

Parallel and Distributed Computing

1º Semester – Academic Year 2013/2014

Stefan Schneider, 79791 - [stefan.schneider@ist.utl.pt](mailto:stefan.schneider@ist.utl.pt)  
Kuganesan Srijeyanthan, 79531 - [kuganesan.srijeyanthan@ist.utl.pt](mailto:kuganesan.srijeyanthan@ist.utl.pt)   
Gayana Ranganatha Chandrasekara Pilana Withanage, 79529 - [gayana.withanage@ist.utl.pt](mailto:gayana.withanage@ist.utl.pt)  
Group 40 - Alameda

# Introduction

This report is presented in order to elaborate and describe the serial and parallel programing aspects followed during the implementation in order to optimize the flow of the program. The quantitative results and comparisons are also included. The objective of this project is to implement the given problem with a program which executes serially and convert that program into a parallel one with OpenMP such that we get the same final results with optimized time efficiency. When you follow the report you will encounter section 1 with serial programing methodology followed by parallel programing methodology in section 2. Section 3 will give the results and comparisons. Finally the section 4, contains the conclusion with result justification.

# Section 1: Serial Programing Methodology

There are two ways of proceeding the program execution are identified, and below Figure 1 and Figure 2 describes them. For our implementation we selected the method in Figure 2 and the reason for the selection will be described in the next Section.

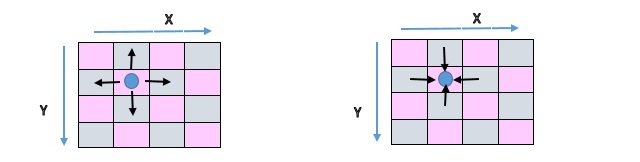


Figure 2: The processing cell (Red) checks adjacent cells and decide which of adjacent cells can come on to it. (Assume red cell is an empty cell) Processing occur to the X direction.

Figure 1: The processing cell (Red) checks adjacent cells and decide where it can moves to any of it. (Assume red cell contains a Wolf or a Squirrel) Processing occur to the X direction.

# Section 2: Parallel Programing Methodology

In this case we are keeping two maps in the memory, where first map is the source map which is initialized with the values given in the input file, while the second map is a temporary copy of the same data structure but it initially empty.

When we process the source map according to the way shown in Figure 2, the updates are made in the temporary copy in a certain sub generation (i.e.: Red sub generation). Once a sub generation is completed the temporary map changes will be populated to the source map and temporary map will be cleaned again. Likewise the program executes for all the generations and gives the final result.

## Methodology Selection Description

1. Parallel execution in method described in Figure 1.

In the parallel execution, T1 (Thread 1) and T2 (Thread 2) both sees a movable position in the source map and tries to update the temporary map creating a raise condition. The whole sub generation is parallelized so that there is a high probability of getting many thread conflicts which will decline the performance of the parallelized version.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **T1** |  | **T2** |
|  |  |  |  |
|  |  |  |  |

Figure 3

1. Parallel execution in method described in Figure 2.

In this method, we only update the current processing cell location in the temporary map so that in a sub generation there is no raise condition unless there is a breading occurs which is a rare conflict comparing to the earlier scenario. Therefore this method makes the parallel execution much efficient.

## Load balancing

# Section 3: Results

The Figure 4 shows the execution time against the number of generations in a map of N size, when executed by a serial version of the program. Meanwhile the Figure 5 indicates the results we obtained when executed the parallel version of the program over the same map of N size.

Figure 5: Parallel execution with 4 threads

Figure 4: Serial execution with 1 thread

# Section 4: Conclusion