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

**Malcolm Roberts**[*,1], John C. Bowman[2]

[1]University of Strasbourg
[2]University 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 Dealiased 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

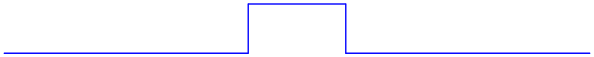
- FFT-based convolutions
  - Conventional dealiasing
  - Implicit dealiasing
- Shared-memory implementation
  - 1/2-padding performance results
  - 2/3-padding performance results
- Distributed-memory implementation
  - `OpenMP/MPI` parallelism
  - Hybrid distributed transpose
  - 1/2-padding performance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

# FFT-based convolutions

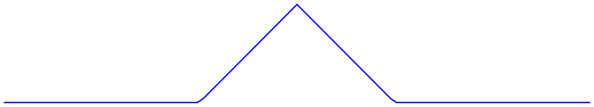
The convolution of $\{F_k\}_{k=0}^{m-1}$ and $\{G_k\}_{k=0}^{m-1}$ is

$$(F * G)_k = \sum_{\ell=0}^{k} F_\ell G_{k-\ell}, \quad k=0, \dots, m-1. \tag{1}$$

For example, if $F$ and $G$ are:



Then $F * G$ is:



## FFT-based convolutions

Applications:

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

The convolution theorem:

$$\mathcal{F}\left[F * G\right] = \mathcal{F}\left[F\right] \odot \mathcal{F}\left[G\right]. \qquad (2)$$

Using FFTs improves speed and accuracy.

## Example: you have all computed convolutions

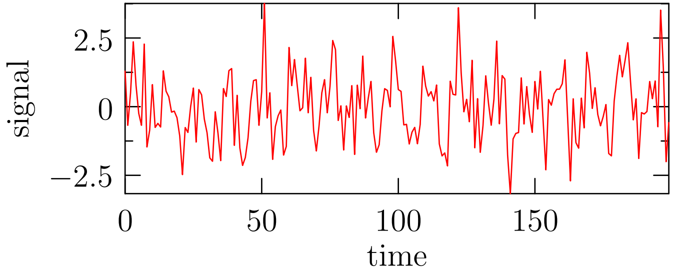
$$42 \times 13 = ? \tag{3}$$

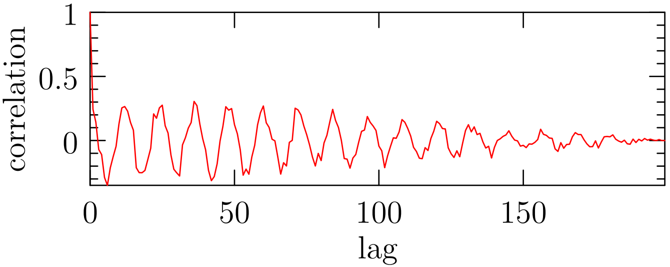
$$\begin{aligned} 42 &= 2 \times 10^0 + 4 \times 10^1 + 0 \times 10^2 \\ &= (2, 4, 0) \end{aligned} \tag{4}$$

$$\begin{aligned} 13 &= 3 \times 10^0 + 1 \times 10^1 + 0 \times 10^2 \\ &= (3, 1, 0) \end{aligned} \tag{5}$$

$$\begin{aligned} (2, 4, 0) * (3, 1, 0) &= (2 \times 3, 4 \times 3 + 2 \times 1, 0 \times 2 + 4 \times 1 + 0 \times 3) \\ &= (6, 14, 4) \\ &= 6 \times 10^0 + 14 \times 10^1 + 4 \times 10^2 \\ &= 6 \times 10^0 + 4 \times 10^1 + 5 \times 10^2 \\ &= 546 \end{aligned} \tag{6}$$

## Example: detecting periodicty





## FFT-based convolutions

Let $\zeta_m = \exp\left(\frac{2\pi i}{m}\right)$. Forward and backward Fourier transforms are given by:

$$f_j = \sum_{k=0}^{m-1} \zeta_m^{jk} F_k, \qquad F_k = \frac{1}{m} \sum_{j=0}^{m-1} \zeta_m^{-kj} f_k, \tag{7}$$

We will use the identity

$$\sum_{j=0}^{m-1} \zeta_m^{\ell j} = \begin{cases} m & \text{if } \ell = sm \text{ for } s \in \mathbb{Z}, \\ \frac{1-\zeta_m^{\ell m}}{1-\zeta_m^{m}} = 0 & \text{otherwise.} \end{cases} \tag{8}$$

# FFT-based convolutions

The convolution theorem works because

$$
\begin{aligned}
\sum_{j=0}^{m-1} f_j g_j \zeta_m^{-jk} &= \sum_{j=0}^{m-1} \zeta_m^{-jk} \left( \sum_{p=0}^{m-1} \zeta_m^{jp} F_p \right) \left( \sum_{q=0}^{m-1} \zeta_m^{jq} G_q \right) \\
&= \sum_{p=0}^{m-1} F_p \sum_{q=0}^{m-1} G_q \sum_{j=0}^{m-1} \zeta_m^{j(-k+p+q)} \qquad (9) \\
&= m \sum_{s} \sum_{p=0}^{m-1} F_p G_{k-p+sm}.
\end{aligned}
$$

The terms $s \neq 0$ are aliases; they are bad.

## Conventional dealiasing: zero padding

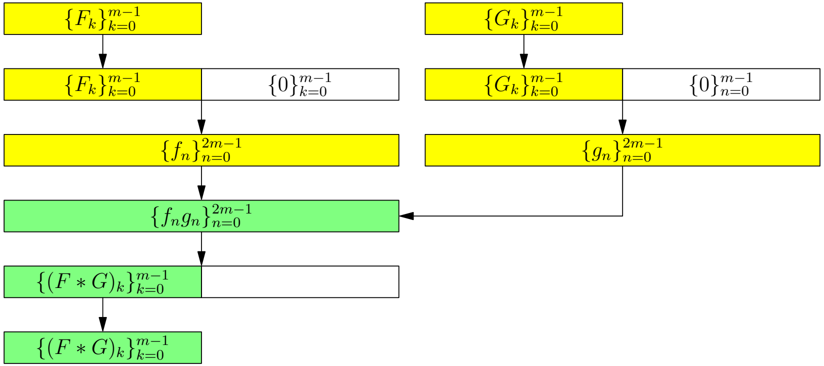
Let

$$\widetilde{F} \doteq \{F_0, F_1, \ldots, F_{m-2}, F_{m-1}, \underbrace{0, \ldots, 0}_{m}\}. \tag{10}$$

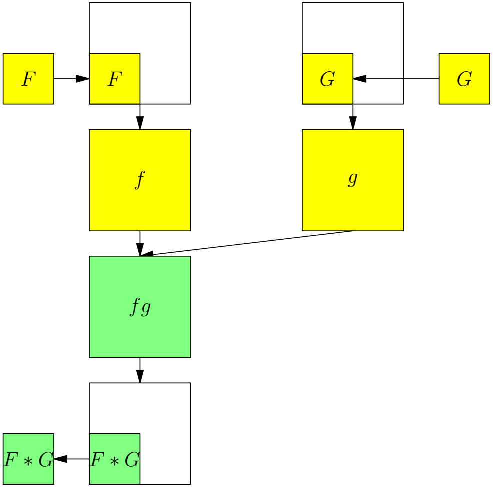
Then,

$$\begin{aligned}\left(\widetilde{F} *_{2m} \widetilde{G}\right)_k &= \sum_{\ell=0}^{2m-1} \widetilde{F}_{\ell \bmod (2m)} \widetilde{G}_{(k-\ell) \bmod (2m)} \\ &= \sum_{\ell=0}^{m-1} F_\ell \widetilde{G}_{(k-\ell) \bmod (2m)} \\ &= \sum_{\ell=0}^{k} F_\ell G_{k-\ell}.\end{aligned} \tag{11}$$

# Explicit zero-padding



## Dealiasing with conventional zero-padding



## 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 $\widetilde{F}$ is

$$f_x = \sum_{k=0}^{2m-1} \zeta_{2m}^{xk} \widetilde{F}_k = \sum_{k=0}^{m-1} \zeta_{2m}^{xk} \widetilde{F}_k \tag{12}$$

We can compute this using two discontiguous buffers:

$$f_{2x} = \sum_{k=0}^{m-1} \zeta_m^{xk} F_k \quad f_{2x+1} = \sum_{k=0}^{m-1} \zeta_m^{xk} \left(\zeta_{2m}^{k} F_k\right). \tag{13}$$

# Implicit zero-padding

$$\begin{tikzcd}
\{F_k\}_{k=0}^{m-1} \arrow[d] \arrow[dr] & & \{G_k\}_{k=0}^{m-1} \arrow[d] \arrow[dr] & \\
\{f_n\}_{n=0}^{m-1}, n\,\text{even} \arrow[d] & \{f_n\}_{n=0}^{m-1}, n\,\text{odd} \arrow[d] & \{g_n\}_{n=0}^{m-1}, n\,\text{even} \arrow[dll] & \{g_n\}_{n=0}^{m-1}, n\,\text{odd} \arrow[dll] \\
\{f_n g_n\}_{n=0}^{N-1}, n\,\text{even} \arrow[d] & \{f_n g_n\}_{n=0}^{N-1}, n\,\text{odd} \arrow[dl] & & \\
\{(F * G)_k\}_{k=0}^{N-1} & & &
\end{tikzcd}$$

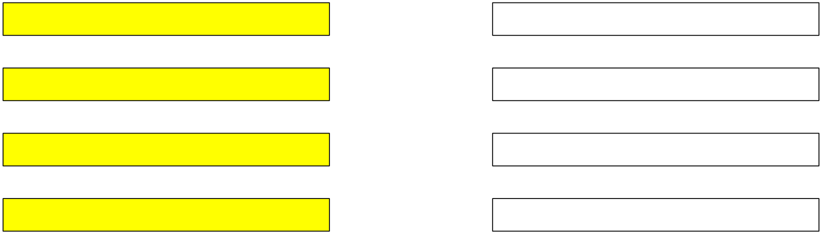
## Shared-memory implementation

Suppose we have $A$ inputs and $B$ outputs.
Examples:

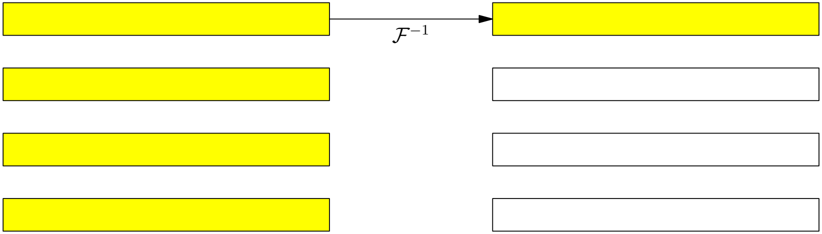
- Binary convolution, $f * g$:
  - $A = 2$, $B = 1$, multiplier: $(f_x, g_x) \to f_x g_x$
- Autocorrelation, $f * f$:
  - $A = 1$, $B = 1$, multiplier: $(f_x) \to f_x f_x^*$
- 2D Navier–Stokes vorticity formulation (2/3 padding):
  - $A = 4$, $B = 1$, multiplier: $\left(u_x, u_y, \frac{\partial\omega}{\partial x}, \frac{\partial\omega}{\partial y}\right) \to u_x \frac{\partial\omega}{\partial x} + u_y \frac{\partial\omega}{\partial y}$
- 3D magneto-hydrodynamic flow:
  - $A = 12$, $B = 6$

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

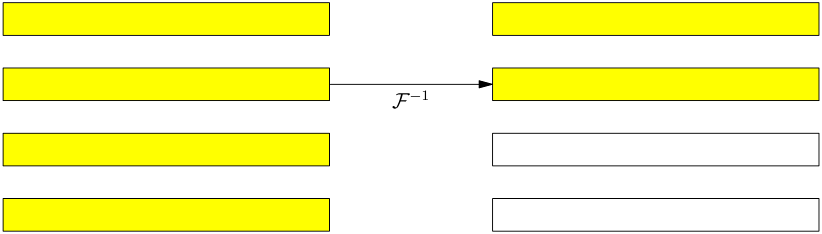
# Shared-memory implementation



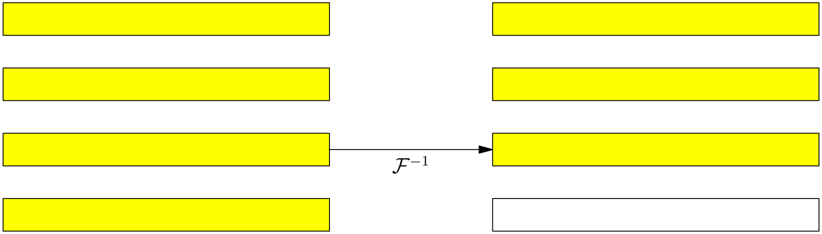
# Shared-memory implementation



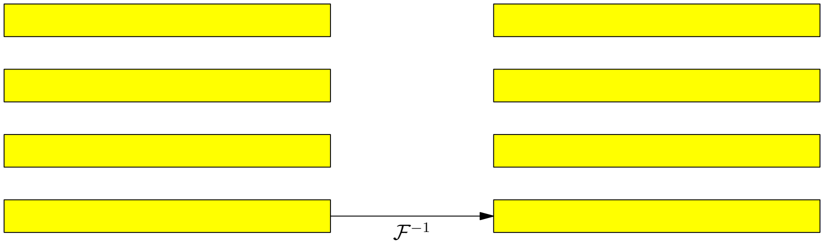
# Shared-memory implementation



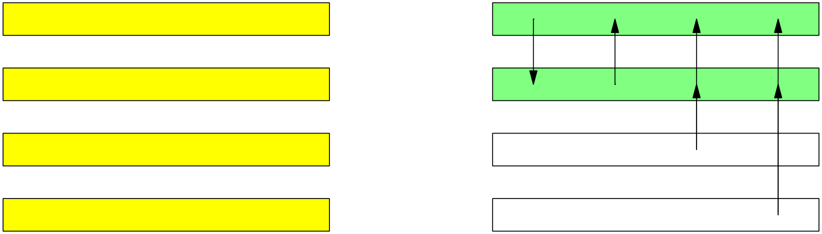
# Shared-memory implementation



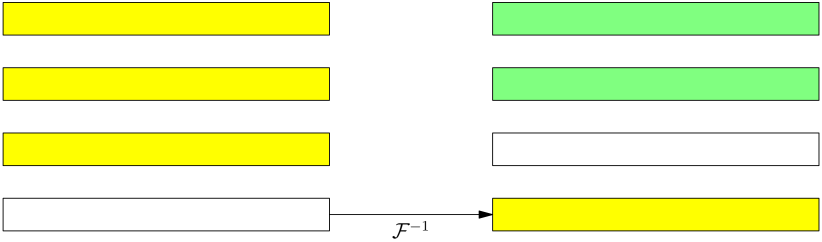
## Shared-memory implementation



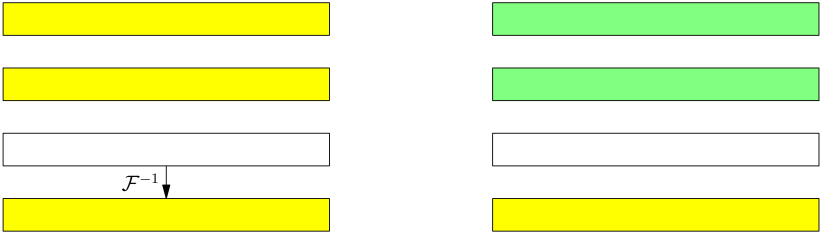
# Shared-memory implementation



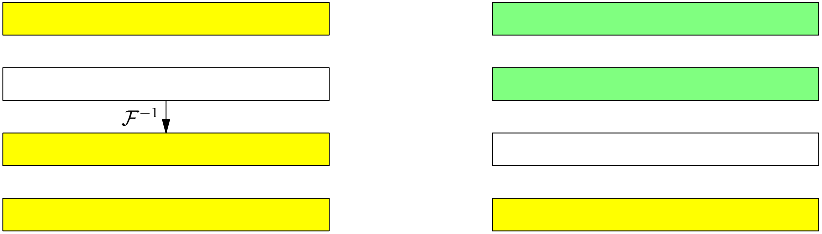
# Shared-memory implementation



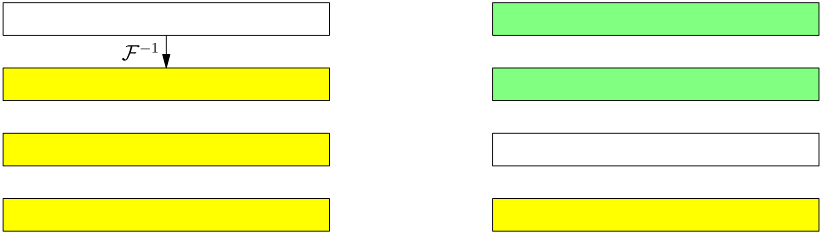
# Shared-memory implementation



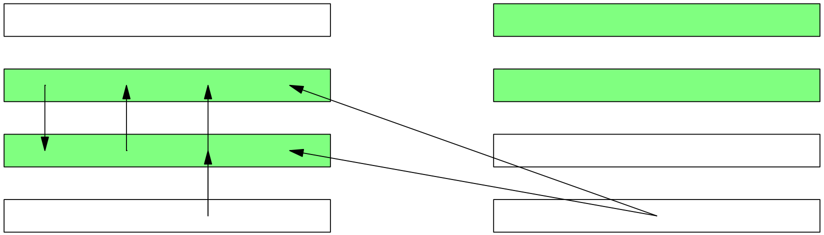
## Shared-memory implementation



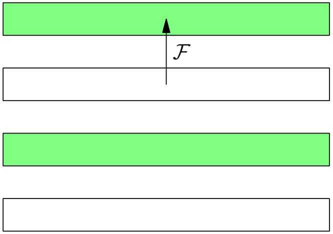
# Shared-memory implementation

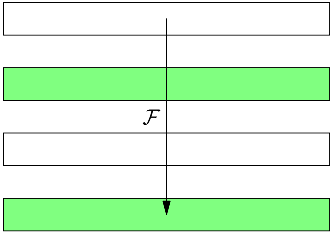


# Shared-memory implementation

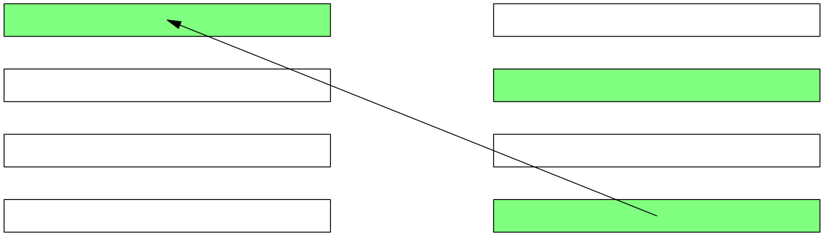


# Shared-memory implementation

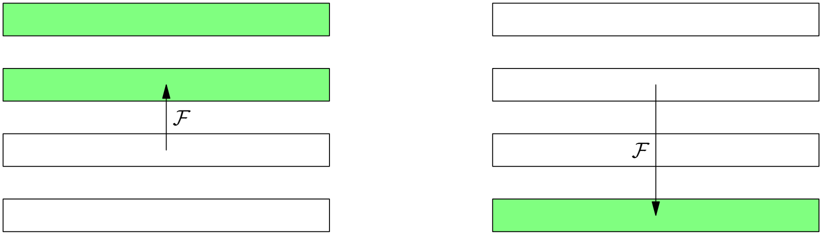




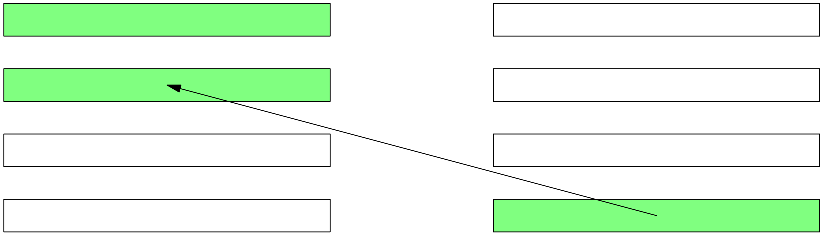
# Shared-memory implementation



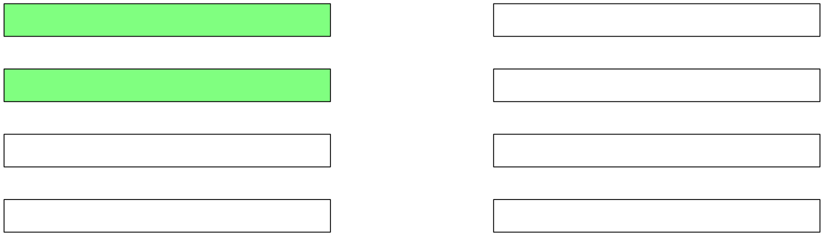
# Shared-memory implementation



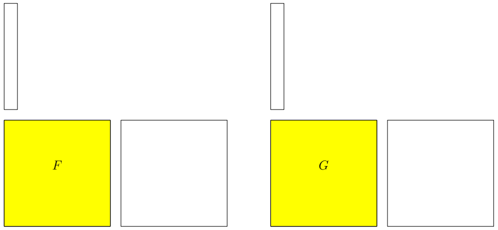
# Shared-memory implementation



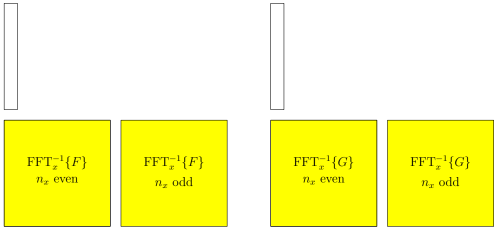
# Shared-memory implementation



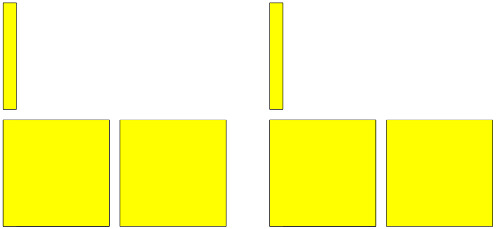
# Dealiasing with implicit zero-padding



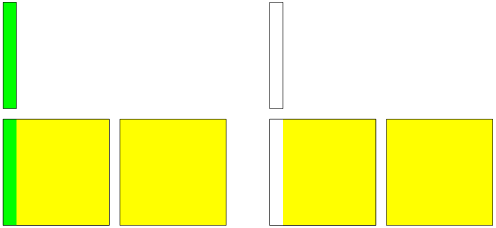
# Dealiasing with implicit zero-padding


$\mathrm{FFT}_x^{-1}\{F\}$ $n_x$ even
$\mathrm{FFT}_x^{-1}\{F\}$ $n_x$ odd
$\mathrm{FFT}_x^{-1}\{G\}$ $n_x$ even
$\mathrm{FFT}_x^{-1}\{G\}$ $n_x$ odd

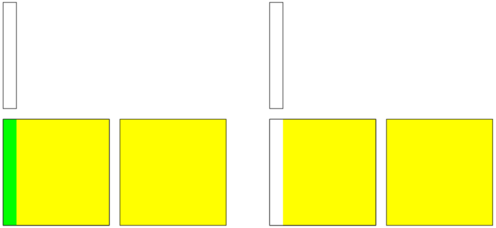
# Dealiasing with implicit zero-padding



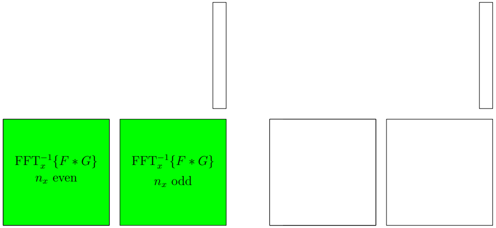
# Dealiasing with implicit zero-padding



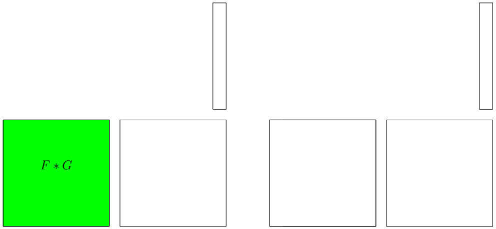
# Dealiasing with implicit zero-padding



# Dealiasing with implicit zero-padding



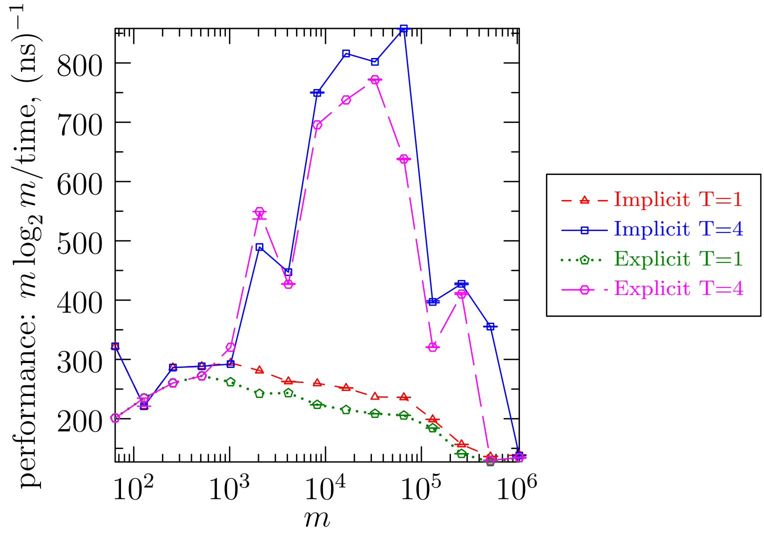
## Dealiasing with implicit zero-padding



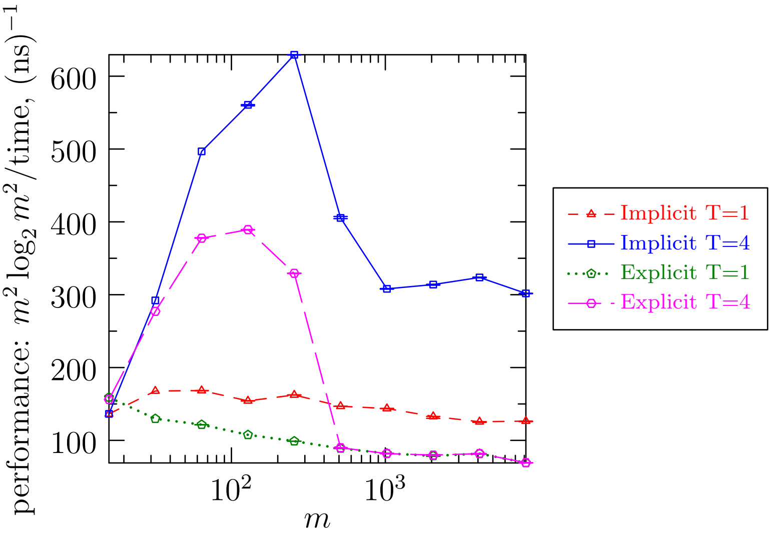
## Shared-memory implementation

- 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.

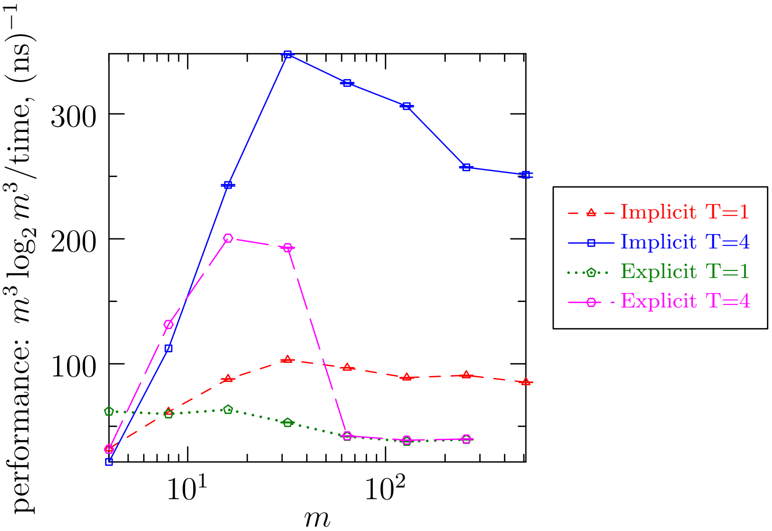
# Multi-threaded performance: 1D



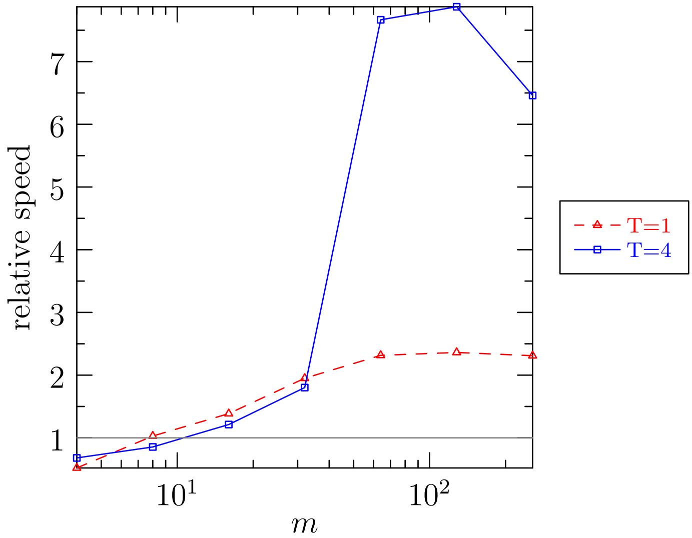
# Multi-threaded performance: 2D



# Multi-threaded performance: 3D



# Multi-threaded speedup: 3D



## 2/3 padding

The Fourier transform of $\{f_x \in \mathbb{R}\}_{x=0}^{2m}$ is

$$F = \{F_k \in \mathbb{C}, F_{-k} = F_k^*\}_{k=-m}^{m-1}. \tag{14}$$

The convolution is

$$(F * G)_k = \sum_{\ell = k-m}^{m-1} F_\ell G_{k-\ell} \tag{15}$$

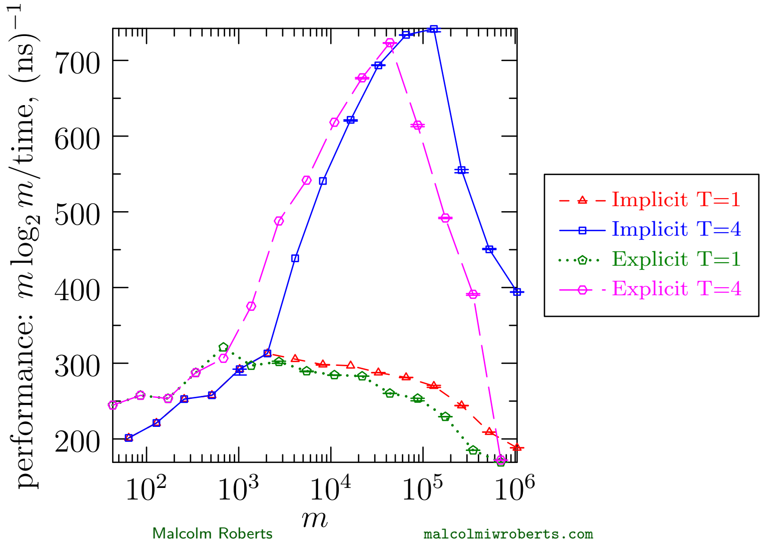
One must pad from length $2m$ to length $3m$.

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 $F_{-m}$.

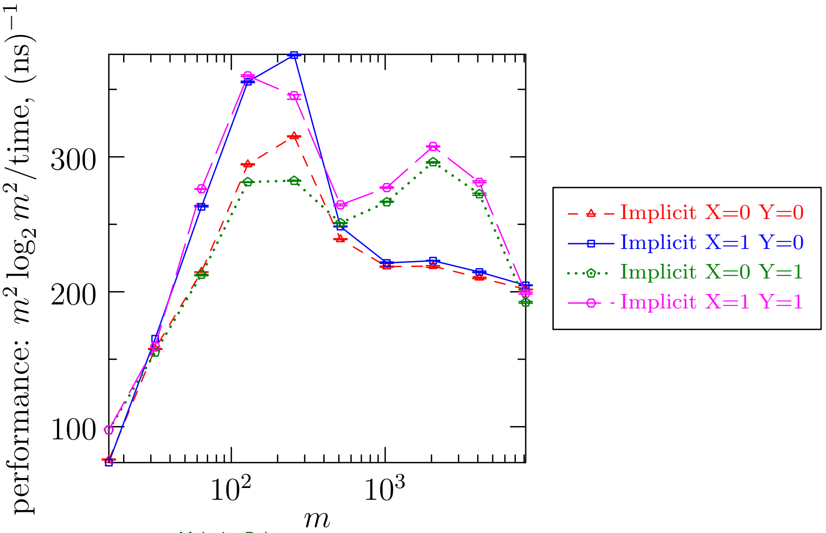
# 2/3 padding: 1D

Implicti vs explicit:



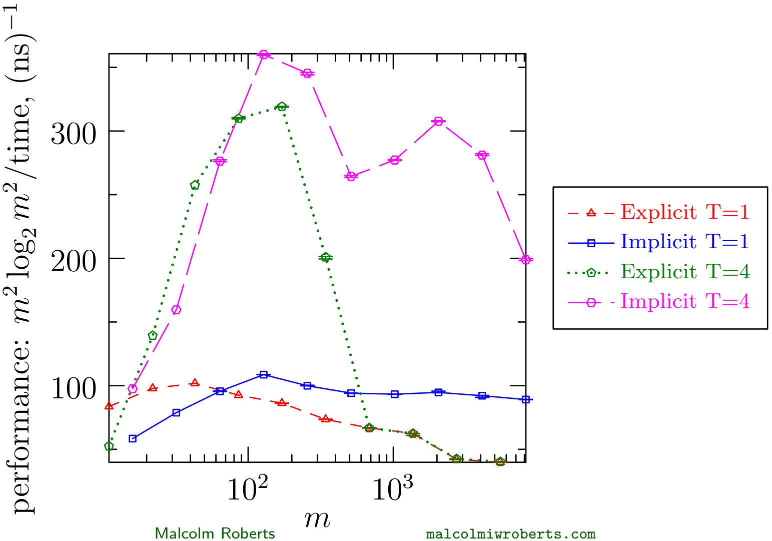
# 2/3 padding: 2D

Compact vs non-compact, T=4:



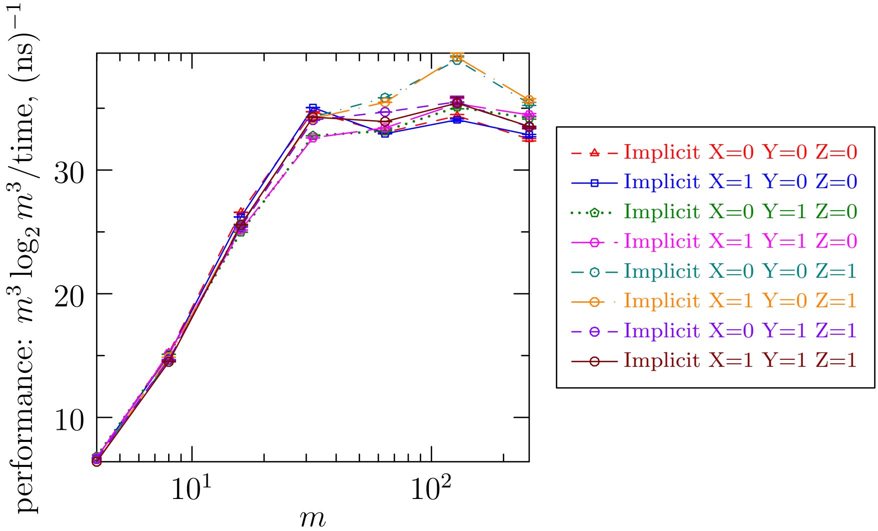
# 2/3 padding: 2D

Implicti vs explicit:



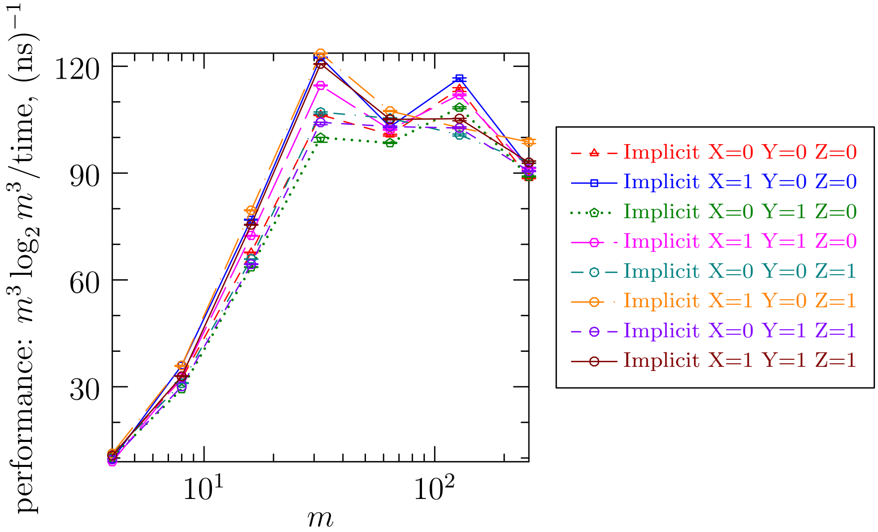
## 2/3 padding: 2D

Compact vs non-compact, T=1:



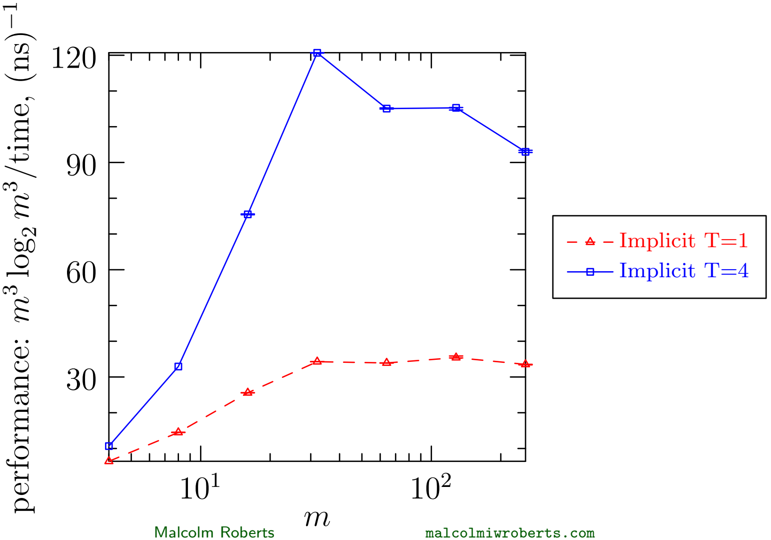
# 2/3 padding: 3D

Compact vs non-compact, T=4:



## 2/3 padding: 3D

Implicit:



# 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 w ill use $P$ `MPI` processes with $T \leq C$ threads per process.
- We launch $C/T$ processes per node.

## Hybrid MPI Transpose

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*.

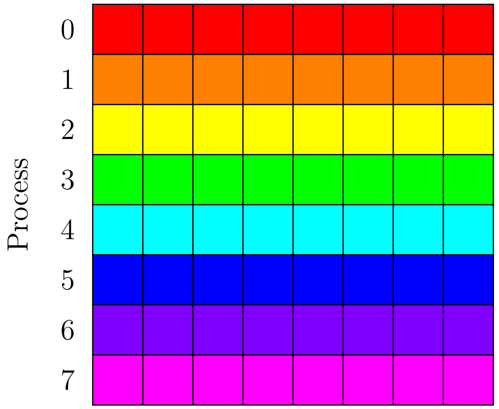
## Direct (AlltoAll) Transpose

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

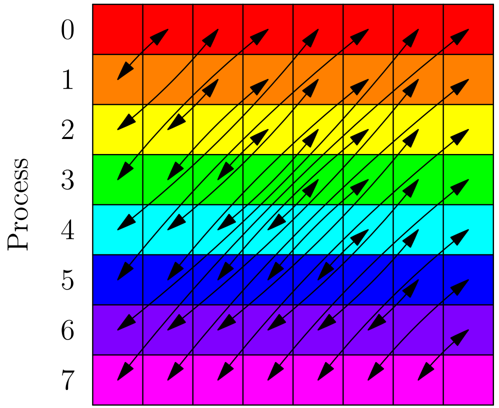
Implementations:

- `MPI_Alltoall`
- `MPI_Send`, `MPI_Recv`

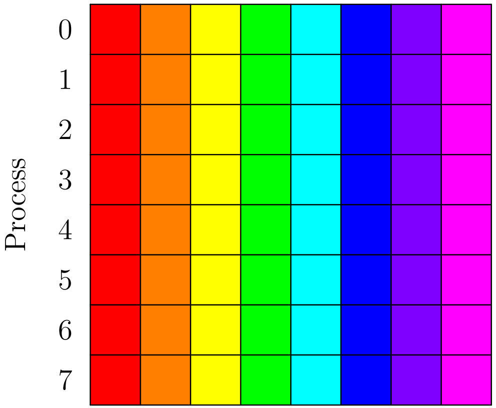
# Direct (AlltoAll) Transpose



# Direct (AlltoAll) Transpose



# Direct (AlltoAll) Transpose



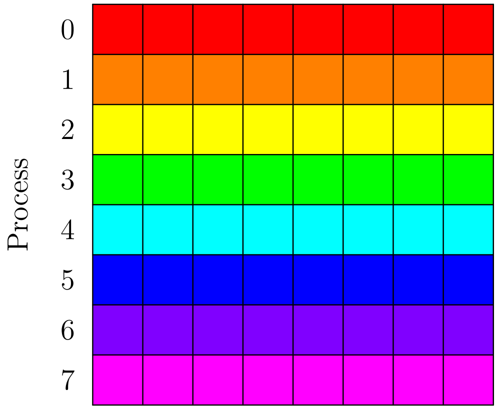
## Recursive Transpose

- 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.

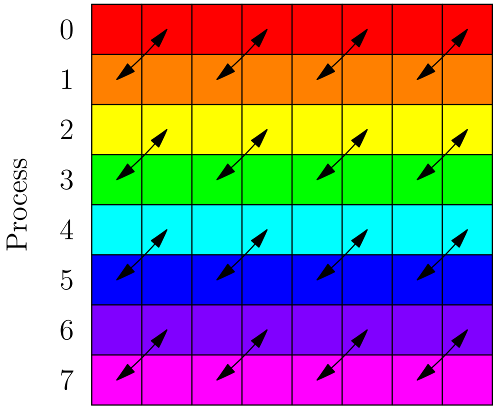
Implementations:

- `FFTW`

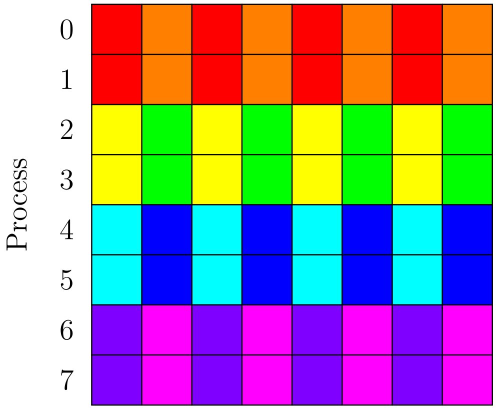
## Recursive Transpose



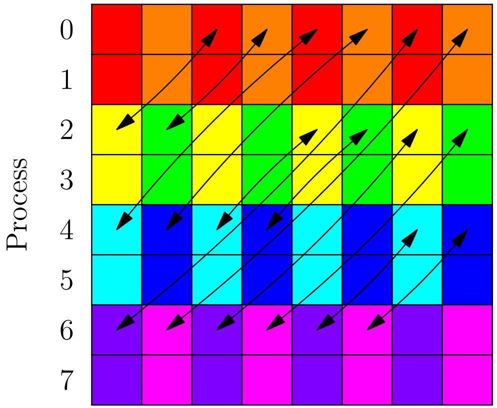
# Recursive Transpose



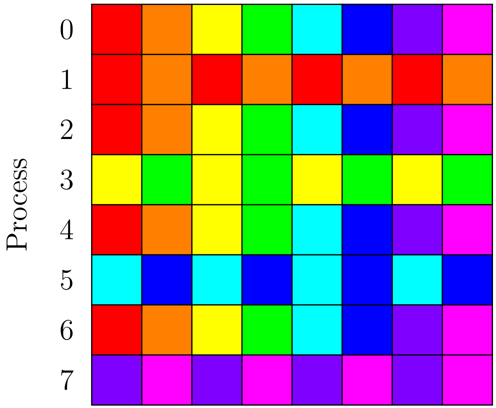
# Recursive Transpose



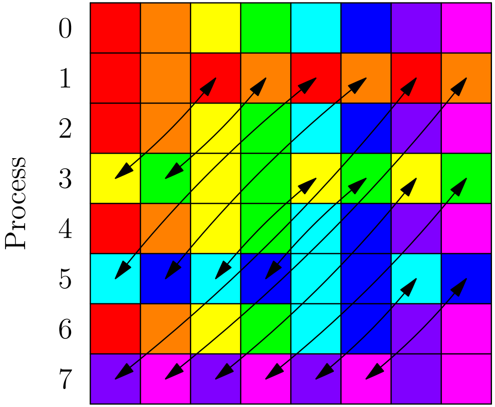
# Recursive Transpose



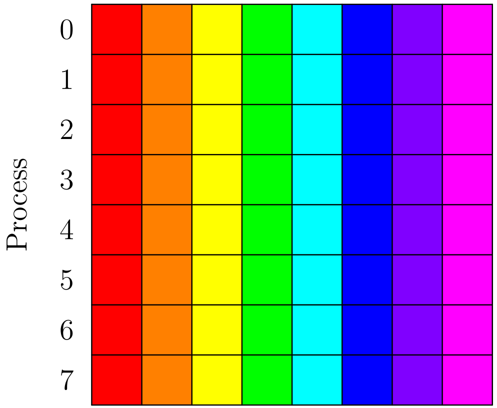
# Recursive Transpose



# 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.

## Hybrid Transpose

Let $\tau_\ell$ be the message latency, and $\tau_d$ the time to send one element. The time to send $n$ elements is

$$\tau_\ell + n\tau_d. \tag{16}$$

The time required to do a direct transpose is

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

The time for a block transpose is

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

## Hybrid Transpose

Let $L = \tau_\ell / \tau_d$.

For $P \gg 1$,

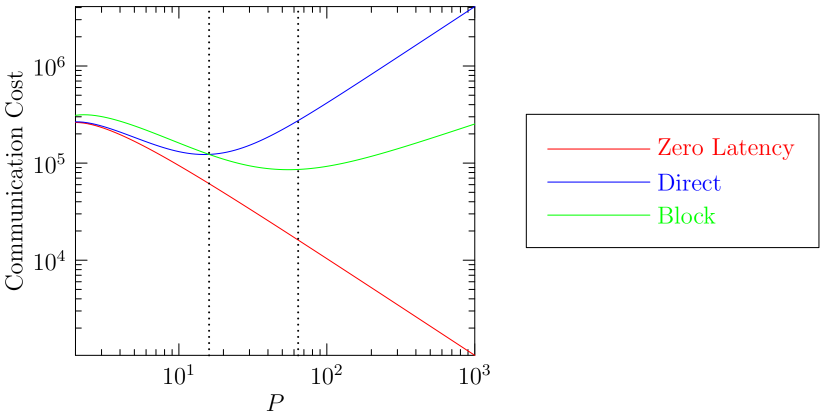
$$T_D \approx \tau_d \left( PL + \frac{m^2}{P} \right) \tag{19}$$

has a global minimum of $2\tau_d m\sqrt{L}$ at $P = m/\sqrt{L}$.

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

$$T_B(a = \sqrt{P}) \approx 2\tau_d \sqrt{P} \left( L + \frac{m^2}{P^{3/2}} \right) \tag{20}$$

# Hybrid Transpose



## Hybrid Transpose: multi-threading

We have $C$ cores per node and $S$ nodes.

We launch $P$ processes with $T$ threads with $PT = SC$.

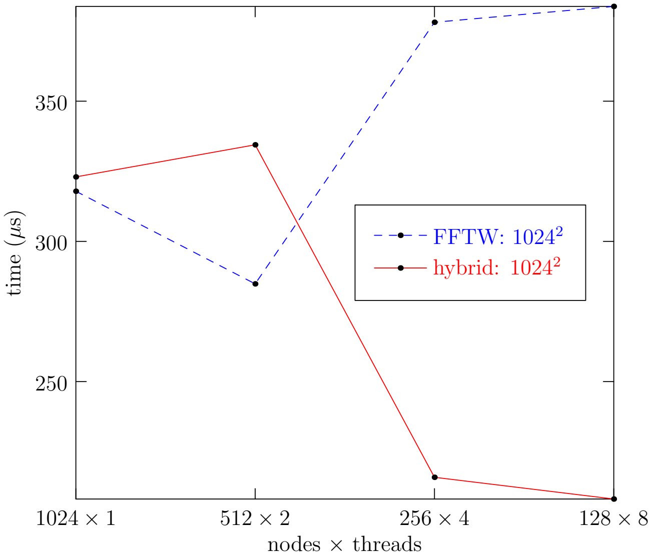
From minimizing $T_B$, the optimal number of threads is

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

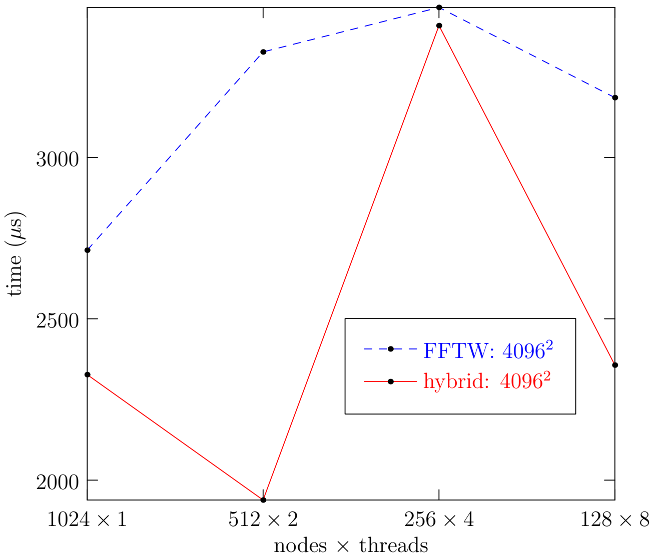
For `stampede` at the Texas Advanced Supercomputer Center, we measured $L = 4096$, so for $S = 128$ and $C = 8$,

- $T = 8$ for $m = 1024$
- $T = 2$ for $m = 4096$

## Hybrid Transpose



# Hybrid Transpose



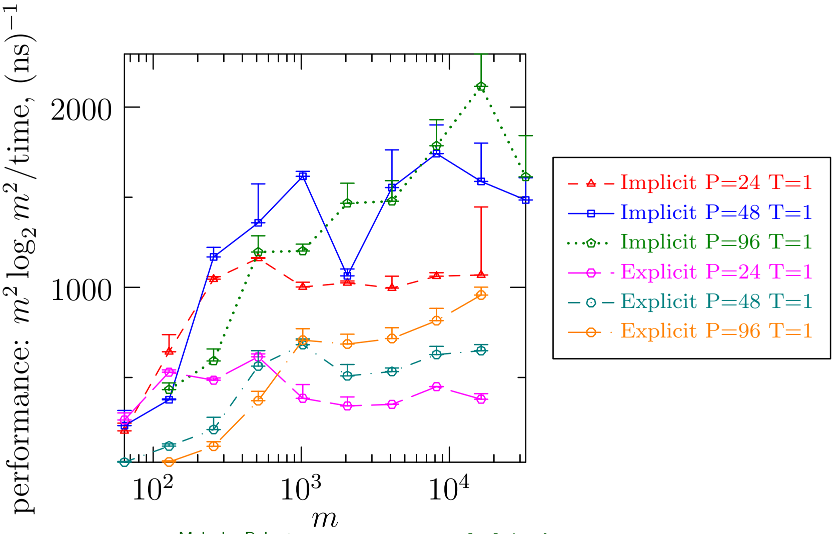
## Hybrid Transpose

The hybrid transpose

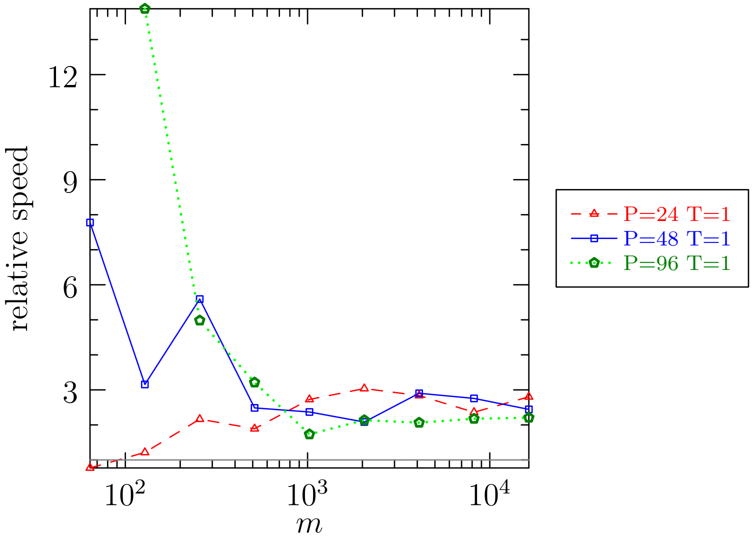
- Uses a direct transpose for large message sizes.
- Uses a block transpose for small message sizes.
- Offers a performance advantage when $P \approx m$.
- Can be tuned based upon the values of $\tau_\ell$ and $\tau_d$ for the cluster.
- Optimal number of threads Depends on the problem size and cluster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

We use the hybrid transpose in for computing convolutions using implicit dealiasing on clusters.

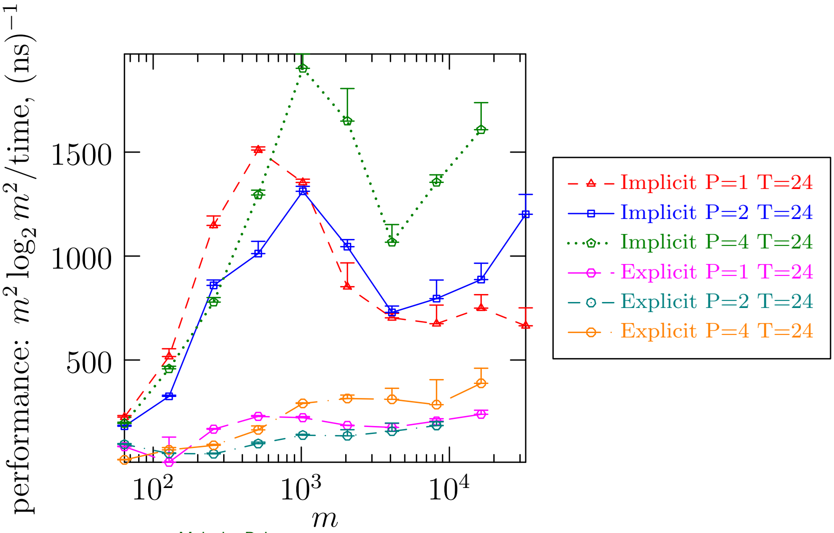
# MPI Convolution: 2D performance



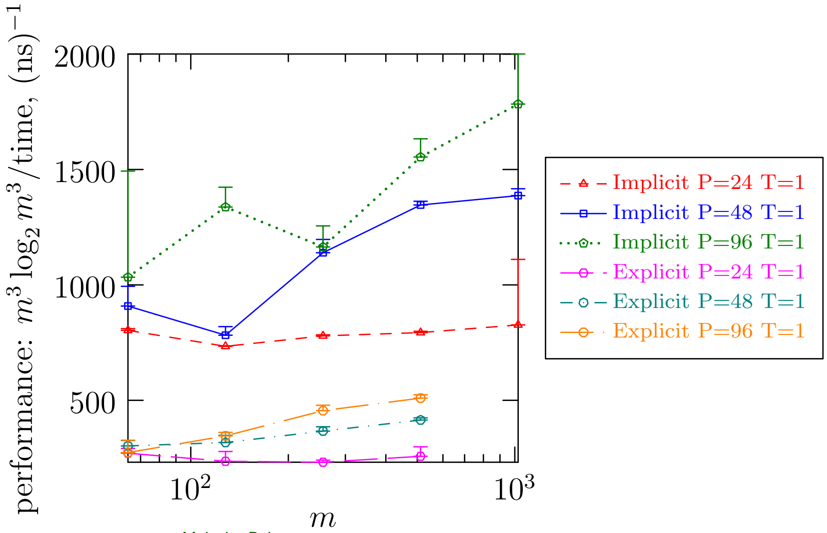
# `MPI` Convolution: 2D performance



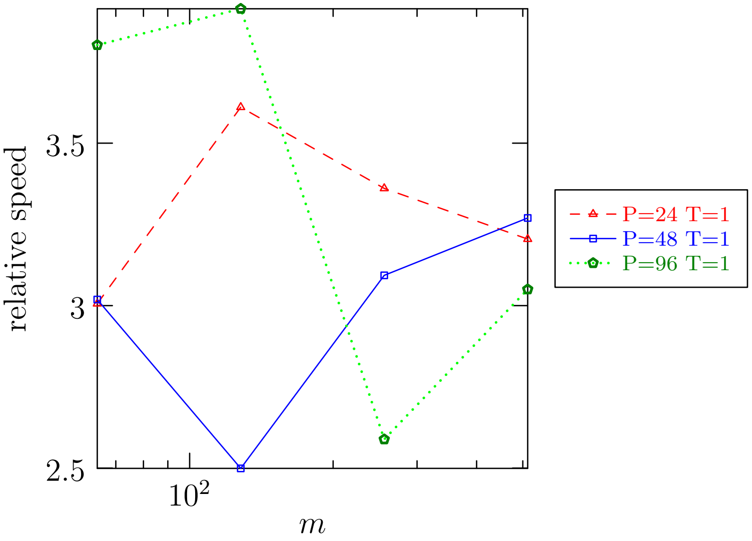
# MPI Convolution: multithreaded 2D performance



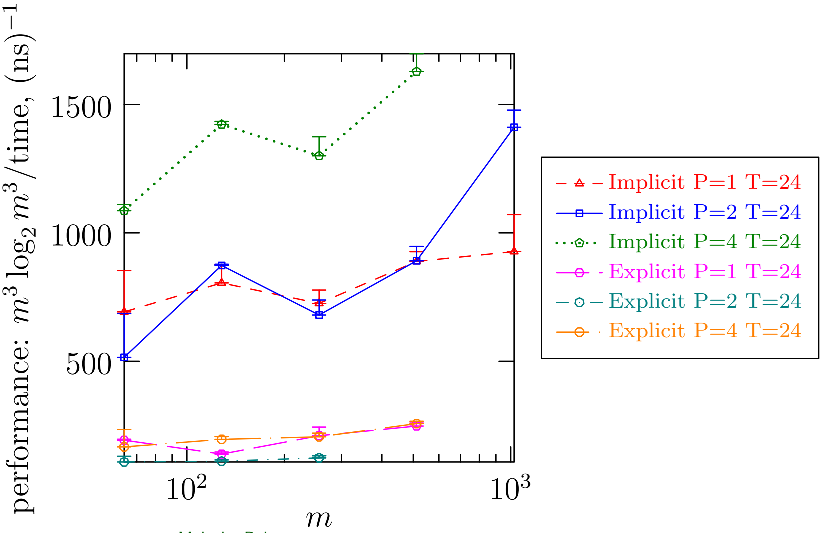
# MPI Convolution: 3D performance



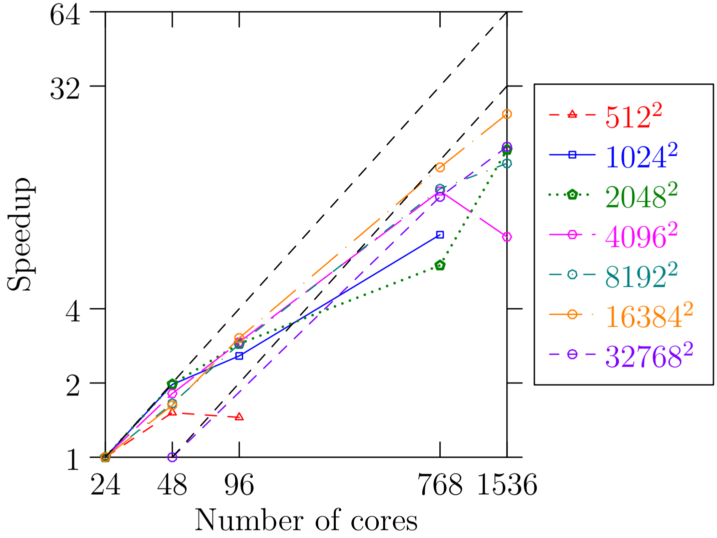
## MPI Convolution: 3D performance



# MPI Convolution: multithreaded 3D performance

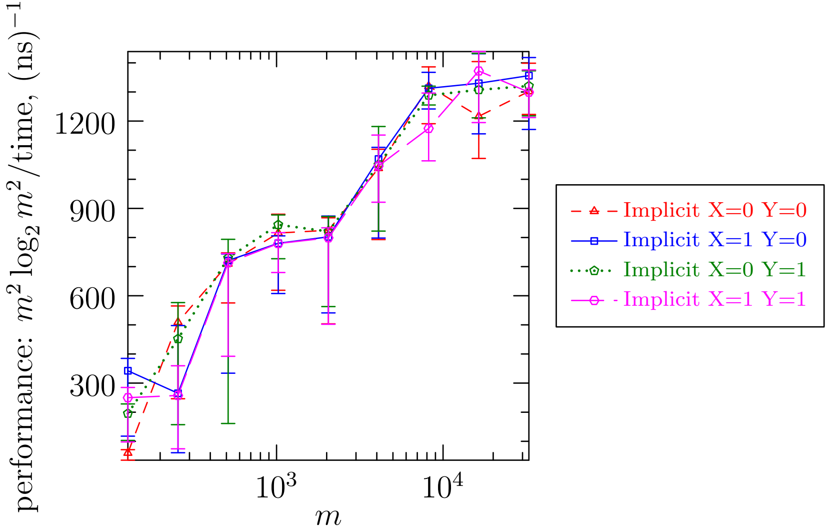


# MPI Convolution: 3D scaling



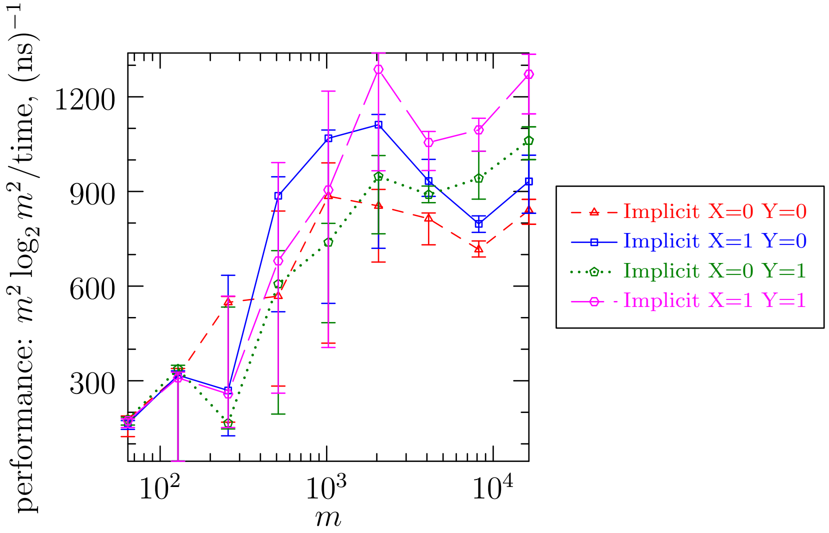
# 2/3 padding: 2D

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



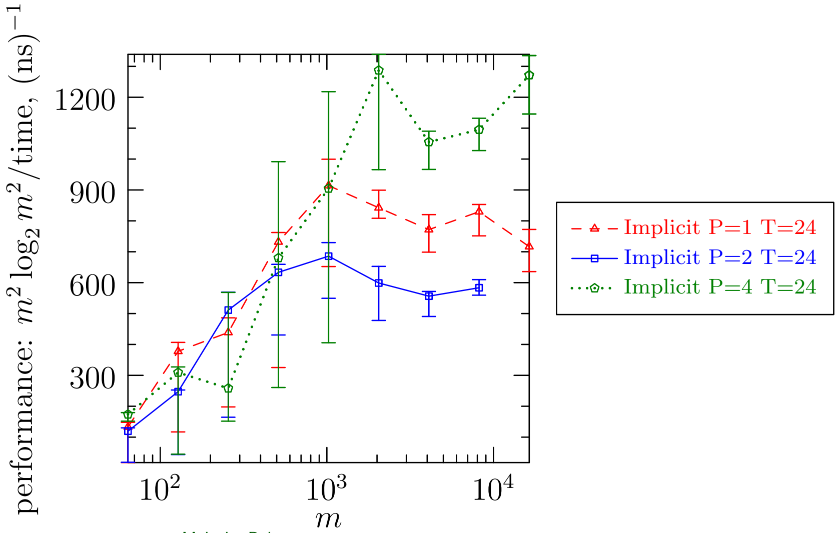
## 2/3 padding: 2D

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



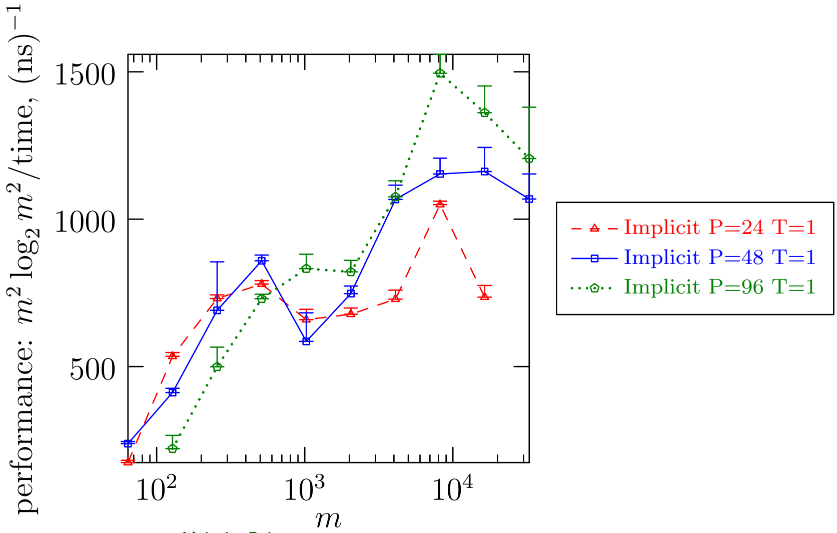
## 2/3 padding: 2D performance

Here we are non-compact in both directions:



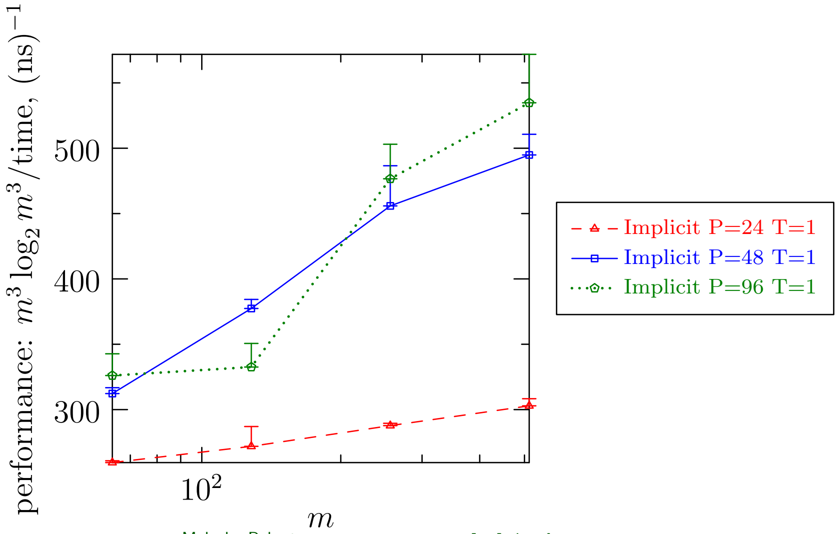
## 2/3 padding: 2D performance

Here we are non-compact in both directions:



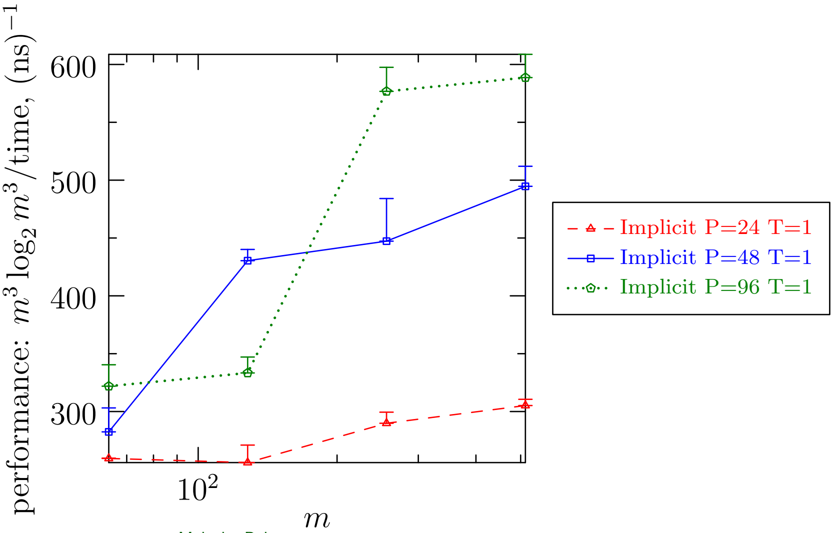
## 2/3 padding: 3D performance

Here we are non-compact in all three directions:



## 2/3 padding: 3D performance

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



## 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).