
5
6
7

Malcolm Roberts malcolmiwroberts.com
Recursive Transpose

Process

0 1 2 3 4 5 6 7

Malcolm Roberts
malcolmiwroberts.com
Recursive Transpose
Recursive Transpose
Hybrid Transpose

- Recursive, but just one level.
- Use the empirical properties of the cluster to determine best parameters.
- **Optionally** group messages to reduce latency.

Implementation:

- FFTW++

Direct transpose communication cost: \( \frac{P-1}{P^2} m^2 \), \( P \) messages.

Hybrid cost with \( P = ab \): \( \frac{(a-1)bm^2}{P^2} + \frac{(b-1)am^2}{P^2} \), \( a + b \) messages.
Let $\tau_\ell$ be the message latency, and $\tau_d$ the time to send one element. The time to send $n$ elements is

$$\tau_\ell + n\tau_d.$$ (16)

The time required to do a direct transpose is

$$T_D = \tau_\ell (P - 1) + \tau_d \frac{P - 1}{P^2} m^2 = (P - 1) \left( \tau_\ell + \tau_d \frac{m^2}{P^2} \right)$$ (17)

The time for a block transpose is

$$T_B(a) = \tau_\ell \left( a + \frac{P}{a} - 2 \right) + \tau_d \left( 2P - a - \frac{P}{a} \right) \frac{m^2}{P^2}.$$ (18)
Hybrid Transpose

Let \( L = \frac{\tau_\ell}{\tau_d} \).

For \( P \gg 1 \),

\[
T_D \approx \tau_d \left( PL + \frac{m^2}{P} \right)
\]

has a global minimum of \( 2\tau_d m\sqrt{L} \) at \( P = \frac{m}{\sqrt{L}} \).

For \( m^2 < P^2 L \), \( T_B \) is convex, with a global minimum at \( a = \sqrt{P} \), with

\[
T_B(a = \sqrt{P}) \approx 2\tau_d \sqrt{P} \left( L + \frac{m^2}{P^{3/2}} \right)
\]
Hybrid Transpose

Communication Cost

Zero Latency
Direct
Block

$P$
Hybrid Transpose: multi-threading

We have $C$ cores per node and $S$ nodes.
We launch $P$ processes with $T$ threads with $PT = SC$.
From minimizing $T_B$, the optimal number of threads is

$$T = \min \left( \frac{SC}{(2m^2/L)^{2/3}}, C \right). \quad (21)$$

For stampede at the Texas Advanced Supercomputer Center, we measured $L = 4096$, so for $S = 128$ and $C = 8$,
- $T = 8$ for $m = 1024$
- $T = 2$ for $m = 4096$
Hybrid Transpose

![Graph showing the comparison between FFTW and Hybrid transpose performance for different node and thread configurations. The x-axis represents the number of nodes multiplied by threads, ranging from 1024x1 to 128x8. The y-axis represents time in microseconds, ranging from 250 to 350. The graph includes two lines: one for FFTW and another for Hybrid, both showing the performance improvement with increasing node and thread configurations.](image-url)
Hybrid Transpose

![Graph showing time (μs) versus nodes × threads for FFTW and hybrid methods. The graph demonstrates performance comparisons for different configurations of nodes and threads.]
The hybrid transpose

- Uses a direct transpose for large message sizes.
- Uses a block transpose for small message sizes.
- Offers a performance advantage when $P \approx m$.
- Can be tuned based upon the values of $\tau_\ell$ and $\tau_d$ for the cluster.
- Optimal number of threads depends on the problem size and cluster characteristics.

We use the hybrid transpose for computing convolutions using implicit dealiasing on clusters.
MPI Convolution: 2D performance

performance: \(m^2 \log_2 m^2 / \text{time}, (\text{ns})^{-1}\)

- Implicit P=24 T=1
- Implicit P=48 T=1
- Implicit P=96 T=1
- Explicit P=24 T=1
- Explicit P=48 T=1
- Explicit P=96 T=1

Malcolm Roberts malcolmiwroberts.com
MPI Convolution: 2D performance

![Graph showing relative speed vs. m for different process counts (P) and time steps (T).]

- Red dashed line: \(P=24, T=1\)
- Blue square line: \(P=48, T=1\)
- Green dotted line: \(P=96, T=1\)

Malcolm Roberts malcolmiwroberts.com
MPI Convolution: multithreaded 2D performance

![Graph showing performance of MPI Convolution with different parameters.]
MPI Convolution: 3D performance

![Graph showing performance](image)

- Implicit $P=24$ $T=1$
- Implicit $P=48$ $T=1$
- Implicit $P=96$ $T=1$
- Explicit $P=24$ $T=1$
- Explicit $P=48$ $T=1$
- Explicit $P=96$ $T=1$

Malcolm Roberts
malcolmiwroberts.com
mpi convolution: 3d performance

\begin{figure}
\centering
\includegraphics[width=\textwidth]{convolution.png}
\end{figure}
MPI Convolution: multithreaded 3D performance

![Graph showing performance: \(\frac{m^3 \log_2 m^3}{\text{time}}, \text{(ns)}^{-1}\) vs. \(m\) for different parallelization schemes.]

- ▲ - Implicit \(P=1\) \(T=24\)
- □ - Implicit \(P=2\) \(T=24\)
- ⋄ - Implicit \(P=4\) \(T=24\)
- □ - Explicit \(P=1\) \(T=24\)
- ○ - Explicit \(P=2\) \(T=24\)
- ○ - Explicit \(P=4\) \(T=24\)
2/3 padding: 2D

Compact / non-compact performance, $P = 96, T = 1$:

![Graph showing performance vs. m^2 \log_2 m^2/time, (ns)^{-1} for different implicit configurations](image)

- ▲ - Implicit X=0 Y=0
- □ - Implicit X=1 Y=0
- ● ● ● - Implicit X=0 Y=1
- ○ - Implicit X=1 Y=1

Malcolm Roberts
malcolmiwroberts.com
2/3 padding: 2D

Compact / non-compact performance, $P = 4$, $T = 24$: 

![Graph showing performance vs $m$ for different implicit settings.](image)

- ▲ - Implicit $X=0$ $Y=0$
- □ - Implicit $X=1$ $Y=0$
- ○ - Implicit $X=0$ $Y=1$
- ● - Implicit $X=1$ $Y=1$
2/3 padding: 2D performance

Here we are non-compact in both directions:

\[ \text{performance: } m^2 \log_2 m^2 / \text{time, (ns)}^{-1} \]

- Implicit P=1 T=24
- Implicit P=2 T=24
- Implicit P=4 T=24
2/3 padding: 2D performance

Here we are non-compact in both directions:

![Graph showing performance vs. m with different line styles for Implicit P=24 T=1, Implicit P=48 T=1, and Implicit P=96 T=1.](image)
2/3 padding: 3D performance

Here we are non-compact in all three directions:

\[ \text{performance: } m^3 \log_2 m^3 \text{/time, (ns)}^{-1} \]

- Implicit P=24 T=1
- Implicit P=48 T=1
- Implicit P=96 T=1
2/3 padding: 3D performance

Here we are compact in the $y$-direction:

![Graph showing performance versus $m$]

- Implicit $P=24$ $T=1$
- Implicit $P=48$ $T=1$
- Implicit $P=96$ $T=1$

Malcolm Roberts
malcolmiwroberts.com
Future Work

To-do:

▶ Test scaling with thousands of cores.
▶ The transpose seems slower for 2D FFTs: fix this.
▶ Write-up and publish results.

Future work:

▶ Convolutions on real-valued data.
▶ Inputs with different sizes.
▶ Do it all again on GPU.
Conclusion

Implicitly dealiased convolutions:
- use less memory
- have less communication costs,
- and are faster than conventional zero-padding techniques.

The hybrid transpose is faster for small message size.

Collaboration with John Bowman, University of Alberta.
Implementation in the open-source project FFTW++:

fftwpp.sf.net

We have around 13 000 downloads (plus clones).