Appendix - Code Generation – Luca Zacharias

Query 1 (Can you write a Java function for me that multiplies Matrices using multithreading?):

ChatGPT-3.5:

Certainly! Here's a Java function that multiplies matrices using multithreading. This function uses the Java `ExecutorService` framework to parallelize the matrix multiplication task. We'll break down the code into several steps:

1. Create a class `MatrixMultiplier` that implements `Runnable` and is responsible for multiplying a portion of the matrices.
2. Create a function `multiplyMatricesWithThreads` that takes two matrices as input and performs matrix multiplication using multithreading.

Here's the code:

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

public class MatrixMultiplier implements Runnable {
    private final int[][] resultMatrix;
    private final int[][] matrixA;
    private final int[][] matrixB;
    private final int rowStart;
    private final int rowEnd;
    private final int colStart;
    private final int colEnd;

    public MatrixMultiplier(int[][] resultMatrix, int[][] matrixA, int[][] matrixB, int rowStart, int rowEnd, int colStart, int colEnd) {
        this.resultMatrix = resultMatrix;
        this.matrixA = matrixA;
        this.matrixB = matrixB;
        this.rowStart = rowStart;
        this.rowEnd = rowEnd;
        this.colStart = colStart;
        this.colEnd = colEnd;
    }

    @Override
    public void run() {
        for (int i = rowStart; i < rowEnd; i++) {
            for (int j = colStart; j < colEnd; j++) {
                resultMatrix[i][j] = 0;
                for (int k = 0; k < matrixA[0].length; k++) {
                    resultMatrix[i][j] += matrixA[i][k] * matrixB[k][j];
                }
            }
        }
    }

    public static int[][] multiplyMatricesWithThreads(int[][] matrixA, int[][] matrixB, int numThreads) {
        int numRowsA = matrixA.length;
        int numColsA = matrixA[0].length;
        int numRowsB = matrixB.length;
        int numColsB = matrixB[0].length;
        if (numColsA != numRowsB) {
            throw new IllegalArgumentException("Matrix dimensions are not compatible for multiplication.");
        }

        int[][] resultMatrix = new int[numRowsA][numColsB];
        ExecutorService executor = Executors.newFixedThreadPool(numThreads);
        int step = numRowsA / numThreads;
        int rowStart = 0;
        int rowEnd = step;
        for (int i = 0; i < numThreads; i++) {
            if (i == numThreads - 1) {
                rowEnd = numRowsA; // Last thread takes the remaining rows
            }
            executor.execute(new MatrixMultiplier(resultMatrix, matrixA, matrixB, rowStart, rowEnd, 0,
numColsB));
            rowStart = rowEnd;
            rowEnd += step;
        }
        executor.shutdown();
        try {
            executor.awaitTermination(Long.MAX_VALUE, TimeUnit.NANOSECONDS);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
        return resultMatrix;
    }
```

In this code:

- `MatrixMultiplier` is a class that represents a worker thread responsible for computing a portion of the result matrix.
- `multiplyMatricesWithThreads` is the main function that divides the matrix multiplication task into multiple threads, assigns each thread a portion of the work, and waits for all threads to complete.
- We use an `ExecutorService` with a fixed number of threads to parallelize the matrix multiplication.

- The `main` method demonstrates how to use the `multiplyMatricesWithThreads` function with sample input matrices. Make sure to adjust the input matrices and the number of threads according to your requirements.

ChatGPT-4.0:

Certainly! To multiply matrices using multithreading, we can divide the resulting matrix's rows among multiple threads. Each thread will then be responsible for computing the values in its assigned rows.

Here's a simple example that multiplies two matrices using multithreading:

```
private static final int NUM_THREADS = 8;  // Adjust based on your requirements

    public static int[][] multiplyWithThreads(int[][] A, int[][] B) {
        int aRows = A.length;
        int aColumns = A[0].length;
        int bRows = B.length;
        int bColumns = B[0].length;

        if (aColumns != bRows) {
            throw new IllegalArgumentException("A:Columns: " + aColumns + " did not match B:Rows " +
bRows + ".");
        }

        int[][] C = new int[aRows][bColumns];

        List<Thread> threads = new ArrayList<>();
        for (int i = 0; i < NUM_THREADS; i++) {
            final int threadId = i;
            Thread thread = new Thread(() -> {
                int fromRow = threadId * aRows / NUM_THREADS;
                int toRow = (threadId + 1) * aRows / NUM_THREADS;
                for (int row = fromRow; row < toRow; row++) {
                    for (int col = 0; col < bColumns; col++) {
                        for (int k = 0; k < aColumns; k++) {
                            C[row][col] += A[row][k] * B[k][col];
                        }
                    }
                }
            });
            threads.add(thread);
            thread.start();
        }

        for (Thread thread : threads) {
            try {
                thread.join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        return C;
    }
```

Note:

1. This example uses a fixed number of threads (`NUM_THREADS`). Depending on the size of your matrices and the specifics of your hardware, you might want to adjust this value.
2. Be sure to handle any potential exceptions appropriately for your specific needs. This example is a basic demonstration and might need further optimizations and safety checks for a more robust application.

Bing Chat:

```
import java.util.concurrent.*;

public class MatrixMultiplier {
    private static class MultiplyTask implements Callable<Integer> {
        private final int[][] matrixA;
        private final int[][] matrixB;
        private final int row;
        private final int col;

        public MultiplyTask(int[][] matrixA, int[][] matrixB, int row, int col) {
            this.matrixA = matrixA;
            this.matrixB = matrixB;
            this.row = row;
            this.col = col;
        }

        @Override
        public Integer call() {
            int sum = 0;
            for (int i = 0; i < matrixA[row].length; i++) {
                sum += matrixA[row][i] * matrixB[i][col];
            }
            return sum;
        }
    }

    public static int[][] multiply(int[][] matrixA, int[][] matrixB, int numThreads) throws InterruptedException, ExecutionException {
        int rows = matrixA.length;
        int cols = matrixB[0].length;

        ExecutorService executor = Executors.newFixedThreadPool(numThreads);
        Future<Integer>[][] futures = new Future[rows][cols];
        int[][] result = new int[rows][cols];

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                futures[i][j] = executor.submit(new MultiplyTask(matrixA, matrixB, i, j));
            }
        }

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                result[i][j] = futures[i][j].get();
            }
        }

        executor.shutdown();
        return result;
    }
}
```

Google Bard:

- *First, it outputs Code which does not compile. Error: Variable used in lambda expression should be final or effectively final.*

```
threads[i] = new Thread(() -> {
    for (int row = finalStartRow; row <= finalEndRow; row++) {
        for (int column = 0; column < resultMatrix[0].length; column++) {
            resultMatrix[row][column] = multiplyRowByColumn(matrix1[row], matrix2[column]);
        }
    }
});
```

- When asked to fix the problem, it finds a code which compiles and produces the right result.

Fixed Code:

```
public static int[][] multiplyMatricesMultithreaded(int[][] matrix1, int[][] matrix2, int numThreads) {

        // Check if the matrices are compatible for multiplication.

        if (matrix1[0].length != matrix2.length) {

            throw new IllegalArgumentException("The matrices must have compatible dimensions for
multiplication.");

        }
```

```
// Create a new matrix to store the product of the two matrices.
int[][] resultMatrix = new int[matrix1.length][matrix2[0].length];

// Create an array of threads to perform the matrix multiplication.
Thread[] threads = new Thread[numThreads];

// Divide the rows of the result matrix into segments based on the number of threads.
int segmentSize = resultMatrix.length / threads.length;
int startRow, endRow;
for (int i = 0; i < threads.length; i++) {
    startRow = i * segmentSize;
    endRow = (i + 1) * segmentSize - 1;
    if (i == threads.length - 1) {
        endRow = resultMatrix.length - 1;
    }

    // Create a new thread to multiply the rows in the assigned segment.
    int finalStartRow = startRow;
    int finalEndRow = endRow;
    threads[i] = new Thread(() -> {
        for (int row = finalStartRow; row <= finalEndRow; row++) {
            for (int column = 0; column < resultMatrix[0].length; column++) {
                resultMatrix[row][column] = 0; // This fixes the bug.
                for (int k = 0; k < matrix1[0].length; k++) {
                    resultMatrix[row][column] += matrix1[row][k] * matrix2[k][column];
                }
            }
        }
    });
}

// Start all of the threads.
for (Thread thread : threads) {
    thread.start();
}
```

```
        // Wait for all of the threads to finish.
        for (Thread thread : threads) {
            try {
                thread.join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        // Return the result matrix.
        return resultMatrix;
    }

    // Multiplies a row of a matrix by a column of a matrix and returns the result.
    private static int multiplyRowByColumn(int[] row, int[] column) {
        int result = 0;
        for (int i = 0; i < row.length; i++) {
            result += row[i] * column[i];
        }
        return result;
    }
```

YouChat:

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

public class MatrixMultiplier {

    public static int[][] multiplyMatricesWithThreads(int[][] matrix1, int[][] matrix2, int numThreads) {
        int rows1 = matrix1.length;
        int columns1 = matrix1[0].length;
        int columns2 = matrix2[0].length;

        int[][] result = new int[rows1][columns2];
        ExecutorService executor = Executors.newFixedThreadPool(numThreads);

        for (int i = 0; i < rows1; i++) {
          final int row = i;
          executor.execute(() -> {
            for (int j = 0; j < columns2; j++) {
              for (int k = 0; k < columns1; k++) {
                result[row][j] += matrix1[row][k] * matrix2[k][j];
              }
            }
          });
        }

        executor.shutdown();
        while (!executor.isTerminated()) {
          // Wait until all threads finish their tasks
        }

        return result;
    }
```

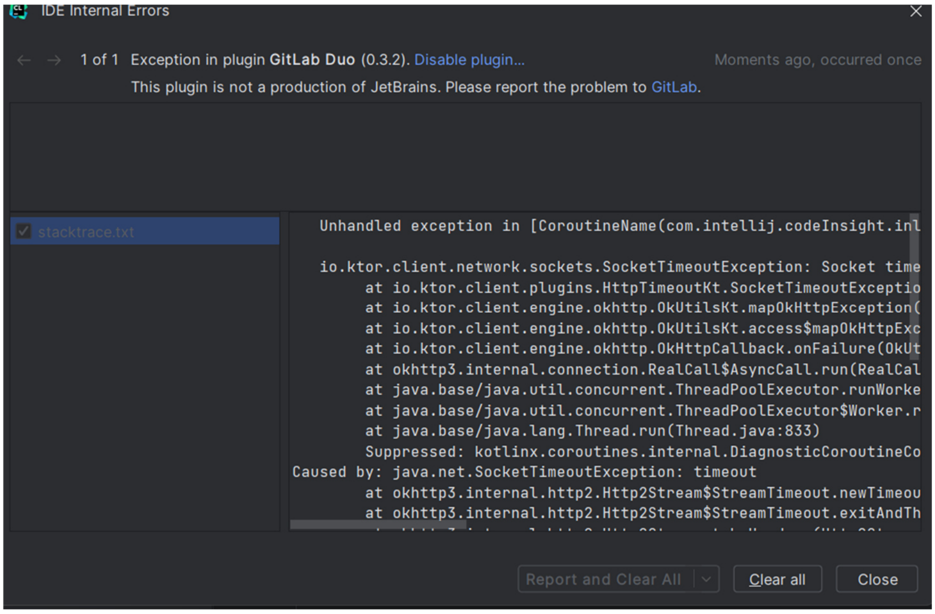
GitLab Duo:

- Tested the capabilities in C because Plugin is easier to use in JetBrains CLion, than in JetBrains IntelliJ IDEA.
- Code completion works only if you give a complete function header and the function has an easy/clear name.
- GitLab Duo was not capable of producing code that multiplies matrices with multi-threading. It only produced code for normal matrix multiplication, no matter how you changed the function header or the comment, describing the function.

```
#define MAX_SIZE 100 // You can define a maximum size for the matrices

//Matrix multiplication with multi-threading
void multiplyMatricesThreads(int mat1[][MAX_SIZE], int mat2[][MAX_SIZE], int result[][MAX_SIZE]){
 int i, j, k;
 for(i = 0; i < MAX_SIZE; i++) {
  for(j = 0; j < MAX_SIZE; j++) {
   result[i][j] = 0;
   for (k=0; k<MAX_SIZE; k++) {
    result[i][j] += mat1[i][k] * mat2[k][j];
   }
  }
 }
}
```

- Sometimes the tool could not make a prediction because it had an error.



1 of 1 Exception in plugin **GitLab Duo** (0.3.2). Disable plugin... Moments ago, occurred once

This plugin is not a production of JetBrains. Please report the problem to GitLab.

```
  Unhandled exception in [CoroutineName(com.intellij.codeInsight.inl

  io.ktor.client.network.sockets.SocketTimeoutException: Socket time
        at io.ktor.client.plugins.HttpTimeoutKt.SocketTimeoutExceptio
        at io.ktor.client.engine.okhttp.OkUtilsKt.mapOkHttpException(
        at io.ktor.client.engine.okhttp.OkUtilsKt.access$mapOkHttpExc
        at io.ktor.client.engine.okhttp.OkHttpCallback.onFailure(OkUt
        at okhttp3.internal.connection.RealCall$AsyncCall.run(RealCal
        at java.base/java.util.concurrent.ThreadPoolExecutor.runWorke
        at java.base/java.util.concurrent.ThreadPoolExecutor$Worker.r
        at java.base/java.lang.Thread.run(Thread.java:833)
        Suppressed: kotlinx.coroutines.internal.DiagnosticCoroutineCo
Caused by: java.net.SocketTimeoutException: timeout
        at okhttp3.internal.http2.Http2Stream$StreamTimeout.newTimeou
        at okhttp3.internal.http2.Http2Stream$StreamTimeout.exitAndTh
```

- Sometimes it produced wrong code with double function header.

```
#define MAX_SIZE 100 // You can define a maximum size for the matrices


//Matrix multiplication with multi-threading
void multiplyMatricesThreads(int mat1[][MAX_SIZE], int mat2[][MAX_SIZE], int result[][MAX_SIZE]){ void multiplyMatricesThreads(int mat1[][MAX_SIZE], int mat2[][MAX_SIZE], int result[][MAX_SIZE]){
 int i, j, k;
 for(i = 0; i < MAX_SIZE; i++) {
  for(j = 0; j < MAX_SIZE; j++) {
   result[i][j] = 0;
   for (k=0; k<MAX_SIZE; k++) {
        result[i][j] += mat1[i][k] * mat2[k][j];
       }
     }
   }
 }
}
```