Adaptive Transpose Algorithms for Distributed Multicore Processors

John C. Bowman and Malcolm Roberts
University of Alberta and Université de Strasbourg

April 15, 2016

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

Acknowledgement: Wendell Horton, Institute for Fusion Studies
Matrix Transposes

- The matrix transpose is an essential primitive of high-performance parallel computing.
Matrix Transposes

• The matrix transpose is an essential primitive of high-performance parallel computing.

• Transposes are used to localize the computation of multidimensional fast Fourier transforms onto individual processors.
The performance of various transpose algorithms depends on:

- communication bandwidth
- communication latency
- network congestion
- communication packet size
- local cache size
- network topology
The performance of various transpose algorithms depends on:

- communication bandwidth
- communication latency
- network congestion
- communication packet size
- local cache size
- network topology

It is hard to estimate the relative importance of these factors at compilation time.
• The performance of various transpose algorithms depends on:
  – communication bandwidth
  – communication latency
  – network congestion
  – communication packet size
  – local cache size
  – network topology

• It is hard to estimate the relative importance of these factors at compilation time.

• An adaptive algorithm, dynamically tuned to take advantage of these specific architectural details, is desirable.
8 × 8 Matrix Transpose over 8 Processors
8 × 8 Matrix Transpose over 8 Processors
8 × 8 Matrix Transpose over 8 Processors
Direct (All-to-All)

- Advantages:
  - efficient for $N \gg P$ (large messages);
  - most direct.
Direct (All-to-All)

- Advantages:
  - efficient for $N \gg P$ (large messages);
  - most direct.

- Disadvantages:
  - many small message sizes when $P \geq N$. 
Direct (All-to-All)

• Advantages:
  - efficient for $N \gg P$ (large messages);
  - most direct.

• Disadvantages:
  - many small message sizes when $P \geq N$.

• Implementation:
  - MPI_ALLTOALL, MPI_SEND/MPI_RECV.
Recursive (Butterfly)

- Advantages:
  - efficient for $N \ll P$ (small messages);
  - recursively subdivides transpose into smaller block transposes;
  - $\log N$ phases;
  - groups messages together to reduce communication latency.
Recursive (Butterfly)

• Advantages:
  - efficient for $N \ll P$ (small messages);
  - recursively subdivides transpose into smaller block transposes;
  - $\log N$ phases;
  - groups messages together to reduce communication latency.

• Disadvantages:
  - requires intermediate communications.
Recursive (Butterfly)

• Advantages:
  – efficient for $N \ll P$ (small messages);
  – recursively subdivides transpose into smaller block transposes;
  – $\log N$ phases;
  – groups messages together to reduce communication latency.

• Disadvantages:
  – requires intermediate communications.

• Implementation:
  – FFTW
8 × 8 Block Transpose over 8 processors
8 × 8 Block Transpose over 8 processors
### 8 × 8 Block Transpose over 8 processors

<table>
<thead>
<tr>
<th>Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
</tbody>
</table>

The diagram illustrates the block transpose operation across 8 processors, arranged in a 8x8 grid. Each color represents a different processor, and the layout shows the data distribution across the 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
Adaptive

- Advantages:
  - best of both worlds
  - uses subdivision at highest level to increase communication block size;
    - *optimally* groups messages together to reduce communication latency.
    - directly communicates several sub-blocks at a time.
Adaptive

● Advantages:
  - best of both worlds
  - uses subdivision at highest level to increase communication block size;
    - \textit{optimally} groups messages together to reduce communication latency.
    - directly communicates several sub-blocks at a time.

● Implementation:
  - FFTW (sub-optimal), FFTW++ (quasi-optimal).
Hybrid Parallel Architectures (nodes \( \times \) threads)

- Advantages:
  - MPI between nodes / OpenMP within node;
  - exploits modern multicore architectures;
  - many algorithms can use memory striding to avoid local transposition within a node;
  - compatible with modern trend of less memory/core;
Hybrid Parallel Architectures (nodes $\times$ threads)

- **Advantages:**
  - MPI between nodes / OpenMP within node;
  - exploits modern multicore architectures;
  - many algorithms can use memory striding to avoid local transposition within a node;
  - compatible with modern trend of less memory/core;

- **Disadvantages:**
  - requires both OpenMP and MPI support.
Communication Costs: Direct Transpose

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

  – Requires $b$ communications per process, each of size $(NM/a)/b^2 = aNM/P^2$, for a total per-process data transfer of $(b - 1)aNM/P^2$. 
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.
  
  - Requires $b$ communications per process, each of size $(NM/a)/b^2 = aNM/P^2$, for a total per-process data transfer of $(b - 1)aNM/P^2$.

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

  - Requires $a$ communications per process, each of size $(NM/b)/a^2 = bNM/P^2$, for a total per-process data transfer of $(a - 1)bNM/P^2$. 
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

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

$$T_B(a) = \tau_\ell \left( a + \frac{P}{a} - 2 \right) + \tau_d \left( 2P - a - \frac{P}{a} \right) \frac{NM}{P^2}.$$
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 $NM \geq P^2 L$, whereas a block transpose should be used when $NM < P^2 L$. 
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 \( NM \geq P^2 L \), whereas a block transpose should be used when \( NM < P^2 L \).

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

\[ T'_B(a) = \tau_d \left( 1 - \frac{P}{a^2} \right) \left( L - \frac{NM}{P^2} \right). \]
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 \( NM \geq P^2 L \), whereas a block transpose should be used when \( NM < P^2 L \).

• 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 \( NM < P^2 L \), 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

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

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

$$P \approx (2NM/L)^{2/3}.$$
Optimal Number of Threads

- The minimum value of $T_B$ is

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

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

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

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

Transpose Communication Costs

![Graph showing communication costs with various latency and threading options.](image-url)
Implementation

- Choose optimal block size $b$ that minimizes effects of communication latency.
Implementation

• Choose optimal block size $b$ that minimizes effects of communication latency.

• Use hybrid OpenMPI/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.
Implementation

• Choose optimal block size $b$ that minimizes effects of communication latency.

• Use hybrid OpenMPI/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.

• The use of nonblocking MPI communications allows us to overlap computation with communication: this can yield an additional 10–30% performance gain for 3D convolutions.
1024 × 1024 Transpose over 1024 Processors

![Graph showing time (µs) vs. nodes × threads for 1024 × 1024 transpose operations using FFTW and hybrid methods.]
4096 × 4096 Transpose over 4096 Processors

![Graph showing time (µs) vs nodes × threads for FFTW and hybrid methods. The graph includes points for 1024 × 1, 512 × 2, 256 × 4, and 128 × 8, with FFTW represented by a dashed blue line and hybrid represented by a red line.](image)
Applications

- FFT in 2 & higher dimensions
Applications

- FFT in 2 & higher dimensions
- Pseudospectral Collocation
  - Explicit Dealiasing via Zero Padding
  - Implicit Dealiasing
Conclusions

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

- 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.
Conclusions

• 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.
References