Improving Communication Lower Bounds for Matrix-Matrix Multiplication

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

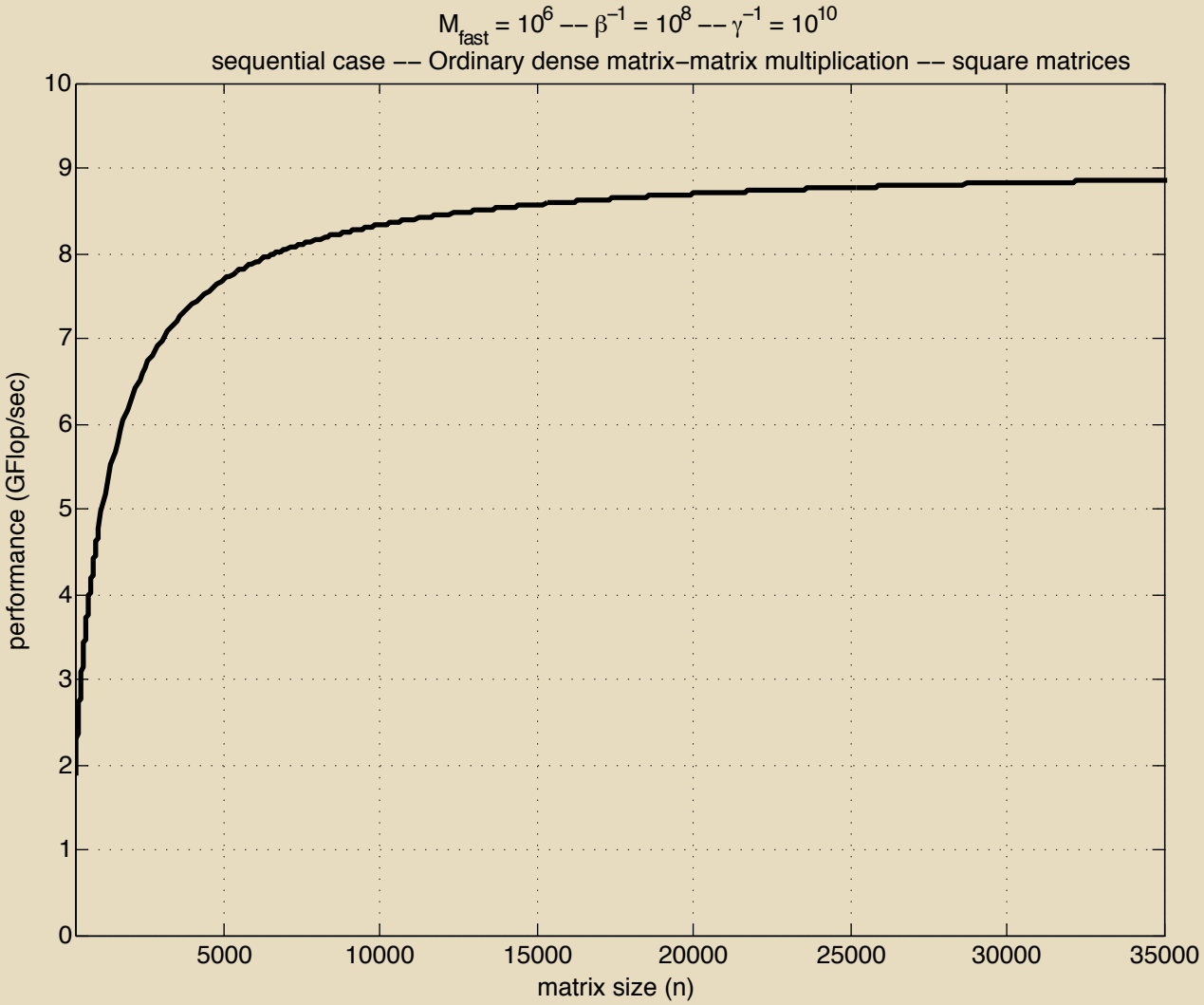
Bradley R. Lowery and Julien Langou

University of Colorado Denver

July 2, 2014

Department of Mathematical
& Statistical Sciences
UNIVERSITY OF COLORADO **DENVER**

$\beta^{-1} = 10^8$words/sec $\quad \gamma^{-1} = 10^{10}$flops/sec $\quad M = 10^6$words



Mission Statement

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

Mission Statement

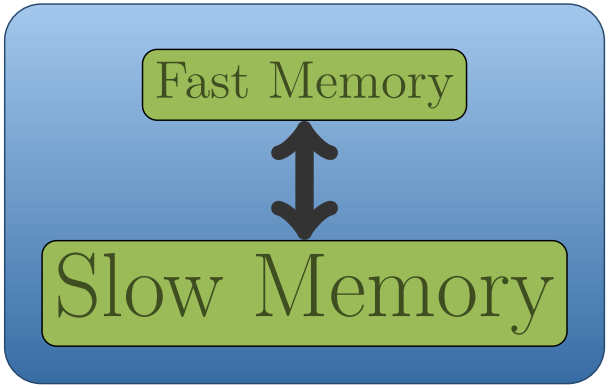
We study communication costs for the ordinary **dense** (OD) matrix-matrix multiplication in the sequential model.

- **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

α (latency), β (inverse of bandwidth), γ (inverse of bandwidth)

time = $\alpha\times$ (# message) + $\beta\times$ (total vol. of communication) + $\gamma\times$ (# 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)

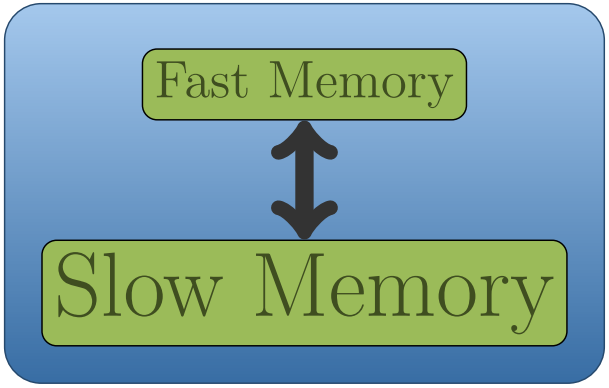
$$c_{ijk} = a_{ik} \cdot b_{kj}$$

(consequence: Strassen-like matrix-matrix multilplications are not allowed.)

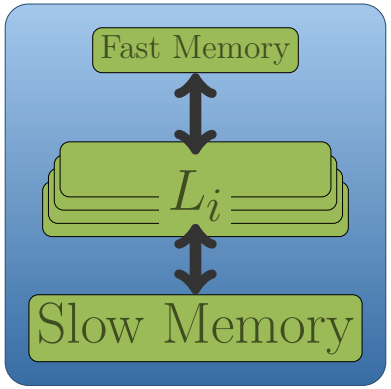
The present study is only (mainly) concerned with the volume of communication (bandwidth term).
Important to realize that this generalizes to

- **# of messages** (latency related) (as opposed to "total volume of messages", bandwidth related)
- **parallel distributed**
- **hierarchical memories**

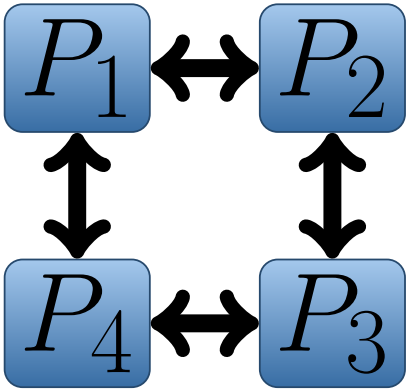
Sequential



Hierarchical



Parallel

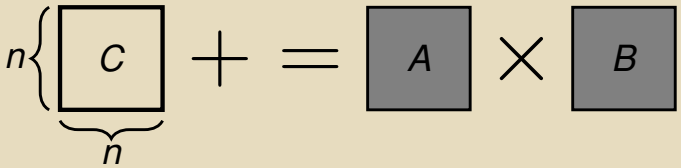


Mission Statement

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.



- 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;

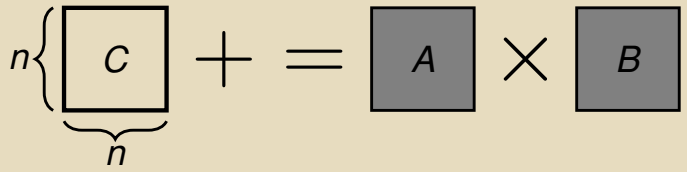
- Communication cost is $3n^2$

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.



- 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-by-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,

$$b\left\{\underbrace{\boxed{C_{ij}}}_{b}\right. + = \boxed{A_{ik}} \times \boxed{B_{kj}}$$

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-by-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,

$$b\left\{\underbrace{\boxed{C_{ij}}}_{b}\right. + = \boxed{A_{ik}} \times \boxed{B_{kj}}$$

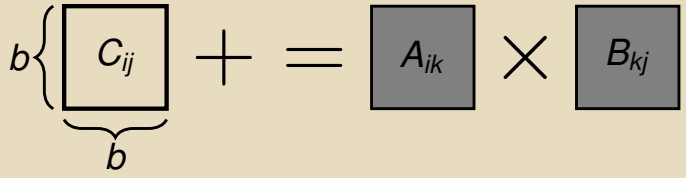
end; end; end;

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

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

- **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 ...)
- **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,



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 $\left(\frac{n}{b}\right)^3 \left(2b^3\right) \quad \rightarrow \quad 2n^3 \quad \rightarrow \quad$ perfect.
- Communication cost is $\left(\frac{n}{b}\right)^3 \left(2b^2\right) \quad \rightarrow \quad \left(\frac{2}{b}\right) n^3 \quad \rightarrow \quad$ 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 β being the time to move one unit of data (inverse of bandwidth) and γ 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 β being the time to move one unit of data (inverse of bandwidth) and γ 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 β being the time to move one unit of data (inverse of bandwidth) and γ 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)

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

$$\frac{1}{2\sqrt{2}}\frac{mnp}{\sqrt{M}} - M.$$

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: $0.35 \approx (2\sqrt{2})^{-1}$

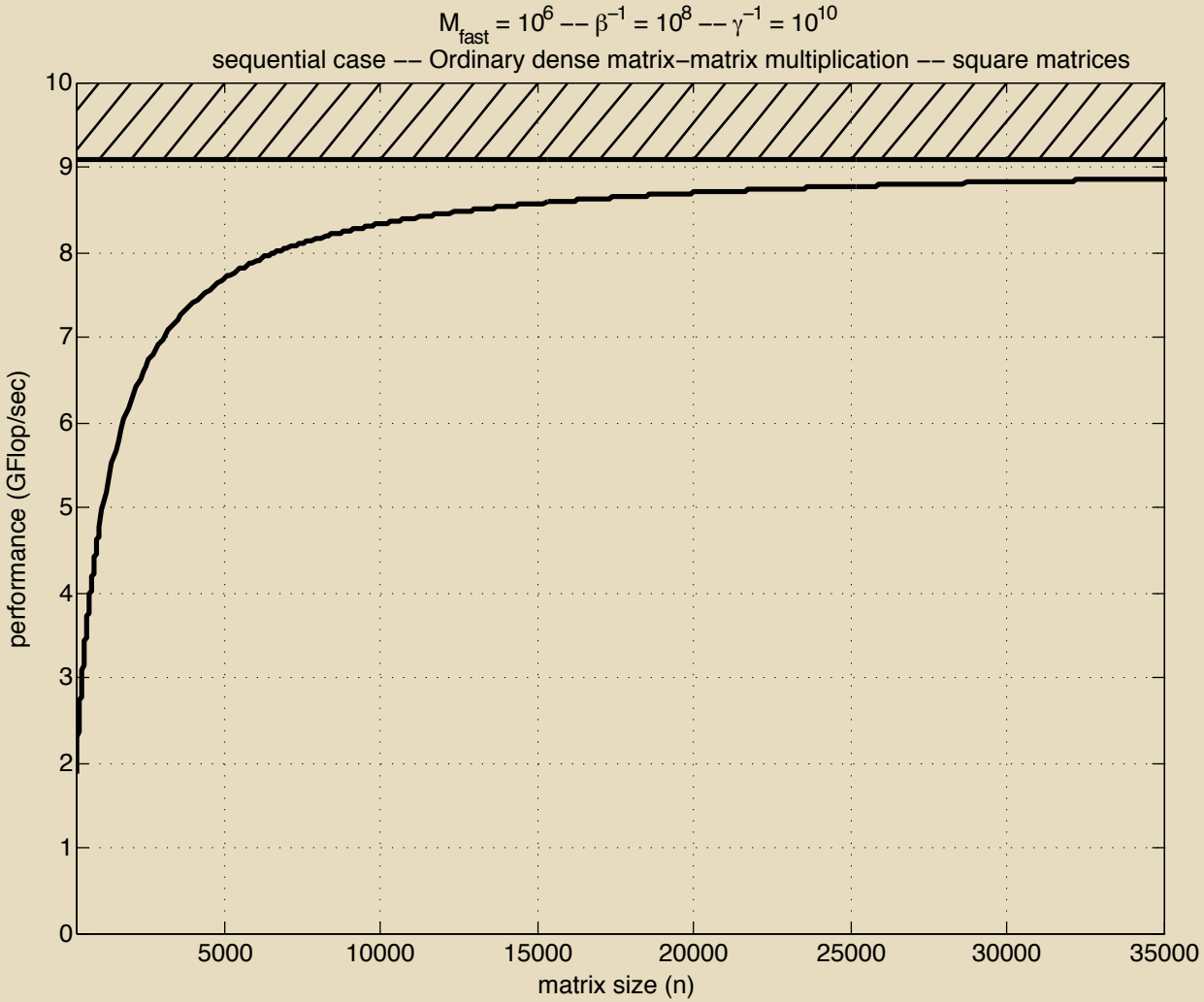
The time of an OD matrix-matrix multiplication is

$$(?)\,\beta n^3 + 2\gamma n^3$$

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

We know that (?) is between 0.35 and 3.46.

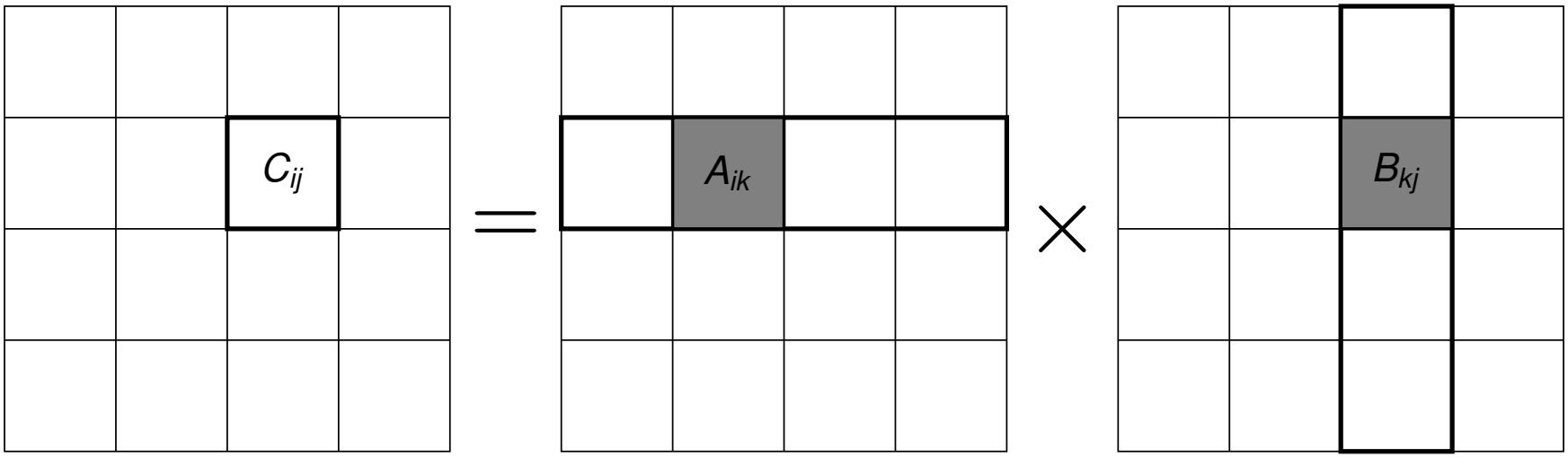
$\beta^{-1} = 10^8$words/sec $\quad \gamma^{-1} = 10^{10}$flops/sec $\quad M = 10^6$words



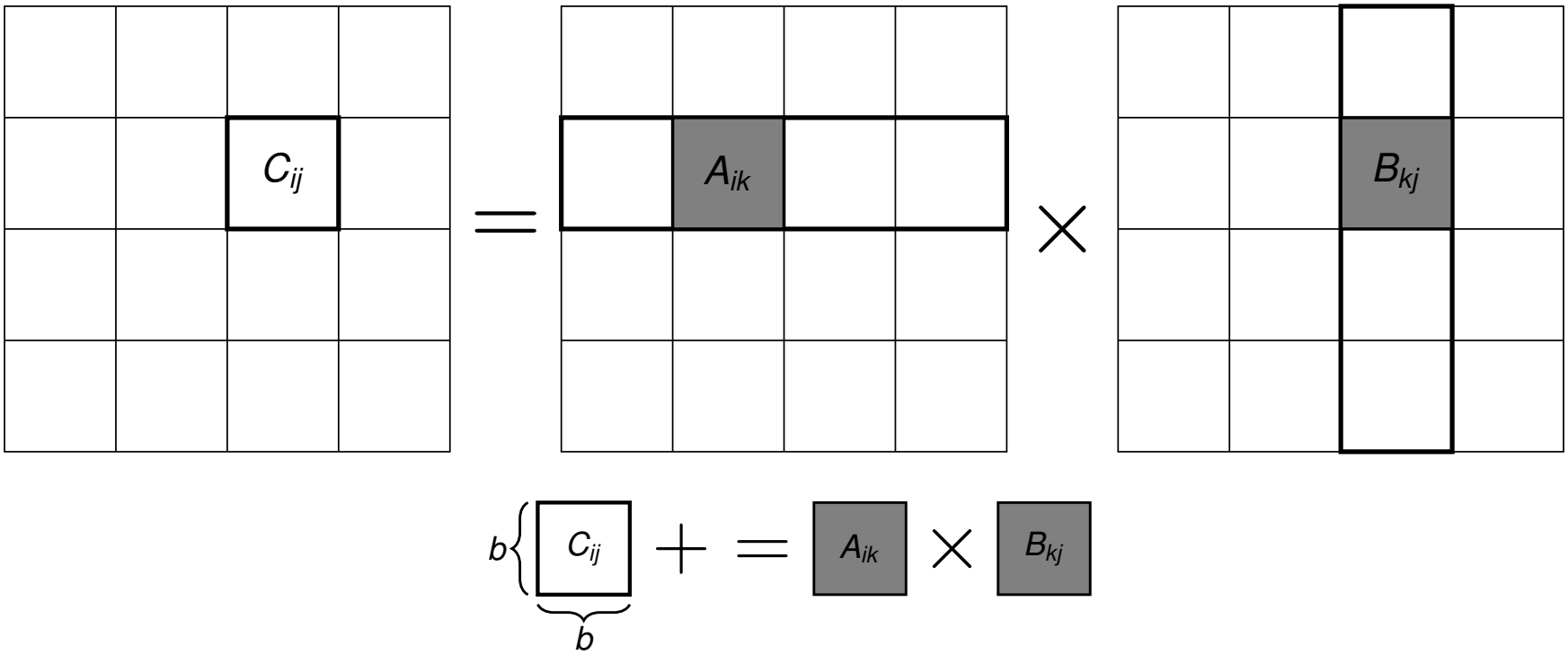
Block matrix-matrix multiplication

See: the *blocked matrix-multiply algorithm* of S. Toledo. A survey of out-of-core algorithms in numerical linear algebra. In External Memory Algorithms and Visualization, pages 161–180. American Mathematical Society Press, 1999.

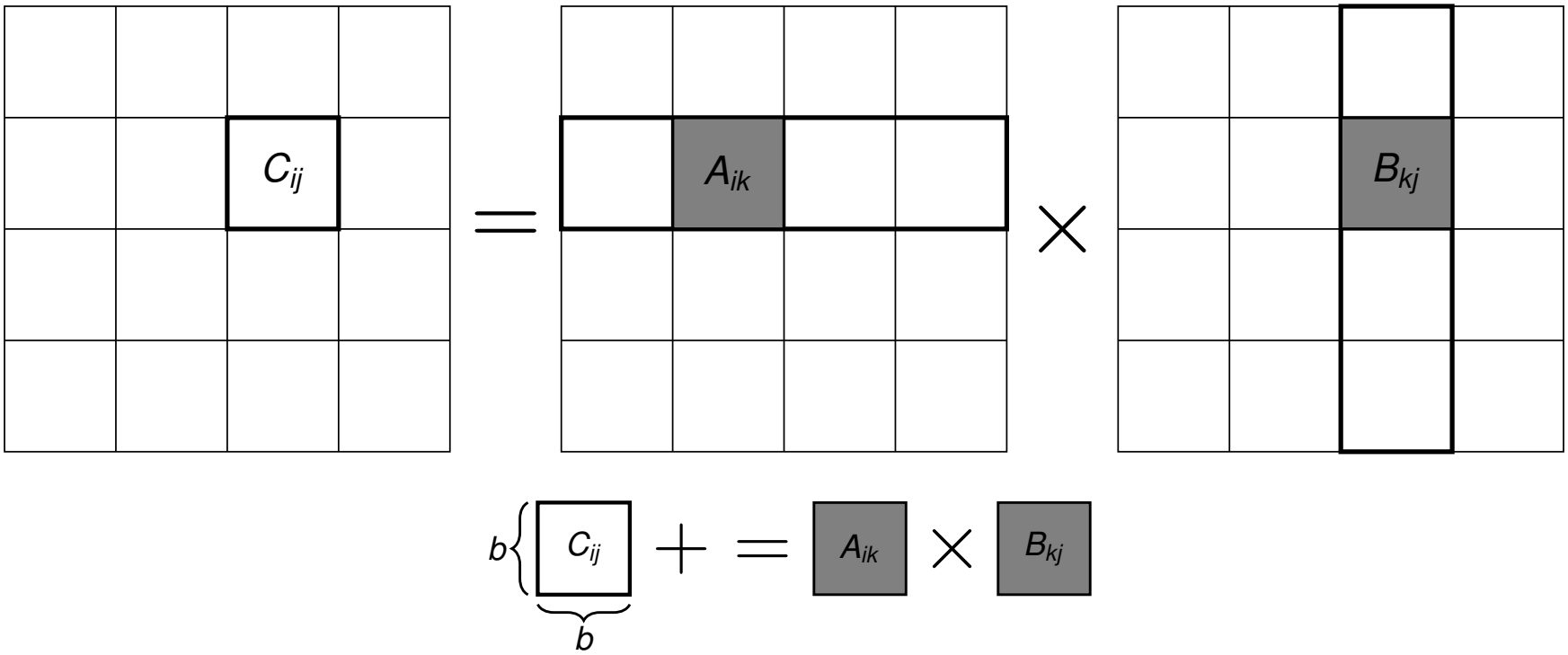
Block matrix-matrix multiplication



Block matrix-matrix multiplication

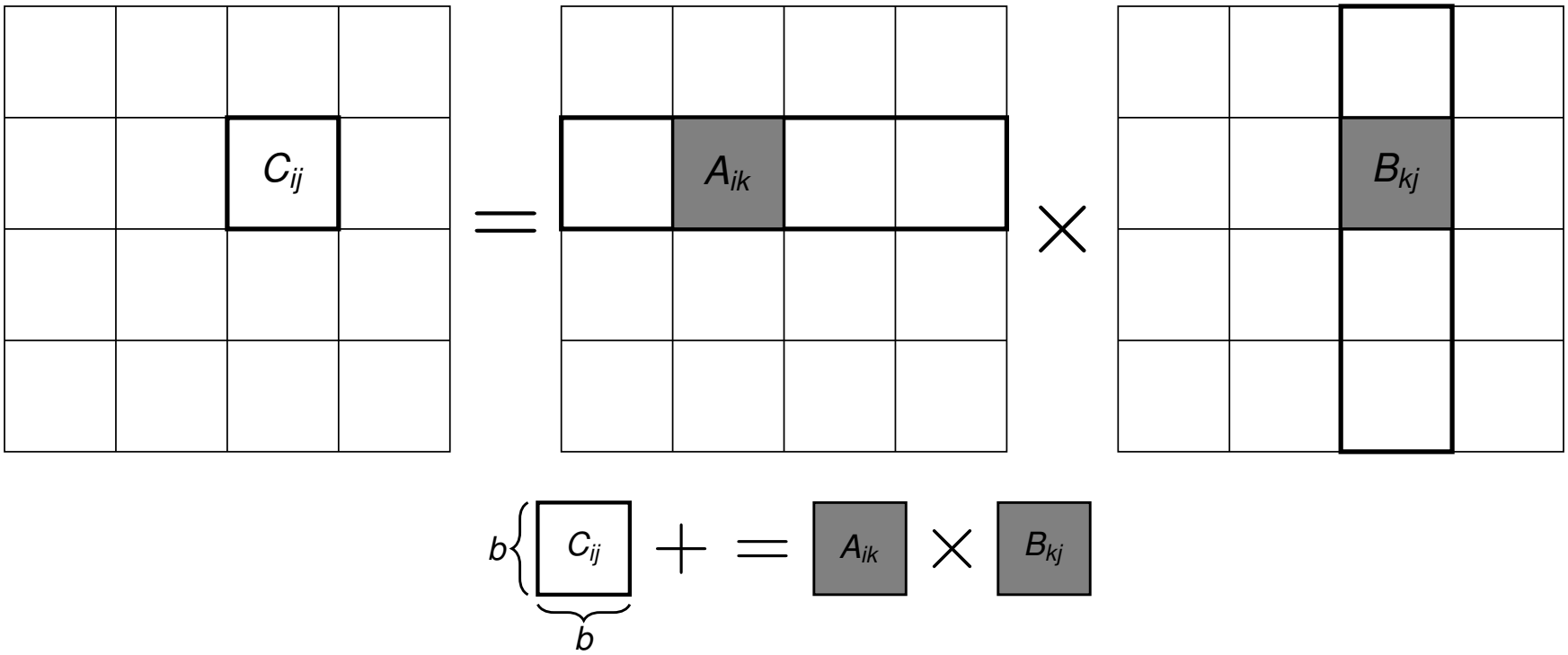


Block matrix-matrix multiplication



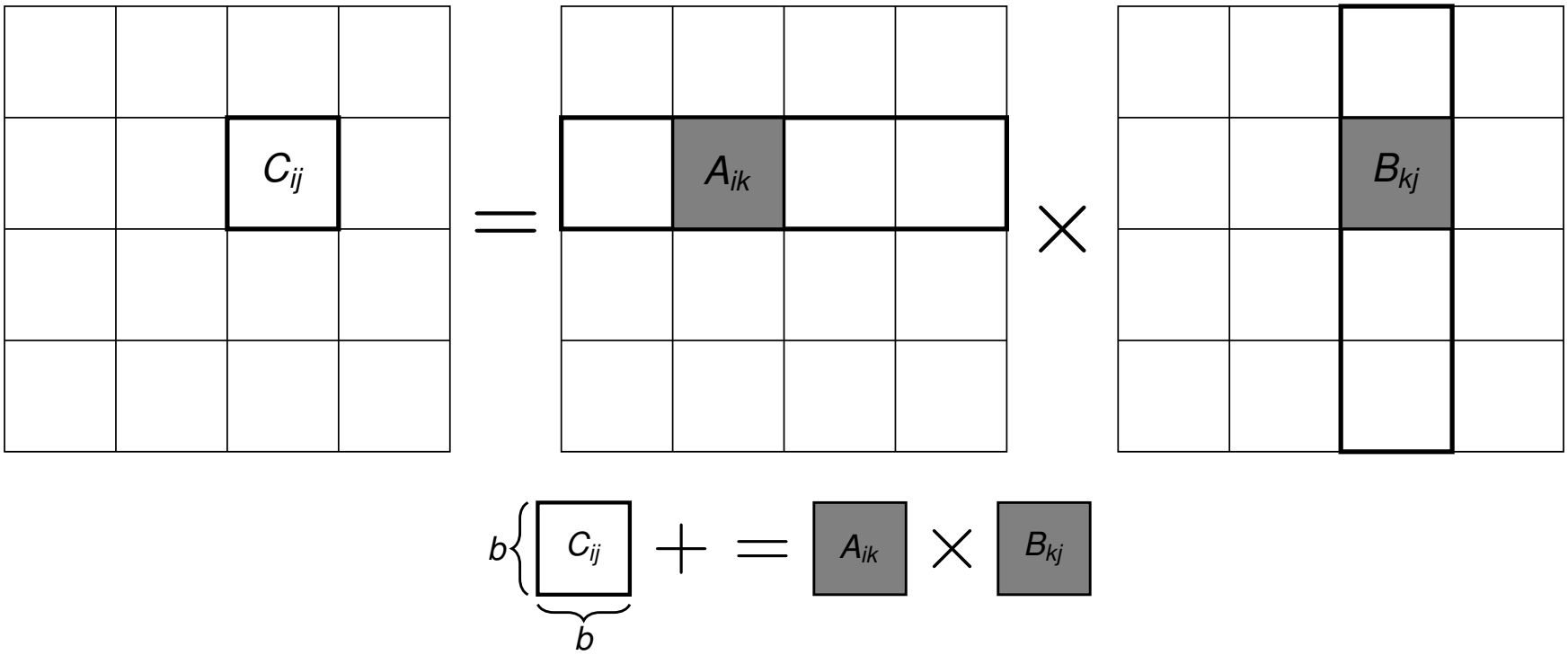
- Three square blocks fit in fast memory: $3b^2 = M$.

Block matrix-matrix multiplication



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

Block matrix-matrix multiplication

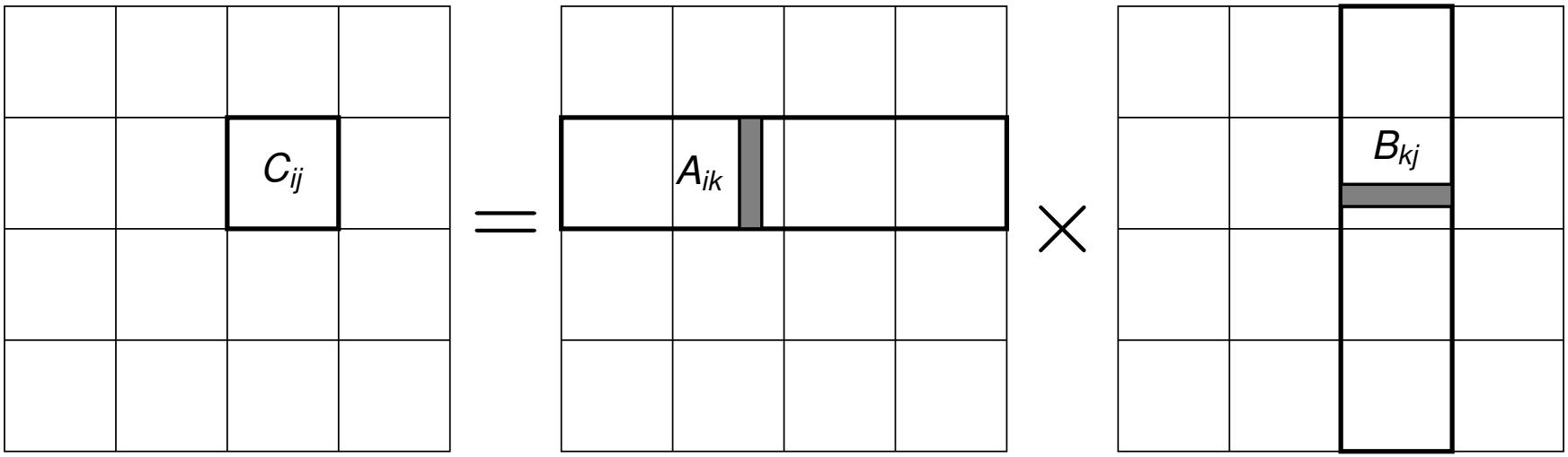


- Three square blocks fit in fast memory: $3b^2 = M$.
- Good bandwidth: Volume $= 2\sqrt{3}\dfrac{mnp}{\sqrt{M}}$
- Good latency: # Messages $= 3\sqrt{3}\dfrac{mnp}{M^{3/2}}$

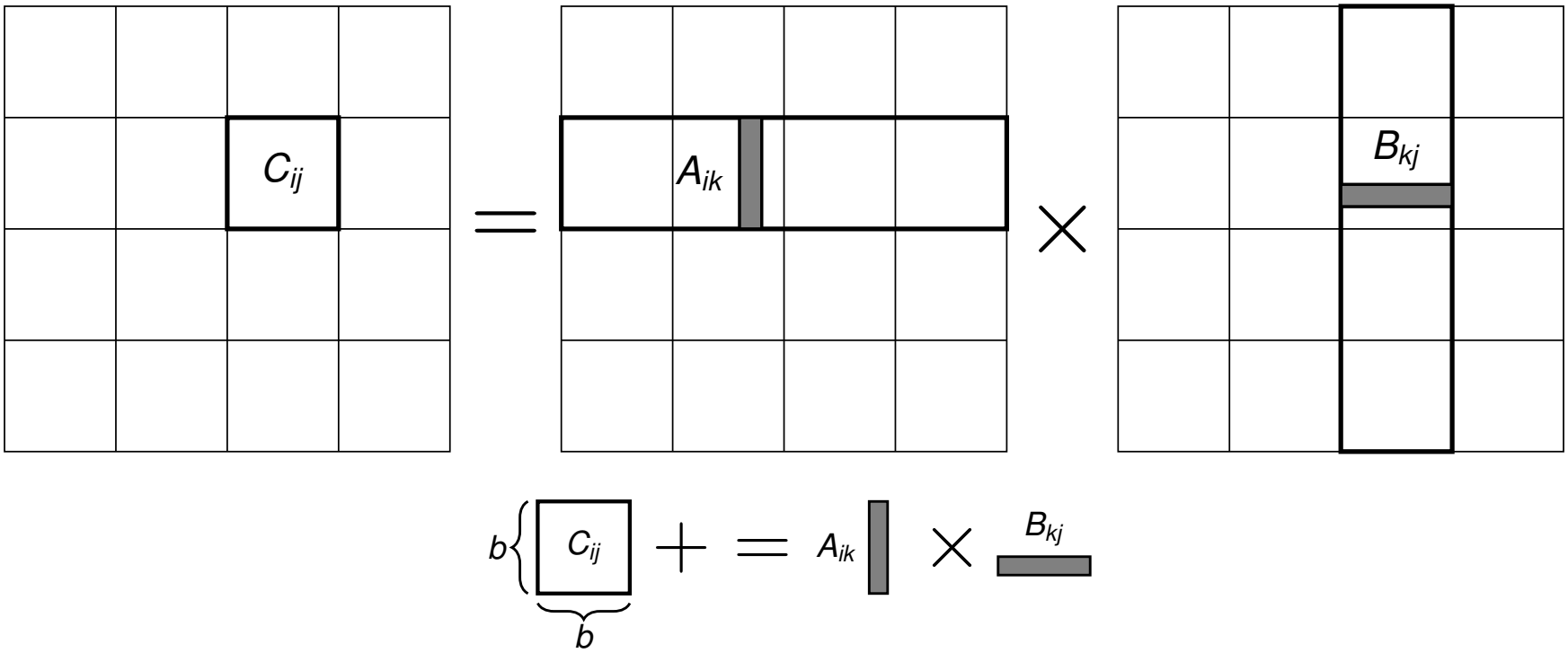
Block matrix-matrix multiplication

See: the *maximum re-use matrix-multiply algorithm* of Dongarra, Pineau, Robert, Shi, and Vivien. “Revisiting Matrix Product on Master-Worker Platforms.” IEEE International Parallel and Distributed Processing Symposium, 2007. IPDPS 2007.
See PUMMA / SUMMA parallel distributed algorithms.

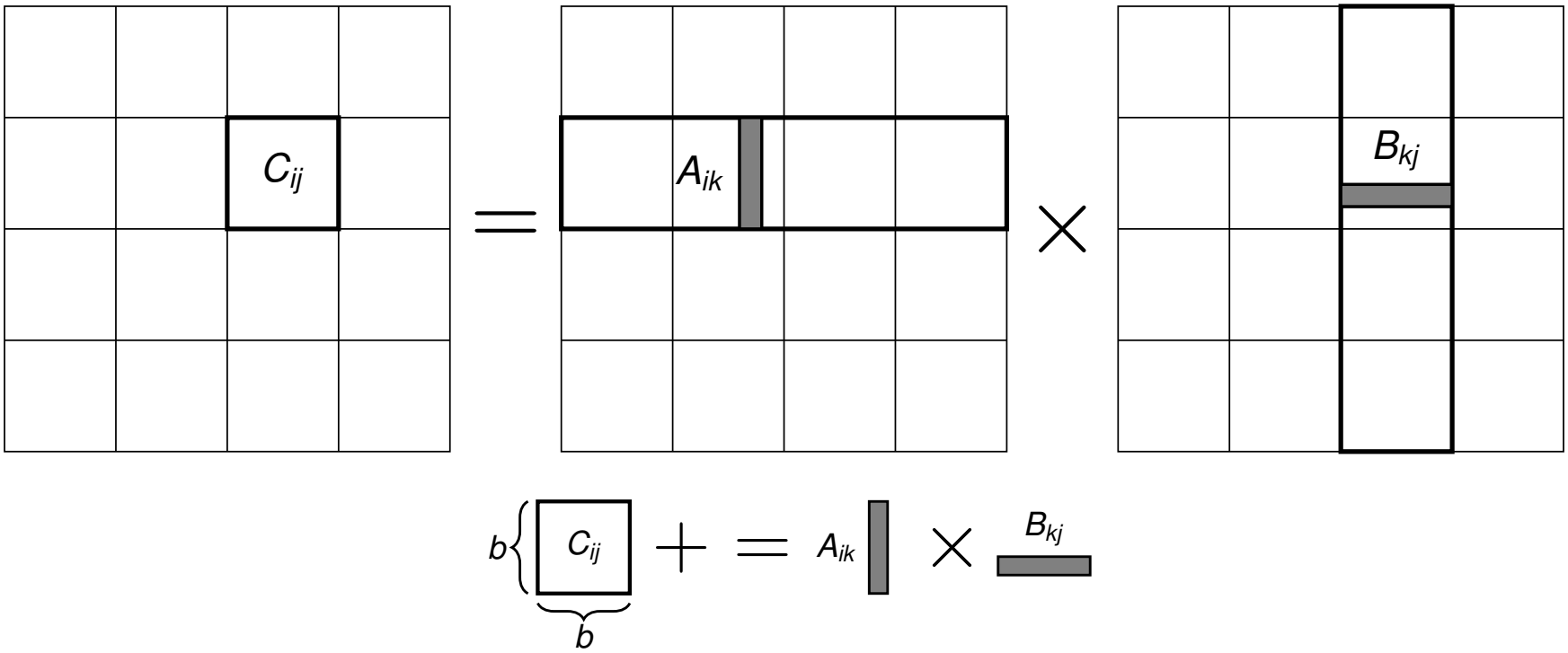
Block matrix-matrix multiplication



Block matrix-matrix multiplication

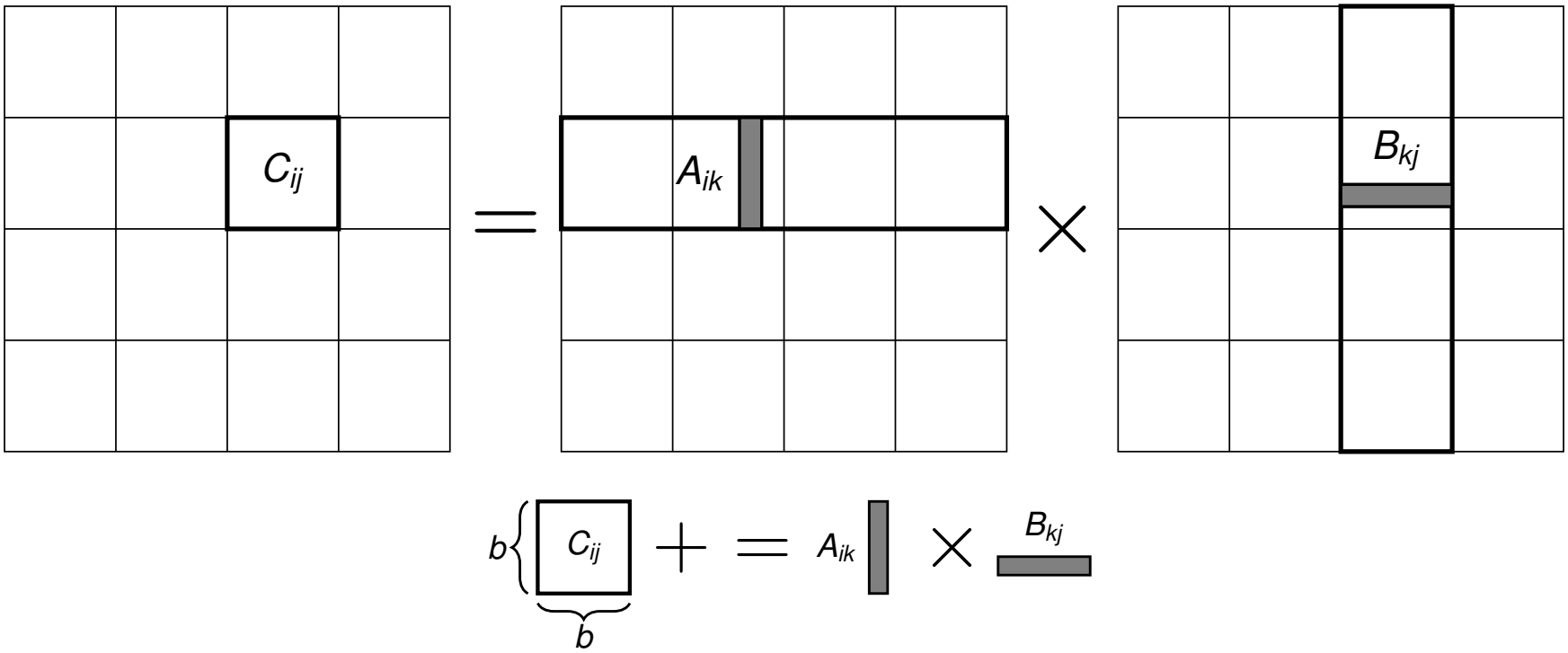


Block matrix-matrix multiplication



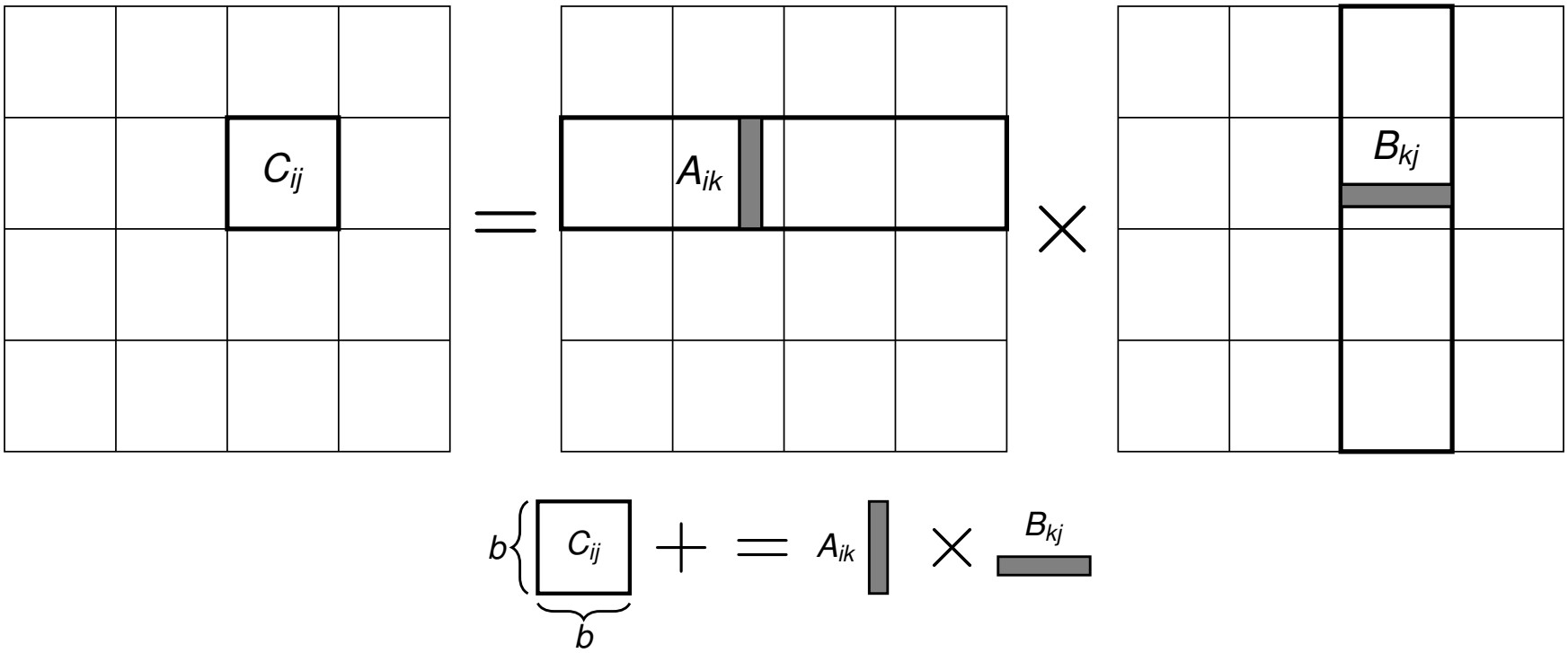
- Block C_{ij} fits in fast memory: $b^2 \approx M$.

Block matrix-matrix multiplication



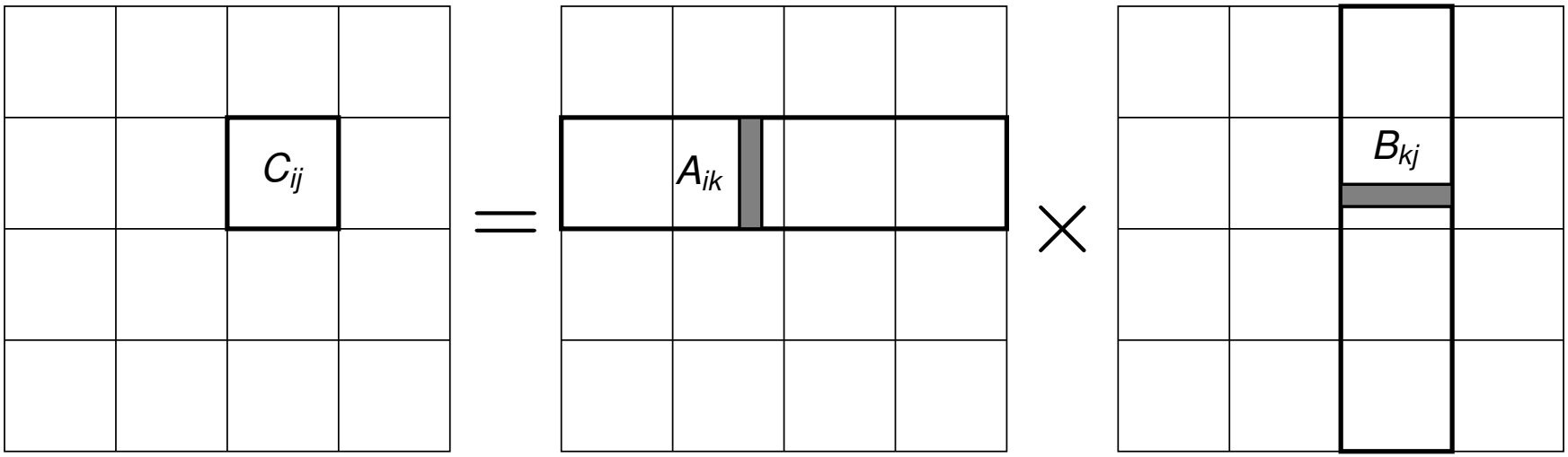
- Block C_{ij} fits in fast memory: $b^2 \approx M$.
- Better bandwidth: Volume $= 2\dfrac{mnp}{\sqrt{M}}$

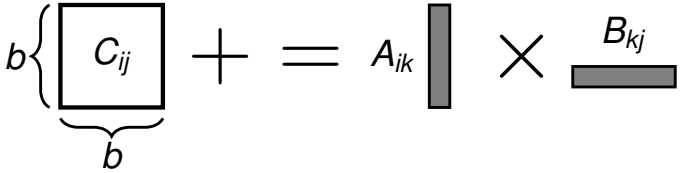
Block matrix-matrix multiplication



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

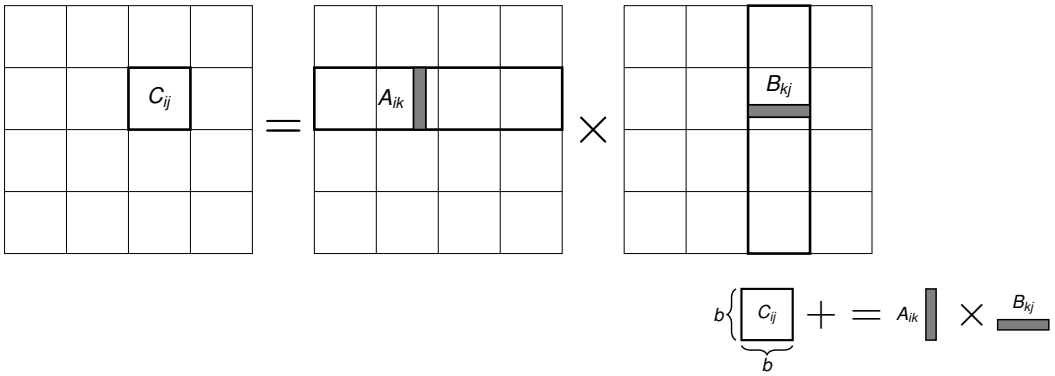
Block matrix-matrix multiplication





	what fits in fast memory	Volume	# Messages
compromise	b-by-b, b-by-ℓ, ℓ-by-b	$2\frac{mnp}{b}$	$\left(\frac{mnp}{b^2\ell}\right)$

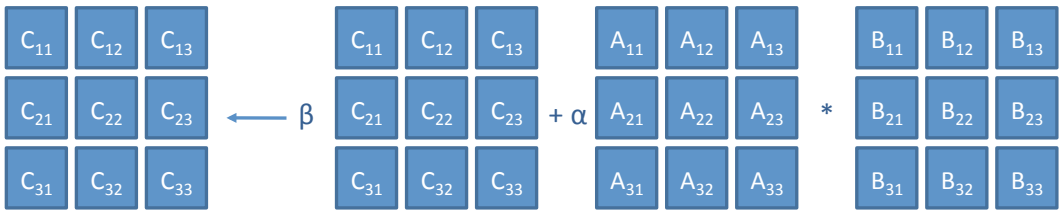
Block matrix-matrix multiplication



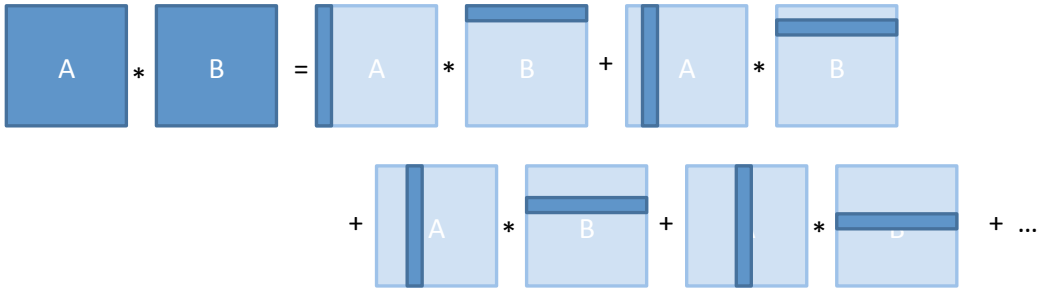
	what fits in fast memory M	Volume	# Messages
compromise	b-by-b, b-by-ℓ, ℓ-by-b	$2\frac{mnp}{b}$	$\left(\frac{mnp}{b^2\ell}\right)$
block MM	b-by-b, b-by-b, b-by-b	$2\sqrt{3}\frac{mnp}{\sqrt{M}}$	$3\sqrt{3}\frac{mnp}{M^{3/2}}$
max re-use MM	b-by-b, b-by-1, 1-by-b	$2\frac{mnp}{\sqrt{M}}$	$\sqrt{M}\left(\frac{mnp}{M^{3/2}}\right)$

block MM	$b = \sqrt{M/3}$	$\ell = \sqrt{M/3}$.
max re-use MM	$b = \sqrt{M}$	$\ell = 1$.

Parallel Distributed MM algorithms

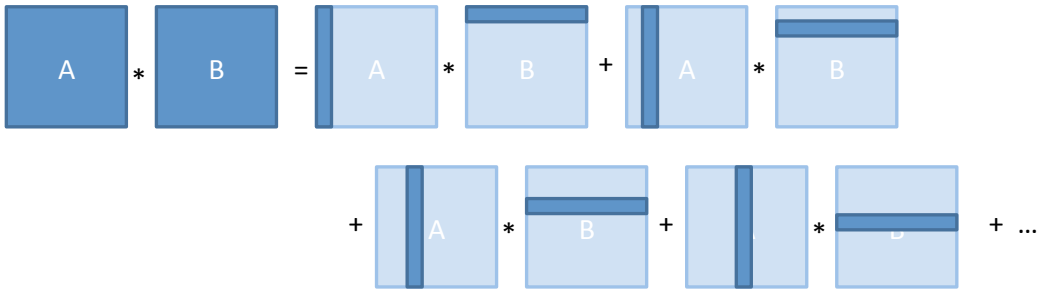


Parallel Distributed MM algorithms



- Use the outer version of the matrix-matrix multiply algorithm.

Parallel Distributed MM algorithms

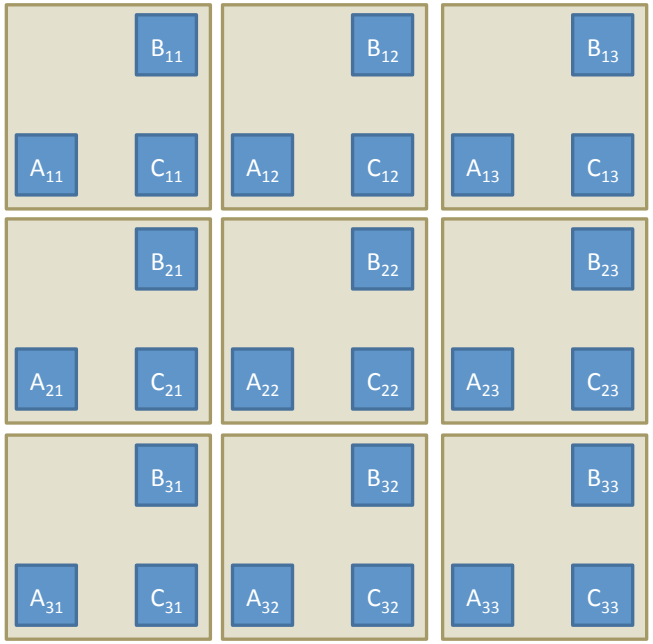


PDGEMM :

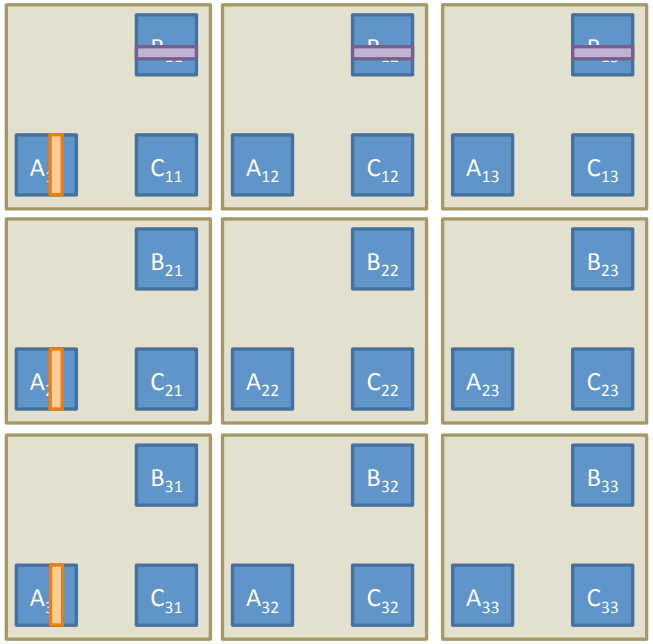
For k = 1:nb:n,

End For

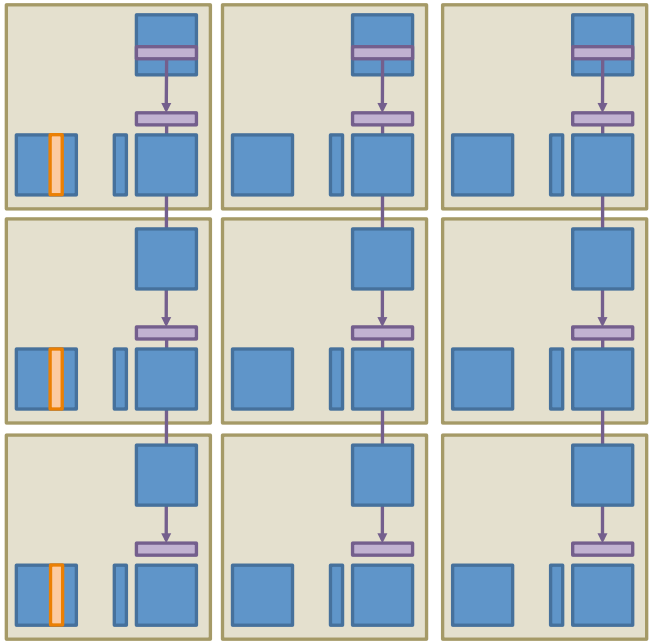
Parallel Distributed MM algorithms



Parallel Distributed MM algorithms

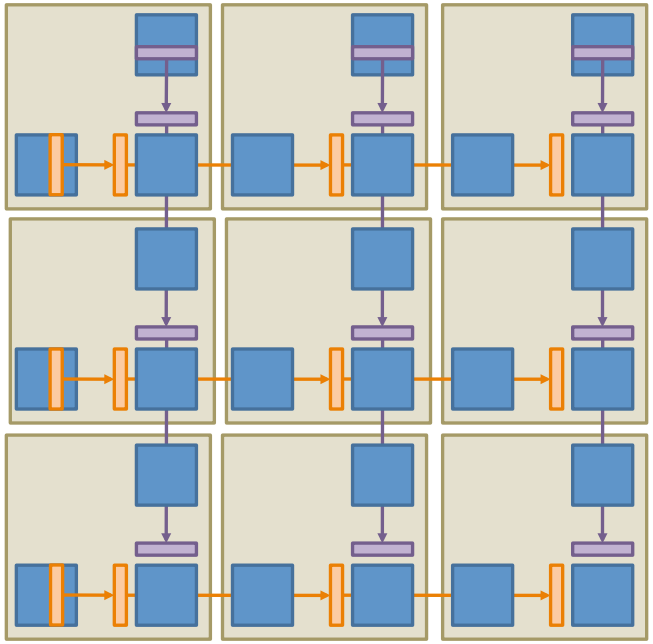


Parallel Distributed MM algorithms



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

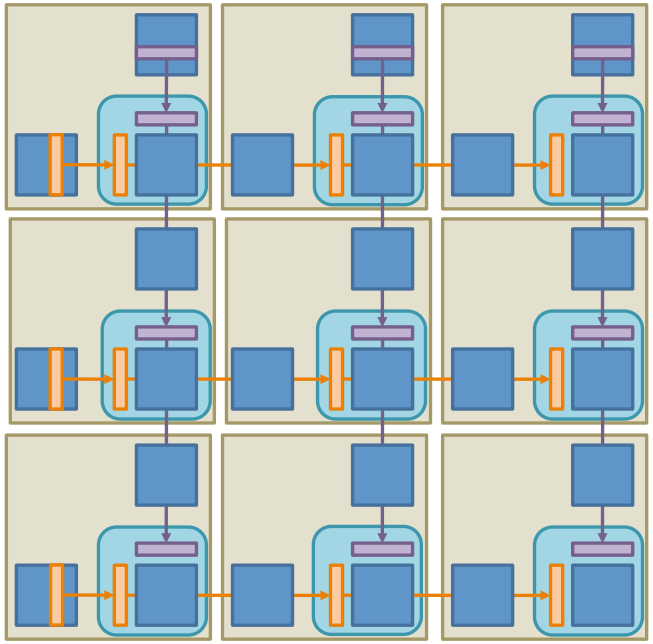
Parallel Distributed MM algorithms



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

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

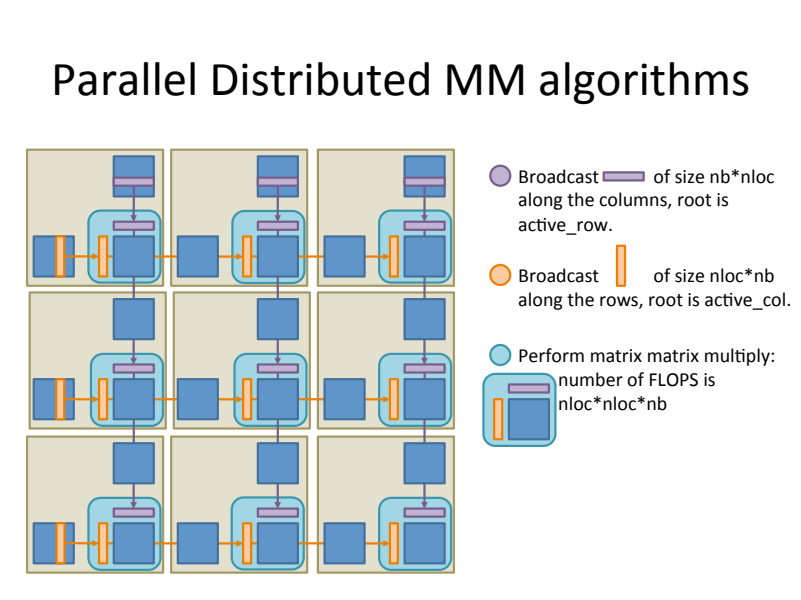
Parallel Distributed MM algorithms



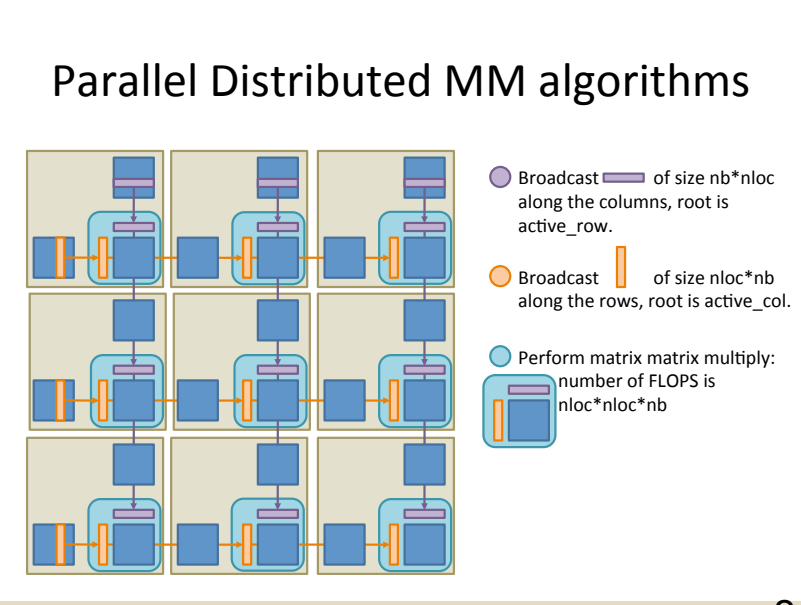
Broadcast of size nb*nloc along the columns, root is active_row.

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

Perform matrix matrix multiply: number of FLOPS is nloc*nloc*nb



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



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

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

where

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

$M_{\text{loc}} =$ em size of one proc $= n^2/P$

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.

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

- M reads and writes.

- Split the instructions into segments so exactly M reads and writes occur in each segment.

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

- M reads and writes.
  - R_a = number of reads for A.

- Split the instructions into segments so exactly M reads and writes occur in each segment.

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

- M reads and writes.
  - R_a = number of reads for A.
  - W_a = number of writes for A.

- Split the instructions into segments so exactly M reads and writes occur in each segment.

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

- M reads and writes.
  - R_a = number of reads for A.
  - W_a = number of writes for A.
  - Similar for B and C.

- Split the instructions into segments so exactly M reads and writes occur in each segment.

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

- M reads and writes.
  - R_a = number of reads for A.
  - W_a = 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.

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

- M reads and writes.
  - R_a = number of reads for A.
  - W_a = number of writes for A.
  - Similar for B and C.
- Maximize number of creates.
- Deletes are free.

- Split the instructions into segments so exactly M reads and writes occur in each segment.

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

- M reads and writes.
  - R_a = number of reads for A.
  - W_a = number of writes for A.
  - Similar for B and C.
- Maximize number of creates.
- Deletes are free.
- M_a = number of A elements in fast memory at the start.

- Split the instructions into segments so exactly M reads and writes occur in each segment.

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

- M reads and writes.
  - R_a = number of reads for A.
  - W_a = number of writes for A.
  - Similar for B and C.
- Maximize number of creates.
- Deletes are free.
- M_a = number of A elements in fast memory at the start.
- N_a = number of A elements in fast memory at the end.

$$\max(\text{Number of Scalar Multiplications}), \quad \text{subject to}$$

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

$$M_a + M_b + M_c \leq M$$

$$N_a + N_b + N_c \leq M$$

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

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

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

$$\max(\text{Number of Scalar Multiplications}), \quad \text{subject to}$$

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

$$M_a + M_b + M_c \leq M$$

$$N_a + N_b + N_c \leq M$$

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

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

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

- Constraint 1: Total number of reads and writes.

$$\max(\text{Number of Scalar Multiplications}), \quad \text{subject to}$$

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

$$M_a + M_b + M_c \leq M$$

$$N_a + N_b + N_c \leq M$$

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

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

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

- Constraint 1: Total number of reads and writes.
- Constraint 2: Total number of elements at start of segment.

$$\max(\text{Number of Scalar Multiplications}), \quad \text{subject to}$$

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

$$M_a + M_b + M_c \leq M$$

$$N_a + N_b + N_c \leq M$$

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

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

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

- 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.

$$\max(\text{Number of Scalar Multiplications}), \quad \text{subject to}$$

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

$$M_a + M_b + M_c \leq M$$

$$N_a + N_b + N_c \leq M$$

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

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

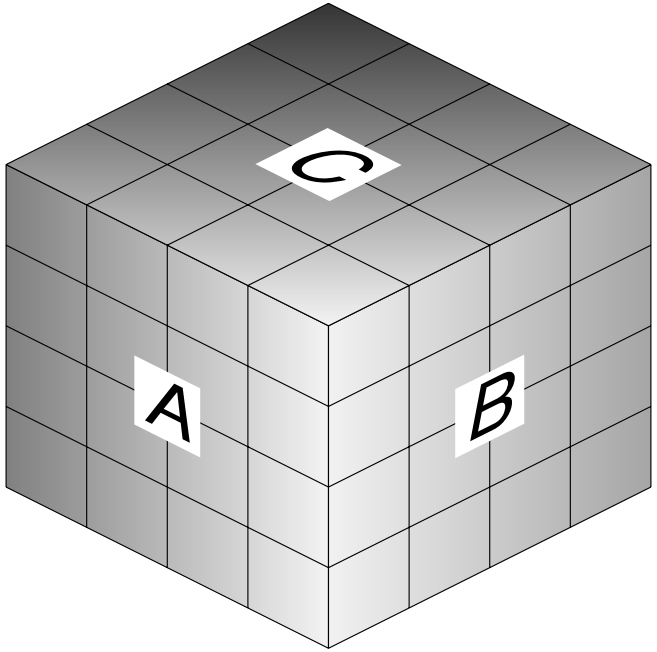
$$M_c \geq 0,\ N_c \geq 0,\ R_c \geq 0,\ W_c \geq 0$$

- 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

$$|V| \leq \sqrt{|V_x| \cdot |V_y| \cdot |V_z|}.$$



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

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

$$M_a + M_b + M_c \leq M$$

$$N_a + N_b + N_c \leq M$$

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

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

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

- Loomis-Whitney inequality.

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

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

- Loomis-Whitney inequality.
- $M_a + R_a$: Maximum number of elements of A in fast memory.

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

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

$$M_a + M_b + M_c \leq M$$

$$N_a + N_b + N_c \leq M$$

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

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

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

- Loomis-Whitney inequality.
- $M_a + R_a$: Maximum number of elements of A in fast memory.
- $M_b + R_b$: Maximum number of elements of B in fast memory.

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

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

$$M_a + M_b + M_c \leq M$$

$$N_a + N_b + N_c \leq M$$

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

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

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

- Loomis-Whitney inequality.
- $M_a + R_a$: 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,

$$\sqrt{(M_a + R_a)(M_b + R_b)(M + W_c)} \leq 2\sqrt{2}M^{3/2}.$$

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

- 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.
4. Change the optimization problem.

1. Change length of a segment.

One segment of length αM.

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

1. Change length of a segment.

One segment of length αM.

- 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

$$Volume \geq \frac{2}{3\sqrt{3}} \frac{mnp}{\sqrt{M}} - 2M.$$

- 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

$$Volume \geq 1 \frac{mnp}{\sqrt{M}} - M.$$

Solving exactly for arbitrary segment length, αM.

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

Solving exactly for arbitrary segment length, αM.

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

- 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.
4. Change the optimization problem.

4. Change the optimization problem.

See: Irony, D., Toledo, S., and Tiskin, A. (2004). 11Communication lower bounds for distributed-memory matrix multiplication." J. Parallel Distrib. Comput., 64(9):1017-1026.

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

$$R_a + R_b + W_c = M$$

$$M_a + M_b \leq M$$

$$N_c \leq M$$

4. Change the optimization problem.

See: Irony, D., Toledo, S., and Tiskin, A. (2004). 11Communication lower bounds for distributed-memory matrix multiplication." J. Parallel Distrib. Comput., 64(9):1017-1026.

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

See: Dongarra, Pineau, Robert, Shi, and Vivien. (2007). "Revisiting Matrix Product on Master-Worker Platforms." IEEE International Parallel and Distributed Processing Symposium.

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

4. Change the optimization problem.

See: Irony, D., Toledo, S., and Tiskin, A. (2004). 11Communication lower bounds for distributed-memory matrix multiplication." J. Parallel Distrib. Comput., 64(9):1017-1026.

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

See: Dongarra, Pineau, Robert, Shi, and Vivien. (2007). "Revisiting Matrix Product on Master-Worker Platforms." IEEE International Parallel and Distributed Processing Symposium.

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

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.
4. Change the optimization problem.

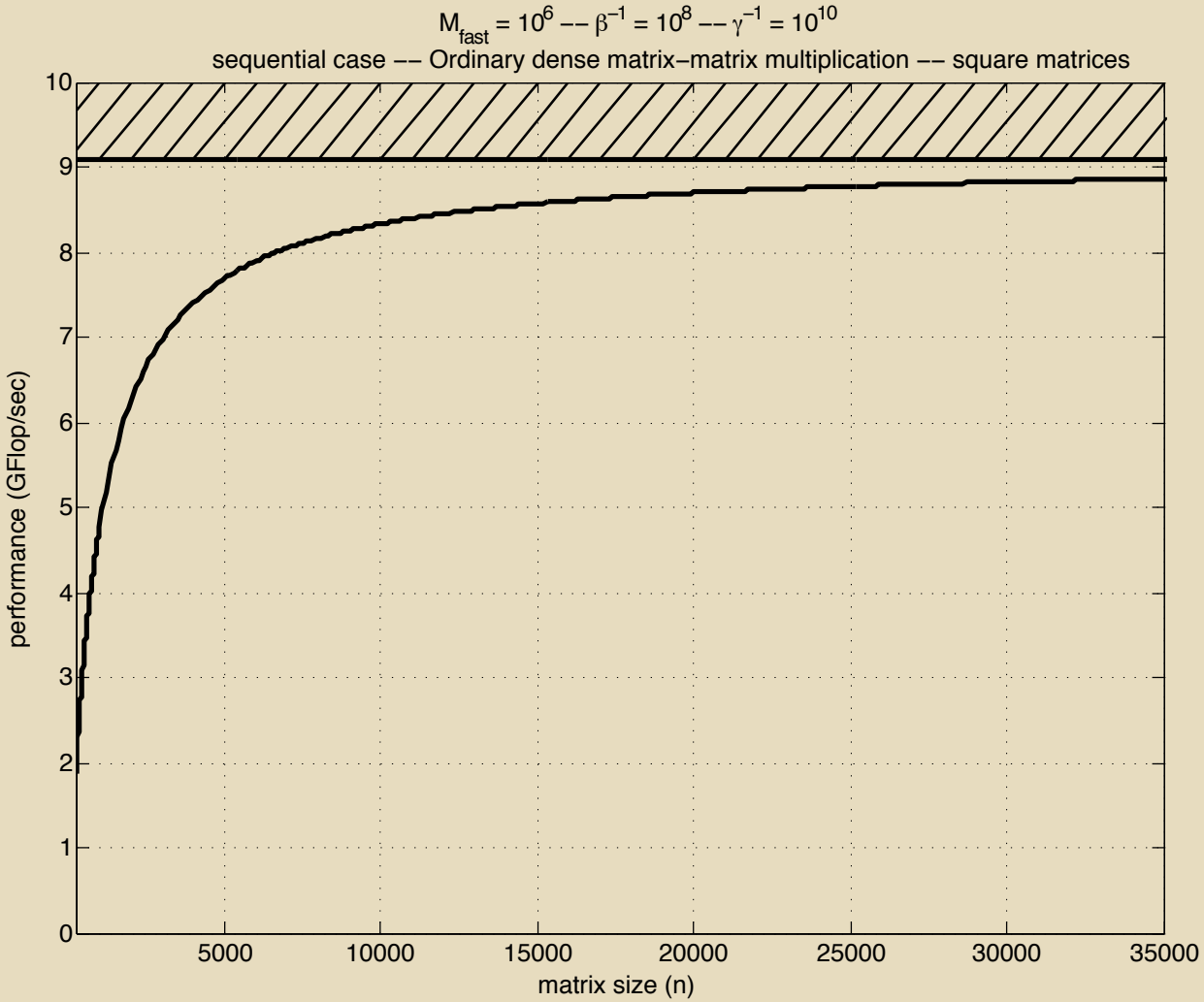
Ways to improve lower bound

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

gives
Lower bound on volume of message transferred is

$$\text{Volume} \geq 2\frac{mnp}{\sqrt{M}} - 2M.$$

$\beta^{-1} = 10^8$words/sec $\gamma^{-1} = 10^{10}$flops/sec $M = 10^6$words



What's next.

1. Work on the latency.

   lowest known upper bound: $3\sqrt{3}\frac{n^3}{M^{\frac{3}{2}}}$

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