Multithreaded Implicitly Dealised Pseudospectral Convolutions

Malcolm Roberts and John C. Bowman

University of Alberta

May 2012
Pseudospectral simulations

- The incompressible 2D vorticity formulation

\[
\frac{\partial \omega}{\partial t} + (u \cdot \nabla) \omega = \nu \nabla^2 \omega
\]

Malcolm Roberts and John C. Bowman  
University of Alberta
The incompressible 2D vorticity formulation

\[ \frac{\partial \omega}{\partial t} + (u \cdot \nabla) \omega = \nu \nabla^2 \omega \]

is Fourier-transformed into

\[ \hat{\omega} \hat{k} \frac{\partial}{\partial t} = \sum_{\hat{p} + \hat{q} = \hat{k}} \hat{\epsilon}_{\hat{k}\hat{p}\hat{q}} \hat{\omega}^* \hat{p} \hat{\omega}^* \hat{q} - \nu \hat{k}^2 \hat{\omega} \hat{k} \hat{\epsilon}_{\hat{k}\hat{p}\hat{q}} = (\hat{z} \cdot \hat{p} \times \hat{q}) \delta(\hat{k} + \hat{p} + \hat{q}) \]

The nonlinearity becomes a convolution:

\[ (F * G) \hat{k} = \sum_{\hat{k}_1, \hat{k}_2} F \hat{k}_1 G \hat{k}_2 \delta(\hat{k}, \hat{k}_1, \hat{k}_2). \]
Pseudospectral simulations

- The incompressible 2D vorticity formulation

\[
\frac{\partial \omega}{\partial t} + (u \cdot \nabla) \omega = \nu \nabla^2 \omega
\]

is Fourier-transformed into

\[
\frac{\partial \omega_k}{\partial t} = \sum_{p+q=k} \frac{\epsilon_{kpq}}{q^2} \omega_p^* \omega_q^* - \nu k^2 \omega_k
\]

where

\[
\epsilon_{kpq} = (\hat{z} \cdot p \times q) \delta(k + p + q)
\]
The incompressible 2D vorticity formulation

\[ \frac{\partial \omega}{\partial t} + (u \cdot \nabla) \omega = \nu \nabla^2 \omega \]

is Fourier-transformed into

\[ \frac{\partial \omega_k}{\partial t} = \sum_{p+q=k} \frac{\epsilon_{kpq}}{q^2} \omega_p^* \omega_q^* - \nu k^2 \omega_k \]

\[ \epsilon_{kpq} = (\hat{z} \cdot p \times q) \delta(k + p + q) \]

The nonlinearity becomes a convolution:

\[ (F \ast G)_k = \sum_{k_1, k_2} F_{k_1} G_{k_2} \delta_{k, k_1, k_2}. \]
Non-centered data

- Input data: \( \{ F_k \}_{k=0}^{N-1} \) and \( \{ G_k \}_{k=0}^{N-1} \).

This produces non-centered convolutions:

\[
(F \ast G)_k = \sum_{\ell=0}^{k} F_{\ell} G_{k-\ell}
\]

For non-centered data, \( F \ast (G \ast H) = F \ast G \ast H \).
Non-centered data

Input data: \( \{ F_k \}_{k=0}^{N-1} \) and \( \{ G_k \}_{k=0}^{N-1} \).

This produces non-centered convolutions:

\[
(F \ast G)_k = \sum_{\ell=0}^{k} F_{\ell} G_{k-\ell}
\]
Non-centered data

- Input data: \( \{F_k\}_{k=0}^{N-1} \) and \( \{G_k\}_{k=0}^{N-1} \).

- This produces non-centered convolutions:

\[
(F \ast G)_k = \sum_{\ell=0}^{k} F_\ell G_{k-\ell}
\]

- For non-centered data, \( \ast(F, G, H) = F \ast (G \ast H) \).
Input data: \( \{ F_k \}_{k=-N+1}^{N-1} \) and \( \{ G_k \}_{k=-N+1}^{N-1} \).
Input data: \( \{ F_k \}_{k=-N+1}^{N-1} \) and \( \{ G_k \}_{k=-N+1}^{N-1} \).

\[
(F * G)_k = \min(N-1, k+N-1) \sum_{\ell=\max(-N+1, k-N+1)} F_\ell G_{k-\ell}
\]
Centered data

- Input data: \( \{F_k\}_{k=-N+1}^{N-1} \) and \( \{G_k\}_{k=-N+1}^{N-1} \).

\[
(F \ast G)_k = \min (N-1,k+N-1) \sum_{\ell = \max (-N+1,k-N+1)} F_\ell G_{k-\ell}
\]

- Considering Hermitian-symmetric data \((F_{-k} = F_k^\ast)\), we compute data for \(k \geq 0\), so

\[
(F \ast G)_k = \sum_{\ell = k-N+1}^{N-1} F_\ell G_{k-\ell}.
\]
Input data: \( \{ F_k \}_{k=-N+1}^{N-1} \) and \( \{ G_k \}_{k=-N+1}^{N-1} \).

\[
(F \ast G)_k = \min(N-1, k+N-1) \sum_{\ell=\max(-N+1, k-N+1)} F_{\ell} G_{k-\ell}
\]

Considering Hermitian-symmetric data \((F_{-k} = F_k^*)\), we compute data for \( k \geq 0 \), so

\[
(F \ast G)_k = \sum_{\ell=k-N+1}^{N-1} F_{\ell} G_{k-\ell}.
\]

For centered data, \((F, G, H) \neq F \ast (G \ast H)\).
The convolution sum involves $O(N^2)$ terms. Using FFTs, we can compute a convolution in $O(N \log N)$ operations.
The convolution sum involves $O(N^2)$ terms. Using FFTs, we can compute a convolution in $O(N \log N)$ operations.

FFTs produce cyclic convolutions. Linear convolutions are attained if one zero-pads the input data.
The convolution sum involves $O(N^2)$ terms. Using FFTs, we can compute a convolution in $O(N \log N)$ operations.

FFTs produce cyclic convolutions. Linear convolutions are attained if one zero-pads the input data.

Non-centered data is padded from length $N$ to length $2N$. 
FFT-based convolutions

- The convolution sum involves $\mathcal{O}(N^2)$ terms. Using FFTs, we can compute a convolution in $\mathcal{O}(N \log N)$ operations.

- FFTs produce cyclic convolutions. Linear convolutions are attained if one zero-pads the input data.

- Non-centered data is padded from length $N$ to length $2N$.

- Centered data is padded from length $2N - 1$ to length $3N$. 
Implicit padding involves using a separate work array to compute the DFT:

\[
f_x = \sum_{k=0}^{2N-1} \zeta_{2N}^{xk} F_k, \quad F_k = 0 \text{ if } k \geq N
\]
Implicit Zero-padding

Implicit padding involves using a separate work array to compute the DFT:

\[ f_x = \sum_{k=0}^{2N-1} \zeta_x^{k} F_k, \quad F_k = 0 \text{ if } k \geq N \]

is attained by computing
Implicit Zero-padding

Implicit padding involves using a separate work array to compute the DFT:

\[ f_x = \sum_{k=0}^{2N-1} \zeta^{xk}_{2N} F_k, \quad F_k = 0 \text{ if } k \geq N \]

is attained by computing

\[ f_{2x} = \sum_{k=0}^{N-1} \zeta^{xk}_N F_k \]
Implicit Zero-padding

Implicit padding involves using a separate work array to compute the DFT:

\[ f_x = \sum_{k=0}^{2N-1} \zeta_{2N}^{xk} F_k, \quad F_k = 0 \text{ if } k \geq N \]

is attained by computing

\[ f_{2x} = \sum_{k=0}^{N-1} \zeta_N^{xk} F_k \]

and

\[ f_{2x+1} = \sum_{k=0}^{N-1} \zeta_N^{xk} (\zeta_{2N}^x F_k) \]
Implicit Zero-padding

$F$

$G$
Implicit Zero-padding

$F^{-1}_x[F]$

$F^{-1}_x[F]$

$F^{-1}_x[G]$

$F^{-1}_x[G]$

Malcolm Roberts and John C. Bowman
University of Alberta
Implicit Zero-padding
Implicit Zero-padding

\[ F^{-1}[F \ast G] \]

Malcolm Roberts and John C. Bowman
University of Alberta
Implicit Zero-padding
Implicit Zero-padding

\[ F^{-1} \ast [F \ast G] \]

Malcolm Roberts and John C. Bowman
University of Alberta
Implicit Zero-padding

\[ F \ast G \]
Work memory required for an $n$-dimensional non-centered convolution:

<table>
<thead>
<tr>
<th>$n$</th>
<th>Explicit</th>
<th>Implicit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$2N_x$</td>
<td>$2N_x$</td>
</tr>
<tr>
<td>2</td>
<td>$6N_xN_y$</td>
<td>$2N_xN_y + 2PN_y$</td>
</tr>
<tr>
<td>3</td>
<td>$14N_xN_yN_z$</td>
<td>$2N_xN_yN_z + 2PN_yN_z$</td>
</tr>
</tbody>
</table>
## Memory requirements

Work memory required for an $n$-dimensional non-centered convolution:

<table>
<thead>
<tr>
<th>$n$</th>
<th>Explicit</th>
<th>Implicit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$2N_x$</td>
<td>$2N_x$</td>
</tr>
<tr>
<td>2</td>
<td>$6N_xN_y$</td>
<td>$2N_xN_y + 2PN_y$</td>
</tr>
<tr>
<td>3</td>
<td>$14N_xN_yN_z$</td>
<td>$2N_xN_yN_z + 2PN_yN_z$</td>
</tr>
</tbody>
</table>

Work memory required for an $n$-dimensional centered convolution:

<table>
<thead>
<tr>
<th>$n$</th>
<th>Explicit</th>
<th>Implicit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$2N_x$</td>
<td>$2N_x$</td>
</tr>
<tr>
<td>2</td>
<td>$5N_xN_y$</td>
<td>$2N_xN_y + PN_y$</td>
</tr>
<tr>
<td>3</td>
<td>$19N_xN_yN_z$</td>
<td>$4N_xN_yN_z + 2PN_xN_y$</td>
</tr>
</tbody>
</table>
Performance: multiple threads

Non-centered 1D convolution.
Performance: multiple threads

Non-centered 2D convolution.

Malcolm Roberts and John C. Bowman

University of Alberta
Performance: multiple threads

Non-centered 3D convolution.

Malcolm Roberts and John C. Bowman  
University of Alberta
Performance: multiple threads

Centered 1D convolution.

Malcolm Roberts and John C. Bowman
University of Alberta
Performance: multiple threads

Centered 2D convolution.

Malcolm Roberts and John C. Bowman
University of Alberta
Performance: multiple threads

Centered 3D convolution.

Malcolm Roberts and John C. Bowman  
University of Alberta
Performance: multiple threads

Centered ternary 1D convolution.
Performance: multiple threads

Centered ternary 2D convolution.

Malcolm Roberts and John C. Bowman
University of Alberta
Performance: multiple threads

- One-dimensional convolutions on four cores are about 2 times as fast as on one core.
Performance: multiple threads

- One-dimensional convolutions on four cores are about 2 times as fast as on one core.

- Two-dimensional convolutions on four cores are about 3 times as fast.
Performance: multiple threads

- One-dimensional convolutions on four cores are about 2 times as fast as on one core.

- Two-dimensional convolutions on four cores are about 3 times as fast.

- Three-dimensional convolutions on four cores are about 3.5 times as fast.
Performance: explicit vs. implicit

Non-centered 1D convolution.

Malcolm Roberts and John C. Bowman
University of Alberta
Performance: explicit vs. implicit

Non-centered 2D convolution.

Malcolm Roberts and John C. Bowman

University of Alberta
Performance: explicit vs. implicit

Non-centered 3D convolution.

Malcolm Roberts and John C. Bowman
University of Alberta
Performance: explicit vs. implicit

Centered 1D convolution.

Malcolm Roberts and John C. Bowman  
University of Alberta
Performance: explicit vs. implicit

Centered 2D convolution.

Malcolm Roberts and John C. Bowman
University of Alberta
Performance: explicit vs. implicit

Centered ternary 1D convolution.

Malcolm Roberts and John C. Bowman  
University of Alberta
Centered ternary 2D convolution.
Summary of Results

- Implicit methods require much less work memory than is required by explicit methods.
Summary of Results

- Implicit methods require much less work memory than is required by explicit methods.

- The implicit method had a speedup of up to 3.6 on four cores, while the explicit method sped-up of up to a factor of 3.
Summary of Results

- Implicit methods require much less work memory than is required by explicit methods.

- The implicit method had a speedup of up to 3.6 on four cores, while the explicit method sped-up of up to a factor of 3.

- The implicit method is around twice as fast as the explicit method for multidimensional convolutions.
Computing the nonlinear source of the 2D incompressible Navier–Stokes equations in a vorticity formulation, which appears in Fourier space as

$$\sum_{\mathbf{p}} \frac{p_x k_y - p_y k_x}{|\mathbf{k} - \mathbf{p}|^2} \omega_{\mathbf{p}} \omega_{\mathbf{k} - \mathbf{p}},$$

is performed as follows:

$$\text{conv2} \left( ik_x \omega, ik_y \omega, ik_y \omega/|\mathbf{k}|^2, -ik_x \omega/|\mathbf{k}|^2 \right).$$

One also has the option of passing work arrays to `conv2`, which can then be used elsewhere.
Computing the nonlinear source of the 2D incompressible Navier–Stokes equations in a vorticity formulation, which appears in Fourier space as

$$\sum_p p_x k_y - p_y k_x \frac{\omega_p \omega_{k-p}}{|k-p|^2},$$

is performed as follows:

$$\text{conv2}(ik_x \omega, ik_y \omega, ik_y \omega/k^2, -ik_x \omega/k^2).$$
Computing the nonlinear source of the 2D incompressible Navier–Stokes equations in a vorticity formulation, which appears in Fourier space as

$$
\sum_p p_x k_y - p_y k_x \frac{\omega_p \omega_{k-p}}{|k-p|^2},
$$

is performed as follows:

$$\text{conv2}(ik_x \omega, ik_y \omega, ik_y \omega/k^2, -ik_x \omega/k^2).$$

One also has the option of passing work arrays to conv2, which can then be used elsewhere.
Implicitly zero-padding multi-dimensional convolutions is faster and requires less memory than explicit routines.
Conclusion

- Implicitly zero-padding multi-dimensional convolutions is faster and requires less memory than explicit routines.
- The algorithm has been successfully implemented on a shared-memory architecture with only a small increase in work memory.
Conclusion

- Implicitly zero-padding multi-dimensional convolutions is faster and requires less memory than explicit routines.
- The algorithm has been successfully implemented on a shared-memory architecture with only a small increase in work memory.
- Convolution algorithms are available for complex non-centered data and centered Hermitian-symmetric data in 1D, 2D, and 3D.
Implicitly zero-padding multi-dimensional convolutions is faster and requires less memory than explicit routines.

The algorithm has been successfully implemented on a shared-memory architecture with only a small increase in work memory.

Convolution algorithms are available for complex non-centered data and centered Hermitian-symmetric data in 1D, 2D, and 3D.

Ternary convolution algorithms are available for centered Hermitian-symmetric in 1D and 2D.
Future work

- Develop a distributed-memory implementation based on openMPI.
Future work

- Develop a distributed-memory implementation based on openMPI.

- Add additional routines, such as convolutions on real data, self-convolution, correlations, etc.
Resources

FFTW++:
http://fftwpp.sourceforge.net

Asymptote:
http://asymptote.sourceforge.net

Malcolm Roberts:
http://www.math.ualberta.ca/~mroberts