COMP 426
Assignment 2
Written Report

Shawn Ostad Abbasi 40191688

Gina Cody School of Engineering and Computer Science

Concordia University

PROBLEM STATEMENT

Intel’s Threading Building Blocks (TBB) is a runtime library that accomplishes multicore programming through the use of tasks. A task is some code that has a goal (e.g. a free function, class functions, lambda functions, etc) which is then mapped to physical threads by TBB. Using this library the processing of the “game of life” can be explicitly implemented using multicore programming rather than C++’s native thread library with the hopes to obtain improved execution.

While certain parts of the “game of life” program require sequential execution it is important to note that other parts would benefit from data parallelism obtained through TBB. Such functions that would benefit from TBB’s approach to data parallelism would be the rule checking within the two dimensional grid of species as well as a color mapping function that maps numbers to a corresponding color. Rule checking for species is accomplished through a two-dimensional array of int8_t (to improve cache locality) which is then later mapped to actual RGB values and stored in a new array meant for OpenGL. Such parts would benefit from TBB due to how the same instruction is performed on different data otherwise known as SIMD behaviour as well as a lack of synchronization in their execution.

PROCEDURE

The general structure of the “game of life” program is as follows. Three two-dimensional arrays are used, the first two store numerical values and act as buffers during computation/rule checking which removes the need for synchronization in the computation. The third array stores RGB values and is passed to OpenGL for rendering. Using the buffers and TBB, the rule checking is performed on the foreground array and results are then stored on the background array which is later used for mapping integers to RGB values. Finally using the array of RGB values the grid of species is then displayed on the screen through OpenGL.

Parallelism often consists of a control thread and a series of computation threads. In this case the computation threads are the physical threads mapped from TBB tasks. In the program there are two different types of tasks. The first one being the logic for the rule checking for each cell in the grid of species. This is accomplished through a class consisting of an overloaded operator function meant for TBB execution, and the logic needed for the rules.The second task that exists in the program is the logic for the mapping of numbers to colors which, similarly to the rule checking, is data parallel in nature due to its lack of synchronization and SIMD structure. Like the rule checking, this task consists of a class definition, operator function overload, and function logic. It is important to note that while no programmer defined computation threads exist, they are created and managed from TBB through user defined tasks.

On the other hand, control threads are used to manage the execution of a program as well as create and schedule tasks. In the "game of life" program the control thread is simply the main function which operates on OpenGL functions, as well as calling the necessary TBB functions to manage data parallelism such as the parallel_for function with the goal of performing data parallelism in for loops. Here the control thread continuously calls TBB functions for creating the computation thread followed by displaying the output of the computation threads through OpenGL.

ANALYSIS

In order to record the performance of the program frames per second (FPS) was measured. Using TBB the FPS averages around 65-80 FPS, I believe that the disparity of FPS may be related to the availability of cores. Compared to the first assignment where FPS averaged

60 but also ranged from 60-75, using TBB shows slight improvement but nothing significant. We can likely attribute the improvement to TBB’s use of multicore programming paradigm, scheduling, and work stealing. Having expected a larger improvement in performance I was quite surprised to see little improvement. As stated earlier the lack of significant improvement may be related to the overhead of TBB, availability of cores, as well as other hardware specifications and may not be a result of differences in code since the structure of the program is relatively the same as the first assignment.

PORTABILITY

Portability of software is important for ensuring that users on different platforms can all benefit from a program. This “game of life” project was initially created for Macs. In order to export the program to Windows a few measures were taken. While none of the source code had to be changed, changes were necessary in the structure of the project as well as the CMakeLists file used for building the project. In terms of program structure, several library, and dynamic linked library files related to TBB and OPenGL were added. While TBB header files, and functions within the source code created no portability issues, the only issue was related to how Mac uses .dylib format and Windows uses a combination of .lib and .dll. On the other hand, in the CMakeLists text file, conditional statements were added to verify the user's operating system in order to include the correct library files needed for execution. While the performance of the “game of life” program is similar on both Mac and Windows, I did notice that Mac on average was running an extra ten to fifteen FPS. While I expected the performance to be better on the Windows machine I was testing on, due to it having a dedicated graphical processing unit (GPU) the slightly enhanced performance on the Mac may be related to several things. The first one being that most of the computation is on the CPU using TBB, therefore leaving only simple

OpenGL functions and computations for the GPU. This being said, having a dedicated GPU may not be as influential on the execution. Secondly, since most of the computation is done on the CPU through TBB, the number of cores and processing power of the CPU is important. In this case we see that the program ran slightly slower on the Windows machine than on the Mac, perhaps due to the Windows machine having two fewer cores than the Mac. Finally, the last reason may be related to the overhead and limitations of using TBB.