Part 4: Communication Avoiding Algorithms

Loris Marchal (CNRS, Lyon, France)
loris.marchal@ens-lyon.fr

November 20, 2019
Part 4: Communication Avoiding Algorithms

Introduction

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

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

Conclusion
Communication Avoiding Algorithms

Context: Distributed Memory

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
Reminder: Matrix Product Lower Bound

Context: Single processor + Memory (size $M$)

- Analysis in phases of $M$ I/O operations
- Bound on the number of elementary product in each phase:
  \[ F = O\left(\frac{M^{3/2}}{2}\right) \]
  Geometric argument: Loomis-Whitney inequality
- At least $\frac{n^3}{F}$ phases, of $M$ I/Os, in total: $\Theta\left(\frac{n^3}{\sqrt{M}}\right)$ I/Os
Part 4: Communication Avoiding Algorithms

Introduction

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

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

Conclusion
Generalization to other Linear Algebra Algorithms

▶ Inputs/Ouput: $n \times n$ 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 $(i, j) \in S_c$,

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

- $f_{i,j}$ and $g_{i,j,k}$ non-trivial:
  - $g_{i,j,k}$ needs to 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
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 $(i, j) \in S_c$,

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

- $f_{i,j}$ and $g_{i,j,k}$ non-trivial:
  - $g_{i,j,k}$ needs to 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
- 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 \((i, j) \in S_c\),

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

- \(f_{i,j}\) and \(g_{i,j,k}\) non-trivial:
  - \(g_{i,j,k}\) needs to 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

- 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

- $f$ and $g$ are not assumed associative or commutative
- Correct computations may require special ordering of computations: no such constraint needed for the lower bound

Analysis based on Loomis-Whitney inequality:

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

Let $V$ be a finite subset of $\mathbb{Z}^D$ and $V_1, V_2, V_3$ denotes 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 (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 \( F = \sqrt[3]{(4M)^4} \) computations of \( g \)
- Total number of loads and stores

\[
M \left\lfloor \frac{3}{4} \right\rfloor \leq M \left\lfloor \frac{G}{\sqrt[3]{(4M)^4}} \right\rfloor \leq \frac{G}{4\sqrt[3]{M^4}} - M
\]
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 (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 is at most $F = \sqrt[3]{(4M)^2}$ computations of $g$
- Total number of loads and stores

$$M \left\lfloor \frac{G}{F} \right\rfloor = M \left\lfloor \sqrt[3]{(4M)^2} \right\rfloor \leq \frac{G}{2N} - M$$
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 (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:

\[
M \left[ \frac{G}{F} \right] = M \left[ \frac{G}{\sqrt{(4M)^3}} \right] \geq \frac{G}{8\sqrt{M}} - M
\]
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 (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:

$$M \left\lceil \frac{G}{F} \right\rceil = M \left\lceil \frac{G}{\sqrt{(4M)^3}} \right\rceil \geq \frac{G}{8\sqrt{M}} - M$$
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 (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:

$$M \left\lfloor \frac{G}{F} \right\rfloor = M \left\lfloor \frac{G}{\sqrt{(4M)^3}} \right\rfloor \geq \frac{G}{8\sqrt{M}} - M$$
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 (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:

$$M \left\lfloor \frac{G}{F} \right\rfloor = M \left\lfloor \frac{G}{\sqrt{(4M)^3}} \right\rfloor \geq \frac{G}{8\sqrt{M}} - M$$
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)
Application to LU Factorization (2/2)

Can be expressed as follows:

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

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

Fits the generalized matrix computations:

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

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: \( O(G/\sqrt{M}) = O(n^3/\sqrt{M}) \)
- A recursive algorithm achieves this bound
Part 4: Communication Avoiding Algorithms

Introduction

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

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

Conclusion
Lemma.

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

Theorem.

Consider a conventional 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 \( M < \frac{n^2}{(2P^{3/2})} \)
Lemma.
Consider a conventional matrix multiplication performed on $P$ processors with distributed memory. A processor with memory $M$ that perform $W$ elementary products must send or receive at least $\frac{W}{2\sqrt{2\sqrt{M}}} - M$ elements.

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

**Theorem.**
Consider a conventional 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 $M < \frac{n^2}{(2P^{3/2})}$
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)

After computation, shift $A$ blocks right, shift $B$ blocks down

Total I/O cost: $O(n^2 \sqrt{P})$

Storage $O(n^2 / P)$ 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

- 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)

- After computation, shift $A$ blocks right, shift $B$ blocks down

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

- SUMMA: Scalable Universal Matrix Multiplication Algorithm
- Same 2D grid distribution
- 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)

Smaller communications \( \Rightarrow \) smaller temporary storage

Same I/O volume: \( O(n^2 \sqrt{P}) \)
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 $\Theta(n^2/\sqrt{P})$.

Proof: Previous result: $O(n^3/P\sqrt{M})$ 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?
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 $\Theta(n^2/\sqrt{P})$.

Proof: Previous result: $O(n^3/P\sqrt{M})$ 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: \( q \times q \times q \) 
  \((q = P^{1/3})\)
- 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: \(O(n^2/q^2) = O(n^2/P^{2/3})\) per processor
- Total I/O volume: \(O(n^2/q^2 \times q^3) = O(n^2q) = O(n^2P^{1/3})\)

Lower Bound:
- Previous theorem does not give useful bound 
  \((M = \Theta(n^2P^{1/3}))\)
- More complex analysis shows that the I/O volume on some processor is \(\Theta(n^2/P^{2/3})\)
2.5D Algorithm (1/2)

- 3D algorithm requires large memory on each processor \((P^{1/3}\) copies of each matrices)
- What if we have space for only \(1 < c < P^{1/3}\) copies?
- Assume each processor has a memory \(M = O(cn^2/P)\)
- 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

\[\text{NB: } c = 1 \text{ gets 2D, } c = P^{1/3} \text{ gives 3D}\]
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: \(O\left(\frac{n^2}{\sqrt{P/c}}\right)\)
    - Replication, initial shift, final sum: \(O(n^2c)\)
    - \(c\) layers of fraction \(1/c\) of Cannon’s alg. with grid size \(\sqrt{P/c}\):
      \[O\left(\frac{n^2}{\sqrt{P/c}}\right)\]
  - Reaches lower bound on I/Os per processor:
    \[O\left(\frac{n^3}{P\sqrt{M}}\right) = O\left(\frac{n^3}{P\sqrt{cn^2/P}}\right) = O\left(\frac{n^2}{\sqrt{cP}}\right)\]
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: $O(n^2 / \sqrt{P/c})$
  - Replication, initial shift, final sum: $O(n^2 c)$
  - c layers of fraction 1/c of Cannon’s alg. with grid size $\sqrt{P/c}$:
    $O\left(n^2 \sqrt{P/c}\right)$
- Reaches lower bound on I/Os per processor:
  $O\left(\frac{n^3}{P\sqrt{M}}\right) = O\left(\frac{n^3}{P\sqrt{cn^2/P}}\right) = O(n^2 / \sqrt{cP})$
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: $O(n^2/\sqrt{P/c})$

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

Reaches lower bound on I/Os per processor:

$$O\left(\frac{n^3}{P\sqrt{M}}\right) = O\left(\frac{n^3}{P\sqrt{cn^2/P}}\right) = O\left(\frac{n^2}{\sqrt{cP}}\right)$$
Performance on Blue Gene P

Matrix multiplication on 16,384 nodes of BG/P

95% reduction in communication

C=16
Part 4: Communication Avoiding Algorithms

Introduction

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

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

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
▶ Write-avoiding algorithms for NVRAMs
  (writes more expensive than reads)