Jarad Humphry

CSC2002S – 2016

Prac\_1: Scientific report investigating the advantages of parallel programming

Introduction

The aim of the project is to investigate parallel programming and compare it to sequential programming. A case study was performed using quick sort and merge sort using a parallel algorithm while benchmarking them against the java Arrays.sort() method.

Method

The mergesort algorithm employed splits the array into equal smaller arrays and invokes two new parallel processes each time. Once the lists are small enough\* they are sorted (using sequential methods). These arrays are then 'merged' by comparing their smallest (or first item) and adding it to a larger array. This recursive processes is repeated until the final array is sorted.

The quicksort algorithm uses a pivot value (in this case the first item in the array is used) and places it above all items that are less than it and below all items greater than it. The array is then split with respect to the pivot value and two new processes are invoked. This process is repeated until the sub arrays are small enough\* and they are sorted sequentially.

These sorting methods were tested using a Junit test while comparing an array sorted using the parallel methods to an array sorted by Arrays.sort() method.

The test was completed for multiple sizes of arrays. In each case the array is comprised of random numbers (selected by the user). For each array size the code runs through different numbers of threads (from 2 to 10000). For each of the number of threads, five different tests are run and the best one is chosen. It then compares the speedup for each of the number of threads and choses the value that is that is highest to add to the list that is printed out.

Since parallel algorithms utilise the computers cores while serial ones only use one. The expected speedup (speed of parallel/speed of serial) for both quicksort and mergesort are dependent on the number of core the computer it is being tested on has. With this in mind, the test was completed on [however many machines]. This allows the results to be compared across different architectures. With the expected speedup being equal to the number of cores (in ideal conditions).

Results

The expected trend for small array sizes is a speedup below 1. This is because the program will spend more time allocating threads than solving the problem, resulting in serial sort being quicker. As the array size increases so too will the speedup due to the size of the array increasing. The speedup will eventually peak and plateau.

Three tests were completed for each sorting algorithm on each architecture, each test using a different range of array sizes and increments. The first test uses the range of 100 to 10 000 with an increment of 100. This is to show the trend of speedup increasing at low array sizes. The second test was completed on a range of 10 000 to 1000 000 with an increment of 10 000. The third test was completed with a range of 1 000 000 to 10 000 000 with an increment of 1 000 000. The data is plotted onto a logarithmic x-axis to accommodate for this large range with highly variable resolution.