Threaded Implementation of Berkowitz Algorithm

Khaled Zaky, Jamie Counsell, Kevin Witlox
& Aaron Chaisson

April 11, 2013

Abstract

In this assignment, we are going to implement Berkowitz's parallel algorithm for computing the characteristic polynomial of a matrix. Berkowitz's algorithm, is the fastest known parallel algorithm for computing the characteristic polynomial of a matrix. For the sake of simplicity we assume that the matrices are over {0,1}, i.e., the two field elements, where plus is XOR and multiplication is AND. The algorithm was designed such that the process takes no more than $\log^2 n$ *sequentially many steps.* The program was then rigorously tested to ensure its accuracy.

Contents

1 Data Types & Storage

We have chosen to store the sets of matrices in lists; One list to store $MMatrices$ and one list to store $CMatrices$. We have also created an ADT $mElement()$ that stores the values of the Matrix, the matrix's lock, and its event (the event is set when the matrix is calculated). Same ADT was created for the $cElement()$ with the same storage concept.

In order to compute all M^i matrices, and all C_j matrices in $\log n$ time, peers must have access to previous powers of M, and previous values of C. This ensures that each peer can calculate the M or C that it is responsible for as soon as the required M and C are available. This is why we chose to lock each element of the list, instead of locking the list in its entirety. This lock allows other list elements to be accessed in parallel, and the event ensures that:

i No element of the list is calculated more than once.

ii Threads can wait on the necessary data to be available (computed elsewhere) by using $mElement.done.wait()$

2 Step Complexity

The overall step complexity of our program is running at $\log^2 n$ steps. The computation runs on different phases, where the program computes all the powers of all the sub-matrices. Then it computes all C_i, and in the final phase it computes their product.

2.1 Computing the Powers of the Matrix M

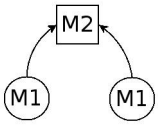
Our implementation works using two types of methods; one for computing powers of a matrix, and one for computing iterated matrix products. That is, one method for computing M_i for any M and any i and one for computing $M_1 \ M_2 \ ... \ M_k$, for compatible matrices. On input M, the output of the powering method is a list of matrices of the form M_1,M_2,...,M_n, where M is $n \times n$, and on input M_1, M_2, ... ,M_k.

We use the following threaded method to compare all M_i^j $(1 \leq i \leq k, 1 \leq j \leq m)$, where $k = \frac{N}{2}$ and m is the size of M_i:

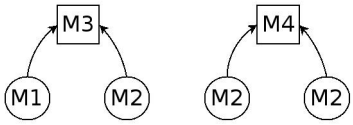
> The main thread launches k peers (called kPeers), and passes them a value i.
>
> Each thread then computes $M_i^2 ... M_i^{N-m}$, where m is the size of M_i.
>
> This process executes in $\log n$ steps. This is because we are utilizing the power of $\log n$ threads, and each thread cycles through a variable i, and attempts to locate an M^i that has not been calculated. When it locates this, it locks the data and calculates the corresponding M^i. Each M^i requires two other M's to calculate. These M's are at M^j and M^k where:
>
> - If i is even, $j = k = \frac{i}{2}$
> - If i is odd, $j = \frac{i+1}{2}$, and $k = \frac{i-1}{2}$
>
> The result of $M^k \times M^j$ is M^i; this result is then stored in a list to be used later on. A small step-by-step analysis, and $\log n$ pattern, can be seen in Figure 1.

Initially:

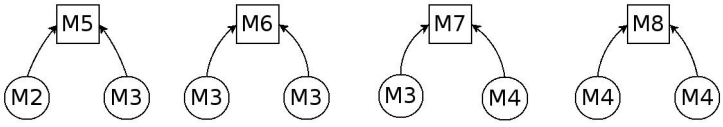
Step 1:



Step 2:



Step 3:



Step 4:

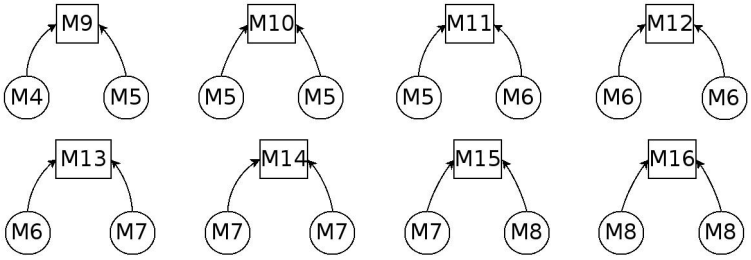


Figure 1: M powers Computed in $\log n$ Steps

2.2 Computing the Matrices for C

Each C_i has a unique list of M_i which was calculated in the previous step. The Thread now uses those M values along with and R and S computed independently of other threads (directly from A) to fill in the values of the first column of the matrix C_i. Then there is a short algorithm to copy the first column into all other columns with a column-wise shift of 1.

$$C = \begin{bmatrix} 1 & \cdots & \cdots & \cdots \\ 0 & \cdots & \cdots & \cdots \\ 1 & \cdots & \cdots & \cdots \\ 0 & \cdots & \cdots & \cdots \\ 0 & \cdots & \cdots & \cdots \\ 0 & \cdots & \cdots & \cdots \end{bmatrix} \qquad \Rightarrow \qquad C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

2.3 Computing the product of the C Matricies

We organize the C Matrices by making an $n+1 \times n+1$ matrix of zeros (where n is the number of C's) and putting all C Matrices objects above the principle axis. We will call this matrix A.We square the matrix n times using a modified algorithm of the one used for computing the powers of M Matrices. Upon squaring the Matrix B a total of n times, the matrix object at $B[0][n+1]$ will be our resultant product vector.

3 Results & Testing

For testing purposes we used two methods to verify our program and algorithm. First we used our test Matrix A, and manually evaluated all our M Matrices and C Matrices to check them against our program output. Later we utilized the algorithm discussed in class to construct the method $testRes(N, A, p)$ which takes in a matrix A of size N and the characteristic polynomial coefficient vector, which was was determined by the program, and outputs *passed* or *failed* using this algorithm. This algorithm could be represented by the following formula $P_nA^n + P_{n-1}A^{n-1}...P_0I = R$, where I is the identity matrix, and R is a zero matrix of size $N \times N$.

For our manual calculations we assumed $A = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{bmatrix}$

The coefficients of the characteristic polynomial of an $n \times n$ matrix A below, are computed in terms of the coefficients of the characteristic polynomial of M; Such that $A = \begin{bmatrix} a_{11} & R \\ S & M \end{bmatrix}$

Therefore $M_1 = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix}$, $M_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$, $M_3 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

$M_1^2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$, $M_2^3 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$

$R_1 = \begin{bmatrix} 1 & 0 & 0 & 1 \end{bmatrix}, \qquad S_1 = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$

$R_2 = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}, \qquad S_2 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$

$R_3 = \begin{bmatrix} 0 & 0 \end{bmatrix}, \qquad S_3 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

Considering if A is a 4×4 matrix, then $p = C_1\ q$ is given by:

$$\begin{bmatrix} p_4 \\ p_3 \\ p_2 \\ p_1 \\ p_0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -a_{11} & 1 & 0 & 0 \\ -RS & -a_{11} & 1 & 0 \\ -RMS & -RS & -a_{11} & 1 \\ -RM^2S & -RMS & -RS & -a_{11} \end{bmatrix} \begin{bmatrix} q_3 \\ q_2 \\ q_1 \\ q_0 \end{bmatrix}$$

Therefore, $C_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$, $C_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$, $C_3 = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$, $C_4 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}$, $C_5 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

By Multiplying all the C Matrices our final resultant matrix is $p = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$

The output for our program agreed with this p.

```
>>> ================================ RESTART ================================
>>>
Enter the size of the matrix (N): 50

Generating matrix...  Done.
Generating kPeers...  Done.
Starting kPeers...  Done. Time: 27.506000
Generating C Product Matrix...  Done.
Calculating Product of C matrices...  Done. 12.680000
Testing...  Passed
>>> ================================ RESTART ================================
>>>
Enter the size of the matrix (N): 100

Generating matrix...  Done.
Generating kPeers...  Done.
Starting kPeers...  Done. Time: 907.477000
Generating C Product Matrix...  Done.
Calculating Product of C matrices...  Done. 375.692000
Testing...  Passed
>>>
```

Figure 2: Program Output