

# Optimizing Linear Algebra Performance for Programmers

Elliot Fiske

April 5, 2016

## 1 Multiplying Many Matrices and One Vector

Consider the following matrix operation:

$$Result = \underbrace{\mathbf{A} * \mathbf{B} * \mathbf{C} * \mathbf{D} * \mathbf{E}}_{100 \text{ by } 100} * \underbrace{\vec{x}}_{100 \text{ by } 1} \quad (1)$$

If this operation was done programmatically, with a compiler that evaluated each operation left-to-right, there would be over 4 million multiplication operations:

$$\underbrace{100 * 100 * 100 * 4}_{\substack{4 \text{ instances of multiply-} \\ \text{ing a } 100 \times 100 \text{ matrix} \\ \text{by a } 100 \times 100 \text{ matrix}}} + \underbrace{100 * 100}_{\substack{1 \text{ instance of} \\ \text{multiplying a} \\ 100 \times 100 \text{ ma-} \\ \text{trix by a } 100 \times 1 \\ \text{vector}}} = 4,010,000 \quad (2)$$

However, if we instead first evaluate  $\mathbf{E} * \vec{x}$ , then evaluate  $\mathbf{D} * (\mathbf{E} * \vec{x})$ , etc. there will be just over 1 million multiplication operations:

$$\underbrace{100 * 100 * 4}_{\substack{4 \text{ instances of multiply-} \\ \text{ing a } 100 \times 100 \text{ matrix} \\ \text{by a } 100 \times 1 \text{ vector}}} + \underbrace{100 * 100 * 100}_{\substack{1 \text{ instance of} \\ \text{multiplying a} \\ 100 \times 100 \text{ matrix} \\ \text{by a } 100 \times 100 \\ \text{matrix}}} = 1,040,000 \quad (3)$$

I used Eigen and C++ to practically test this, timing how long it took to do the approach from Equation 2 vs. the approach from Equation 3 with various matrix sizes.

Matrix Dimension	Slow Method Time ( $\mu$ s)	Fast Method Time ( $\mu$ s)
10	162	52
50	13,291	738
100	32,515	2561
300	537,397	17,399
1000	19,730,814	173,000

Table 1: Timing the dumb way vs. the smart way

For each dimension, the 'smart' method is orders of magnitude faster, becoming better as the dimension increases.

## 2 Calculating the Inverse of a Matrix

Consider the following matrix operation:

$$Result = \vec{x}^T * \mathbf{A}^T * \mathbf{B}^{-1} * \mathbf{A} * \vec{x} \quad (4)$$

One approach is to simply calculate the inverse of  $\mathbf{B}$  then multiply it with the rest of the terms. However, calculating the inverse of a matrix is a very expensive operation. Instead, consider rearranging the equation like so:

$$\begin{aligned} \mathbf{B}b &= \mathbf{A} * \vec{x} \\ Result &= \vec{x}^T * \mathbf{A}^T * b \end{aligned} \quad (5)$$

It is a much cheaper operation to solve a  $\mathbf{A}\vec{x} = \mathbf{B}$  matrix problem than it is to find an inverse of a matrix. I used Eigen and C++ to practically test this, timing how long it takes to do the approach from Equation 4 vs. the approach from Equation 5.

Below a 50x50 matrix, the two methods are fairly equal in running time. However, at dimension 100 and above the solve() method is clearly superior.

Matrix Dimension	<i>inverse()</i> Time ( $\mu$ s)	<i>solve()</i> Time ( $\mu$ s)
10	61	70
50	278	267
100	670	935
300	9599	17,104
1000	320,196	244,522

Table 2: Using *inverse()* vs. using *solve()*