James Uttaro
C.K Obieyisi
December 12, 2015
CSC 22000 Algorithms
Rosario Gennaro

Matrix Multiplication Programming Assignment

Objective: To write a program to test and improve time analysis performance of generic matrix by vector multiplication $O(N^2)$ against matrix by vector multiplication using FFT O(NlogN).

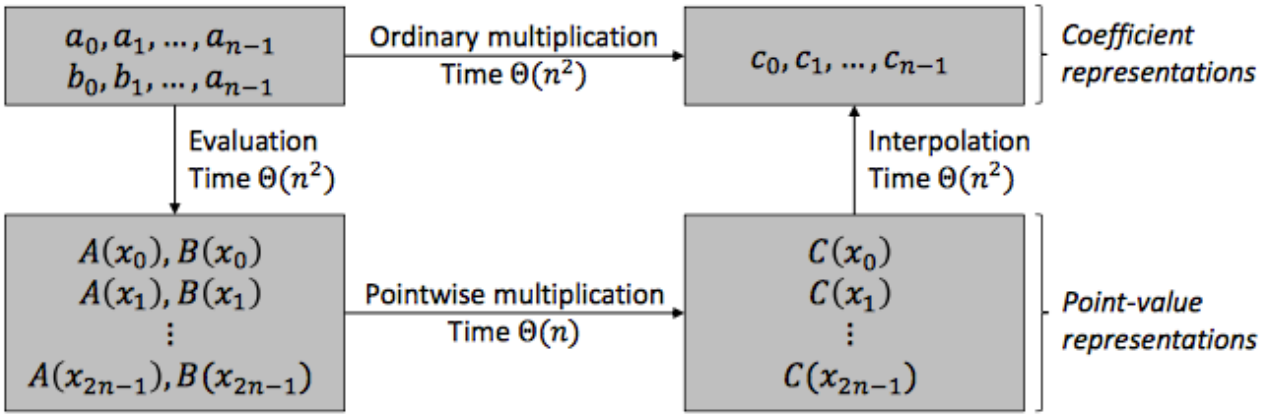
The Comparison of the two different matrix by vector multiplication algorithms is done using the Toeplitz Matrix. As the Toeplitz matrix is a diagonalized NxN matrix with the property that each descending diagonal from left to right is constant.

$$A_{i,j} = A_{i+1,j+1} = a_{i-j}.$$

$$A = \begin{bmatrix} a_0 & a_{-1} & a_{-2} & \dots & \dots & a_{-(n-1)} \\ a_1 & a_0 & a_{-1} & \ddots & & \vdots \\ a_2 & a_1 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & a_{-1} & a_{-2} \\ \vdots & & \ddots & a_1 & a_0 & a_{-1} \\ a_{n-1} & \dots & \dots & a_2 & a_1 & a_0 \end{bmatrix}$$

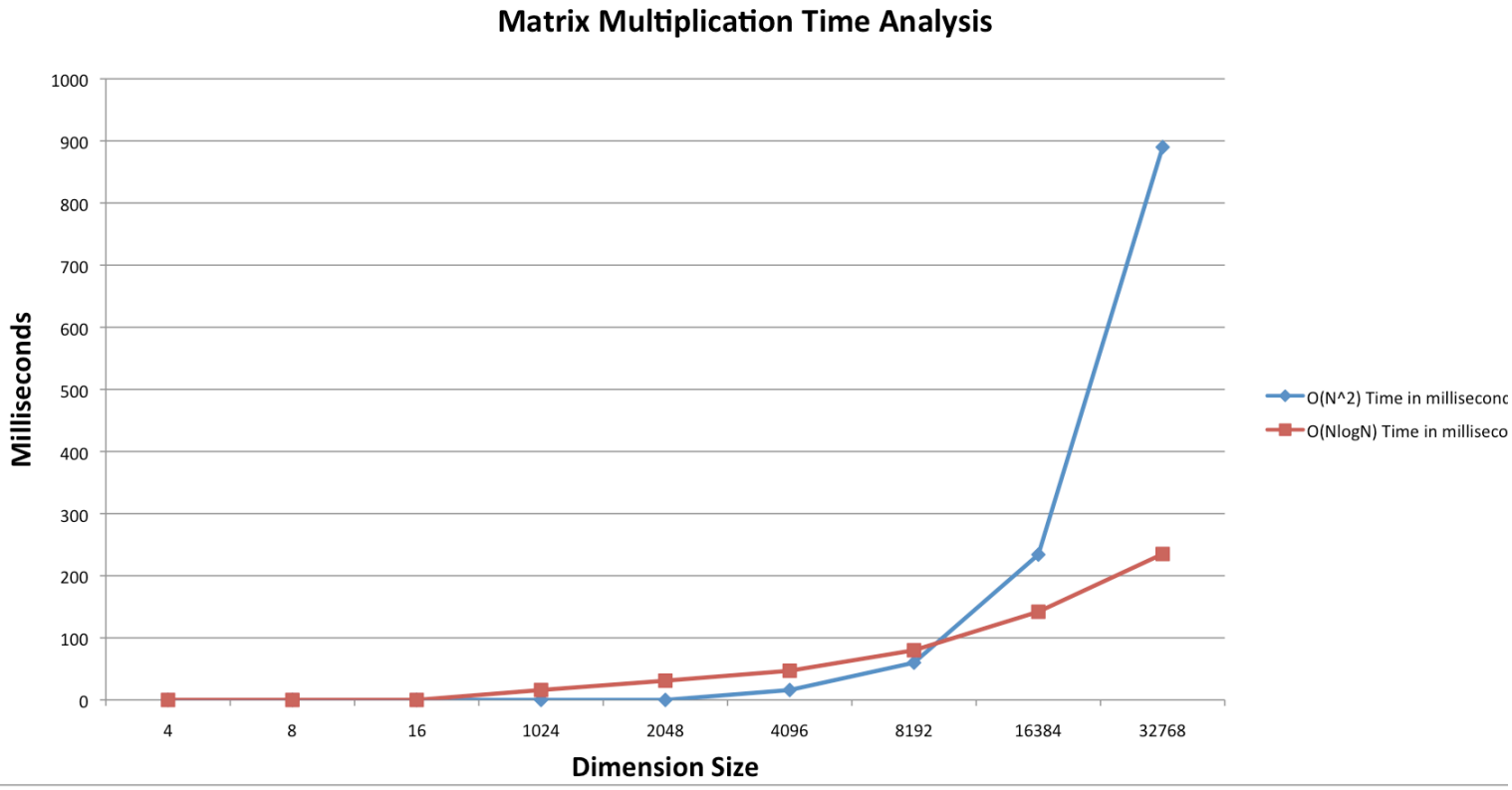
Toeplitz Matrix A (NxN) multiplied by a Vector B size N we can compute the this as a multiplication of Two Polynomials A(x) of size 2N-1 padded with one zero and B(x) as a vector of all 1's (1,1...,1,0…0) with N 1's and the rest of the vector padded with 0's . When we multiply these two Polynomials we can look at the resulting polynomials coefficients at specific degrees within the polynomial C(x). The coefficients we will need to look at with in C(x) are the coefficients of degrees ((N/2)-1) to (N-1). In Example of Dimension 4 , Look at C(3 to 7).

Below we can see the Process in which the FFT will follow to achieve a faster matrix by vector multiplication, allowing the algorithm to run in O(NlogN) time instead of ordinary multiplications $O(N^2)$.



The Implementation of this assignment was done in Java, The Fast Fourier Transform that was used is from
http://introcs.cs.princeton.edu/java/97data/FFT.java.html
An off the shelf FFT which was used in Combination with a Complex class and a Toeplitz Class which was created for this project. Within the Toeplitz class we implemented Ordinary Matrix Multiplication, Toeplitz Matrix Initialization function, and a couple of other functions that were needed for functionality. To run this code passed a Dimension of 8192, VM argument -Xmx4096m or larger will need to be passed to increase the heap size large enough to perform the FFT Convolution.

Below is the Time Analysis Comparison Of $O(N^2)$ versus O(NlogN) Matrix by Vector Multiplication



Below are two outputs of dimensions 4 and 8 confirming the outputs correctness.

```
<terminated> Test [Java Application] C:\Program Files\Java\jre1.8.0_66\bin\javaw.exe (Dec 12, 2015, 1:56:25 PM)

Toeplitz Matrix
14151617
13141516
12131415
11121314

Resultant Vector Standard multiplication
{ 62 58 54 50 }

Ordinary Multiplication O(N^2) Run Time: 0 milliseconds

FFT Function Time Cost: 0 milliseconds
Point multiplication Time Cost:  0 milliseconds
FFT O(NLogN) Run Time: 0 milliseconds
Result
-------------------
50

54

58

62
```

```
<terminated> Test [Java Application] C:\Program Files\Java\jre1.8.0_66\bin\javaw.exe (Dec 12, 2015, 1:59:09 PM)

Toeplitz Matrix
1819202122232425
1718192021222324
1617181920212223
1516171819202122
1415161718192021
1314151617181920
1213141516171819
1112131415161718

Resultant Vector Standard multiplication
{ 172 164 156 148 140 132 124 116 }

Ordinary Multiplication O(N^2) Run Time: 0 milliseconds

FFT Function Time Cost: 0 milliseconds
Point multiplication Time Cost:  0 milliseconds
FFT O(NLogN) Run Time: 0 milliseconds
Result
--------------------
116

124

132

140

148

156

164

172
```

We Can see that using the FFT Method for Matrix Multiplication has a High overhead and does not run better than $O(N^2)$ until an Matrix Dimension of around 9000 x 9000. The limiting factor for us to create more data points going passed 32768 Dimension size was not being able to allocate a large enough heap space, But we can see clear difference in performance at the test with dimension size 32768. The major downside to this Algorithm is the cost of space that it requires in comparison to O(N^2).

4096

```
<terminated> Test [Java Application] C:\Program Files\Java\jre1.8.0_66\bin\javaw.exe (Dec 12, 2015, 2:14:56 PM)

Ordinary Multiplication O(N^2) Run Time: 16 milliseconds

FFT Function Time Cost: 31 milliseconds
Point multiplication Time Cost:  0 milliseconds
FFT O(NLogN) Run Time: 47 milliseconds
```

8192

```
<terminated> Test [Java Application] C:\Program Files\Java\jre1.8.0_66\bin\javaw.exe (Dec 12, 2015, 2:04:02 PM)

Ordinary Multiplication O(N^2) Run Time: 60 milliseconds

FFT Function Time Cost: 60 milliseconds
Point multiplication Time Cost:  0 milliseconds
FFT O(NLogN) Run Time: 80 milliseconds
```

16384

```
<terminated> Test [Java Application] C:\Program Files\Java\jre1.8.0_66\bin\javaw.exe (Dec 12, 2015, 2:05:37 PM)

Ordinary Multiplication O(N^2) Run Time: 234 milliseconds

FFT Function Time Cost: 95 milliseconds
Point multiplication Time Cost:  0 milliseconds
FFT O(NLogN) Run Time: 142 milliseconds
```

32768

```
<terminated> Test [Java Application] C:\Program Files\Java\jre1.8.0_66\bin\javaw.exe (Dec 12, 2015, 2:06:34 PM)

Ordinary Multiplication O(N^2) Run Time: 890 milliseconds

FFT Function Time Cost: 157 milliseconds
Point multiplication Time Cost:  0 milliseconds
FFT O(NLogN) Run Time: 235 milliseconds
```