Laboratory Exercise 1

Shane House (749524), Benjamin Rosen (858324)

ELEN4020, 23 February 2018

I. OpenMP

When the sample code was run, it was found that 4 "hello world"s were printed. OpenMP runs the code in a number of parallel processes, which defaults to the number of threads capable by the machine. The code was run on a 2 core machine, with each core capable of 2 threads, which is why it printed 4 times.

II. Lab

The N-dimensional array was constructed using one single array. This is because a multi-dimensional array with dynamic dimension cannot be passed to a function. In addition, the computer stores a multi-dimensional array in contiguous memory, so it is effectively a 1D array.

First, a helper function was declared, which calculated the total length of the 1D array by multiplying the dimensions together.

For the first function, a simple for loop was used that runs through the entire 1D array and sets all values to 0.

The second function multiplies the array length by 0.1 (i.e. 10%), and uses that value as an increment in the for loop. The for loop sets each value that it encounters to 1. This ensures that there is a uniform distribution of 1s throughout the array.

The final function multiplies the array length by 0.05 (5%) to obtain the number of elements to check. For each iteration, a new random position in the linear (1D) array is selected and its value printed. The actual position in the multi-dimensional array is calculated using Algorithm **??**.

A for loop prints out the multi-dimensional position to console. The reason we worked backwards, by generating a random number and then working out the N-dimensional position, was to ensure that the distribution would be uniform for the **entire** array. For the other method, where a random number is determined for each dimension and the linear position calculated after, the distribution will only occur over each dimension, but not necessarily the entire N-dimensional array.

Algorithm 1 Method to Calculate Multi-Dimensional Position from Linear Array Position

```
Begin Algorithm:
  linearPos = linear array position
  for each dimension, working backwards, do:
    dimensionLength = length of dimension
    position for multi-dimensional array = linearPos mod dimensionLength
    linearPos /= dimensionLength
  end for
End Algorithm
```