Communication-Avoiding Algorithms

Grégoire Pichon, Bora Uçar & Frédéric Vivien
(Original slides by Loris Marchal)

CNRS, INRIA, Université Lyon 1 & ENS Lyon

CR15: January 2023
https://gpichon.gitlabpages.inria.fr/m2if-numerical_algorithms/
Yet Another Motivation...

...for limiting communications

Source: John Shalf, LBL
Communication-Avoiding Algorithms

**Context:** Distributed Memory

![Diagram showing distributed memory model with processors and memories connected to a disk]

**Communications:** Data movements between:
- one processor and its memory
- different processors/memories

**Objective:**
- Derive communication *lower bounds* for many linear algebra operations
- Design communication-optimal algorithms
Context: Single processor + Memory (size $M$)

- Analysis in phases of $M$ I/O operations
- Bound on the number of elementary product in each phase: $F = O(M^{3/2})$
  
  **Geometric argument:** Loomis-Whitney inequality
- At least $n^3/F$ phases, of $M$ I/Os, in total: $\Omega(n^3/\sqrt{M})$ I/Os
Communication-Avoiding Algorithms

1. Generalization to other Linear Algebra Algorithms
   - Generalized Matrix Computations
   - I/O Analysis
   - Application to LU Factorization

2. Analysis and Lower Bounds for Parallel Algorithms
   - Matrix Multiplication Lower Bound for $P$ processors
   - 2D and 3D Algorithms for Matrix Multiplication
   - 2.5D Algorithm for Matrix Multiplication

3. Conclusion
Communication-Avoiding Algorithms

1. Generalization to other Linear Algebra Algorithms
   - Generalized Matrix Computations
   - I/O Analysis
   - Application to LU Factorization

2. Analysis and Lower Bounds for Parallel Algorithms
   - Matrix Multiplication Lower Bound for $P$ processors
   - 2D and 3D Algorithms for Matrix Multiplication
   - 2.5D Algorithm for Matrix Multiplication

3. Conclusion
Generalization to other Linear Algebra Algorithms

- Inputs/Output: $n \times n$ matrices $A, B, C$
- Any mapping of the matrices to the memory (possibly overlapping)
Generalization to other Linear Algebra Algorithms

- Inputs/Output: $n \times n$ matrices $A, B, C$
- Any mapping of the matrices to the memory (possibly overlapping)

**General computation**

For all $(i, j) \in S_C$,

$$C_{i,j} \leftarrow f_{i,j} \left( g_{i,j,k}(A_{i,k}, B_{k,j}) \text{ for } k \in S_{i,j}, \text{ any other arguments} \right)$$
Generalization to other Linear Algebra Algorithms

- Inputs/Output: $n \times n$ matrices $A, B, C$
- Any mapping of the matrices to the memory (possibly overlapping)

**General computation**

For all $(i, j) \in S_C$,

$$C_{i,j} \leftarrow f_{i,j}\left(g_{i,j,k}(A_{i,k}, B_{k,j}) \text{ for } k \in S_{i,j}, \text{ any other arguments}\right)$$

- For matrix multiplication:
Generalization to other Linear Algebra Algorithms

- Inputs/Output: $n \times n$ matrices $A, B, C$
- Any mapping of the matrices to the memory (possibly overlapping)

General computation

For all $(i, j) \in S_C,$

$$C_{i,j} \leftarrow f_{i,j}(g_{i,j,k}(A_{i,k}, B_{k,j}) \text{ for } k \in S_{i,j}, \text{ any other arguments})$$

- For matrix multiplication:
  - $f_{i,j}$: summation, $g_{i,j,k}$: product
  - $S_{i,j} = [1, n], \ S_C = [1, n] \times [1, n]$
Generalized Matrix Computations

General computation

For all \((i, j) \in S_C\),

\[
C_{i,j} \leftarrow f_{i,j}\left( g_{i,j,k}(A_{i,k}, B_{k,j}) \text{ for } k \in S_{i,j}, \text{ any other arguments} \right)
\]

- \(f_{i,j}\) and \(g_{i,j,k}\) non-trivial:
  - \(g_{i,j,k}\) needs to load the value of \(A_{i,k}\) and \(B_{k,j}\) in memory
  - \(f_{i,j}\) needs at least an “accumulator” while results of \(g_{i,j,k}(\ldots)\) are loaded/computed in memory one after the other

- \(S_C, S_{i,j}, f_{i,j}, g_{i,j,k}\) possibly determined at runtime

- Correct computations may require special ordering of computations:
  - no such constraint needed for the lower bound:
    - any order for computing the \(g_{i,j,k}\)’s
    - any order for computing and storing the \(f_{i,j}\)’s
Generalized Matrix Computations

General computation

For all \((i, j) \in S_C\),

\[
C_{i,j} \leftarrow f_{i,j}\left(g_{i,j,k}(A_{i,k}, B_{k,j}) \text{ for } k \in S_{i,j}, \text{ any other arguments}\right)
\]

- \(f_{i,j}\) and \(g_{i,j,k}\) non-trivial:
  - \(g_{i,j,k}\) needs to load the value of \(A_{i,k}\) and \(B_{k,j}\) in memory
  - \(f_{i,j}\) needs at least an “accumulator” while results of \(g_{i,j,k}(\ldots)\) are loaded/computed in memory one after the other

- \(S_C, S_{i,j}, f_{i,j}, g_{i,j,k}\) possibly determined at runtime

- Correct computations may require special ordering of computations: no such constraint needed for the lower bound:
  - any order for computing the \(g_{i,j,k}\)’s
  - any order for computing and storing the \(f_{i,j}\)’s
Generalized Matrix Computations

General computation

For all \((i, j) \in S_C\),

\[
C_{i,j} \leftarrow f_{i,j}\left(g_{i,j,k}(A_{i,k}, B_{k,j}) \text{ for } k \in S_{i,j}, \text{ any other arguments}\right)
\]

- \(f_{i,j}\) and \(g_{i,j,k}\) non-trivial:
  - \(g_{i,j,k}\) needs to load the value of \(A_{i,k}\) and \(B_{k,j}\) in memory
  - \(f_{i,j}\) needs at least an “accumulator” while results of \(g_{i,j,k}(\ldots)\) are loaded/computed in memory one after the other

- \(S_C, S_{i,j}, f_{i,j}, g_{i,j,k}\) possibly determined at runtime

- Correct computations may require special ordering of computations:
  - no such constraint needed for the lower bound:
    - any order for computing the \(g_{i,j,k}\)’s
    - any order for computing and storing the \(f_{i,j}\)’s
Geometric analysis

Analysis based on Loomis-Whitney inequality:

**Theorem (Discrete Loomis-Whitney Inequality)**

*Let $V$ be a finite subset of $\mathbb{Z}^3$ and $V_1, V_2, V_3$ denote the orthogonal projections of $V$ on each coordinate planes, we have:*

$$|V|^2 \leq |V_1| \cdot |V_2| \cdot |V_3|,$$
I/O Analysis

One phase: \( M \) I/Os operations (loads and stores)

Classify operands based on their root and destination:

- **R1**: operands present in fast memory at the beginning of the phase or loaded during the phase (at most \( 2M \) such operands)
- **R2**: operands computed during the phase
- **D1**: operands left in fast memory at the end of the phase or written (at most \( 2M \) such operands)
- **D2**: operands discarded

Forget about R2/D2 operands

At most \( 4M \) operands available in one phase, for each matrix

Loomis-Whitney \( \Rightarrow \) at most \( F = \sqrt{(4M)^2} \) computations of \( g \)

Total number of loads and stores:

\[
M \left| \frac{\mathcal{G}}{F} \right| \leq M \left| \frac{\mathcal{G}}{\sqrt{(4M)^2}} \right| \leq \frac{\mathcal{G}}{8N/M} 
\]
I/O Analysis

One phase: $M$ I/Os operations (loads and stores)

Classify operands based on their root and destination:

- **R1**: operands present in fast memory at the beginning of the phase or loaded during the phase (at most $2M$ such operands)
- **R2**: operands computed during the phase
- **D1**: operands left in fast memory at the end of the phase or written (at most $2M$ such operands)
- **D2**: operands discarded

- Forget about R2/D2 operands
- At most $4M$ operands available in one phase, for each matrix
- Loomis-Whitney: at most $F = ((4M)^{2/3} \cdot \text{computations of } g)^{1/3}$
- Total number of loads and stores:

$$M \geq \frac{G}{\sqrt[3]{(4M)^{2/3}}} \geq \frac{G}{\sqrt[3]{RVM}} - M$$
I/O Analysis

One phase: $M$ I/Os operations (loads and stores)

Classify operands based on their root and destination:

- **R1**: operands present in fast memory at the beginning of the phase or loaded during the phase (at most $2M$ such operands)
- **R2**: operands computed during the phase
- **D1**: operands left in fast memory at the end of the phase or written (at most $2M$ such operands)
- **D2**: operands discarded

Forget about R2/D2 operands

At most $4M$ operands available in one phase, for each matrix

- Loomis-Whitney $\Rightarrow$ at most $F = \sqrt{(4M)^3}$ computations of $g$
- Total number of loads and stores:

$$M \left\lceil \frac{G}{F} \right\rceil \geq M \left\lceil \frac{G}{\sqrt{(4M)^3}} \right\rceil \geq \frac{G}{8\sqrt{M}} - M$$
I/O Analysis

One phase: $M$ I/Os operations (loads and stores)

Classify operands based on their root and destination:

- **R1**: operands present in fast memory at the beginning of the phase or loaded during the phase (at most $2M$ such operands)
- **R2**: operands computed during the phase
- **D1**: operands left in fast memory at the end of the phase or written (at most $2M$ such operands)
- **D2**: operands discarded

Forget about R2/D2 operands

At most $4M$ operands available in one phase, for each matrix

- Loomis-Whitney $\Rightarrow$ at most $F = \sqrt{(4M)^3}$ computations of $g$
- Total number of loads and stores:

$$M \left\lceil \frac{G}{F} \right\rceil \geq M \left\lceil \frac{G}{\sqrt{(4M)^3}} \right\rceil \geq \frac{G}{8\sqrt{M}} - M$$
I/O Analysis

One phase: $M$ I/Os operations (loads and stores)

Classify operands based on their root and destination:

- **R1**: operands present in fast memory at the beginning of the phase or loaded during the phase (at most $2M$ such operands)
- **R2**: operands computed during the phase
- **D1**: operands left in fast memory at the end of the phase or written (at most $2M$ such operands)
- **D2**: operands discarded

- Forget about R2/D2 operands
- At most $4M$ operands available in one phase, for each matrix
- Loomis-Whitney $\Rightarrow$ at most $F = \sqrt{(4M)^3}$ computations of $g$

Total number of loads and stores:

$$M \left\lceil \frac{G}{F} \right\rceil \geq M \left\lceil \frac{G}{\sqrt{(4M)^3}} \right\rceil \geq \frac{G}{8\sqrt{M}} - M$$
I/O Analysis

One phase: \( M \) I/Os operations (loads and stores)

Classify operands based on their root and destination:

- **R1**: operands present in fast memory at the beginning of the phase or loaded during the phase (at most \( 2M \) such operands)
- **R2**: operands computed during the phase
- **D1**: operands left in fast memory at the end of the phase or written (at most \( 2M \) such operands)
- **D2**: operands discarded

Forget about R2/D2 operands

At most \( 4M \) operands available in one phase, for each matrix

Loomis-Whitney \( \Rightarrow \) at most \( F = \sqrt{(4M)^3} \) computations of \( g \)

Total number of loads and stores:

\[
M \left[ \frac{G}{F} \right] \geq M \left[ \frac{G}{\sqrt{(4M)^3}} \right] \geq \frac{G}{8\sqrt{M}} - M
\]
Application to LU Factorization (1/2)

LU factorization (Gaussian elimination):
- Convert a matrix $A$ into product $L \times U$
- $L$ is lower triangular with diagonal 1
- $U$ is upper triangular
- $(L - D + U)$ stored in place with $A$

LU Algorithm

For $k = 1 \ldots n - 1$:
- For $i = k + 1 \ldots n$,
  $A_{i,k} \leftarrow A_{i,k}/A_{k,k}$ (column/panel preparation)
- For $i = k + 1 \ldots n$,
  For $j = k + 1 \ldots n$,
  $A_{i,j} \leftarrow A_{i,j} - A_{i,k}A_{k,j}$ (update)
Can be expressed as follows:

\[ U_{i,j} = A_{i,j} - \sum_{k < i} L_{i,k} \cdot U_{k,j} \quad \text{for } i \leq j \]

\[ L_{i,j} = \left( A_{i,j} - \sum_{k < j} L_{i,k} \cdot U_{k,j} \right) / U_{j,j} \quad \text{for } i > j \]

Fits the generalized matrix computations:

\[ C(i,j) = f_{i,j} \left( g_{i,j,k}(A(i,k), B(k,j)) \right) \text{ for } k \in S_{i,j}, K \]

with:
Can be expressed as follows:

\[ U_{i,j} = A_{i,j} - \sum_{k<i} L_{i,k} \cdot U_{k,j} \]

\[ L_{i,j} = \left( A_{i,j} - \sum_{k<j} L_{i,k} \cdot U_{k,j} \right) / U_{j,j} \]

Fits the generalized matrix computations:

\[ C(i,j) = f_{i,j} \left( g_{i,j,k}(A(i,k), B(k,j)) \right) \text{ for } k \in S_{i,j}, K \]

with:

- \( A = B = C \)
- \( g_{i,j,k} \) multiplies \( L_{i,k} \cdot U_{k,j} \)
- \( f_{i,j} \) performs the sum, subtracts from \( A_{i,j} \) (divides by \( U_{j,j} \))
- I/O lower bound: \( \Omega(G/\sqrt{M}) = \Omega(n^3/\sqrt{M}) \)
- Some algorithms attain this bound (hard because of pivoting)
Communication-Avoiding Algorithms

1. Generalization to other Linear Algebra Algorithms
   - Generalized Matrix Computations
   - I/O Analysis
   - Application to LU Factorization

2. Analysis and Lower Bounds for Parallel Algorithms
   - Matrix Multiplication Lower Bound for $P$ processors
   - 2D and 3D Algorithms for Matrix Multiplication
   - 2.5D Algorithm for Matrix Multiplication

3. Conclusion
Lemma

Consider a conventional \( N \times N \) matrix multiplication performed on \( P \) processors with distributed memory. A processor with memory \( M \) that performs \( W \) elementary products must send or receive at least 
\[
\frac{W}{2\sqrt{2\sqrt{M}}} - M \text{ elements.}
\]

Theorem

Consider a conventional \( N \times N \) matrix multiplication on \( P \) processors, each with a memory \( M \). Some processor has a volume of I/O at least 
\[
\frac{N^3}{2\sqrt{2P}\sqrt{M}} - M.
\]

NB: bound useful only when \( M < \frac{N^2}{(2P^{3/2})} \).
Matrix Multiplication Lower Bound for $P$ processors

**Lemma**
Consider a conventional $N \times N$ matrix multiplication performed on $P$ processors with distributed memory. A processor with memory $M$ that performs $W$ elementary products must send or receive at least
\[
\frac{W}{2\sqrt{2}\sqrt{M}} - M \text{ elements.}
\]

**Theorem**
Consider a conventional $N \times N$ matrix multiplication on $P$ processors, each with a memory $M$. Some processor has a volume of I/O at least
\[
\frac{N^3}{2\sqrt{2}P\sqrt{M}} - M.
\]

NB: bound useful only when $M < \frac{N^2}{(2P^{3/2})}$.
Matrix Multiplication Lower Bound for $P$ processors

Lemma

Consider a conventional $N \times N$ matrix multiplication performed on $P$ processors with distributed memory. A processor with memory $M$ that performs $W$ elementary products must send or receive at least
\[
\frac{W}{2\sqrt{2\sqrt{M}}} - M \text{ elements.}
\]

Theorem

Consider a conventional $N \times N$ matrix multiplication on $P$ processors, each with a memory $M$. Some processor has a volume of I/O at least
\[
\frac{N^3}{2\sqrt{2P\sqrt{M}}} - M.
\]

NB: bound useful only when $M < N^2/(2P^{3/2})$
Cannon’s 2D algorithm

- Processors organized on a square 2D grid of size $\sqrt{P} \times \sqrt{P}$
- $A$, $B$, $C$ matrices distributed by blocks of size $N/\sqrt{P} \times N/\sqrt{P}$
  
  Processor $P_{i,j}$ initially holds matrices $A_{i,j}$, $B_{i,j}$, computes $C_{i,j}$
- At each step, each proc. performs a $A_{i,k} \times B_{k,j}$ block product

First reallign matrices:
- Shift $A_{i,j}$ blocks to the left by $i$ (wraparound)
- Shift $B_{i,j}$ blocks to the top by $j$ (wraparound)

Then $P_{i,j}$ holds blocks $A_{i,i+j}$ and $B_{i+j,j}$

At each step:
- Compute one block product
  
  Total I/O cost: $\Theta(N^2 \sqrt{P})$
- Storage $\Theta(N^2/P)$ per proc.
Cannon's 2D algorithm

- Processors organized on a **square 2D grid** of size $\sqrt{P} \times \sqrt{P}$
- $A, B, C$ matrices distributed by blocks of size $N/\sqrt{P} \times N/\sqrt{P}$
  - Processor $P_{i,j}$ initially holds matrices $A_{i,j}$, $B_{i,j}$, computes $C_{i,j}$
  - At each step, each proc. performs a $A_{i,k} \times B_{k,j}$ block product

![Diagram of Cannon's 2D algorithm]

- **First realign matrices:**
  - Shift $A_{i,j}$ blocks to the left by $i$ (wraparound)
  - Shift $B_{i,j}$ blocks to the top by $j$ (wraparound)

  Then $P_{i,j}$ holds blocks $A_{i,i+j}$ and $B_{i+j,j}$

- At each step:
  - Compute one block product
  - Shift $A$ blocks right
  - Shift $B$ blocks down

- Total I/O cost: $\Theta(N^2 \sqrt{P})$
- Storage $\Theta(N^2/P)$ per proc.
Cannon’s 2D algorithm

- Processors organized on a **square 2D grid** of size $\sqrt{P} \times \sqrt{P}$
- $A, B, C$ matrices distributed by blocks of size $N/\sqrt{P} \times N/\sqrt{P}$
- Processor $P_{i,j}$ initially holds matrices $A_{i,j}, B_{i,j}$, computes $C_{i,j}$
- At each step, each proc. performs a $A_{i,k} \times B_{k,j}$ block product

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Matrix A" /></td>
<td><img src="image2.png" alt="Matrix B" /></td>
</tr>
<tr>
<td><strong>Starting position</strong></td>
<td><strong>Starting position</strong></td>
</tr>
<tr>
<td><img src="image3.png" alt="Stagger left" /></td>
<td><img src="image4.png" alt="Stagger up" /></td>
</tr>
<tr>
<td><img src="image5.png" alt="Shift right" /></td>
<td><img src="image6.png" alt="Shift down" /></td>
</tr>
</tbody>
</table>

At each step:
- First realign matrices:
  - Shift $A_{i,j}$ blocks to the left by $i$ (wraparound)
  - Shift $B_{i,j}$ blocks to the top by $j$ (wraparound)

Then $P_{i,j}$ holds blocks $A_{i,i+j}$ and $B_{i+j,j}$

| ![Shifts](image7.png) |
| **Shifts** |

- At each step:
  - Compute one block product
  - Shift $A$ blocks right
  - Shift $B$ blocks down

- Total I/O cost: $\Theta(N^2 \sqrt{P})$
- Storage $\Theta(N^2/P)$ per proc.
Cannon’s 2D algorithm

- Processors organized on a square 2D grid of size $\sqrt{P} \times \sqrt{P}$
- $A$, $B$, $C$ matrices distributed by blocks of size $N/\sqrt{P} \times N/\sqrt{P}$
- Processor $P_{i,j}$ initially holds matrices $A_{i,j}$, $B_{i,j}$, computes $C_{i,j}$
- At each step, each proc. performs a $A_{i,k} \times B_{k,j}$ block product

![Diagram of Cannon’s 2D algorithm]

- First reallign matrices:
  - Shift $A_{i,j}$ blocks to the left by $i$ (wraparound)
  - Shift $B_{i,j}$ blocks to the top by $j$ (wraparound)

Then $P_{i,j}$ holds blocks $A_{i,i+j}$ and $B_{i+j,j}$

- At each step:
  - Compute one block product
  - Shift $A$ blocks right
  - Shift $B$ blocks down

- Total I/O cost: $\Theta(N^2 \sqrt{P})$
- Storage $\Theta(N^2/P)$ per proc.
Other 2D Algorithm: SUMMA

- SUMMA: Scalable Universal Matrix Multiplication Algorithm
- Same 2D grid distribution: $P_{i,j}$ holds $A_{i,j}$, $B_{i,j}$, computes $C_{i,j}$
- At each step $k$, column $k$ of $A$ and row $k$ of $B$ are broadcasted (from processors owning the data)
- Each processor computes a local contribution (outer-product)

![Diagram showing the process of SUMMA algorithm]

- Smaller communications $\Rightarrow$ smaller temporary storage
- Same I/O volume: $\Theta(N^2 \sqrt{P})$
I/O Lower Bound for 2D algorithms

**Theorem**

Consider a conventional matrix multiplication on $P$ processors each with $O(N^2/P)$ storage, some processor has a I/O volume at least $\Omega(N^2/\sqrt{P})$.

**Proof:**

Previous result: $\Omega(N^3/P\sqrt{M})$ with $M = N^2/P$.

- When balanced, total I/O volume: $\Theta(N^2\sqrt{P})$
- Both Cannon’s algorithm and SUMMA are optimal

Can we do better?
Theorem

Consider a conventional matrix multiplication on $P$ processors each with $O(N^2/P)$ storage, some processor has a I/O volume at least $\Omega(N^2/\sqrt{P})$.

Proof: Previous result: $\Omega(N^3/P\sqrt{M})$ with $M = N^2/P$.

- When balanced, total I/O volume: $\Theta(N^2\sqrt{P})$
- Both Cannon’s algorithm and SUMMA are optimal among 2D algorithms (memory limited to $O(N^2/P)$)

Can we do better?
3D Algorithm

- Consider 3D grid of processor: \( q \times q \times q \) 
  \((q = P^{1/3} = 3\sqrt[3]{P})\)
- Processor \(i, j, k\) owns blocks \(A_{i,k}, B_{k,j}, C_{i,j}^{(k)}\)
- Matrices are replicated (including \(C\))
- Each processor computes its local contribution
- Then summation of the various \(C_{i,j}^{(k)}\) for all \(k\)

- Memory needed: \( \Theta(N^2/q^2) = \Theta(N^2/P^{2/3}) \) per processor
- Total I/O volume: \( \Theta(N^2/q^2 \times q^3) = \Theta(N^2q) = \Theta(N^23\sqrt[3]{P}) \)

Lower Bound:

- Previous theorem does not give useful bound 
  (only when \( M < N^2/(2\sqrt{6}P^{2/3}) \))
- More complex analysis shows that the I/O volume on some processor is \( \Theta(N^2/P^{2/3}) \)
- In total, when balanced \( \Theta(N^23\sqrt[3]{P}) \) \( \Rightarrow \) 3D algo. is optimal
- Can we do better?
3D Algorithm

- Consider 3D grid of processor: \( q \times q \times q \) 
  \((q = P^{1/3} = \sqrt[3]{P})\)

- Processor \( i, j, k \) owns blocks \( A_{i,k}, B_{k,j}, C_{i,j}^{(k)} \)

- Matrices are replicated (including \( C \))

- Each processor computes its local contribution

- Then summation of the various \( C_{i,j}^{(k)} \) for all \( k \)

- Memory needed: \( \Theta(N^2/q^2) = \Theta(N^2/P^{2/3}) \) per processor

- Total I/O volume: \( \Theta(N^2/q^2 \times q^3) = \Theta(N^2q) = \Theta(N^2\sqrt[3]{P}) \)

Lower Bound:

- Previous theorem does not give useful bound 
  (only when \( M < N^2/(2\sqrt{6}P^{2/3}) \))

- More complex analysis shows that the I/O volume on some processor is \( \Theta(N^2/P^{2/3}) \)

- In total, when balanced \( \Theta(N^2\sqrt[3]{P}) \Rightarrow 3D \) algo. is optimal

- Can we do better?
3D Algorithm

- Consider 3D grid of processor: $q \times q \times q$
  $(q = P^{1/3} = \sqrt[3]{P})$
- Processor $i, j, k$ owns blocks $A_{i,k}, B_{k,j}, C_{i,j}^{(k)}$
- Matrices are replicated (including $C$)
- Each processor computes its local contribution
- Then summation of the various $C_{i,j}^{(k)}$ for all $k$
- Memory needed: $\Theta(N^2/q^2) = \Theta(N^2/P^{2/3})$ per processor
- Total I/O volume: $\Theta(N^2/q^2 \times q^3) = \Theta(N^2q) = \Theta(N^2\sqrt[3]{P})$

Lower Bound:

- Previous theorem does not give useful bound
  (only when $M < N^2/(2\sqrt{6}P^{2/3})$)
- More complex analysis shows that the I/O volume on some processor is $\Theta(N^2/P^{2/3})$
- In total, when balanced $\Theta(N^2\sqrt[3]{P}) \Rightarrow$ 3D algo. is optimal
- Can we do better?
3D Algorithm

- Consider 3D grid of processor: $q \times q \times q$
  
  \[ q = P^{1/3} = \sqrt[3]{P} \]
  
- Processor $i, j, k$ owns blocks $A_{i,k}, B_{k,j}, C^{(k)}_{i,j}$

- Matrices are replicated (including $C$)

- Each processor computes its local contribution

- Then summation of the various $C^{(k)}_{i,j}$ for all $k$

- Memory needed: $\Theta(N^2/q^2) = \Theta(N^2/P^{2/3})$ per processor

- Total I/O volume: $\Theta(N^2/q^2 \times q^3) = \Theta(N^2q) = \Theta(N^2\sqrt[3]{P})$

Lower Bound:

- Previous theorem does not give useful bound
  (only when $M < N^2/(2\sqrt{6}P^{2/3})$)

- More complex analysis shows that the I/O volume on some processor
  is $\Theta(N^2/P^{2/3})$

- In total, when balanced $\Theta(N^2\sqrt[3]{P}) \Rightarrow$ 3D algo. is optimal

- Can we do better?
2.5D Algorithm (1/2)

- 3D algorithm requires large memory on each processor \((\sqrt[3]{P} \text{ copies of each matrices})\)
- What if we have space for only \(1 < c < \sqrt[3]{P}\) copies?
- Assume each processor has a memory \(M = O(c \cdot N^2 / P)\)
- Arrange processors in \(\sqrt{P/c} \times \sqrt{P/c} \times c\) grid:
  - \(c\) layers, each layer with \(P/c\) processors in square grid
- \(A, B, C\) distributed by blocks of size \(N\sqrt{c/P} \times N\sqrt{c/P}\), replicated on each layer

NB: \(c = 1\) gets 2D, \(c = P^{1/3}\) gives 3D
Each layer responsible for a fraction $1/c$ of Cannon’s alg.: Different initial shifts of $A$ and $B$

Finally, sum $C$ over layers

- Total I/O volume: $\Theta(N^2\sqrt{P/c})$
  - Replication, initial shift, final sum: $\Theta(N^2c)$
  - $c$ layers of fraction $1/c$ of Cannon’s alg. with grid size $\sqrt{P/c}$:
    $\Theta\left(N^2\sqrt{P/c}\right)$

- Reaches lower bound on I/Os per processor:
  $$\Omega\left(\frac{N^3}{P\sqrt{M}}\right) = \Omega\left(\frac{N^3}{P\sqrt{cN^2/P}}\right) = \Omega(N^2/\sqrt{cP})$$
2.5D Algorithm (2/2)

- Each layer responsible for a fraction $1/c$ of Cannon’s alg.: Different initial shifts of $A$ and $B$
- Finally, sum $C$ over layers
- Total I/O volume: $\Theta(N^2 \sqrt{P/c})$
  - Replication, initial shift, final sum: $\Theta(N^2 c)$
  - $c$ layers of fraction $1/c$ of Cannon’s alg. with grid size $\sqrt{P/c}$:
    $\Theta\left(N^2 \sqrt{P/c}\right)$
- Reaches lower bound on I/Os per processor:
  $$\Omega\left(\frac{N^3}{P \sqrt{M}}\right) = \Omega\left(\frac{N^3}{P \sqrt{cN^2/P}}\right) = \Omega(N^2/\sqrt{cP})$$
2.5D Algorithm (2/2)

- Each layer responsible for a fraction $1/c$ of Cannon’s alg.: Different initial shifts of $A$ and $B$
- Finally, sum $C$ over layers
- Total I/O volume: $\Theta(N^2\sqrt{P/c})$
  - Replication, initial shift, final sum: $\Theta(N^2c)$
  - $c$ layers of fraction $1/c$ of Cannon’s alg. with grid size $\sqrt{P/c}$: $\Theta(N^2\sqrt{P/c})$
- Reaches lower bound on I/Os per processor:
  $$\Omega\left(\frac{N^3}{P\sqrt{M}}\right) = \Omega\left(\frac{N^3}{P\sqrt{cN^2/P}}\right) = \Omega(N^2/\sqrt{cP})$$
Performance on Blue Gene P

Matrix multiplication on 16,384 nodes of BG/P

95% reduction in communication

C=16
Communication-Avoiding Algorithms

1. Generalization to other Linear Algebra Algorithms
   - Generalized Matrix Computations
   - I/O Analysis
   - Application to LU Factorization

2. Analysis and Lower Bounds for Parallel Algorithms
   - Matrix Multiplication Lower Bound for $P$ processors
   - 2D and 3D Algorithms for Matrix Multiplication
   - 2.5D Algorithm for Matrix Multiplication

3. Conclusion
Conclusion

Generalized I/O lower bound for matrix computations:
- Apply to most linear algebra algorithms
- Design of I/O-optimal algorithms

Parallel algorithms with distributed memory:
- Adapted I/O lower bounds (depends on $M$ on each processor)
- Asymptotically optimal algorithm for matrix multiplication... and many other matrix computations “communication-avoiding algorithms”

Here: focus on the total I/O volume
- Similar lower bound and analysis for the number of messages: also important factor for performance
- Variant: Write-avoiding algorithms for NVRAMs (writes more expensive than reads)