**Student:** Hong Quan Doan

**Student ID:** 986956

**Day 20, 21 – Mar 19, 20**

**Environment**

MacBook, Core i5, 4 CPU, 8GB

**Jacobi Relaxation project using Stream**

**Test results:**

**With 100 rows x 100 cols array**

Generating ... 100 rows, cols with tolerance = 0.10

Sequential scan with FOR loop time: 8 ms

Regenerating ... 100 rows, cols with tolerance = 0.10

Sequential scan with IntStream time: 30 ms

Regenerating ... 100 rows, cols with tolerance = 0.10

Parallel with FOR loop time: 24 ms

Speedup with FOR loop time (comparing with sequential - FOR): 0.333333 ms

Speedup with FOR loop time (comparing with sequential - IntStream): 1.250000 ms

Regenerating ... 100 rows, cols with tolerance = 0.10

Parallel time with IntStream: 29 ms

Speedup with IntStream loop time (comparing with sequential - FOR): 0.275862 ms

Speedup with IntStream loop time (comparing with sequential - IntStream): 1.034483 ms

Speedup with IntStream loop time (comparing with parallel - FOR): 0.827586 ms

**With 1,000 rows x 1,000 cols array**

Generating ... 1,000 rows, cols with tolerance = 0.10

Sequential scan with FOR loop time: 81 ms

Regenerating ... 1,000 rows, cols with tolerance = 0.10

Sequential scan with IntStream time: 645 ms

Regenerating ... 1,000 rows, cols with tolerance = 0.10

Parallel with FOR loop time: 56 ms

Speedup with FOR loop time (comparing with sequential - FOR): 1.446429 ms

Speedup with FOR loop time (comparing with sequential - IntStream): 11.517857 ms

Regenerating ... 1,000 rows, cols with tolerance = 0.10

Parallel time with IntStream: 293 ms

Speedup with IntStream loop time (comparing with sequential - FOR): 0.276451 ms

Speedup with IntStream loop time (comparing with sequential - IntStream): 2.201365 ms

Speedup with IntStream loop time (comparing with parallel - FOR): 0.191126 ms

**With 10,000 rows x 10,000 cols array**

Generating ... 10,000 rows, cols with tolerance = 0.10

Sequential scan with FOR loop time: 8,316 ms

Regenerating ... 10,000 rows, cols with tolerance = 0.10

Sequential scan with IntStream time: 50,539 ms

Regenerating ... 10,000 rows, cols with tolerance = 0.10

Parallel with FOR loop time: 1,890 ms

Speedup with FOR loop time (comparing with sequential - FOR): 4.400000 ms

Speedup with FOR loop time (comparing with sequential - IntStream): 26.740212 ms

Regenerating ... 10,000 rows, cols with tolerance = 0.10

Parallel time with IntStream: 30,121 ms

Speedup with IntStream loop time (comparing with sequential - FOR): 0.276086 ms

Speedup with IntStream loop time (comparing with sequential - IntStream): 1.677866 ms

Speedup with IntStream loop time (comparing with parallel - FOR): 0.062747 ms

From the above results, we can clearly see for the Jacobi relaxation method for computing the convergence test:

* If we use the FOR loop, the access to each element in the array is instance, because the elements are primitive value. However, when we use the IntStream to loop, the access likely adds overhead to the computation by boxing and unboxing the Integer to int, then process the loop with each 2D array.
* The computation is likely CPU heavily bounded operation, that is another burden for Stream if the operation on each element is for heavy CPU computation.
* FOR loop is long time implemented and through many versions of Java generation, Stream is just recently introduced, hence, that could be another factor that Stream cannot be faster in this case (or Stream is not too optimized yet.)

From the above results, there are few important points to consider when we choose a solution.

* What operation will be for the iteration?
* Would the index, position of each element is important?
* Would there be a heavy CPU computation based on the primitive value?
* Can the array split into binary tree? Or distinction is considered?
* Is the operation stateful?

Hence, Streams are NOT always slower than loops. Sometimes, streams are slower than loops, but they can also be equally fast; it depends on the circumstances.

Another point is sequential and parallel Stream application does depend very much on the conditions and environment too.

**Source code:**

**private** **static** **void** sequentialScanWithStream() {

*isDone* = **false**;

**while** (!*isDone*) {

IntStream.*range*(1, *n* - 1).sequential().forEach((i) -> {

IntStream.*range*(1, *n* - 1).forEach((j) -> {

*B*[i][j] = (**double**) (0.25 \* (*A*[i - 1][j] + *A*[i + 1][j] + *A*[i][j - 1] + *A*[i][j + 1])); }); });

*isDone* = **true**;

IntStream.*range*(1, *n* - 1).boxed().sequential().forEach((i) -> {

IntStream.*range*(1, *n* - 1).boxed().forEach((j) -> {

**if** (Math.*abs*(*B*[i][j] - *A*[i][j]) > *tolerance*)

*isDone* = **false**;

*A*[i][j] = *B*[i][j]; }); });

}

}

**private** **static** **void** parallelScan() {

*isDone* = **false**;

**while** (!*isDone*) {

IntStream.*range*(1, *n* - 1).parallel().forEach((i) -> {

IntStream.*range*(1, *n* - 1).forEach((j) -> {

*B*[i][j] = (**double**) (0.25 \* (*A*[i - 1][j] + *A*[i + 1][j] + *A*[i][j - 1] + *A*[i][j + 1])); }); });

*isDone* = **true**;

IntStream.*range*(1, *n* - 1).boxed().parallel().forEach((i) -> {

IntStream.*range*(1, *n* - 1).boxed().forEach((j) -> {

**if** (Math.*abs*(*B*[i][j] - *A*[i][j]) > *tolerance*)

*isDone* = **false**;

*A*[i][j] = *B*[i][j]; }); });

}

}