Communication costs of sequential matrix multiplications

Suraj Kumar

Inria & ENS Lyon

Email:*suraj.kumar@inria.fr*

CR12: September 2023

https://surakuma.github.io/courses/daamtc.html

Why so much stress on matrix multiplication?

- Basic in almost all computational domains
- Everyone knows about it
    - Still there are many open research questions
- Easy to understand and explain ideas with this computation

BLAS: Basic Linear Algebra Subprograms

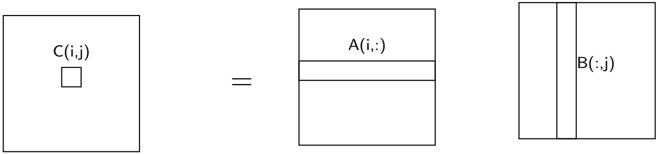
- Introduced in the 80s as a standard for LA computations
- Organized by levels:
    - Level 1: vector/vector operations ($x \cdot y$)
    - Level 2: vector/matrix (Ax)
    - Level 3: matrix/matrix (AB^T, blocked algorithms)
- Implementations:
    - Vendors (MKL from Intel, CuBLAS from NVidia, etc.)
    - Automatic Tuning: ATLAS
    - GotoBLAS

Table of Contents

Traditional matrix multiplication

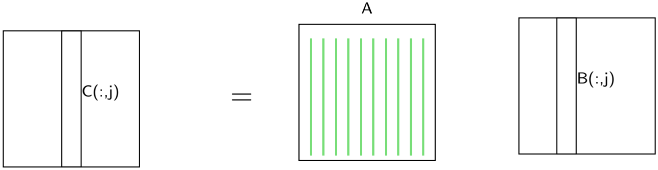
- $C = AB$, where $A \in \mathbb{R}^{m \times k}$, $B \in \mathbb{R}^{k \times n}$, and $C \in \mathbb{R}^{m \times n}$.
- $C_{ij} = \sum_{\ell} A_{i\ell} \cdot B_{\ell j}$

For simplicity, we assume $m = k = n$.



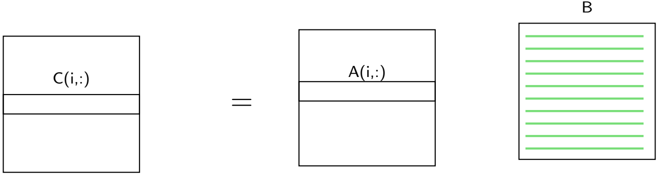
Matrix multiplication: linear combination of columns

- A column of C is obtained by linear combination of columns of A.



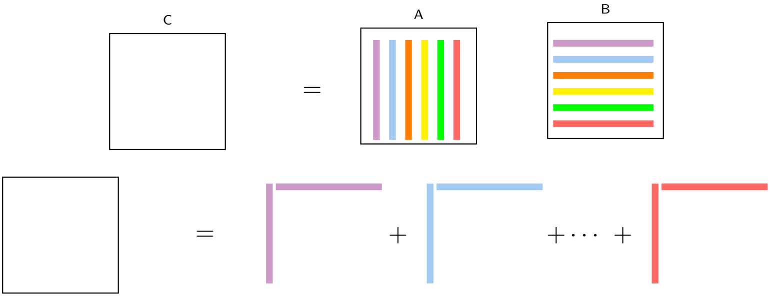
Matrix multiplication: linear combination of rows

- A row of C is obtained by linear combination of rows of B.



Matrix multiplication: sum of n matrices

- Matrix multiplication can also be viewed as sum of n matrices.



Matrix multiplication: recursive calls on submatrices

- Matrix is divided into 2×2 blocks

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\ C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\ C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\ C_{22} &= A_{21}B_{12} + A_{22}B_{22} \end{aligned}$$

Matrix multiplication: recursive calls on submatrices

Operation count recurrence,

$$T(n) = 8T\left(\frac{n}{2}\right) + \mathcal{O}(n^2)$$
$$T(n) = 1$$

Here $\mathcal{O}(n^2)$ refers that $\exists c \in \mathbb{N}$ such that this term is less than or equal to cn^2 for every n.

After solving, we obtain $T(n) = \mathcal{O}(n^3)$.

Table of Contents

Matrix multiplication: Strassen's algorithm

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$\begin{aligned} M_1 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\ M_2 &= (A_{21} + A_{22})B_{11} \\ M_3 &= A_{11}(B_{12} - B_{22}) \\ M_4 &= A_{22}(B_{21} - B_{11}) \\ M_5 &= (A_{11} + A_{12})B_{22} \\ M_6 &= (A_{21} - A_{11})(B_{11} + B_{12}) \\ M_7 &= (A_{12} - A_{22})(B_{21} + B_{22}) \end{aligned}$$

$$\begin{aligned} C_{11} &= M_1 + M_4 - M_5 + M_7 \\ C_{12} &= M_3 + M_5 \\ C_{21} &= M_2 + M_4 \\ C_{22} &= M_1 - M_2 + M_3 + M_6 \end{aligned}$$

Matrix multiplication: Strassen's algorithm

Operation count recurrence,

$$T(n) = 7T\left(\frac{n}{2}\right) + \mathcal{O}(n^2)$$
$$T(n) = 1$$

After solving, we obtain $T(n) = \mathcal{O}(n^{\log_2 7})$.
$$\log_2 7 \approx 2.81$$

Open questions

- Is there a way to perform matrix multiplication in less number of operations than this algorithm?
- What is the minimum number of operations to perform matrix multiplication?

Table of Contents

Analysis of traditional matrix multiplication algorithm

```
//implements C=C+AB
for i=1 to n
  for j=1 to n
    for k=1 to n
      C(i,j) = C(i,j) + A(i,k) * B(k,j);
```

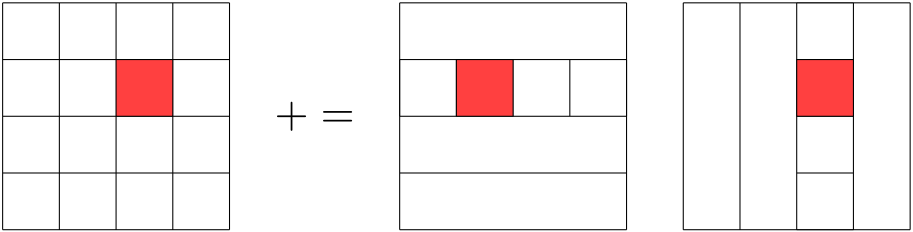
Analysis of traditional matrix multiplication algorithm

```
//implements C=C+AB
for i=1 to n
 for j=1 to n
  // read row i of C into fast memory (total n^2 reads)
  for k=1 to n
   // read row i of A into fast memory  (total n^3 reads)
    // read column j of B into fast memory (total n^3 reads)
    C(i,j) = C(i,j) + A(i,k) * B(k,j);
  // write row i of C back to slow memory (total n^2 writes)
```

$2n^3 + 2n^2$ reads/writes combined dominates $2n^3$ computations.

Tiled matrix multiplication

- A, B, C are $n/b \times n/b$ matrices of $b \times b$ subblocks
- 3 $b \times b$ blocks fit in the fast memory



Tiled matrix multiplication

```
for i=1 to n/b
  for j=1 to n/b
    // read block C(i,j) into fast memory
    // (total bˆ2 * n/b * n/b = nˆ2 reads)
    for k=1 to n/b
      // read block A(i,k) into fast memory
      // (total bˆ2 * n/b * n/b * n/b = nˆ3/b reads)
      // read block B(k,j) into fast memory
      // (total bˆ2 * n/b * n/b * n/b = nˆ3/b reads)
      //perform block matrix multiplication
      C(i,j) = C(i,j) + A(i,k) * B(k,j);
    // write block C(i,j) into fast memory
    // (total bˆ2 * n/b * n/b = nˆ2 writes)
```

$\frac{2n^3}{b} + 2n^2$ reads/writes $<<$ $2n^3$ computations.

Amount of volume in matrix multiplication

- Let M be the size of the fast memory, make b as large as possible, $3b^2 \leq M$
- Number of reads/writes $\geq 2\sqrt{3}n^3/\sqrt{M} + 2n^2$

Question : Is this optimal?

Assignment 1 – deadline Sept. 21

```
for i=1 to m
  for j=1 to n
    for k=1 to l
      C(i,j) = C(i,j) + A(i,k) * B(k,j);
```

Here $A \in \mathbb{R}^{m\times\ell}$, $B \in \mathbb{R}^{\ell\times n}$, and $C \in \mathbb{R}^{m\times n}$. The computation is performed with infinite precision.

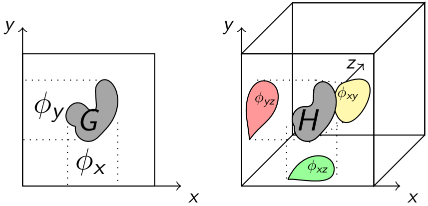
Questions:

1. Prove that all the 6 permutations of the loops produce the same output.
2. Compute the number of cache misses for each permutation of the loops. All matrices are stored in the row-major order in the slow memory. Size of each cache line is L and the cache capacity $<< \min(m, n, \ell)$. Assume that the cache is fully associative and the least recently used (LRU) strategy is employed to evict a cache line.

Table of Contents

Approach to obtain communication lower bounds

- Loomis-Whitney inequalitiy: for $d - 1$ dimensional projections
  - For the 2d object G, $Area(G) \leq \phi_x \phi_y$
  - For the 3d object H, $Volume(H) \leq \sqrt{\phi_{xy}\phi_{yz}\phi_{xz}}$



- Hölder-Brascamp-Lieb (HBL) inequality – generalization for arbitrary dimensional projections
  - Provide exponent for each projection

Number of iterations with a phases of R reads ($\neq M$)

Theorem

During a phase of R reads with memory M, the number of computed iterations is bounded by

$$F_{M+R} \leq \left(\frac{1}{3}(M+R)\right)^{3/2}$$

Maximize F_{M+R} constrained by:

$$\begin{cases} F_{M+R} \leq \sqrt{N_A N_B N_C} \\ 0 \leq N_A, N_B, N_C \\ N_A + N_B + N_C \leq M + R \end{cases}$$

Using Lagrange multipliers, maximal value obtained when $N_A = N_B = N_C$

Selection of R

```
for i=1 to m
  for j=1 to n
    for k=1 to l
      C(i,j) = C(i,j) + A(i,k) * B(k,j);
```

Total number of iterations in one phase: $F_{M+R} \leq \left(\frac{1}{3}(M+R)\right)^{3/2}$

Total volume of reads:

$$V_{\text{read}} \geq \left\lfloor \frac{mn\ell}{F_{M+R}} \right\rfloor \cdot R \geq \left(\frac{mn\ell}{F_{M+R}} - 1 \right) \cdot R$$

Valid for all values of R, maximized when $R = 2M$:

$$V_{\text{read}} \geq 2mn\ell/\sqrt{M} - 2M$$

Communication bounds

$$V_{\text{read}} \geq 2mn\ell/\sqrt{M} - 2M$$

All elements of the output matrix are in the slow memory in the end. Each element of C is written at least once: $V_{\text{write}} \geq mn$

Theorem

The total volume of I/Os is bounded by:

$$V_{I/O} \geq 2mn\ell/\sqrt{M} + mn - 2M$$

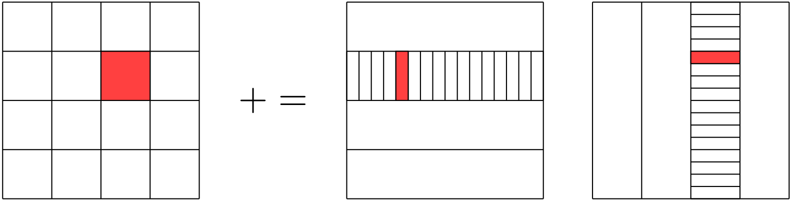
Our tiled algorithm (explained previously)

- With square matrices, total number of reads/writes $\geq 2\sqrt{3}n^3/\sqrt{M} + 2n^2$
- How far it is from the lower bound?

Structure of the optimal algorithm (attaining the same constant for the leading term)

Consider the following algorithm sketch:

- Partition C into blocks of size $(\sqrt{M}-1)\times(\sqrt{M}-1)$
- Partition A into block-columns of size $(\sqrt{M}-1)\times 1$
- Partition B into block-rows of size $1\times(\sqrt{M}-1)$
- For each block C_b of C:
  - Load the corresponding blocks of A and B on after the other
  - For each pair of blocks A_b, B_b, compute $C_b \leftarrow C_b + A_b B_b$
  - When all computations for C_b are performed, write back C_b



Another approach to computer communication bound

Red-Blue pebble game (Hong and Kung, 1981):

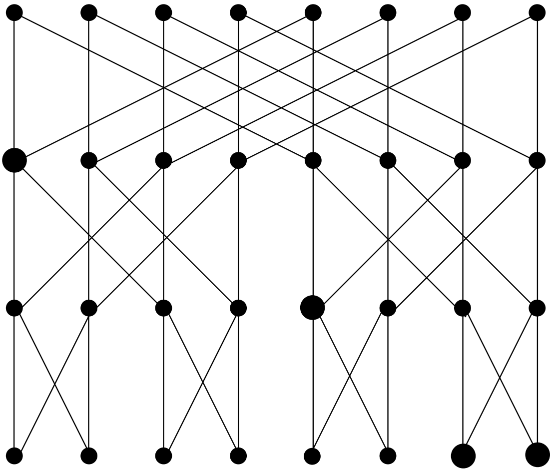
- Red pebbles: limited number S (slots in fast memory)
- Blue pebbles: unlimited number, only for slow memory

Rules:

1. A red pebble may be placed on a vertex that has a blue pebble.
2. A blue pebble may be placed on a vertex that has a red pebble.
3. If all predecessors of a vertex v have a red pebble, a red pebble may be placed on v.
4. A pebble (red or blue) may be removed at any time.
5. No more than S red pebbles may be used at any time.
6. A blue pebble can be placed on an input vertex at any time

Objective: put a red pebble on each target (not necessary simultaneously) using a minimum rules 1 and 2 (I/O operations)

Example: FFT graph



k levels, $n = 2^k$ vertices at each level

Minimum number S of red pebbles ?

How many I/Os for this minimum number S ?