Benchmark

The technique that we are working on needs to test such problems that use the standard divide and conquer technique. The problems are suppose to be simple problems that use the solvng techniques of the divide and conquer algorithm. This methodology woul give a chnace to allot the cpu cores is an efficient manner. Not every divide and conquer problem can be used for this purpose. The ones that we are looking for should be dividing the problem solution in a random manner. Some divide and conquer problems divide the problem in an equal manner. Like the merge sort. The array that is porvided to be solved is divided into two equal halves, and then this procedure is continued untilla very small problem length is achieved. Then the problem is solved in a recursive manner and the solutions is merged into one another. These kind of problems are not useful for our implmentations.

The porblems which gets divided in a haphazard manner are the ones which do not consider load balancing. Eac divided part of the problem is allotted to a core that runs on a default frequency. Due to thios haphazard nature the length of the generated problems are not the same. Some gets the major part of the original problem and others gets only a small part. For instance, while solving quicksort using divide and conquer, the position of the pivot decides the place where the division of the array is to be performed. Most of the quicksort implementations pick the pivot among the numbers in the array in a random manner. This results in the array getting divided by the postion in which the pivot is placed. As the pivot is picked randomly is the begining, the array ends up getting divided in a random place. This random place can be anywhere in the whole array. So the array is divided in unequal parts.

In a parallel environment, the divide and conquer algorithm has a way of getting implemented. As the porblem statement is divided into small parts and solved seperately. In a parallel environment the problem portions is alloted to the independent cores and they are solved parallely. As the cores finish processing and executing on the part of the porblem statement that was given to it, the solution is then returned to the main core. The main core gathers all the parts of the solution from the other cores and them merges them together to get the total solution. Here is where the porblem lies, as the main core has to wait for the part of the solution to return, it is almost always a waste of time. The solutions allotted are not of the same length, as a result some of the cores witha small solution fnishes up solving the problem and waits ofr the otehr cores. If the otehr coresw are solving a part that has a very large length, then this time is very high. This happens cause all the cores are running in the same frequency and the problem is divided in an unequal manner.

So we pick up these kind of problems where the solutions are divided in an unequal manner. The problems that we have chosen to run for our proposed approach are Quicksort, Tree Traversal and Prime Number Calculation.

Quicksort

Quicksort is one of the sorting algorithms that uses divide and conquer. We select this problem because it shows the exact problems that we have discussed. The pivot is also chosen randomly so that the problem statement gets divided in an unequal manner. We use the recursive version of the quicksort algorithm to solve the problem.

We first initialize an array of 10000 numbers. Each number is generated randomly and is placed in the array. Chosing the numbers randomly disperses it evenly all over the array. This problem is then sent to a function that solves it recursively.

Tree traversal

Tree traversal is a problem that goes through all the nodes of a given tree, either to process it or just display it. The tree is divided into two parts, the right half and the left half. The tree is divided in the same manner untill all the cores are satisfied. Then the cores work upon there problems and process it to the end. We pick this problem because the tree is not a abalanced one. It may so happen that the right side has too many nodes and the left one has too few. This would result in an imbalance among the cores. This is where our approach woulf work.

We are taking a level 10 tree, and initializing in a random manner. At every leaf node a random number would decide if that node would have more children or would end up there. This results in a tree that has random number of children on both the sides. This random way of generating the tree makes sure that the tree is not a balnced one. This property is very essential for the approach that we are implementing.

Prime Number Calculation

This problem takes a limit that is usually very large. And then calculates all the prime nunber within that limit. Essentially this is not a divide and conquer porblem but we are solving it in a way such that it can be used in an divide and conquer algorithm. We are working on a array set of 10000 numbers. These numbers are saved in an array and hen iterated upon. These numbers are checked one by one, and of they are prime it is displayed on the screnn.

To implement this problem in a divide and conquer environment, we first divide the numbers into 4 sub-arrays. These sub-arrays are then allotted into each cores. The cores then start working on them. In the ned, each cores display there solutions one by one, thus completing the solution.

All these benchmark problems are used to test the porpesed approach. The benchmark is run in a normal environment, with a base default frequency. It is then run in a custon set of frequencies and the difference in running time is noted.

The idea of this appproach is that the core which has the smaller problem part is given a lesser frequency. And the core with the bigger part in given a larger frequency. This results in the balance of time taken by the cores to execute and compute the solutions. As the compution is executing parallely, we consider the largest time taken by the core as the final time. Our results will show the improvements achjieved.

Methodology

The approach that we are dealing with is a parallel implmentaion of an algorithm that is related to green computing. The idea is to use the problem size and decide the frequency that the cpu core will be running at. This is called dynamic voltage and frequency scaling. It is dynamic because the frequency is decided during the runtime, when the problem is allotted to the cpu. There is always a master core (core 1) which decides the allotment of the problem parts to the other cores. This cpu only calculates the size of the porblem part and then decides which frequency the other processot should run at.

The methodology starts with the selecting a baseline frequency. This frequency will be used to compare the execution time of all the problem parts. We choose a set of 5 frequencies which we will use to set to the cores. At the generation of every porblem sub-part, the running time for the each nad every frequency will be checked. Thus check is done to make sure that the best frequency is selected among all the freuqencies. The result of the cehck is the best frequency, and at the end the problem sub-part is allotted ot a core with the frequency as the resultant frequency.

The problems that we hvae chosen are recursive. This means that there isnt just one or two sub-problems that are porduced. Instead, at every iteration, two sub-problems are produced. For a big problem statement, the number of total sub-problems that would e porduced is very large.But we acnnot allot so many subproblems to different processor cores. Therefore, we keep alloting the sub-problems and the resultant frequency (for every specific sub-problem) to the cores untill they are available. Once all the cores have been allotted a sub-problem, this process stops. Then the cores work independtly to solve their sub-problems and return back the solution.

For our implementation, we have selected 4 number of cores. The master core is the core 1, which begins the execution and allots the first sub-problem to other core. The other core in turn allots its sub-problem to another one. This continues untill all the 4 cores are busy working on their own sub-problem. After the execution and processing is complete, all the cores return the solution to the master core, which does the last processing step and displays the results.

To test the approach that we have implmented, we start with a practical approach. We run the porblem that we have in the default environment without the dynamic voltage and frequency scaling. This gives the execution time that it would take for the porblem to run in the parallel environment, but without the porposed approach. Then we apply our mehtodolgy and constructed algorithm. The execution time using this approach is also calculated. In the end, when both the solutions have been processed, we display the excution time using both the approaches. The results acquired are discussed in the result and anlysis section. The difference in execution time is clearly visible.

Implementation

The algorithm starts by calculating the balanced execution time for each of the 3 benchmarks that we have selected. This execution time is calculated if the problem was to run at the max frequency that the processor has to offer. This balanced execution time is used later to decide the frequency of the other cores to which the sub-problems will be allotted. There is also a default frequency on whichthe processor runs. This is the frequency on which the porcessor runs if there is no dynamic voltage and frequency scaling. This is the frequency on which the processors would run if the problem is allotted to them without any implementation of the proposed approach. This default frequency is used to calculate the reduction of execution time that we have achieved while using dynamic voltage and frequency scaling.

The program starts with displaying the choice to the user. The choice is among the benchmarks that we have selected. Each benchmark is tested and solved without the implementation of the dynamic voltage and frequency scaling. After the execution is completed, the running time for this approach is displayed. After this the same algorithm and the data is used to execute the problem is the implemented environment. This uses the dynamic voltage and frequency scaling concept along with the algorithm designed to allot frequency to the cores based on the size of the sub-problem. After the execution is over with this proposed approach, the execution time is displayed then.

The program uses the concept of recursion mostly to solve the problems. It keeps tracks of the number of units that the sub-problem has, and to which core each sub-problem is allotted to. This information is then used to allot the frequency for the cores. This data is always present for the master cpu core to access and process. Four variables keep track of the information regarding these values. These values are very essential as they are used to calculate the execution time for the cores.