COMP 426

Assignment 2

Design Documentation

Christopher Mukherjee

6291929

November 2, 2014

TBB Optimization

In this assignment, I took the code that was developed in Assignment 1 and optimized it by replacing the multithreading achieved using the standard thread library with TBB task parallelization. Instead of explicitly creating threads to manage the parallelization of the cancer simulation, a TBB task scheduler was initialized and a *parallel\_for* generic parallel algorithm was implemented. In the *parallel\_for*, the TBB tasks executed the updating of cell states in parallel.

Architecture

The entirety of the program is contained in one C++ file (.cpp). This file consists of a main function, 5 GLUT-related helper functions, 3 general helper functions to achieve the updating of the cells, and one class with an overloaded parenthesis () operator. The individual functions and class are:

**main**: This function is called when the program is initially run. It represents the main control thread of the application.

**Initialize**: Initialization function for GLUT.

**Keyboard**: GLUT-related function that handles keyboard buttons being pressed.

**MouseClicks**: GLUT-related function that handles mouse buttons being clicked.

**RenderBitmapString**: GLUT-related function that renders bitmap strings to display text on screen

**Display**: Function that displays the cells and text in a window on screen using GLUT

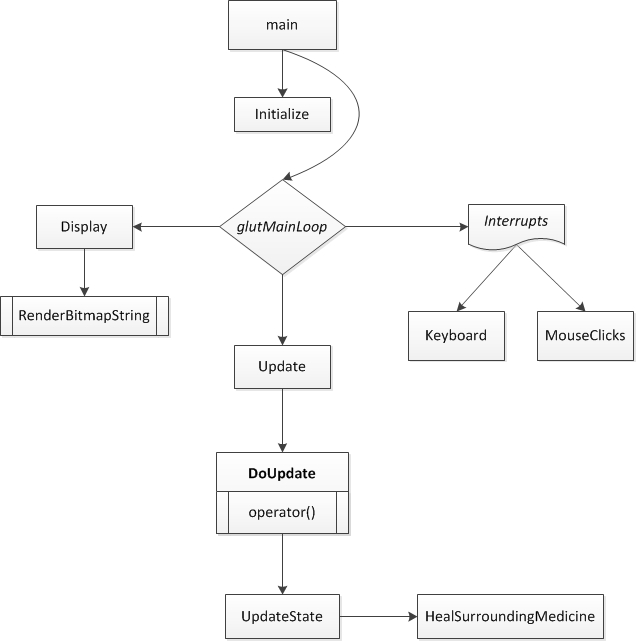
**Update**: Function that uses TBB to initialize a task scheduler and perform a parallel loop. At the end of the function, the function calls itself (to update again).

**UpdateState**: Function that updates each cell state. This function is called by each computational thread, which are generated using TBB).

**HealSurroundingMedicine**: Recursive function that is used to heal all surrounding medicine cells when a cancer cell turns into a healthy cell.

**DoUpdate**: Class with overloaded parenthesis () operator

**operator()**:Overloaded parenthesis () operator, which is used to start the updating of the cell states in the parallel loop.

Control Flow Diagram:

Techniques Used

A TBB *parallel\_for* generic parallel algorithm was implemented in order to parallelize the updating of the cells. This involved using a TBB *blocked\_range2d* and blocking the entire cancer simulation 2-dimensional array. Along with this, I had to create a class and overload its parenthesis () operator. The overloaded parenthesis operator calls the function that is responsible for updating the cells. Finally, I also used the TBB *auto\_partitioner* to heuristically adjust grain size.

The 2-dimensional area of 1024 x 768 cells is represented by a global array. The reason I used an array is because the size of the problem was already known. This means that resizing (which would be a problem for an array) would never be needed. Therefore, all I would need to do is change the values of specific cells, which is easy to do with an array. Also, since the area is a 2-dimensional array, the entire area can be blocked using the TBB *blocked\_range2d* function.

Finally, recursion is used in the “HealSurroundingMedicine” function. This function is used to heal all the surrounding medicine cells when a cancer cell is healed. Starting with the cancer cell that is being healed, the surrounding cells are all checked to see if they are medicine cells. If they are, then they are turned into healthy cells and their surrounding cells are checked. With recursion, I can easily ensure that all the surrounding medicine cells can be healed when a cancer cell is healed.

Test Scenarios

**Test Scenario #1:** Directly healing a single cancer cell

**User Input:** Click on a red cancer cell.

**Expected Result:** Medicine is injected into the cancer cell and is instantly absorbed. The cancer cell is healed and turns green.

**Test Scenario #2:** Injecting medicine into a healthy cell

**User Input:** Click on a green healthy cell.

**Expected Result:** Medicine is injected into the healthy cell and the cell becomes yellow. The medicine is not absorbed and moves radially outwards by one cell position in each direction. Note that all the surrounding cells turn into yellow medicine cells regardless of their previous state.

**Test Scenario #3:** Indirectly healing a single cancer cell

**User Input:** Click on the cells around a single cancer cell. Be sure to click at a distance of two (2) cells away due to the radial expansion of the medicine. It should take three (3) clicks in order for the cancer cell to be sufficiently surrounded by medicine.

**Expected Result:** The cancer cell is cured and turns into a green healthy cell due to the high number of surrounding yellow medicine cells. The surrounding yellow medicine cells also become green healthy cells.

**Test Scenario #4:** Injecting medicine into a healthy cell at the edge of the area

**User Input:** Click on a green healthy cell that is at the edge of the area (either on the left, right, top, or bottom edges, or in a corner).

**Expected Result:** Medicine is injected into the healthy cell and the cell becomes yellow. The medicine is not absorbed and moves radially outwards by one cell position in each possible direction, being sure to not expand outside of the boundaries of the area. Note that all the affected surrounding cells turn into yellow medicine cells regardless of their previous state.

**Test Scenario #5:** A healthy cell turns into a cancer cell

**User Input:** Click on a red cancer cell that is surrounded by more than the majority (≥ 6) red cancer cells.

**Expected Result:** Medicine is injected into the cancer cell and is instantly absorbed. The cancer cell is temporarily healed and turns green. However, the cell quickly turns back into a red cancer cell due to the high number of surrounding red cancer cells.

**Test Scenario #6:** Simulation starts with ≥ 25% cancer cells randomly placed.

**User Input:** Restart the simulation several times (checking expected result each time).

**Expected Result:** The simulation starts with approximately the same amount of cancer cells each time (being ≈25%). The cells appear in different locations each time, due to the fact that they are randomly placed.

Most Important Part of Assignment

The most important part of this assignment is replacing the standard thread library with TBB tasks. Using TBB tasks, we can be sure that the program will use the appropriate number of threads, scaled to the number of cores available. This means that the simulation is now more parallelized and more optimized.

Most Difficult Part of Assignment

Personally, the most difficult part of the assignment was setting up TBB. Even though I though I thought I had set it up as was shown in the tutorial, I still ran into some linker problems. After these problems were solved, I ran into problems having to do with Visual Studio not being able to find necessary DLLs. All problems were eventually solved through online research, but the fact remains that I ran into more problems when setting up TBB than when trying to replace my standard thread library code with TBB code.