MONASH University

FIT3143 Lab Week 4

OPENMP

OBJECTIVES

- Design and develop parallel algorithms for various parallel computing architectures
- Analyse and evaluate the performance of parallel algorithms

INSTRUCTIONS

- Download and set up software applications used in this unit [Refer to Lab Week 1]
- Setup eFolio (including Git) and share with tutor and partner [Refer to Lab Week 1]

TASKS

DESCRIPTION:

- Parallel computing practice using OpenMP in C/C++.
- Parallel algorithm design and development

WHAT TO SUBMIT:

1. E-folio document containing algorithm or code description, analysis of results, screenshot of the running programs and git repository URL. E-folio template for this lab can be found in Week 04 of Moodle.
2. Code and supporting files in the Git.
3. This is an assessed lab. Therefore, you are required to submit both the E-folio document, code file(s) and text files into Moodle. Submission link is available in Week 04 of Moodle. Each student makes a submission. Although you are working in a team of two (or three) members and your submitted files will be the same within a team, each team member is required to make a submission independently in Moodle.

MONASH University

EVALUATION CRITERIA

This Lab-work is part of grading, with 10 maximum marks, which is then scaled to 2 percentage points of the overall unit marks.

	Code compiles without errors and executed correctly (2 marks)	Sufficient code comments (2 marks)	Questions or instructions fully answered (4 marks)	Proper tabulation of results and analysis (2 marks)
Activity 1	1	1	1	1
Activity 2	1	1	3	1

LAB ACTIVITIES (10 MARKS)

The aim of this task is to develop a gaming machine using OpenMP as a workshare programming model. The machine has a dynamic digit display. This means that the machine can produce a display sequence of 10 numbers, 100 numbers, 1000 numbers, 1,000,000 numbers, etc. The size of the display is determined by the user during runtime.

The following rule applies to register a win with the machine.

- At least two of the displayed digits must represent the same value to register a win.
- The machine may generate more than one win after a play. It will depend on the number of (different) repeated digits appearing on the display, as illustrated in Figure 1.

Figure 1 – An example of a 10-digit display. Number '5' appears thrice (a win), number '1' appears twice (a win), and number '7' appears twice as well (a win). The total number of wins counted by the machine in this case will be 3.

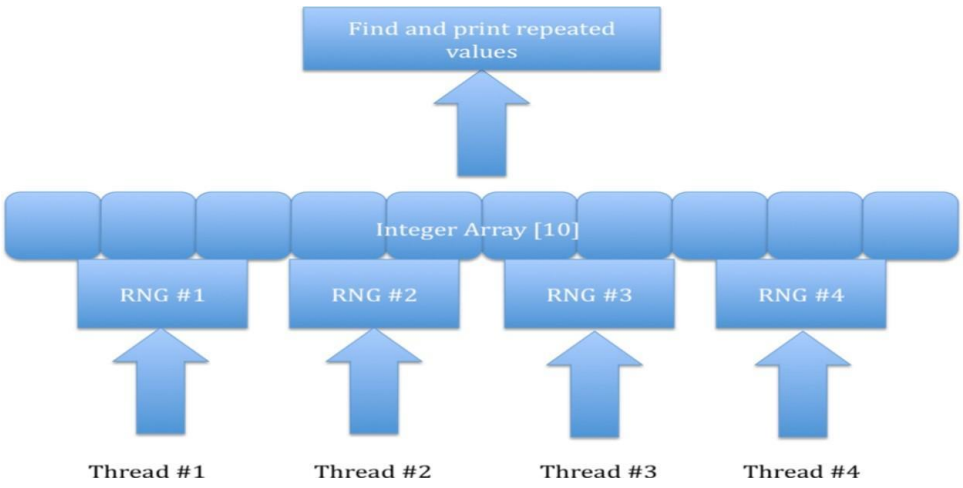

Figure 2 – In this example, the code creates four concurrent threads to workshare for a sample display size of 10. Each thread independently generates a random number over the prescribed range (or chunk size). The threads work collaboratively and in parallel to populate the shared integer array. The value of the prescribed range or chunk size varies based on the number of threads, p, and the size of the display.

The machine is to be controlled by a parallel software code using OpenMP. Complete the following activities.

Activity-1:

- The code forks *p* number of threads. The value of *p* is determined by the user at runtime.
- Each thread executes an independent pseudorandom number generator function that produces single digit random numbers. Each random number represents a value of the display digit.
- The threads work on a shared heap array to fill each of the array elements with a random number. The size of this array is based on the size of the display which is determined by the user during runtime.
- The threads join back to the master thread (i.e., main process). The main process writes the content of the heap array into a text file (i.e., *display.txt*) and clears the heap array.
- Use the appropriate OpenMP constructs for this activity. Demonstrate the use of the schedule clause. In your analysis, compare the time required to generate values for very large displays (e.g., display size = 10,000,000) when using different schedule clauses (i.e., static, and dynamic, with and without chunk-sizes). Observe and explain the measured time performance. You may use the appropriate OpenMP API functions to set the number of threads (i.e., *p*) at runtime.

Activity-2:

- The content of *display.txt* file is checked for matching entries. The code forks out a new set of *p* number of threads to check the matching entries in parallel.
- A win is counted if two or more occurrences of a number are found in the array. The total number of such occurrences is written to another text file (i.e., *wins.txt*).
- Use the appropriate OpenMP constructs for this activity. Demonstrate the use of the schedule, reduction, and any other clauses to help you in this activity. Once more, compare the time required to count the wins for very large display arrays (e.g., display array size = 10,000,000) when using different schedule clauses (i.e., static, and dynamic, with and without chunk-sizes). Observe and explain the measured time performance.

Note: Although both Activities 1 and 2 have their own tasks, a single application (or executable) is produced as a result of this lab work. You may specify an appropriate format to write the content of the display array into *display.txt*, and the number of wins into *wins.txt*. Please ensure that you submit the text files along with the code and E-folio document in Moodle.