Algoritmic Complexity of Matrix Operations
Matrix Computations — CPSC 5006 E

Julien Dompierre

Department of Mathematics and Computer Science
Laurentian University

Sudbury, September 27, 2010

Algorithm 1.1.1 — Dot Product (p. 5)

Algorithm 1 (Dot Product) If $x, y \in \mathbb{R}^n$, then this algorithm computes their dot product $c = x^T y$.

$c = 0$
for $i = 1 : n$
  $c = c + x(i)y(i)$
end

The dot product of two vector gives a scalar. The dot product of two n-vectors involves n multiplications and n additions. It is an $O(n)$ operation, meaning that the amount of work is linear in the dimension.

Algorithm 1.1.2 — Saxpy (p. 5)

Saxpy is the scalar multiplication of a scalar with a vector, and it results in a vector. One can think of "saxpy" as a mnemonic for "scalar a x plus y."

Algorithm 2 (Saxpy) If $x, y \in \mathbb{R}^n$ and $a \in \mathbb{R}$, then this algorithm overwrites y with $ax + y$.

for $i = 1 : n$
  $y(i) = ax(i) + y(i)$
end

The saxpy computation is also an $O(n)$ operation, but it returns a vector instead of a scalar.

Algorithm 1.1.3 — Gaxpy (Row Version) (p. 5)

A standard way to compute the matrix-vector multiplication $y = Ax + y$ is to update the components one at a time

$$y_i = \sum_{j=1}^{n} a_{ij} x_j + y_i, \quad \text{for } i = 1 : m.$$

The *generalized* saxpy operation is referred to as a *gaxpy*.

Algorithm 3 (Gaxpy: Row Version) If $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$, and $y \in \mathbb{R}^m$, then this algorithm overwrites y with $Ax + y$.

for $i = 1 : m$
  for $j = 1 : n$
    $y(i) = A(i, j)x(j) + y(i)$
  end
end

The gaxpy computation is also an $O(mn)$ operation.

Algorithm 1.1.3 Using Colon Notation

If $A \in \mathbb{R}^{m \times n}$, then $A(i,:)$ designates the ith row of A, i.e.,

$$A(i,:) = \left[\begin{array}{cccc} a_{i1} & a_{i2} & \cdots & a_{in} \end{array} \right].$$

Then the Algorithm 1.1.3 — Gaxpy (row version)

for $i = 1 : m$
  for $j = 1 : n$
    $y(i) = A(i,j)x(j) + y(i)$
  end
end

can be written as follow

for $i = 1 : m$
  $y(i) = A(i,:)x(:) + y(i)$
end

Algorithm 1.1.3 Using Row Notation

Algorithm 1.1.3 access the data in A by row. From a row point of view, a matrix is a stack of row vectors:

$$A \in \mathbb{R}^{m \times n} \quad \Longleftrightarrow \quad A = \begin{bmatrix} r_1^T \\ \vdots \\ r_m^T \end{bmatrix}, \text{where } r_i \in \mathbb{R}^n.$$

Then the Algorithm 1.1.3 — Gaxpy (row version)

for $i = 1 : m$
  $y(i) = A(i,:)x(:) + y(i)$
end

can be written as follow

for $i = 1 : m$
  $y_i = r_i^T x + y_i$
end

The inner loop is a scalar product of the row i with the vector x.

Algorithm 1.1.4 — Gaxpy (Column Version) (p. 6)

If we regard the matrix-vector multiplication Ax as a linear combination of A's columns, then we get the column version of gaxpy:

Algorithm 4 (Gaxpy: Column Version) If $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$, and $y \in \mathbb{R}^m$, then this algorithm overwrites y with $Ax + y$.

for $j = 1 : n$
  for $i = 1 : m$
    $y(i) = A(i,j)x(j) + y(i)$
  end
end

Algorithm 1.1.4 Using the Colon Notation

If $A \in \mathbb{R}^{m \times n}$, then $A(:,j)$ designates the jth column of A, i.e.,

$$A(j,:) = \begin{bmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{mj} \end{bmatrix}.$$

Then the Algorithm 1.1.4 — Gaxpy (column version)

for $j = 1 : n$
  for $i = 1 : m$
    $y(i) = A(i,j)x(j) + y(i)$
  end
end

can be written as follow

for $j = 1 : n$
  $y = x(j)A(:,j) + y$
end

Algorithm 1.1.4 Using the Column Notation

Algorithm 1.1.4 access the data in A by column. From a column point of view, a matrix is a collection of column vectors:

$$A \in \mathbb{R}^{m\times n} \iff A = \begin{bmatrix} c_1 & \cdots & c_n \end{bmatrix}, \text{where } c_j \in \mathbb{R}^m.$$

Then the Algorithm 1.1.4 — Gaxpy (column version)

for $j = 1 : n$
  $y = x(j)A(:, j) + y$
end

can be written as follow

for $j = 1 : n$
  $y = x_j c_j + y$
end

The inner loop is a saxpy, the scalar multiplication of the scalar x_j with the vector c_j.

Matrix-Vector Multiplication. Loop Ordering and Properties

Loop Order	Inner Loop	Inner Loop Data Access
ij	dot	A by row, x by column, y constant
ji	saxpy	A by column, x constant, y by column

Algorithm 1.1.5 Matrix Multiplication — *ijk* Variant (p. 9)

Consider the following matrix multiplication update:

$$C = AB + C, \quad A \in \mathbb{R}^{m\times p}, B \in \mathbb{R}^{p\times n}, C \in \mathbb{R}^{m\times n}.$$

The *ijk* variant is the standard familiar triply-nested loop algorithm:

Algorithm 5 (Matrix Multiplication: *ijk* Variant) If $A \in \mathbb{R}^{m\times p}$, $B \in \mathbb{R}^{p\times n}$, and $C \in \mathbb{R}^{m\times n}$ are given, then this algorithm overwrites C with $AB + C$.

for $i = 1 : m$
  for $j = 1 : n$
    for $k = 1 : p$
      $C(i, j) = A(i, k)B(k, j) + C(i, j)$
    end
  end
end

This algorithm is $O(mnp)$.

Algorithm 1.1.5 Matrix Multiplication Using Colon Notation (p. 11)

If $A \in \mathbb{R}^{m\times p}$, then $A(i, :)$ designates the ith row of A, i.e.,

$$A(i, :) = \begin{bmatrix} a_{i1} & a_{i2} & \cdots & a_{ip} \end{bmatrix}.$$

If $B \in \mathbb{R}^{p\times n}$, then $B(:, j)$ designates the jth column of B, i.e.,

$$B(:, j) = \begin{bmatrix} b_{1j} \\ \vdots \\ b_{pj} \end{bmatrix}.$$

Then the Algorithm 1.1.5 can be written as follow

for $i = 1 : m$
  for $j = 1 : n$
    $C(i, j) = A(i, :)B(:, j) + C(i, j)$
  end
end

Algorithm 1.1.5 Matrix Multiplication Using Row and Column Notation (p. 11)

In the language of partitioned matrices, if

$$A = \begin{bmatrix} a_1^T \\ a_2^T \\ \vdots \\ a_m^T \end{bmatrix} \text{ and } B = \begin{bmatrix} b_1 & b_2 & \cdots & b_n \end{bmatrix},$$

with $a_i \in \mathbb{R}^p$ and $b_j \in \mathbb{R}^p$, then the Algorithm 1.1.5 can be written as follow

for $i = 1 : m$
  for $j = 1 : n$
    $c_{ij} = a_i^T b_j + c_{ij}$
  end
end

The inner loop is the scalar product of row i of A with the column j of B.

Algorithm 1.1.7 Matrix Multiplication — Saxpy Version (jki Variant) (p. 12)

If we regard the matrix-matrix multiplication AB as a linear combination of A's columns, then we get the column version of gaxpy: the jki variant:

Algorithm 6 (Matrix Multiplication: Saxpy Version (jki Variant)) If $A \in \mathbb{R}^{m\times p}$, $B \in \mathbb{R}^{p\times n}$, and $C \in \mathbb{R}^{m\times n}$ are given, then this algorithm overwrites C with $AB + C$.

for $j = 1 : n$
  for $k = 1 : p$
    for $i = 1 : m$
      $C(i,j) = A(i,k)B(k,j) + C(i,j)$
    end
  end
end

Algorithm 1.1.7 Matrix Multiplication Using Colon Notation (p. 12)

If $A \in \mathbb{R}^{m\times p}$, then $A(:,k)$ designates the kth column of A, i.e.,

$$A(:,k) = \begin{bmatrix} a_{1k} \\ \vdots \\ a_{mk} \end{bmatrix}.$$

If $C \in \mathbb{R}^{m\times n}$, then $C(:,j)$ designates the jth column of C, i.e.,

$$C(:,j) = \begin{bmatrix} a_{1j} \\ \vdots \\ c_{mj} \end{bmatrix}.$$

Then the Algorithm 1.1.7 can be written as follow

for $j = 1 : n$
  for $k = 1 : p$
    $C(:,j) = A(:,k)B(k,j) + C(:,j)$
  end
end

Algorithm 1.1.7 Matrix Multiplication Using Row and Column Notation (p. 12)

In the language of partitioned matrices, if

$$A = \begin{bmatrix} a_1 & a_2 & \cdots & a_p \end{bmatrix} \text{ and } C = \begin{bmatrix} c_1 & c_2 & \cdots & c_n \end{bmatrix},$$

with $a_k \in \mathbb{R}^m$ and $c_j \in \mathbb{R}^m$, then the Algorithm 1.1.7 can be written as follow

for $j = 1 : n$
  for $k = 1 : p$
    $c_j = b_{kj} a_k + c_j$
  end
end

The inner loop is a saxpy, the scalar multiplication of the scalar b_{kj} with the vector a_k. Also, the middle loop over k is a column gaxpy, the multiplication of the matrix A times the column j of B.

The Outer Product (p. 8)

Let $x \in \mathbb{R}^m$ and $y \in \mathbb{R}^n$, Then the **outer product** of x and y, denoted by $x \otimes y$, is given by

$$x \otimes y = xy^T = C,$$

where $C \in \mathbb{R}^{m \times n}$ with $c_{ij} = x_i y_j$.

For example,

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \begin{bmatrix} 4 & 5 \end{bmatrix} = \begin{bmatrix} 4 & 5 \\ 8 & 10 \\ 12 & 15 \end{bmatrix}.$$

Algorithm 1.1.8 Matrix Multiplication — Outer Product Version (kji Variant) (p. 13)

If we regard the matrix-matrix multiplication AB as a sum of outer products of A's columns times B's rows, then we get the outer product version: the kji variant:

Algorithm 7 (Matrix Multiplication: Outer Product Version (*kji* Variant)) If $A \in \mathbb{R}^{m \times p}$, $B \in \mathbb{R}^{p \times n}$, and $C \in \mathbb{R}^{m \times n}$ are given, then this algorithm overwrites C with $AB + C$.

for $k = 1 : p$
  for $j = 1 : n$
    for $i = 1 : m$
      $C(i,j) = A(i,k)B(k,j) + C(i,j)$
    end
  end
end

Algorithm 1.1.8 Matrix Multiplication Using Colon Notation (p. 13)

If $A \in \mathbb{R}^{m \times p}$, then $A(:,k)$ designates the kth column of A, i.e.,

$$A(:,k) = \begin{bmatrix} a_{1k} \\ \vdots \\ a_{mk} \end{bmatrix}.$$

If $B \in \mathbb{R}^{p \times n}$, then $B(k,:)$ designates the kth row of B, i.e.,

$$B(k,:) = \begin{bmatrix} b_{k1} & b_{k2} & \cdots & b_{kn} \end{bmatrix}.$$

Then the Algorithm 1.1.8 can be written as follow

for $k = 1 : p$
  $C = A(:,k)B(k,:) + C$
end

The inner loop is the outer product of the kth column of A by the kth row of B.

Algorithm 1.1.8 Matrix Multiplication Using Row and Column Notation (p. 13)

In the language of partitioned matrices, if

$$A = \begin{bmatrix} a_1 & a_2 & \cdots & a_p \end{bmatrix} \text{ and } B = \begin{bmatrix} b_1^T \\ \vdots \\ b_p^T \end{bmatrix},$$

with $a_k \in \mathbb{R}^m$ and $b_k \in \mathbb{R}^n$, then the Algorithm 1.1.8 can be written as follow

for $k = 1 : p$
  $C = a_k b_k^T + C$
end

Matrix Multiplication: Loop Ordering and Properties (p. 10)

Loop Order	Inner Loop	Middle Loop	Inner Loop Data Access
ijk	dot	vector × matrix	A by row, B by column C constant
jik	dot	matrix × vector	A by row, B by column C constant
ikj	saxpy	row gaxpy	B by row, C by row A constant
jki	saxpy	column gaxpy	A by column, C by column B constant
kij	saxpy	row outer product	B by row, C by row A constant
kji	saxpy	column outer product	A by column, C by column B constant

Memory Access Time

Time for memory access in function of memory size.

	CPU	cache	SIMM	hard drive	mass storage
Speed	$++$	$+$	0	$-$	$--$
Size	$--$	$-$	0	$+$	$++$
Cost	$++$	$+$	0	$-$	$--$

Talk also here about row and column storage, pipelining arithmetic operations, multiplication/addition processor operations and non square matrices.

Block Matrix Multiplication

Suppose that $A \in \mathbb{R}^{m \times n}$, that $0 < p < m$ and $0 < q < n$. Then the matrix A can be divided into four blocks:

$$A_{m\times n} = \left[\begin{array}{c|c} A_{p\times q} & A_{p\times n-q} \\ \hline A_{m-p\times q} & A_{m-p\times n-q} \end{array}\right]$$

The block matrix multiplication by a vector can be expressed as

$$\begin{aligned} & \left[\begin{array}{c|c} A_{p\times q} & A_{p\times n-q} \\ \hline A_{m-p\times q} & A_{m-p\times n-q} \end{array}\right] \left[\begin{array}{c} x_{q\times 1} \\ \hline x_{n-q\times 1} \end{array}\right] \\ = & \left[\begin{array}{c} A_{p\times q}x_{q\times 1} + A_{p\times n-q}x_{n-q\times 1} \\ \hline A_{m-p\times q}x_{q\times 1} + A_{m-p\times n-q}x_{n-q\times 1} \end{array}\right] \end{aligned}$$

A Divide and Conquer Matrix Multiplication (p. 31)

The starting point in the discussion is the 2-by-2 block matrix multiplication, where each block is square:

$$\begin{aligned} \left[\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array}\right] &= \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right] \left[\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right] \\ &= \left[\begin{array}{c|c} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ \hline A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{array}\right] \end{aligned}$$

There are 8 matrix multiplications and 4 matrix additions. As the submatrices are half-size, the cost is $(n/2)^3 = n^3/8$.

Stassen Multiplication Algorithm 1.3.1 (p. 32)

Volker Strassen (1969) has shown how to compute C with just 7 multiplies and 18 adds:

$$
\begin{aligned}
P_1 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\
P_2 &= (A_{21} + A_{22})B_{11} \\
P_3 &= A_{11}(B_{12} - B_{22}) \\
P_4 &= A_{22}(B_{21} - B_{11}) \\
P_5 &= (A_{11} + A_{12})B_{22} \\
P_6 &= (A_{21} - A_{11})(B_{11} + B_{12}) \\
P_7 &= (A_{12} - A_{22})(B_{21} + B_{22}) \\
C_{11} &= P_1 + P_4 - P_5 + P_7 \\
C_{12} &= P_3 + P_5 \\
C_{21} &= P_2 + P_4 \\
C_{22} &= P_1 + P_3 - P_2 + P_6
\end{aligned}
$$

These equations are easily confirmed by substitution.