Parsa Yadollahi - 260869949
Nicholas Nikas - 260870980

ECSE 420 - Assignment 3 Report

Question 1:

1.1) *L'* represents the cache size/4 since we assume a cache line size of 4 words.
Time t_0 represents the cache hit time.

1.2) When *L* is larger than *L'* then we have more words than the amount we can store in the cache which implies that we will have a cache miss. Hence time t_1 represents the average time to access the array when an element is not in the cache.

1.3) Part 1 of the graph is when the entire array fits within the cache which implies a constant access time.

Part 2 of the graph is when access can result in either a cache hit or a cache miss therefore we do not have a constant access time. We can see that as the stride increases, the access time increases which makes sense as we will more likely have a cache miss.

Part 3 of the graph is when every access will result in a cache miss hence we get a constant access time but it is much larger than part 1 where every access is a cache hit.

1.4) The padding technique used in Anderson Lock can degrade the overall performance of the lock by causing more cache misses since we are adding more elements to the array. This would undo any performance benefit we get from using Anderson Lock.

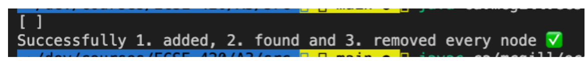
Question 2:

2.1)

The code for the *contains()* function can be found in the Appendix. The method is similar to how *add()* and *remove()* methods were implemented in chapter 9 of the textbook. The function simply iterates over the nodes and verifies if the current node is the appropriate key. If we've found the node we return True if not we return False.

2.2)

The code for testing the *contains()* function can be found in the Appendix. The test first creates *X* amount of threads. Then add items to the Linked List using *add()*. Once it has added them, the thread sleeps for some time. We then verify if the element was added using *contains()*, and remove that node from the Linked List using *remove()*. Finally, we print the LinkedList. If our implementation and test are correct, we should print an empty LinkedList like so:



If any of the functions do not succeed we print an error message. These error message are for the add, remove and contains function. We ran this implementation with various numbers of threads and items and it always returned a successful result.

Question 3:

3.1) Please see the code under LockBasedQueue Appendix for our implementation.

3.2) When transforming our algorithm to be lock-free we ran into difficulty where the value of the head, tail, and size variables would be incorrect due to race conditions. In order to fix this, we changed these variables from regular integers to atomic integers. This ensured we had accurate values for head, tail, and size and that the queue was not empty during dequeuing and not full during enqueuing.

Question 4:

4.1)

The algorithm for sequential multiplication has a similar approach to that done in Assignment 1 when sequentially multiplying two matrices together. We first create a random matrix using code from assignment 1 and a random vector that is based on assignment 1. We then compute the dot product on rows of the matrix with the vector.

4.2)

The algorithm for parallel multiplication splits the problem into multiple tasks where each task was in charge of calculating one entry of the resulting vector. That is multiplying one row of the matrix by one column of the vector. Similar to 4.1 where we multiply every row by the vector. We do this in parallel.

4.3)

We recorded the execution time of the sequential multiplication and parallel multiplication using a 2000x2000 dimension matrix multiplicated by a 2000x1 dimension vector. The matrix and vector were generated using random numbers from 0 to 10 using methods created in assignment 1. The execution time for the sequential multiplication was 0.023 seconds and the parallel multiplication was 0.005 seconds using 3 threads. We ran the parallel multiplication experiment with 10 different threads and recorded their results and returned the number of threads that were the least time-consuming. We plotted the number of threads vs. execution time in figure 1. The speedup on the processor is the time taken by one processor divided by the time taken for *x* number of processors. Hence the speedup using 3 processors was 0.023 / 0.005 = 4.6.

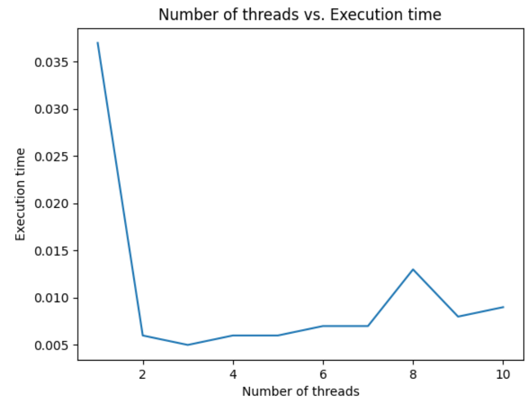


Figure 1: Number of threads vs. Execution time(s)

4.4)

Each of the subtasks performs *n* multiplications and *n-1* additions where *n* is the number of entries in the matrix. There are *n* subtasks meaning the work is $n * (n + n - 1)$ which is $O(n^2 + n^2 - n) = O(n^2)$. Since the critical path can be executed in parallel, the critical path would be the cost of one subtask being $O(n)$. The parallelism is the work divided by the critical path which is $O(n^2) / O(n) = O(n)$.

Appendix

Figure 2 - Fine Grain:

To run this file, simply run the following commands from the src folder

- javac ca/mcgill/ecse420/a2/FineGarin.java ca/mcgill/ecse420/a2/Node.java ca/mcgill/ecse420/a2/FineGarin.java
- java ca/mcgill/ecse420/a2/TestFineGrain

FineGrain.java

```java
You, 4 minutes ago | 1 author (You)
package ca.mcgill.ecse420.a3;

import ca.mcgill.ecse420.a3.Node;

You, 4 minutes ago | 1 author (You)
public class FineGrain<T> {
  private static Node head;

  public FineGrain() {
    head = new Node<>(Integer.MAX_VALUE);
    head.next = new Node<>(Integer.MAX_VALUE);
  }

  // Code taken from chapter 9
  public boolean add(T item) {
    int key = item.hashCode();

    head.lock();
    Node prev = head;
    try {
      Node curr = prev.next;
      curr.lock();
      try {
        while (curr.key < key) {
          prev.unlock();
          prev = curr;
          curr = curr.next;
          curr.lock();
        }

        if (curr.key == key) {
          return false;
        }

        Node newNode = new Node<T>(item);
        newNode.next = curr;
        prev.next = newNode;
        return true;
      } finally {
        curr.unlock();
      }
    } finally {
      prev.unlock();
    }
  }

```

```java
// Code taken from chapter 9
public boolean remove(T item) {
  Node prev = null;
  Node curr = null;

  int key = item.hashCode();
  head.lock();

  try {
    prev = head;
    curr = prev.next;
    curr.lock();
    try {
      while (curr.key < key) {
        prev.unlock();
        prev = curr;
        curr = curr.next;
        curr.lock();
      }
      if (curr.key == key) {
        prev.next = curr.next;
        return true;
      }
      return false;
    } finally {
      curr.unlock();
    }
  } finally {
    prev.unlock();
  }
}
```

```java

  // Question 2.1
  public boolean contains(T item) {
    Node prev = null;
    Node curr = null;

    int key = item.hashCode();
    head.lock();
      You, last week • A3 Node …
    try {
      prev = head;
      curr = prev.next;
      curr.lock();
      try {
        while(curr.key < key) {
          prev.unlock();
          prev = curr;
          curr = curr.next;
          curr.lock();
        }
        /*
        Only modification to add and remove.
        If we've found the node, we return true
        */
        if (curr.key == key){
          return true;
        }
      } finally {
        curr.unlock();
      }
    } finally {
      prev.unlock();
    }
    return false;
  }

  public static void printLinkedList() {
    Node curr = head.next;
    String linkedList = "";

    while(curr.item != null) {
      linkedList += "[ " + curr.item.toString() + " ]";
      curr = curr.next;

      if (curr.next != null){
        linkedList += " -> ";
      }
    }

    // If we have no nodes, add an empty one to make it look nice nice
    if (linkedList == ""){
      linkedList = "[ ]";
    }

    System.out.println(linkedList);
    if (linkedList == "[ ]") {
      System.out.println("Successfully 1. added, 2. found and 3. removed every node ✅");
    }
  }
}
```

Node.java

```java
You, last week | 1 author (You)
package ca.mcgill.ecse420.a3;

import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

You, last week | 1 author (You)
public class Node<T> {
    T item;
    int key;
    Node next;
    Lock lock;


    Node(int key) {
        this.item = null;
        this.key = key;
        this.lock = new ReentrantLock();
    }

    Node(T item) {
        this.item = item;
        this.key = item.hashCode();
        this.lock = new ReentrantLock();
    }


    public void lock() {
        this.lock.lock();
    }

    public void unlock() {        You, last week • A3 No
        this.lock.unlock();
    }
}

```

TestFineGrain.java

```java
You, 2 minutes ago | 1 author (You)
package ca.mcgill.ecse420.a3;

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

import ca.mcgill.ecse420.a3.FineGrain;
import ca.mcgill.ecse420.a3.Node;

You, 2 minutes ago | 1 author (You)
public class TestFineGrain {
  public static int NUM_THREADS = 3;
  public static int NUM_ITEMS = 10;
  public static int THREAD_ITEMS = NUM_ITEMS / NUM_THREADS;

  public static FineGrain<Integer> fineGrain= new FineGrain<>();

  Run | Debug
  public static void main(String[] args) {
    ExecutorService executorService = Executors.newFixedThreadPool(NUM_THREADS);

    for (int i = 0; i < NUM_THREADS; i++){
      executorService.execute(new NodeRunnable(i));
    }
    executorService.shutdown();

    try {
      executorService.awaitTermination(10, TimeUnit.SECONDS);
    } catch (InterruptedException e) {
      e.printStackTrace();
    }

    fineGrain.printLinkedList();
  }

```

```java
You, 2 minutes ago | 1 author (You)
public static class NodeRunnable implements Runnable {
  int thread_num;

  public NodeRunnable(int t_num) {
    this.thread_num = t_num;
  }

  @Override
  public void run() {
    // Each thread adds THREAD_ITEMS to the linked list
    for (int thread_item = 0; thread_item < THREAD_ITEMS; thread_item++) {
      // Get unique integers
      int item = thread_num + THREAD_ITEMS * thread_item;

      if (!fineGrain.add(item)) {
        System.out.println("Failed to add " + item);
      }

      try {
        Thread.sleep(10);       You, 2 minutes ago • Uncommitted changes
      } catch (InterruptedException e) {
        e.printStackTrace();
      }

      if (fineGrain.contains(item)) {
        if (!fineGrain.remove(item)) {
          System.out.println("Failed to Remove " + item);
        }
      } else {
        System.out.println("Failed to Find " + item);
      }
    }
  }
}
}
```

Figure 4 - Bounded Lock-Based Queue:

To run this file, simply run the following commands from the src folder

- javac ca/mcgill/ecse420/a3/LockBasedQueue.java ca/mcgill/ecse420/a3/LockFreeQueue.java ca/mcgill/ecse420/a3/TestQueue.java
- ca.mcgill.ecse420.a3.TestQueue

LockBasedQueue.java

```java
Nicholas Nikas, 18 hours ago | 1 author (Nicholas Nikas)
package ca.mcgill.ecse420.a3;

import java.util.concurrent.atomic.AtomicInteger;
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.ReentrantLock;

Nicholas Nikas, 18 hours ago | 1 author (Nicholas Nikas)
// Bounded lock-based blocking queue
public class LockBasedQueue<T> {

    public ReentrantLock enqLock, deqLock;    // Locks for enqueue and dequeue
    public Condition notEmptyCondition, notFullCondition;   // Conditions whether queue is empty or not
    public AtomicInteger size;                          // Amount of used slots in queue
    public int head;                            // Head entry
    public int tail;                            // Tail entry
    public int capacity;                        // Max capacity in queue
    public Object[] queue;                      // Array version of linked list

    public LockBasedQueue(int capacity) {
        this.queue = new Object[capacity];      Nicholas Nikas, 18 hours ago • Q3 …
        this.head = 0;
        this.tail = this.head;
        this.capacity = capacity;
        this.size = new AtomicInteger(0);
        this.enqLock = new ReentrantLock();
        this.deqLock = new ReentrantLock();
        this.notFullCondition = enqLock.newCondition();
        this.notEmptyCondition = deqLock.newCondition();
    }

    // Add object to the end of the queue
    public void enqueue(T value) {
        if (value == null) {
            throw new NullPointerException();
        }

        // boolean to track if queue becomes empty
        boolean isQueueEmpty = false;

        enqLock.lock();

        try {
            // ensure queue is not full when enqueuing, otherwise wait until not full
            while (this.size.get() == this.capacity) {
                try {
                    this.notFullCondition.await();
                } catch (InterruptedException e) {}
            }

            add(value);

            // verify if queue is empty to prevent false dequeue later
            if (this.size.getAndIncrement() == 0) {
                isQueueEmpty = true;
            }
        } finally {
            this.enqLock.unlock();
        }
```

```java

            // set the dequeue locks to prevent a dequeue in an empty queue
            if (isQueueEmpty) {
                this.deqLock.lock();
                try {
                    this.notEmptyCondition.signalAll();
                } finally {
                    this.deqLock.unlock();
                }
            }
        }

        // Remove and return the head of the queue
        public T dequeue() {
            T value;

            // boolean to track if queue becomes full
            boolean isQueueFull = false;

            this.deqLock.lock();

            try {
                // ensure queue is not empty when dequeuing, otherwise wait until not empty
                while (this.size.get() == 0) {
                    try {
                        this.notEmptyCondition.await();
                    } catch (InterruptedException e) {}
                }

                value = remove();

                // verify if queue is full to prevent false enqueue later
                if (this.size.getAndDecrement() == this.capacity) {
                    isQueueFull = true;
                }
            } finally {
                this.deqLock.unlock();
            }

            // set the enqueue locks to prevent a enqueue in a full queue
            if (isQueueFull) {
                this.enqLock.lock();
                try {
                    this.notFullCondition.signalAll();
                } finally {
                    this.enqLock.unlock();
                }
            }
            return value;
        }

```

```java
    // add element to tail of queue
    public void add(T element) {
        final Object[] items = this.queue;
        items[this.tail] = element;
        // set tail to first element if reach capacity
        if (++this.tail == items.length) {
            this.tail = 0;
        }
    }

    // remove element in the head of the queue
    public T remove() {
        final Object[] items = this.queue;
        T element = (T) items[this.head];
        items[this.head] = null;
        // set head to first element if reach capacity
        if (++this.head == items.length)
            this.head = 0;
        return element;
    }
}
```

LockFreeQueue.java

```java
Nicholas Nikas, 18 hours ago | 1 author (Nicholas Nikas)
package ca.mcgill.ecse420.a3;      Nicholas Nikas, 18 hours ago • Q3 …

import java.util.concurrent.atomic.AtomicInteger;
import java.util.concurrent.atomic.AtomicReferenceArray;

Nicholas Nikas, 18 hours ago | 1 author (Nicholas Nikas)
// bounded lock-free blocking queue
public class LockFreeQueue<T> {

    public AtomicReferenceArray<T> queue;
    public AtomicInteger head, tail, size;
    public int capacity;

    public LockFreeQueue(int maxSize) {
        this.capacity = maxSize;
        this.queue = new AtomicReferenceArray<>(maxSize);
        this.head = new AtomicInteger(0);
        this.tail = new AtomicInteger(0);
        this.size = new AtomicInteger(0);
    }

    // add object to the end of the queue
    public void enqueue(T value) {
        int size = this.size.get();

        // ensure queue is not full when we enqueue, otherwise wait until not full
        while (size == this.capacity || !this.size.compareAndSet(size, size + 1)) {
            size = this.size.get();
        }

        this.queue.set(this.tail.getAndIncrement(), value);

        // if we reach capacity set tail to 0
        if (this.tail.get() == this.capacity) {
            this.tail.set(0);
        }
    }

    // remove and return the head of the queue
    public T dequeue() {
        int size = this.size.get();

        // ensure queue is not empty when we dequeue, otherwise wait until not empty
        while (size == 0 || !this.size.compareAndSet(size, size - 1)) {
            size = this.size.get();
        }
        T value = this.queue.getAndSet(this.head.getAndIncrement(), null);

        // set head to beginning of queue if we reach capacity
        if (this.head.get() == this.capacity) {
            this.head.set(0);
        }

        return value;
    }
}
```

TestQueue.java

```java
You, 7 minutes ago | 2 authors (Nicholas Nikas and others)
package ca.mcgill.ecse420.a3;

You, 7 minutes ago | 2 authors (Nicholas Nikas and others)
public class TestQueue {

    Run | Debug
    public static void main(String[] args) {
        LockBasedQueue<Integer> lbq = new LockBasedQueue<>(4);
        lbq.enqueue(1);
        lbq.enqueue(2);
        lbq.enqueue(3);
        lbq.enqueue(4);

        System.out.println(lbq.queue[lbq.head]); // Should output 1
        System.out.println(lbq.queue[lbq.tail]); // Should output 1

        lbq.dequeue();
        lbq.enqueue(5);
        lbq.dequeue();
        lbq.enqueue(6);      Nicholas Nikas, 19 hours ago • Q3 …

        System.out.println(lbq.queue[lbq.head]); // Should output 3
        System.out.println(lbq.queue[lbq.tail]); // Should output 3

        lbq.dequeue();
        lbq.dequeue();
        lbq.dequeue();
        lbq.dequeue();

        System.out.println(lbq.queue[lbq.head]); // Should output null
        System.out.println(lbq.queue[lbq.tail]); // Should output null

        LockFreeQueue<Integer> lfq = new LockFreeQueue<>(4);
        lfq.enqueue(1);
        lfq.enqueue(2);
        lfq.enqueue(3);
        lfq.enqueue(4);

        System.out.println(lfq.queue.get(lfq.head.get())); // Should output 1
        System.out.println(lfq.queue.get(lfq.tail.get())); // Should output 1

        lfq.dequeue();
        lfq.enqueue(5);
        lfq.dequeue();
        lfq.enqueue(6);

        System.out.println(lfq.queue.get(lfq.head.get())); // Should output 3
        System.out.println(lfq.queue.get(lfq.tail.get())); // Should output 3

        lfq.dequeue();
        lfq.dequeue();
        lfq.dequeue();
        lfq.dequeue();

        System.out.println(lfq.queue.get(lfq.head.get())); // Should output null
        System.out.println(lfq.queue.get(lfq.tail.get())); // Should output null
    }
}
```

Figure 5 - Sequential and Parallel Multiplication:

To run this file, simply run the following commands from the src folder

- javac javac ca/mcgill/ecse420/a3/Vector.java ca/mcgill/ecse420/a3/Matrix.java ca/mcgill/ecse420/a3/ParallelMatrixVectorMulti.java ca/mcgill/ecse420/a3/TestMatrixVectorMulitply.java ca/mcgill/ecse420/a3/ParallelMatrixVectorMulti.java
- java ca.mcgill.ecse420.a3.TestMatrixVectorMulitply

Matrix.java

src > ca > mcgill > ecse420 > a3 > Matrix.java > Matrix > Matrix(double[][], int, int, int)

```java
You, 5 days ago | 1 author (You)
package ca.mcgill.ecse420.a3;

You, 5 days ago | 1 author (You)
public class Matrix {
  int dimension;
  double[][] data;
  int rowDisplace, colDisplace;

  public Matrix(int d) {
    dimension = d;
    rowDisplace = 0;
    colDisplace = 0;
    data = new double[d][d];
  }

  Matrix(double[][] matrix, int x, int y, int d) {
    data = matrix;
    rowDisplace = x;      You, 6 days ago • Q4.2 added Matrix.java and Vector.java …
    colDisplace = y;
    dimension = d;
  }

  public double get(int row, int col) {
    return data[row + rowDisplace][col + colDisplace];
  }

  public void set(int row, int col, double val) {
    data[row + rowDisplace][col + colDisplace] = val;
  }

  public int getDim() {
    return dimension;
  }

  public Matrix[][] split() {
    Matrix[][] result = new Matrix[2][2];
    int newDimension = dimension / 2;
    result[0][0] = new Matrix(data, rowDisplace, colDisplace, newDimension);
    result[0][1] = new Matrix(data, rowDisplace, colDisplace + newDimension, newDimension);
    result[1][0] = new Matrix(data, rowDisplace + newDimension, colDisplace, newDimension);
    result[1][1] = new Matrix(data, rowDisplace + newDimension, colDisplace + newDimension, newDimension);
    return result;
  }

  /**
   * Code taken from Assignment 1
   * Populates a matrix of given size with randomly generated integers between 0-10.
   * @param numRows number of rows
   * @param numCols number of cols
   * @return matrix
   */
  public void generateRandomMatrix() {
    for (int row = 0 ; row < dimension  ; row++ ) {
      for (int col = 0 ; col < dimension ; col++ ) {
        data[row][col] = (double) ((Math.random() * 10.0));
      }
    }
  }
```

```java

    /**
     * Prints a matrix
     * @params None
     * @return None
     */
    public void printMatrix() {
        for (int row = 0 ; row < dimension  ; row++ ) {
            for (int col = 0 ; col < dimension ; col++ ) {
                System.out.print(data[row][col] + " ");
            }
            System.out.println();
        }
    }
}

```

Vector.java

```
You, 3 days ago | 1 author (You)
package ca.mcgill.ecse420.a3;

You, 3 days ago | 1 author (You)
public class Vector {
  int dimension;
  double[] data;
  int rowDisplace;

  public Vector(int d) {
    dimension = d;
    rowDisplace = 0;
    data = new double[d];
  }

  Vector(double[] matrix, int x, int d) {
    data = matrix;
    rowDisplace = x;
    dimension = d;
  }

  public double get(int row) {
    return data[row + rowDisplace];
  }

  public void set(int row, double value) {
    data[row + rowDisplace] = value;
  }

  public int getDim() {
    return dimension;
  }

  public Vector[] split() {
    Vector[] result = new Vector[2];
    int newDimension = dimension / 2;
    result[0] = new Vector(data, rowDisplace, newDimension);
    result[1] = new Vector(data, rowDisplace + newDimension, newDimension);
    return result;
  }

  /**
   * Code taken from Assignment 1
   * Populates a matrix of given size with randomly generated integers between 0-10.
   * @param numRows number of rows
   * @param numCols number of cols
   * @return matrix
   */
  public void generateRandomVector() {
    for (int row = 0 ; row < dimension ; row++ ) {
      data[row] = (double) ((Math.random() * 10.0));
    }
  }

  /**
   * Prints content of a Vector
   * @params None
   * @return None
   */
  public void printVector() {
    for (int row = 0 ; row < dimension ; row++ ) {
      System.out.println(data[row]);
    }
    System.out.println();
  }
```

```java

    public boolean isSame(Vector v) {
      if (v.dimension != dimension) {
        return false;
      }
      for (int row = 0; row < v.dimension; row ++) {
        if (v.get(row) != data[row]) {
          return false;
        }      You, 3 days ago • Q4 DONE …
      }
      return true;
    }
  }

```

SeqMatrixVectorMultiplic.java

```
You, 1 hour ago | 1 author (You)
package ca.mcgill.ecse420.a3;

You, 1 hour ago | 1 author (You)
public class SeqMatrixVectorMultiplic {

  Run | Debug
  public static void main(String[] args) {}

  /**
   * Returns the result of a sequential matrix and vector multiplication
   * The Matrix and vector are randomly generated
   * @param matrix is the matrix
   * @param vector is the vector
   * @return the result of the multiplication
   * */
  public static Vector multiply(Matrix matrix, Vector vector) {
    int rows = matrix.dimension;
    int cols = vector.dimension;

    Vector vector_res = new Vector(rows);      You, 3 days ago • Q4 DONE …

    if (rows != cols) {
      throw new ArithmeticException("Invalid Matrix dimensions");
    }

    for (int row = 0; row < rows; row++) {
      for (int col = 0; col < cols; col++) {
        vector_res.set(row, vector_res.get(row) + matrix.get(row, col) * vector.get(col));
      }
    }
    return vector_res;
  }
}
```

ParallelMatrixMulti.java

src > ca > mcgill > ecse420 > a3 > ParallelMatrixVectorMulti.java > ParallelMatrixVectorMulti > multiply(Matrix, Vector, int)

```
You, 1 hour ago | 1 author (You)
package ca.mcgill.ecse420.a3;

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

You, 1 hour ago | 1 author (You)
public class ParallelMatrixVectorMulti {
  public int NUM_THREADS;
  public Matrix matrix;
  public Vector vector;

  public ParallelMatrixVectorMulti(Matrix matrix, Vector vector, int NUM_THREADS) {
    this.matrix = matrix;
    this.vector = vector;
    this.NUM_THREADS = NUM_THREADS;
  }

  public static Vector multiply(Matrix a, Vector b, int num_threads) {
    Vector c = new Vector(b.dimension);
    int m_size = a.dimension;
    ExecutorService exec = Executors.newFixedThreadPool(num_threads);
      You, 6 days ago • Q4.2 no tests …
    for (int col = 0; col < m_size; col++) {
      exec.execute(new OneEntryMultiply(a, b, c, col));
    }
    exec.shutdown();

    try {
      exec.awaitTermination(10, TimeUnit.SECONDS);
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
    return c;
  }

  You, 3 days ago | 1 author (You)
  static class OneEntryMultiply implements Runnable {
    private Matrix a;
    private Vector b, c;
    private int MATRIX_SIZE, col;

    public OneEntryMultiply(Matrix a, Vector b, Vector c, int col){
      this.a = a;
      this.b = b;
      this.c = c;
      this.MATRIX_SIZE = a.dimension;
      this.col = col;
    }

    @Override
    public void run() {
      for (int row = 0; row < MATRIX_SIZE; row++) {
        c.set(col, c.get(col) + a.get(col, row) * b.get(row));
      }
    }
  }
}
```

TestMatrixVectorMultiply.java

```
src > ca > mcgill > ecse420 > a3 > TestMatrixVectorMulitply.java > TestMatrixVectorMulitply > main(String[])
You, 20 minutes ago | 1 author (You)
package ca.mcgill.ecse420.a3;

import java.util.concurrent.ExecutionException;
import java.util.Arrays;


You, 20 minutes ago | 1 author (You)
public class TestMatrixVectorMulitply {
  private static final int MAX_NUM_THREADS = 10;
  private static final int MATRIX_SIZE = 2000;

  Run | Debug
  public static void main(String [] args) throws InterruptedException, ExecutionException {
    Matrix m = new Matrix(MATRIX_SIZE);
    m.generateRandomMatrix();
    // if (MATRIX_SIZE < 15) {
    //   System.out.println("The matrix to multiply");
    //   System.out.println("----------------------");
    //   m.printMatrix();
    // }

    Vector v = new Vector(MATRIX_SIZE);
    v.generateRandomVector();
    // if (MATRIX_SIZE < 15) {
    //   System.out.println();
    //   System.out.println("The vector to multiply");
    //   System.out.println("----------------------");
    //   v.printVector();
    // }

    System.out.println();

    double start = System.currentTimeMillis();
    Vector res_seq = SeqMatrixVectorMultiplic.multiply(m, v);
    double end = System.currentTimeMillis();
    // if (MATRIX_SIZE < 15) {
    //   System.out.println("Sequential Multiply");
    //   System.out.println("-------------------");
    //   res_seq.printVector();
    // }

    System.out.println("[TIME] Sequential time: " + (end - start) / 1000.0 + " seconds");
    System.out.println("-----------------------");


    double[] parallelMultiplyTimes = new double[MAX_NUM_THREADS];
    Vector res_parallel = new Vector(10);

    for (int i = 1; i < MAX_NUM_THREADS+1; i++) {
      start = System.currentTimeMillis();
      res_parallel = ParallelMatrixVectorMulti.multiply(m, v, i);
      end = System.currentTimeMillis();
      // if (MATRIX_SIZE < 15) {
      //   System.out.println("Parallel Multiply");
      //   System.out.println("-----------------");
      //   res_parallel.printVector();
      // }
      parallelMultiplyTimes[i - 1] = (end - start) / 1000.0;     You, 20 minutes ago • Uncommitted changes
    }

```

```java


        double fastestTime = Arrays.stream(parallelMultiplyTimes).min().getAsDouble();
        System.out.println("[TIME] Parallel time: " + fastestTime + " seconds with " + find(parallelMultiplyTimes, fastestTime));
        System.out.println("---------------------");

        System.out.println("The Sequential and Parallel vectors are the same: " + res_seq.isSame(res_parallel));
    }


    public static int find(double[] array, double value) {
        for(int i=0; i<array.length; i++)
            if(array[i] == value) return i + 1;
        return 3;
    }
}

```