**Student:** Hong Quan Doan

**Student ID:** 986956

**Day 19 – Mar 18**

**Environment**

MacBook, Core i5, 4 CPU, 8GB

**Histogram**

**Test results:**

Image array generating done: 20,000 rows, cols, and max = 10

Sequential time for AtomicInteger with max = 10: 2,524 ms

Parallel time for AtomicInteger with max = 10: 5,556 ms

Speedup: 0.454284

Image array generating done: 20,000 rows, cols, and max = 100

Sequential time for AtomicInteger with max = 100: 2,773 ms

Parallel time for AtomicInteger with max = 100: 2,541 ms

Speedup: 1.091303

Parallel time for ordinary Integer with max = 100 (WRONG values): 525 ms

Speedup: 5.281905

Parallel time for local ordinary Integer with max = 100: 1,890 ms

Speedup: 1.467196

Each of the successive parallel versions takes longer than the sequencial version due to the AtomicInteger item able to pick up the execution and increasing per one, so we need a lock there for reading and writing. The lock is from AtomicInteger itself.

If we don’t use the lock, the value will be read and writing wrongly between threads. To fix this, we should use local variable then add back to the global result.

**Source code:**

**public** **static** **void** runHistogramSequential(**int**[][] image, **int** max) {

*hist* = **new** AtomicInteger[max];

**for** (**int** i = 0; i < max; i++) *hist*[i] = **new** AtomicInteger();

**for** (**int** i = 0; i < *n*; i++)

**for** (**int** j = 0; j < *n*; j++)

*hist*[image[i][j]].getAndIncrement();

**for** (**int** i = 0; i < max; i++)

**if** (*hist*[i].get() != *n* \* *n* / max)

System.***err***.println("Error");

}

**public** **static** **void** runHistogramParallelWithAtomicInteger(**int**[][] image, **int** max)

**throws** InterruptedException, ExecutionException {

*hist* = **new** AtomicInteger[max];

**for** (**int** i = 0; i < max; i++) *hist*[i] = **new** AtomicInteger();

**int** numThreads = Runtime.*getRuntime*().availableProcessors();

ThreadPoolExecutor executor = (ThreadPoolExecutor) Executors.*newFixedThreadPool*(numThreads);

List<Future<Boolean>> listFutures = **new** ArrayList<>();

**int** step = *n* / numThreads;

**int** remainder = *n* % numThreads;

**for** (**int** t = 0; t < numThreads; t++) {

**final** **int** startIndex = t \* step;

**final** **int** endIndex = t \* step + step + ((t == numThreads - 1) ? remainder : 0);

Callable<Boolean> c = () -> {

**for** (**int** i = startIndex; i < endIndex; i++)

**for** (**int** j = 0; j < *n*; j++)

*hist*[image[i][j]].incrementAndGet();

**return** **true**;

};

Future<Boolean> future = executor.submit(c);

listFutures.add(future);

}

executor.shutdown();

executor.awaitTermination(Integer.***MAX\_VALUE***, TimeUnit.***SECONDS***);

**for** (**int** i = 0; i < max; i++)

**if** (*hist*[i].get() != *n* \* *n* / max)

System.***err***.println("Error");

}

**public** **static** **void** runHistogramParallelWithInteger(**int**[][] image, **int** max) **throws** InterruptedException {

**int**[] hist = **new** **int**[max];

**for** (**int** i = 0; i < max; i++)

hist[i] = 0;

**int** numThreads = Runtime.*getRuntime*().availableProcessors();

ThreadPoolExecutor executor = (ThreadPoolExecutor) Executors.*newFixedThreadPool*(numThreads);

**int** step = *n* / numThreads;

**int** remainder = *n* % numThreads;

**for** (**int** t = 0; t < numThreads; t++) {

**final** **int** startIndex = t \* step;

**final** **int** endIndex = t \* step + step + ((t == numThreads - 1) ? remainder : 0);

**final** Integer[] localHist = **new** Integer[max];

**for** (**int** i = 0; i < max; i++) localHist[i] = 0;

Runnable r = () -> {

**for** (**int** i = startIndex; i < endIndex; i++)

**for** (**int** j = 0; j < *n*; j++)

hist[image[i][j]] = hist[image[i][j]] + 1;

};

executor.submit(r);

}

executor.shutdown();

executor.awaitTermination(Integer.***MAX\_VALUE***, TimeUnit.***SECONDS***);

}

**public** **static** **void** runHistogramParallelWithIntegerLocal(**int**[][] image, **int** max) **throws** InterruptedException {

**int** numThreads = Runtime.*getRuntime*().availableProcessors();

ThreadPoolExecutor executor = (ThreadPoolExecutor) Executors.*newFixedThreadPool*(numThreads);

**int** step = *n* / numThreads;

**int** remainder = *n* % numThreads;

List<Integer[]> listInts = **new** ArrayList<>();

**for** (**int** t = 0; t < numThreads; t++) {

**final** **int** startIndex = t \* step;

**final** **int** endIndex = t \* step + step + ((t == numThreads - 1) ? remainder : 0);

**final** Integer[] localHist = **new** Integer[max];

**for** (**int** i = 0; i < max; i++) localHist[i] = 0;

Runnable r = () -> {

**for** (**int** i = startIndex; i < endIndex; i++)

**for** (**int** j = 0; j < *n*; j++)

localHist[image[i][j]] = localHist[image[i][j]] + 1;

};

listInts.add(localHist);

executor.submit(r);

}

executor.shutdown();

executor.awaitTermination(Integer.***MAX\_VALUE***, TimeUnit.***SECONDS***);

**int**[] hist = **new** **int**[max];

**for** (**int** i = 0; i < max; i++) hist[i] = 0;

**for** (Integer[] ints : listInts)

**for** (**int** i = 0; i < max; i++)

hist[i] += ints[i];

**for** (**int** i = 0; i < max; i++)

**if** (hist[i] != *n* \* *n* / max)

System.***err***.println("Error");

}