**Student:** Hong Quan Doan

**Student ID:** 986956

**Day 09, 10 – Mar 06, 07**

**Environment**

MacBook, Core i5, 4 CPU, 8GB

**Jacobi Relaxation**

Result

Number of row and length of the array: 10,000

Sequential time: 1,127 ms

Parallel time: 311 ms

Speedup: 3.623794

Processor utilization: 0.750000

**Source code:** (for Parallel version)

**public** **class** ParallelScanArray {

**private** **float** A[][];

**private** **float** B[][];

**private** CyclicBarrier globalBarrier, localBarrier;

**public** ParallelScanArray(**float**[][] a, **float**[][] b) {

A = a;

B = b;

}

**public** **void** scan() {

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

**int** step = JacobiRelaxation.*n* / parties;

List<Thread> threads = **new** ArrayList<Thread>();

List<ParallelScanArrayTask> tasks = **new** ArrayList<ParallelScanArrayTask>();

**for** (**int** i = 0; i < parties; i++) {

tasks.add(**new** ParallelScanArrayTask(A, B, (i \* step) + 1, (i \* step) + step));

}

Status s = **new** Status(tasks);

**this**.globalBarrier = **new** CyclicBarrier(parties);

**this**.localBarrier = **new** CyclicBarrier(parties, s);

**while**(!s.doneAll) {

threads.clear();

**for** (**int** i = 0; i < parties; i++) {

Thread t = **new** Thread(tasks.get(i));

t.start();

threads.add(t);

}

**try** {

**for** (**int** i = 0; i < parties; i++) {

threads.get(i).join();

}

} **catch** (InterruptedException ex) {

ex.printStackTrace();

}

}

}

**private** **class** ParallelScanArrayTask **implements** Runnable {

**private** **int** startIndex, endIndex;

**private** **boolean** done;

**private** **float** A[][];

**private** **float** B[][];

**public** ParallelScanArrayTask(**float**[][] a, **float**[][] b, **int** startIndex, **int** endIndex) {

**this**.A = a;

**this**.B = b;

**this**.startIndex = startIndex;

**this**.endIndex = endIndex;

}

@Override

**public** **void** run() {

**this**.done = **false**;

**try** {

**for** (**int** i = startIndex; i < endIndex - 1; i++) {

**for** (**int** j = startIndex; j < endIndex - 1; j++) {

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

}

}

**this**.done = **true**;

globalBarrier.await();

**for** (**int** i = startIndex; i < endIndex - 1; i++) {

**for** (**int** j = startIndex; j < endIndex - 1; j++) {

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

**this**.done = **false**;

}

A[i][j] = B[i][j];

}

}

localBarrier.await();

} **catch** (Exception e) {

**throw** **new** RuntimeException(e);

}

}

**public** **boolean** getDone() {

**return** **this**.done;

}

}

**private** **class** Status **implements** Runnable {

**public** **boolean** doneAll = **false**;

**private** List<ParallelScanArrayTask> tasks;

**public** Status(List<ParallelScanArrayTask> tasks) {

**this**.tasks = tasks;

}

@Override

**public** **void** run() {

doneAll = **true**;

**for** (ParallelScanArrayTask t : tasks) {

doneAll = doneAll && t.getDone();

}

}

}

}