Experiment 1

Sinhgad Institutes | Date 1 1 | 1.

Title: Design and implement parallel Breadth first search and depth first search.

Problem Statement: Design and implement Parallel Breadth First search and Depth First search based on existing algorithms using OpenMP. Use a Tree or an Undirected graph for BFS and DFS.

Prerequisite: 64-bit open source Linux or its derivative

Programming Tools: Java / Perl / PHP / Python / Ruby / .net, MongoDB / MYSQL / oracle

Objectives: To offload parallel computations to the graphics card, when it is appropriate to do so, and to give some idea of how to think about code running in the massively parallel environment presented by today's graphic card.

Outcome: students should understand the basic of OpenMP Programming Module using existing techniques.

Theory:

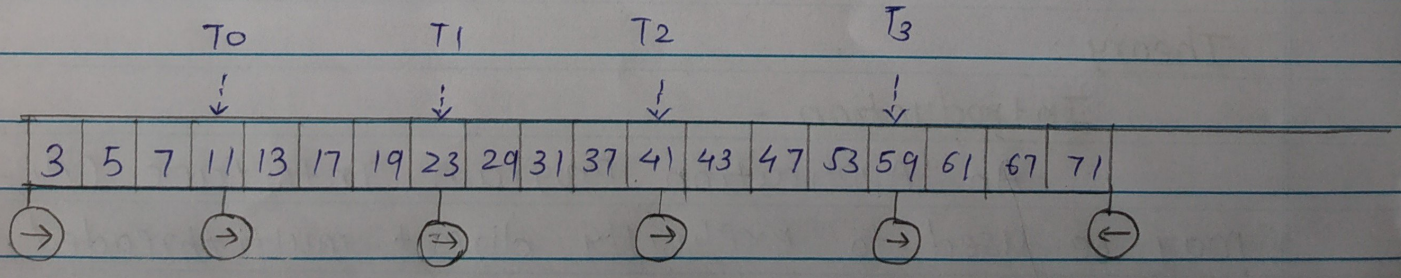
Introduction -

An Application Program Interface (API) that may be used to explicitly direct multi threaded, shared memory parallelism. An open Multi-Processing (OpenMP) consists of a set of compiler #pragmas that control how the program works.

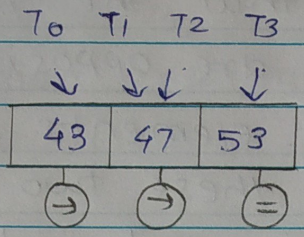
The pragmas are designed so that even if compiler does not support them, the program will still yield correct behavior, but without any Parallelism. searching identifies N well-spaced Points within the search array bounds and compares the key of the corresponding records to the search key. Each thread does one of the N comparisons.

There are three possible outcomes from these comparisons. The first is that the item of interest is found and the search is complete; the second is that the item key examined is less than the search key; the third is that the item key examined is greater than the search key. If no search key match is found, a new, smaller search array is defined by two consecutive index points whose record keys were found to be less than the search key and greater than search key.

The N-ary search is then performed on this refined search array.



(a)



(b)

Given the sorted array of prime numbers in fig. (a), let's say we want to determine whether the value 53 in the array and where it can be found. If there are four threads (T0 to T3), each computes an index into the array and compares the key value found there to the search key.

Threads T0, T1 and T2 all find that the key value at the examined position is less than the search key value. Thus, an item with the matching key value must lie somewhere to the right of each of these thread's current search positions. Thread T3 determines that the examined key value is greater that the search key and the matching key can be found to the left of this thread's search position.

Notice the circled arrows at each end of the array. These are attached to "phantom" elements just outside the array bounds. The results of the individual key tests define the subarray that is to become the new search array. Where we find two consecutive test results with opposite outcomes the corresponding indexes will be just outside of lower and upper bounds of new search array.

Fig (a) shows that the test results from threads T2 and T3 are opposite. The new search array is the array elements between the elements tested by these two threads. Fig (b) shows this sub array and index positions that are tested by each thread. The fig. shows that during this second test of element key values, thread T3 has found the element that matches the search key.

Consider the case, where find a composite value, like 52, in a list of prime numbers. The individual key results by four threads would be the same. The sub array shown in fig (b) would have the same results, except that the test by T3 would find that the key value in the assigned position was greater than the search key.

The next round of key comparison by threads would be from a sub array with no elements, bounded by the array slots holding the key values of 47 and 53. From quick description, it is clear that it needs some globally accessible data and more importantly, it needs a barrier between completion of key and examination of the result of those tests.

Applications :

1. Information retrieval in the required format is the central activity in all computer applications.

2. Searching methods are designed to take advantage of file organization and optimize the search.

Conclusion :

We have tested both the searching algorithm for different number elements with respective to time. As the number of elements increased time required for openMP is reduced.