Improving Communication Lower Bounds for Matrix-Matrix Multiplication

The 9th Scheduling for Large Scale Systems Workshop, Lyon, France

Bradley R. Lowery and Julien Langou

University of Colorado Denver

July 2, 2014
\[ \beta^{-1} = 10^8 \text{words/sec} \quad \gamma^{-1} = 10^{10} \text{flops/sec} \quad M = 10^6 \text{words} \]
Mission Statement

We study communication costs for the ordinary dense (OD) matrix-matrix multiplication in the sequential model.
Mission Statement

We study communication costs for the ordinary dense (OD) matrix-matrix multiplication in the sequential model.

- dense.
Mission Statement

We study communication costs for the ordinary dense (OD) matrix-matrix multiplication in the **sequential** model.

- **dense.**
- **sequential**: two levels of memory
  - not parallel!
  - fast memory of size $M$
  - slow memory
  - computation happens in fast memory
Mission Statement

We study **communication costs** for the ordinary dense (OD) matrix-matrix multiplication in the sequential model.

- **dense.**
- **sequential:** two levels of memory
- **communication cost:**

Algorithms have two costs:

- **Computation:** Cost to perform computation
  - # of operations to be performed
- **Communication:** Cost to move data
  - volume of data to be moved (bandwidth)
  - # of messages (latency)
- No overlap computation / communication
- Cost can be time, energy or power, for time, we get

\[
\alpha \text{ (latency)}, \beta \text{ (inverse of bandwidth)}, \gamma \text{ (inverse of bandwidth)}
\]

\[
time = \alpha \times (# \text{ message}) + \beta \times \text{(total vol. of communication)} + \gamma \times (# \text{ of flops})
\]
Mission Statement

We study communication costs for the ordinary dense (OD) matrix-matrix multiplication in the sequential model.

- dense.
- sequential: two levels of memory
- communication cost:
- ordinary: we compute all \( n^3 \)

\[
c_{ijk} = a_{ik} \cdot b_{kj}
\]

(consequence: Strassen-like matrix-matrix multiplications are not allowed.)
The present study is only (mainly) concerned with the volume of communication (bandwidth term). Important to realize that this generalizes to

- **# of messages** (latency related) (as opposed to “total volume of messages”, bandwidth related)
- **parallel distributed**
- **hierarchical memories**
Mission Statement
We study communication costs for the ordinary dense (OD) matrix-matrix multiplication in the sequential model.

Communication Cost for (OD) Matrix-Matrix Multiplication
Dense matrix-matrix multiplication moves $n^2$ data for $n^3$ computation.

\[
\begin{align*}
\begin{array}{c}
\hline \\
C \\
\hline 
\end{array}
\end{align*}
\begin{align*}
\begin{array}{c}
\hline \\
A \\
\hline 
\end{array} \times 
\begin{array}{c}
\hline \\
B \\
\hline 
\end{array}
\end{align*}
\]

- Computation cost is $2n^3$
  for $i=1:n$, for $j=1:n$, for $k=1:n$, $c_{ij} = c_{ij} + a_{ik}b_{kj}$; end; end; end;
- Communication cost is $3n^2$
Mission Statement

We study communication costs for the ordinary dense (OD) matrix-matrix multiplication in the sequential model.

Communication Cost for (OD) Matrix-Matrix Multiplication

Dense matrix-matrix multiplication moves $n^2$ data for $n^3$ computation.

\[
\begin{array}{c}
\begin{bmatrix} C \\ n \end{bmatrix} \\
+ \\
\begin{array}{c}
\begin{bmatrix} A \\ n \end{bmatrix} \\
\begin{bmatrix} B \\ n \end{bmatrix} \\
\end{array}
\end{array}
\]

- Computation cost is $2n^3$
  for $i=1:n$, for $j=1:n$, for $k=1:n$, $c_{ij} = c_{ij} + a_{ik}b_{kj}$; end; end; end;
- Communication cost is $3n^2$

Conclusion of the study

When $n$ increases, communication cost ($n^2$) becomes negligible with respect to computation cost ($n^3$).
Limitation of the previous study: The previous study assumes that the three $n$-by-$n$ matrix $A$, $B$, and $C$ fit in cache.
Limitation of the previous study: The previous study assumes that the three $n$-by-$n$ matrix $A$, $B$, and $C$ fit in cache.

Note: this is a pretty serious limitation ...
(In particular when $n$ goes to infinity ...)
- **Limitation of the previous study:** The previous study assumes that the three \( n \times n \) matrix \( A \), \( B \), and \( C \) fit in cache.

- **Note:** This is a pretty serious limitation ... (In particular when \( n \) goes to infinity ...)

- **Easy fix:** A common easy fix is to block the matrix-matrix multiplication with square blocks so that the square blocks fit in cache.

Let \( M \) be the size of our cache. Let \( b = \sqrt{\frac{M}{3}} \) (so that \( 3b^2 = M \)). Then,

\[
C_{ij} = A_{ik} \times B_{kj}
\]

for \( i = 1:n/b \), for \( j = 1:n/b \), for \( k = 1:n/b \),

end; end; end;

Then, at each loop, we are moving \( 2b^2 \) data and computing \( 2b^3 \) so ... (Note: \( C_{ij} \) stays in cache.)
- **Limitation of the previous study**: The previous study assumes that the three \( n \times n \) matrix \( A \), \( B \), and \( C \) fit in cache.

- **Note**: this is a pretty serious limitation ...(In particular when \( n \) goes to infinity ...)

- **Easy fix**: A common easy fix is to block the matrix-matrix multiplication with square blocks so that the square blocks fit in cache. Let \( M \) be the size of our cache. Let \( b = \sqrt[3]{\frac{M}{3}} \) (so that \( 3b^2 = M \)). Then,

\[
C_{ij} = B_{kj} \times A_{ik}
\]

for \( i = 1:n/b \), for \( j = 1:n/b \), for \( k = 1:n/b \),

end; end; end;

Then, at each loop, we are moving \( 2b^2 \) data and computing \( 2b^3 \) so ...
(Note: \( C_{ij} \) stays in cache.)

- Computation cost is \( \left( \frac{n}{b} \right)^3 (2b^3) \rightarrow 2n^3 \rightarrow \) perfect.
- **Limitation of the previous study**: The previous study assumes that the three \( n \times n \) matrix \( A, B, \) and \( C \) fit in cache.

- **Note**: this is a pretty serious limitation ... (In particular when \( n \) goes to infinity ... )

- **Easy fix**: A common easy fix is to block the matrix-matrix multiplication with square blocks so that the square blocks fit in cache.

Let \( M \) be the size of our cache. Let \( b = \sqrt{\frac{M}{3}} \) (so that \( 3b^2 = M \)). Then,

for \( i=1:n/b \), for \( j=1:n/b \), for \( k=1:n/b \),

\[
\begin{bmatrix}
C_{ij}
\end{bmatrix}_b + \begin{bmatrix}
A_{ik}
\end{bmatrix}_b \times \begin{bmatrix}
B_{kj}
\end{bmatrix}_b
\]

end; end; end;

Then, at each loop, we are moving \( 2b^2 \) data and computing \( 2b^3 \) so ... (Note: \( C_{ij} \) stays in cache.)

- Computation cost is \( \left(\frac{n}{b}\right)^3 (2b^3) \rightarrow 2n^3 \rightarrow \) perfect.

- Communication cost is \( \left(\frac{n}{b}\right)^3 (2b^2) \rightarrow \left(\frac{2}{b}\right)n^3 \rightarrow \) oopsee.
We see that the previous algorithm
- performs $2n^3$ floating point operations
- performs a volume of data movement of

$$\left( \frac{2\sqrt{3}}{\sqrt{M}} \right) n^3.$$ 

Therefore the time of this OD matrix-matrix multiplication is

$$\left( \frac{2\sqrt{3}}{\sqrt{M}} \right) \beta n^3 + 2\gamma n^3$$

(1) assuming no overlap between communication and computations; (2) with $\beta$ being the time to move one unit of data (inverse of bandwidth) and $\gamma$ being the time to perform one floating-point operation.
We see that the previous algorithm

- performs \(2n^3\) floating point operations
- performs a volume of data movement of

\[
\left( \frac{2\sqrt{3}}{\sqrt{M}} \right) n^3.
\]

Therefore the time of this OD matrix-matrix multiplication is

\[
\left( \frac{2\sqrt{3}}{\sqrt{M}} \right) \beta n^3 + 2\gamma n^3
\]

(1) assuming no overlap between communication and computations; (2) with \(\beta\) being the time to move one unit of data (inverse of bandwidth) and \(\gamma\) being the time to perform one floating-point operation.

**Study with** \(n\). Communication is not negligible against computation. Both computation and communication are of order \(n^3\).
We see that the previous algorithm
- performs $2n^3$ floating point operations
- performs a volume of data movement of

\[
\left( \frac{2\sqrt{3}}{\sqrt{M}} \right) n^3.
\]

Therefore the time of this OD matrix-matrix multiplication is

\[
\left( \frac{2\sqrt{3}}{\sqrt{M}} \right) \beta n^3 + 2\gamma n^3
\]

(1) assuming no overlap between communication and computations; (2) with $\beta$ being the time to move one unit of data (inverse of bandwidth) and $\gamma$ being the time to perform one floating-point operation.

**Study with** $n$. Communication is not negligible against computation. Both computation and communication are of order $n^3$.

*If $\beta/\sqrt{M} << \gamma$ then*, communication is negligible against computation.
Consider any ordinary dense matrix-matrix multiplication algorithm for multiplying an $m$–by–$n$ matrix with an $n$–by–$p$ matrix, consider a computer with fast memory of size $M$, then

**Theorem (Hong and Kung, 1981)**

The number of words transferred between slow and fast memory is at least $\frac{1}{2} \sqrt{2} mnp \sqrt{M - M}$. 
Consider any ordinary dense matrix-matrix multiplication algorithm for multiplying an $m$–by–$n$ matrix with an $n$–by–$p$ matrix, consider a computer with fast memory of size $M$, then

Theorem (Hong and Kung, 1981)

The number of words transferred between slow and fast memory is at least

$$\frac{1}{2\sqrt{2}} \frac{mnp}{\sqrt{M}} - M.$$
Consider any ordinary dense matrix-matrix multiplication algorithm for multiplying an $n$–by–$n$ matrix with an $n$–by–$n$ matrix, consider a computer with fast memory of size $M$, then

**Upper bound :: square tile matrix-matrix multiplication**

The number of words transferred between slow and fast memory is at most

$$3.46 \left( \frac{n^3}{\sqrt{M}} \right).$$

**Lower Bound :: Irony, Toledo, and Tiskin, 2004**

The number of words transferred between slow and fast memory is at least

$$0.35 \left( \frac{n^3}{\sqrt{M}} \right) - M.$$

Note: $3.46 \approx 2\sqrt{3}$

Note: $0.35 \approx (2\sqrt{2})^{-1}$
The time of an OD matrix-matrix multiplication is

\[ (?) \beta n^3 + 2 \gamma n^3 \]

(1) assuming no overlap between communication and computations; (2) with \( \beta \) being the time to move one unit of data (inverse of bandwidth) and \( \gamma \) being the time to perform one floating-point operation.

We know that (?) is between 0.35 and 3.46.
\[ \beta^{-1} = 10^8 \text{ words/sec} \quad \gamma^{-1} = 10^{10} \text{ flops/sec} \quad M = 10^6 \text{ words} \]
Block matrix-matrix multiplication

Block matrix-matrix multiplication

\[
C_{ij} = A_{ik} \times B_{kj}
\]
Block matrix-matrix multiplication

\[ C_{ij} = A_{ik} \times B_{kj} \]

Three square blocks fit in fast memory: \( b^2 = M \).

Good bandwidth: Volume = \( 2\sqrt{3} mnp \sqrt{M} \).

Good latency: # Messages = \( 3\sqrt{3} mnp M^{3/2} \).
Block matrix-matrix multiplication

\[
\begin{align*}
C_{ij} &= A_{ik} \times B_{kj} \\
\begin{array}{c}
\begin{array}{c}
C_{ij} \\
b
\end{array}
\end{array} + &= \begin{array}{c}
\begin{array}{c}
A_{ik} \\
b
\end{array}
\end{array} \times \begin{array}{c}
\begin{array}{c}
B_{kj}
\end{array}
\end{array}
\end{align*}
\]

- Three square blocks fit in fast memory: \(3b^2 = M\).
Block matrix-matrix multiplication

- Three square blocks fit in fast memory: $3b^2 = M$.
- Good bandwidth: Volume $= 2\sqrt{3} \frac{mnp}{\sqrt{M}}$
Block matrix-matrix multiplication

\[
\begin{align*}
C_{ij} &= A_{ik} \times B_{kj} \\

\left( \begin{array}{c}
\begin{array}{c}
\begin{array}{c}
C_{ij}
\end{array}
\end{array}
\end{array} \right) + \left( \begin{array}{c}
\begin{array}{c}
\begin{array}{c}
A_{ik}
\end{array}
\end{array} \right) \times \left( \begin{array}{c}
\begin{array}{c}
\begin{array}{c}
B_{kj}
\end{array}
\end{array} \right) 
\end{align*}
\]

- Three square blocks fit in fast memory: \( 3b^2 = M \).
- Good bandwidth: Volume \( = 2\sqrt{3} \frac{mnp}{\sqrt{M}} \).
- Good latency: # Messages \( = 3\sqrt{3} \frac{mnp}{M^{3/2}} \).
Block matrix-matrix multiplication

See PUMMA / SUMMA parallel distributed algorithms.
Block matrix-matrix multiplication

\[ C_{ij} = A_{ik} \times B_{kj} \]

- **Block** \( C_{ij} \) fits in fast memory: \( b_2 \approx M \).
- **Better bandwidth**: Volume \( = 2^{mnp} \sqrt{M} \).
- **Horrible latency**: Number of messages \( = \sqrt{M} (\frac{mnp}{2} M^3) \).
Block matrix-matrix multiplication

\[
C_{ij} = A_{ik} \times B_{kj}
\]

- Block matrix-matrix multiplication
- Block $C_{ij}$ fits in fast memory: $b^2 \approx M$
- Better bandwidth: Volume $= 2mnp \sqrt{M}$
- Horrible latency: Number of messages $= \sqrt{M} (mnpM^3/2)$
Block matrix-matrix multiplication

Block $C_{ij}$ fits in fast memory: $b^2 \approx M$. 
Block matrix-matrix multiplication

- Block $C_{ij}$ fits in fast memory: $b^2 \approx M$.
- Better bandwidth: Volume $= 2 \frac{mnp}{\sqrt{M}}$
Block matrix-matrix multiplication

- Block $C_{ij}$ fits in fast memory: $b^2 \approx M$.
- Better bandwidth: Volume $= 2 \frac{mnp}{\sqrt{M}}$.
- Horrible latency: # Messages $= \sqrt{M} \left( \frac{mnp}{M^{3/2}} \right)$.
Block matrix-matrix multiplication

\[
\begin{bmatrix}
C_{ij} \\
b
\end{bmatrix} = \begin{bmatrix}
A_{ik} \\
b
\end{bmatrix} \times \begin{bmatrix}
B_{kj}
\end{bmatrix}
\]

what fits in fast memory

compromise \( b \)-by-\( b \), \( b \)-by-\( \ell \), \( \ell \)-by-\( b \)

Volume \( 2 \frac{mnp}{b} \)

# Messages \( \left( \frac{mnp}{b^2 \ell} \right) \)
Block matrix-matrix multiplication

\[ \begin{array}{c|c|c}
C_{ij} & A_{ik} \times & B_{kj} \\
\hline
\end{array} \]

\[ b \left( C_C \right) + = A_k \times B_{kj} \]

<table>
<thead>
<tr>
<th>what fits in fast memory M</th>
<th>Volume</th>
<th># Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>compromise</td>
<td>(b)-by-(b), (b)-by-(\ell), (\ell)-by-(b)</td>
<td>(2 \frac{mnp}{b})</td>
</tr>
<tr>
<td>block MM</td>
<td>(b)-by-(b), (b)-by-(b), (b)-by-(b)</td>
<td>(2 \sqrt{3} \frac{mnp}{\sqrt{M}})</td>
</tr>
<tr>
<td>max re-use MM</td>
<td>(b)-by-(b), (b)-by-(1), (1)-by-(b)</td>
<td>(2 \frac{mnp}{\sqrt{M}})</td>
</tr>
</tbody>
</table>

block MM: \(b = \sqrt{M/3}\), \(\ell = \sqrt{M/3}\).
max re-use MM: \(b = \sqrt{M}\), \(\ell = 1\).
Parallel Distributed MM algorithms
Parallel Distributed MM algorithms

- Use the outer version of the matrix-matrix multiply algorithm.
Parallel Distributed MM algorithms

For $k = 1:nb:n$,

End For
Parallel Distributed MM algorithms

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{11}$</td>
<td>$B_{11}$</td>
<td>$C_{11}$</td>
</tr>
<tr>
<td>$A_{21}$</td>
<td>$B_{21}$</td>
<td>$C_{21}$</td>
</tr>
<tr>
<td>$A_{31}$</td>
<td>$B_{31}$</td>
<td>$C_{31}$</td>
</tr>
<tr>
<td>$A_{12}$</td>
<td>$B_{12}$</td>
<td>$C_{12}$</td>
</tr>
<tr>
<td>$A_{22}$</td>
<td>$B_{22}$</td>
<td>$C_{22}$</td>
</tr>
<tr>
<td>$A_{32}$</td>
<td>$B_{32}$</td>
<td>$C_{32}$</td>
</tr>
<tr>
<td>$A_{13}$</td>
<td>$B_{13}$</td>
<td>$C_{13}$</td>
</tr>
<tr>
<td>$A_{23}$</td>
<td>$B_{23}$</td>
<td>$C_{23}$</td>
</tr>
<tr>
<td>$A_{33}$</td>
<td>$B_{33}$</td>
<td>$C_{33}$</td>
</tr>
</tbody>
</table>
Parallel Distributed MM algorithms
Parallel Distributed MM algorithms

Broadcast of size nb*nloc along the columns, root is active_row.
Parallel Distributed MM algorithms

Broadcast of size \( nb \times nloc \) along the columns, root is active_row.

Broadcast of size \( nloc \times nb \) along the rows, root is active_col.
Parallel Distributed MM algorithms

Broadcast of size nb*nloc along the columns, root is active_row.

Broadcast of size nloc*nb along the rows, root is active_col.

Perform matrix matrix multiply: number of FLOPS is nloc*nloc*nb
Parallel Distributed MM algorithms

1. Broadcast of size nb*nloc along the columns, root is active_row.
2. Broadcast of size nloc*nb along the rows, root is active_col.
3. Perform matrix matrix multiply: number of FLOPS is nloc*nloc*nb

bandwidth term for SUMMA: $2\frac{n^2}{\sqrt{P}}\beta$. 
Parallel Distributed MM algorithms

bandwidth term for SUMMA: $2 \frac{n^2}{\sqrt{P}} \beta$. 

bandwidth term for SUMMA: $2 \frac{n_{loc}^3}{\sqrt{M_{loc}}} \beta$.

where

$n_{loc}^3 = \# \text{ multiplications on one proc} = \frac{n^3}{P}$

$M_{loc} = \text{em size of one proc} = \frac{n^2}{P}$
Sequential Lower Bounds for Matrix-Matrix Multiplication

Consider any ordinary dense matrix-matrix multiplication algorithm for multiplying an $m$–by–$n$ matrix with an $n$–by–$p$ matrix, consider a computer with fast memory of size $M$, then

Theorem (Hong and Kung, 1981)
Sequential Lower Bounds for Matrix-Matrix Multiplication

Consider any ordinary dense matrix-matrix multiplication algorithm for multiplying an $m$–by–$n$ matrix with an $n$–by–$p$ matrix, consider a computer with fast memory of size $M$, then

**Theorem (Hong and Kung, 1981)**

**Theorem (Irony, Toledo, and Tiskin, 2004)**

The number of words transferred between slow and fast memory is at least

$$\frac{1}{2\sqrt{2}} \frac{mnp}{\sqrt{M}} - M.$$
Improving Sequential Lower Bound for Matrix-Matrix multiplication

- Ordinary matrix-matrix multiplication algorithm for $C = AB$. 
Improving Sequential Lower Bound for Matrix-Matrix multiplication

- Ordinary matrix-matrix multiplication algorithm for $C = AB$.
  - Explicitly compute each $c_{ijk} = a_{ik}b_{kj}$. 
Improving Sequential Lower Bound for Matrix-Matrix multiplication

- Ordinary matrix-matrix multiplication algorithm for $C = AB$.
  - Explicitly compute each $c_{ijk} = a_{ik}b_{kj}$.
- A sequence of the following instructions define an algorithm:
Improving Sequential Lower Bound for Matrix-Matrix multiplication

- Ordinary matrix-matrix multiplication algorithm for $C = AB$.
  - Explicitly compute each $c_{ijk} = a_{ik}b_{kj}$.
- A sequence of the following instructions define an algorithm:
  - Read an element of $A$, $B$, or $C$ from slow memory.
Improving Sequential Lower Bound for Matrix-Matrix multiplication

- Ordinary matrix-matrix multiplication algorithm for $C = AB$.
  - Explicitly compute each $c_{ijk} = a_{ik}b_{kj}$.
- A sequence of the following instructions define an algorithm:
  - **Read** an element of $A$, $B$, or $C$ from slow memory.
  - **Create** an element of $C$ in fast memory.
Improving Sequential Lower Bound for Matrix-Matrix multiplication

- Ordinary matrix-matrix multiplication algorithm for $C = AB$.
  - Explicitly compute each $c_{ijk} = a_{ik}b_{kj}$.

- A sequence of the following instructions define an algorithm:
  - **Read** an element of $A$, $B$, or $C$ from slow memory.
  - **Create** an element of $C$ in fast memory.
  - **Write** an element of $C$ to slow memory.
Improving Sequential Lower Bound for Matrix-Matrix multiplication

- Ordinary matrix-matrix multiplication algorithm for $C = AB$.
  - Explicitly compute each $c_{ijk} = a_{ik}b_{kj}$.
- A sequence of the following instructions define an algorithm:
  - **Read** an element of $A$, $B$, or $C$ from slow memory.
  - **Create** an element of $C$ in fast memory.
  - **Write** an element of $C$ to slow memory.
  - **Delete** an element of $A$ or $B$ from fast memory.
Split the instructions into segments so exactly $M$ reads and writes occur in each segment.

- $M$ reads and writes.

Segment

\[
\begin{align*}
\text{Read} & \quad a_{11} \\
\text{Read} & \quad b_{11} \\
\text{Create} & \quad c_{111} = a_{11} b_{11} \\
\text{Read} & \quad a_{12} \\
\text{Read} & \quad b_{21} \\
\text{Create} & \quad c_{112} = a_{12} b_{21} \\
\text{Write} & \quad c_{11} \\
\text{Delete} & \quad c_{11}, a_{11}, b_{11} \\
\vdots & \\
\end{align*}
\]
Split the instructions into segments so exactly $M$ reads and writes occur in each segment.

Segment:

- **Read** $a_{11}$
- **Read** $b_{11}$
- **Create** $c_{111} = a_{11}b_{11}$
- **Read** $a_{12}$
- **Read** $b_{21}$
- **Create** $c_{112} = a_{12}b_{21}$
- **Write** $c_{11}$
- **Delete** $c_{11}, a_{11}, b_{11}$
- $\vdots$

- **M reads and writes.**
  - $R_a = \text{number of reads for } A.$

- **Maximize number of creates.**
- **Deletes are free.**
- **$M_a$** is the number of $A$ elements in fast memory at the start.
- **$N_a$** is the number of $A$ elements in fast memory at the end.
Split the instructions into segments so exactly $M$ reads and writes occur in each segment.

- **M reads and writes.**
  - $R_a = \text{number of reads for } A$.
  - $W_a = \text{number of writes for } A$.

Segment

- **Read** $a_{11}$
- **Read** $b_{11}$
- **Create** $c_{111} = a_{11}b_{11}$
- **Read** $a_{12}$
- **Read** $b_{21}$
- **Create** $c_{112} = a_{12}b_{21}$
- **Write** $c_{11}$
- **Delete** $c_{11}, a_{11}, b_{11}$
- $\vdots$
- Split the instructions into segments so exactly $M$ reads and writes occur in each segment.

<table>
<thead>
<tr>
<th>Segment</th>
<th>Read</th>
<th>$a_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Read</td>
<td>$b_{11}$</td>
</tr>
<tr>
<td></td>
<td>Create</td>
<td>$c_{111} = a_{11} b_{11}$</td>
</tr>
<tr>
<td></td>
<td>Read</td>
<td>$a_{12}$</td>
</tr>
<tr>
<td></td>
<td>Read</td>
<td>$b_{21}$</td>
</tr>
<tr>
<td></td>
<td>Create</td>
<td>$c_{112} = a_{12} b_{21}$</td>
</tr>
<tr>
<td></td>
<td>Write</td>
<td>$c_{11}$</td>
</tr>
<tr>
<td></td>
<td>Delete</td>
<td>$c_{11}, a_{11}, b_{11}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- $M$ reads and writes.
  - $R_a = \text{number of reads for } A$.
  - $W_a = \text{number of writes for } A$.
  - Similar for $B$ and $C$. 

- Maximize number of creates.
- Deletes are free.

- $M_a = \text{number of } A \text{ elements in fast memory at the start}$.
- $N_a = \text{number of } A \text{ elements in fast memory at the end}$. 
- Split the instructions into segments so exactly $M$ reads and writes occur in each segment.

\[
\begin{align*}
\text{Segment} & \quad \begin{cases}
\text{Read} & a_{11} \\
\text{Read} & b_{11} \\
\text{Create} & c_{111} = a_{11} b_{11} \\
\text{Read} & a_{12} \\
\text{Read} & b_{21} \\
\text{Create} & c_{112} = a_{12} b_{21} \\
\text{Write} & c_{11} \\
\text{Delete} & c_{11}, a_{11}, b_{11} \\
\vdots & \vdots
\end{cases}
\end{align*}
\]

- $M$ reads and writes.
  - $R_a = \text{number of reads for } A$.
  - $W_a = \text{number of writes for } A$.
  - Similar for $B$ and $C$.

- Maximize number of creates.
Split the instructions into segments so exactly $M$ reads and writes occur in each segment.

- **M reads and writes.**
  - $R_a = \text{number of reads for } A$.
  - $W_a = \text{number of writes for } A$.
  - Similar for $B$ and $C$.

- Maximize number of creates.
- Deletes are free.

Segment:

- **Read** $a_{11}$
- **Read** $b_{11}$
- **Create** $c_{111} = a_{11}b_{11}$
- **Read** $a_{12}$
- **Read** $b_{21}$
- **Create** $c_{112} = a_{12}b_{21}$
- **Write** $c_{11}$
- **Delete** $c_{11}, a_{11}, b_{11}$
- ...
Split the instructions into segments so exactly $M$ reads and writes occur in each segment.

- **Segment**
  - **Read** $a_{11}$
  - **Read** $b_{11}$
  - **Create** $c_{111} = a_{11}b_{11}$
  - **Read** $a_{12}$
  - **Read** $b_{21}$
  - **Create** $c_{112} = a_{12}b_{21}$
  - **Write** $c_{11}$
  - **Delete** $c_{11}, a_{11}, b_{11}$
  - $\vdots$

- **$M$ reads and writes.**
  - $R_a = \text{number of reads for } A$.
  - $W_a = \text{number of writes for } A$.
  - Similar for $B$ and $C$.

- **Maximize number of creates.**

- **Deletes are free.**

- **$M_a = \text{number of } A \text{ elements in fast memory at the start.}$**
Split the instructions into segments so exactly $M$ reads and writes occur in each segment.

- **$M$ reads and writes.**
  - $R_a = \text{number of reads for } A$.
  - $W_a = \text{number of writes for } A$.
  - Similar for $B$ and $C$.

- Maximize number of creates.
- Deletes are free.
- $M_a = \text{number of } A \text{ elements in fast memory at the start}$.
- $N_a = \text{number of } A \text{ elements in fast memory at the end}$.
max(Number of Scalar Multiplications), subject to

\[ R_a + R_b + R_c + W_a + W_b + W_c = M \]

\[ M_a + M_b + M_c \leq M \]

\[ N_a + N_b + N_c \leq M \]

\[ M_a \geq 0, \quad N_a \geq 0, \quad R_a \geq 0, \quad W_a \geq 0 \]

\[ M_b \geq 0, \quad N_b \geq 0, \quad R_b \geq 0, \quad W_b \geq 0 \]

\[ M_c \geq 0, \quad N_c \geq 0, \quad R_c \geq 0, \quad W_c \geq 0 \]
\[
\max(\text{Number of Scalar Multiplications}), \quad \text{subject to}
\]
\[
R_a + R_b + R_c + W_a + W_b + W_c = M
\]
\[
M_a + M_b + M_c \leq M
\]
\[
N_a + N_b + N_c \leq M
\]
\[
M_a \geq 0, \quad N_a \geq 0, \quad R_a \geq 0, \quad W_a \geq 0
\]
\[
M_b \geq 0, \quad N_b \geq 0, \quad R_b \geq 0, \quad W_b \geq 0
\]
\[
M_c \geq 0, \quad N_c \geq 0, \quad R_c \geq 0, \quad W_c \geq 0
\]

- Constraint 1: Total number of reads and writes.
\[ \text{max(Number of Scalar Multiplications), \hspace{1cm} subject to} \]

\[ R_a + R_b + R_c + W_a + W_b + W_c = M \]

\[ M_a + M_b + M_c \leq M \]

\[ N_a + N_b + N_c \leq M \]

\[ M_a \geq 0, \hspace{0.5cm} N_a \geq 0, \hspace{0.5cm} R_a \geq 0, \hspace{0.5cm} W_a \geq 0 \]

\[ M_b \geq 0, \hspace{0.5cm} N_b \geq 0, \hspace{0.5cm} R_b \geq 0, \hspace{0.5cm} W_b \geq 0 \]

\[ M_c \geq 0, \hspace{0.5cm} N_c \geq 0, \hspace{0.5cm} R_c \geq 0, \hspace{0.5cm} W_c \geq 0 \]

- **Constraint 1:** Total number of reads and writes.
- **Constraint 2:** Total number of elements at start of segment.
max(\text{Number of Scalar Multiplications}), \text{ subject to}

\begin{align*}
R_a + R_b + R_c + W_a + W_b + W_c &= M \\
M_a + M_b + M_c &\leq M \\
N_a + N_b + N_c &\leq M \\
M_a &\geq 0, \ N_a \geq 0, \ R_a \geq 0, \ W_a \geq 0 \\
M_b &\geq 0, \ N_b \geq 0, \ R_b \geq 0, \ W_b \geq 0 \\
M_c &\geq 0, \ N_c \geq 0, \ R_c \geq 0, \ W_c \geq 0
\end{align*}

- Constraint 1: Total number of reads and writes.
- Constraint 2: Total number of elements at start of segment.
- Constraint 3: Total number of elements at end of segment.
\[
\max(\text{Number of Scalar Multiplications}), \quad \text{subject to}
\]
\[
R_a + R_b + R_c + W_a + W_b + W_c = M
\]
\[
M_a + M_b + M_c \leq M
\]
\[
N_a + N_b + N_c \leq M
\]
\[
M_a \geq 0, \quad N_a \geq 0, \quad R_a \geq 0, \quad W_a \geq 0
\]
\[
M_b \geq 0, \quad N_b \geq 0, \quad R_b \geq 0, \quad W_b \geq 0
\]
\[
M_c \geq 0, \quad N_c \geq 0, \quad R_c \geq 0, \quad W_c \geq 0
\]

- **Constraint 1**: Total number of reads and writes.
- **Constraint 2**: Total number of elements at start of segment.
- **Constraint 3**: Total number of elements at end of segment.
- **Constraint 4**: Nonnegative.
Lemma (Loomis-Whitney Inequality)

Let $V \in \mathbb{Z}^3$ be a finite set, and let $V_x$, $V_y$, and $V_z$ be orthogonal projections of $V$ onto the coordinate planes. The cardinality of $V$, $|V|$, satisfies

$$|V| \leq \sqrt{|V_x| \cdot |V_y| \cdot |V_z|}.$$
max $\sqrt{(M_a + R_a)(M_b + R_b)(N_c + W_c)}$, subject to

$R_a + R_b + R_c + W_a + W_b + W_c = M$

$M_a + M_b + M_c \leq M$

$N_a + N_b + N_c \leq M$

$M_a \geq 0$, $N_a \geq 0$, $R_a \geq 0$, $W_a \geq 0$

$M_b \geq 0$, $N_b \geq 0$, $R_b \geq 0$, $W_b \geq 0$

$M_c \geq 0$, $N_c \geq 0$, $R_c \geq 0$, $W_c \geq 0$

- Loomis-Whitney inequality.
\[ \max \sqrt{(M_a + R_a)(M_b + R_b)(N_c + W_c)}, \quad \text{subject to} \]
\[ R_a + R_b + R_c + W_a + W_b + W_c = M \]
\[ M_a + M_b + M_c \leq M \]
\[ N_a + N_b + N_c \leq M \]
\[ M_a \geq 0, \quad N_a \geq 0, \quad R_a \geq 0, \quad W_a \geq 0 \]
\[ M_b \geq 0, \quad N_b \geq 0, \quad R_b \geq 0, \quad W_b \geq 0 \]
\[ M_c \geq 0, \quad N_c \geq 0, \quad R_c \geq 0, \quad W_c \geq 0 \]

- Loomis-Whitney inequality.
- \( M_a + R_a \): Maximum number of elements of \( A \) in fast memory.
\[
\max \sqrt{(M_a + R_a)(M_b + R_b)(N_c + W_c)}, \quad \text{subject to}
\]
\[
R_a + R_b + R_c + W_a + W_b + W_c = M
\]
\[
M_a + M_b + M_c \leq M
\]
\[
N_a + N_b + N_c \leq M
\]
\[
M_a \geq 0, \quad N_a \geq 0, \quad R_a \geq 0, \quad W_a \geq 0
\]
\[
M_b \geq 0, \quad N_b \geq 0, \quad R_b \geq 0, \quad W_b \geq 0
\]
\[
M_c \geq 0, \quad N_c \geq 0, \quad R_c \geq 0, \quad W_c \geq 0
\]

- Loomis-Whitney inequality.
- \(M_a + R_a\): Maximum number of elements of \(A\) in fast memory.
- \(M_b + R_b\): Maximum number of elements of \(B\) in fast memory.
\[
\max \sqrt{(M_a + R_a)(M_b + R_b)(N_c + W_c)}, \quad \text{subject to}
\]
\[
R_a + R_b + R_c + W_a + W_b + W_c = M \\
M_a + M_b + M_c \leq M \\
N_a + N_b + N_c \leq M
\]
\[
M_a \geq 0, \quad N_a \geq 0, \quad R_a \geq 0, \quad W_a \geq 0 \\
M_b \geq 0, \quad N_b \geq 0, \quad R_b \geq 0, \quad W_b \geq 0 \\
M_c \geq 0, \quad N_c \geq 0, \quad R_c \geq 0, \quad W_c \geq 0
\]

- Loomis-Whitney inequality.
- \(M_a + R_a\): Maximum number of elements of \(A\) in fast memory.
- \(M_b + R_b\): Maximum number of elements of \(B\) in fast memory.
- \(N_c + W_c\): Maximum number of elements of \(C\) in fast memory.
\[
\max \sqrt{(M_a + R_a)(M_b + R_b)(N_c + W_c)}, \quad \text{subject to}
\]
\[
R_a + R_b + R_c + W_a + W_b + W_c = M \\
M_a + M_b + M_c \leq M \\
N_a + N_b + N_c \leq M
\]

- \(R_c, W_a, W_b, M_c, N_a,\) and \(N_b\) do not appear in objective function and \(N_c \leq M\).
\[
\max \sqrt{(M_a + R_a)(M_b + R_b)(M + W_c)}, \quad \text{subject to}
\]
\[
R_a + R_b + W_c = M
\]
\[
M_a + M_b \leq M
\]

- \(R_c, W_a, W_b, M_c, N_a, \) and \(N_b\) do not appear in objective function and \(N_c \leq M\).
- Set each to zero since nonzero values will only reduce objective function. Set \(N_c\) to \(M\).
\[
\max \sqrt{(M_a + R_a)(M_b + R_b)(M + W_c)}, \quad \text{subject to}
\]
\[
R_a + R_b + W_c = M \\
M_a + M_b \leq M
\]

- \(R_c, W_a, W_b, M_c, N_a, \) and \(N_b\) do not appear in objective function and \(N_c \leq M\).
- Set each to zero since nonzero values will only reduce objective function. Set \(N_c\) to \(M\).
- Each variable is bounded by \(M\). Therefore,
\[
\sqrt{(M_a + R_a)(M_b + R_b)(M + W_c)} \leq 2\sqrt{2}M^{3/2}.
\]
\[
\text{max} \sqrt{(M_a + R_a)(M_b + R_b)(M + W_c)}, \quad \text{subject to} \\
R_a + R_b + W_c = M \\
M_a + M_b \leq M
\]

- \(R_c, W_a, W_b, M_c, N_a, \) and \(N_b\) do not appear in objective function and \(N_c \leq M\).
- Set each to zero since nonzero values will only reduce objective function. Set \(N_c\) to \(M\).
- Each variable is bounded by \(M\). Therefore,
\[
\sqrt{(M_a + R_a)(M_b + R_b)(M + W_c)} \leq 2\sqrt{2}M^{3/2}.
\]
- A lower bound for the number of words transferred is
\[
\left\lfloor \frac{mnp}{2\sqrt{2}M^{3/2}} \right\rfloor (M) \leq \frac{1}{2\sqrt{2}} \frac{mnp}{\sqrt{M}} - M.
\]
Ways to improve lower bound

1. Change length of a segment.
2. Improve majorization or solve exactly.
3. Increase the number of segments.
1. **Change length of a segment.**

One segment of length $\alpha M$.

$$\max \sqrt{(M_a + R_a)(M_b + R_b)(M + W_c)}, \quad \text{subject to}$$

$$R_a + R_b + W_c = \alpha M$$

$$M_a + M_b \leq M$$
1. Change length of a segment.

One segment of length $\alpha M$.
- Upper bound number of scalar multiplications in one segment is
  $$(1 + \alpha)^{3/2} M^{3/2}.$$ 
- The minimum number of reads and writes is
  $$\left\lfloor \frac{mnp}{(1 + \alpha)^{3/2} M^{3/2}} \right\rfloor (\alpha M) \geq \frac{\alpha}{(1 + \alpha)^{3/2}} \frac{mnp}{\sqrt{M}} - \alpha M.$$ 
- $\alpha = 2$ maximizes the constant.
- A lower bound for the volume of words transferred is
  $$\text{Volume} \geq \frac{2}{3\sqrt{3}} \frac{mnp}{\sqrt{M}} - 2M.$$ 
- Increased constant from about 0.35 to about 0.38. Yeah!
2. Improve majorization or solve exactly.

\[
\max \sqrt{(M_a + R_a)(M_b + R_b)(M + W_c)}, \quad \text{subject to}
\]
\[
R_a + R_b + W_c = M
\]
\[
M_a + M_b \leq M
\]
2. Improve majorization or solve exactly.

\[
\max \sqrt{(M_a + R_a)(M_b + R_b)(M + W_c)} \quad \text{subject to}
\]
\[
R_a + R_b + W_c = M
\]
\[
M_a + M_b \leq M
\]

- Solving exactly, a lower bound for the number of words transferred is

\[
\left\lfloor \frac{mnp}{M^{3/2}} \right\rfloor (\alpha M)
\]

- A lower bound for the volume of words transferred is

\[
\text{Volume} \geq 1 \frac{mnp}{\sqrt{M}} - M.
\]
Solving exactly for arbitrary segment length, $\alpha M$.

$$\max \sqrt{(M_a + R_a)(M_b + R_b)(M + W_c)}, \quad \text{subject to}$$

$$R_a + R_b + W_c = \alpha M$$

$$M_a + M_b \leq M$$
Solving exactly for arbitrary segment length, $\alpha M$.

$$\max \sqrt{(M_a + R_a)(M_b + R_b)(M + W_c)}, \quad \text{subject to}$$

$$R_a + R_b + W_c = \alpha M$$

$$M_a + M_b \leq M$$

- A lower bound for the number of words transferred is:

$$\left\lfloor \frac{mnp}{\left(\frac{2+\alpha}{3}\right)^{3/2}M^{3/2}} \right\rfloor (\alpha M)$$

- $\alpha = 4$ maximizes the constant.

- A lower bound for the volume of words transferred is:

$$Volume \geq \sqrt{2} \frac{mnp}{\sqrt{M}} - 4M.$$
3. Increase number of segments.

(Two segment problem formulation)

\[
\max \sqrt{(M_a^{(0)} + R_a^{(1)})(M_b^{(0)} + R_b^{(1)})(M_c^{(1)} + W_c^{(1)})} \\
+ \sqrt{(M_a^{(1)} + R_a^{(2)})(M_b^{(1)} + R_b^{(2)})(M_c^{(2)} + W_c^{(2)})},
\]

subject to

\[
M_a^{(0)} + M_b^{(0)} + M_c^{(0)} \leq M \\
R_a^{(1)} + R_b^{(1)} + R_c^{(1)} + W_a^{(1)} + W_b^{(1)} + W_c^{(1)} = \alpha M \\
M_a^{(1)} + M_b^{(1)} + M_c^{(1)} \leq M \\
R_a^{(2)} + R_b^{(2)} + R_c^{(2)} + W_a^{(2)} + W_b^{(2)} + W_c^{(2)} = \alpha M \\
M_a^{(2)} + M_b^{(2)} + M_c^{(2)} \leq M
\]
Two segment solution

- Used GAMS global optimization solver lindoglobal.
  - Returns a global solution.
  - Branch-and-cut global optimization procedure.
- Maximized the lower bound when \((s = 2, \alpha = 2)\).
- Lower bound on volume of message transferred is
  \[
  \text{Volume} \geq 1.57 \frac{mnp}{\sqrt{M}} - 4M.
  \]
- \((s = 3, \alpha = 2)\) gives 1.65.
- \((s = 4, \alpha = 2)\) gives 1.73.
Ways to improve lower bound

1. Change length of a segment.
2. Improve majorization or solve exactly.
3. Increase the number of segments.

\[
\max \sqrt{(M_a + R_a)(M_b + R_b)(N_c + W_c)}, \quad \text{subject to}
\]

\[
R_a + R_b + W_c = M \\
M_a + M_b \leq M \\
N_c \leq M
\]

\[
\max \sqrt{(M_a + R_a)(M_b + R_b)(N_c + W_c)}, \quad \text{subject to}
\]

\[
R_a + R_b + W_c = M
\]
\[
M_a + M_b \leq M
\]
\[
N_c \leq M
\]


\[
\max \sqrt{(M_a + R_a)(M_b + R_b)(M_c + R_c)}, \quad \text{subject to}
\]

\[
R_a + R_b + R_c = M
\]
\[
M_a + M_b + M_c \leq M
\]


\[
\max \sqrt{(M_a + R_a)(M_b + R_b)(N_c + W_c)}, \quad \text{subject to} \\
R_a + R_b + W_c = M \\
M_a + M_b \leq M \\
N_c \leq M
\]


\[
\max \sqrt{(M_a + R_a)(M_b + R_b)(M_c + R_c)}, \quad \text{subject to} \\
R_a + R_b + R_c = M \\
M_a + M_b + M_c \leq M
\]

Right away, this formulation gives \(\sqrt{\frac{27}{8}} \approx 1.84\).
Ways to improve lower bound

1. Change length of a segment.
2. Improve majorization or solve exactly.
3. Increase the number of segments.
Ways to improve lower bound

1. Change length of a segment.
2. Improve majorization or solve exactly.
3. Increase the number of segments.

gives

Lower bound on volume of message transferred is

$$\text{Volume} \geq 2 \frac{mnp}{\sqrt{M}} - 2M.$$
$\beta^{-1} = 10^8 \text{words/sec}$  \hspace{0.5cm} $\gamma^{-1} = 10^{10} \text{flops/sec}$  \hspace{0.5cm} $M = 10^6 \text{words}$
What’s next.

1. Work on the latency.
   lowest known upper bound: $3\sqrt{3} \frac{n^3}{M^{3/2}}$
   greatest known lower bound: $2 \frac{n^3}{M^{3/2}}$
2. Work on the latency/bandwidth compromise.