University of Pittsburgh

JavaLife

CS 1699 – Deliverable 4
Performance Testing Conway's Game of Life

John Abraham
Bill Laboon
3/31/2015

TABLE OF CONTENTS

1. SUMMARY:

The purpose of this deliverable is to profile a Conways's Game of Life simulation, and improve its performance by refactoring a single method (to be determined by the results of the profiling). The tool that I used for profiling was a tool called JProfiler. The first figure on the next page shows a screenshot of the JProfiler profiling results of the simulation before any refactoring was done. This was used to determine which method is the most resource-intensive. The command line arguments for the program when I ran it were: 80, 56, 50, and 100000. Looking at the results in the first figure on the next page we can see that, in terms of CPU usage, the top four CPU "hot spots" of the application where the following methods:

java.lang.StringBuilder.append(char) (35%)
java.lang.StringBuilder.append(java.lang.String) (19%)
java.lang.StringBuilding.toString (16%)
com.laboon.World.getNumNeighbors (8%)

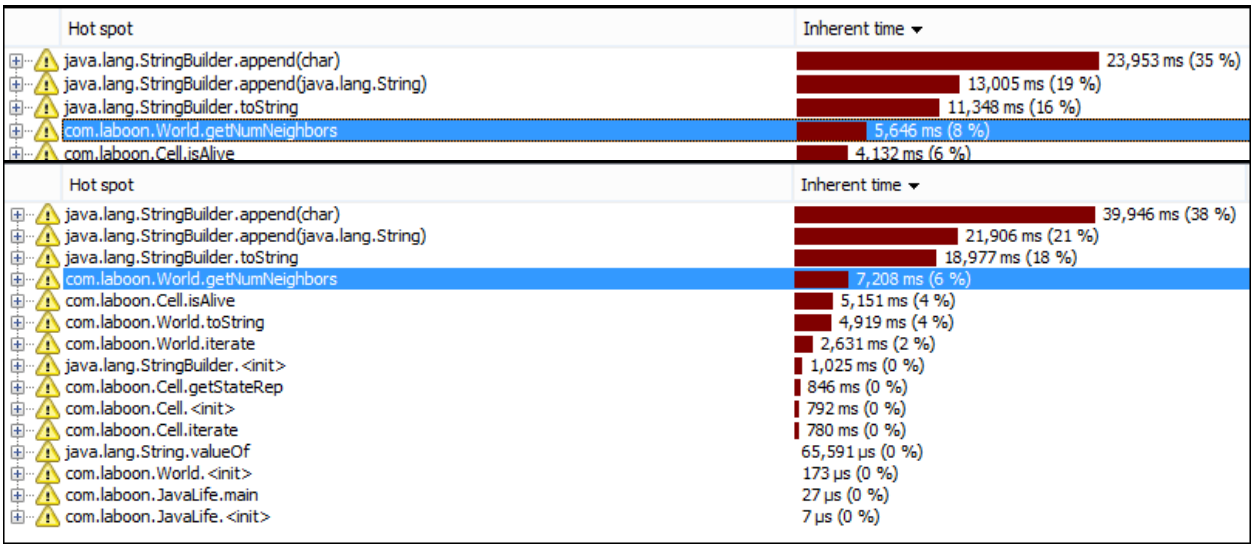
Based on these results the method that I chose to refactor was the getNumNeighbors method in the World class. Although this method is not the biggest hot spot of the application in terms of CPU usage, I chose to refactor it because the other three with higher percentages were from the java StringBuilder class and we're not refactoring built-in java methods. The getNumNeighbors method is the biggest hot spot of the JavaLife application, and what it does is takes in a 2d array of Cell objects, an x position, and a y position and returns an integer representing the number of neighbors of a given cell that are alive. This number of alive neighbors is used to determine whether the cell is in a state of being under-populated or over-crowing. To refactor this method I condensed the code by first removing all lines of code that made use of the modulus operator because the modulus operator finds the remainder of division and that involves dividing numbers which can significantly slow down the program, and it was useful to remove those operators because the getNumNeighbors method gets called for as many times as there are cells in the game's word on each iteration.

The biggest challenge of this assignment wasn't finding what method to refactor because the profiler gives us that information. The biggest challenge was determining how to refactor and optimize the currently given code. The changes I made had a small effect on performance but it was noticeable. The CPU usage of the getNumNeighbors method went down from 8% to 6% using the same command line arguments to set up the word.

Unit tests were also written and executed to ensure that the functionality wasn't changed after refactoring the method to verify that the program still meets the functional-requirements. For this program 5 unit tests were written. They check if the number of neighbors is correct for a simple default 3x3 grid. The unit tests check the middle cell, mid-right, mid-left, mid-top, and mid-bottom cells. The corners are also accounted for but not tested. Also the getNumNeighbors method was changed from private to public for testing purposes.

2. PROFILER BEFORE AND AFTER SCREENSHOTS:

Figure 1: Profiler Results before Refactoring getNumNeighbors:



Hot spot	Inherent time
java.lang.StringBuilder.append(char)	23,953 ms (35 %)
java.lang.StringBuilder.append(java.lang.String)	13,005 ms (19 %)
java.lang.StringBuilder.toString	11,348 ms (16 %)
com.laboon.World.getNumNeighbors	5,646 ms (8 %)
com.laboon.Cell.isAlive	4,132 ms (6 %)

Hot spot	Inherent time
java.lang.StringBuilder.append(char)	39,946 ms (38 %)
java.lang.StringBuilder.append(java.lang.String)	21,906 ms (21 %)
java.lang.StringBuilder.toString	18,977 ms (18 %)
com.laboon.World.getNumNeighbors	7,208 ms (6 %)
com.laboon.Cell.isAlive	5,151 ms (4 %)
com.laboon.World.toString	4,919 ms (4 %)
com.laboon.World.iterate	2,631 ms (2 %)
java.lang.StringBuilder.<init>	1,025 ms (0 %)
com.laboon.Cell.getStateRep	846 ms (0 %)
com.laboon.Cell.<init>	792 ms (0 %)
com.laboon.Cell.iterate	780 ms (0 %)
java.lang.String.valueOf	65,591 µs (0 %)
com.laboon.World.<init>	173 µs (0 %)
com.laboon.JavaLife.main	27 µs (0 %)
com.laboon.JavaLife.<init>	7 µs (0 %)

found in the same repo under WorldTest.java. Also the getNumNeighbors method was changed from private to public for testing purposes.

Screenshots of the executed Junit tests before and after refactoring are on the next page:

Executed JUnit Tests before Refactoring: