Assignment No. 7

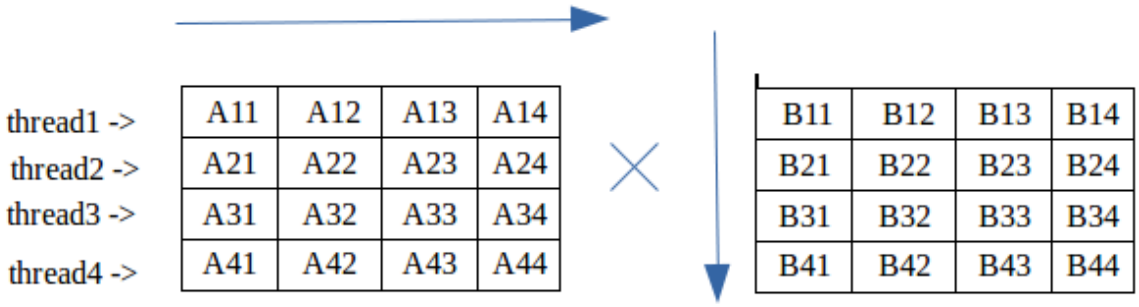
Aim: Implement multithreaded matrix multiplication.
Theory:
Multiplication of matrix does take time surely. Time complexity of matrix multiplication is O(n^3) using normal matrix multiplication. And the Strassen **algorithm** improves it and its time complexity is O(n^(2.8074)).

But, Is there any way to improve the performance of matrix multiplication using the normal method.

Multi-threading can be done to improve it. In multi-threading, instead of utilizing a single core of your processor, we utilize all or more cores to solve the problem.
We create different threads, each thread evaluating some part of matrix multiplication.

Depending upon the number of cores your processor has, you can create the number of threads required. Although you can create as many threads as you need, a better way is to create each thread for one core.

In the second approach,we create a separate thread for each element in the resultant matrix. Using **pthread_exit()** we return computed value from each thread which is collected by **pthread_join()**. This approach does not make use of any global variables.



Examples:

Input :
Matrix A
1 0 0
0 1 0

0 0 1

Matrix B
2 3 2
4 5 1
7 8 6

Output : Multiplication of A and B
2 3 2
4 5 1
7 8 6

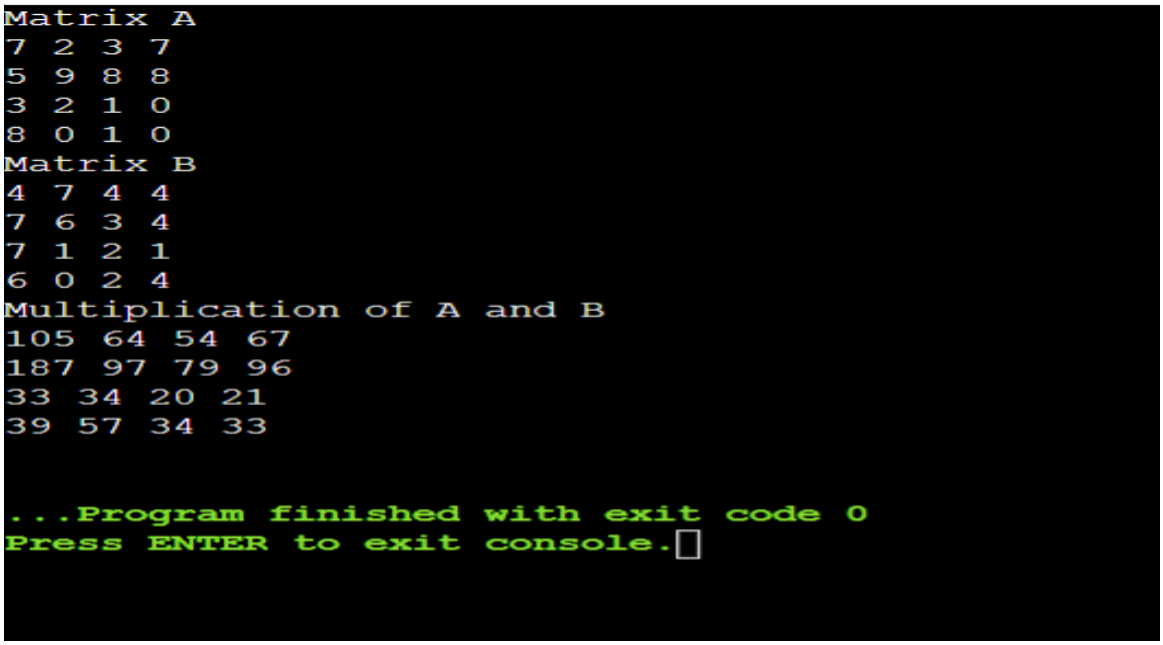
Code:

```
import java.util.Random;
public class Main {
	static final int MAX = 4;
	static final int MAX_THREAD = 4;
	static int[][] matA = new int[MAX][MAX];
	static int[][] matB = new int[MAX][MAX];
	static int[][] matC = new int[MAX][MAX];
	static int step_i = 0;
	static class Worker implements Runnable {
		int i;
		Worker(int i) {
			this.i = i;
		}
		@Override
		public void run() {
			for (int j = 0; j < MAX; j++) {
				for (int k = 0; k < MAX; k++) {
					matC[i][j] += matA[i][k] * matB[k][j];
				}
			}
		}
	}
	public static void main(String[] args) {
		Random rand = new Random();

		for (int i = 0; i < MAX; i++) {
```

```
                for (int j = 0; j < MAX; j++) {
                        matA[i][j] = rand.nextInt(10);
                        matB[i][j] = rand.nextInt(10);
                }
        }
        System.out.println("Matrix A");
        for (int i = 0; i < MAX; i++) {
                for (int j = 0; j < MAX; j++) {
                        System.out.print(matA[i][j] + " ");
                }
                System.out.println();
        }
        System.out.println("Matrix B");
        for (int i = 0; i < MAX; i++) {
                for (int j = 0; j < MAX; j++) {
                        System.out.print(matB[i][j] + " ");
                }
                System.out.println();
        }
        Thread[] threads = new Thread[MAX_THREAD];
        for (int i = 0; i < MAX_THREAD; i++) {
                threads[i] = new Thread(new Worker(step_i++));
                threads[i].start();
        }
        for (int i = 0; i < MAX_THREAD; i++) {
                try {
                        threads[i].join();
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }
        System.out.println("Multiplication of A and B");
        for (int i = 0; i < MAX; i++) {
                for (int j = 0; j < MAX; j++) {
                        System.out.print(matC[i][j] + " ");
                }
                System.out.println();
        }
    }
}
```

Output:


Matrix A
7 2 3 7
5 9 8 8
3 2 1 0
8 0 1 0
Matrix B
4 7 4 4
7 6 3 4
7 1 2 1
6 0 2 4
Multiplication of A and B
105 64 54 67
187 97 79 96
33 34 20 21
39 57 34 33
...Program finished with exit code 0
Press ENTER to exit console.