MLMITU002

CSC2002S

Parallel programming assignment

Introduction

Simulating vegetation patterns in botany using computerized models can take an excessive amount of time due to the amount of input terrain which can span entire continents. Therefore this project aims to use parallel programming to accelerate the computing efficiency of such algorithms by maximizing CPU core usage on machines. This is to investigate also the difference in time taken to run such algorithms on single core machines using sequential programming as compared to running them with parallel programming on multiple core machines.

This also investigates the optimal sequential cutoff based on input size and machine architecture to have efficient and best time complexity for the algorithm. Sequential cutoff is investigated as it plays a huge role in parallel programming because it allows to find the optimal number of threads operating at an instance which can also vary with different computer architectures and affects program execution time excessively.

Method

The sequential program as well as the parallel program are implemented to investigate the difference in time taken by the two algorithms on different computer architectures. In the sequential program the main thread does all the work and it calls a “Sum” method that sums up the sun exposure of all the trees in the terrain. The parallel program is implemented using the Java Fork/Join framework   which uses light weight threads .The main class extends the “RecursiveAction” class and overrides the “compute” method of the class.

The sun exposure calculation algorithm is implemented in both a serial and a parallel program to compare performance. The “currentTimeMillis()” method from System class is used to precisely time the sun exposure calculation code . The parallel and the sequential program are tested on a quad- core computer architecture with a significantly large data set of 1 million trees to prove speed up. The programs are furthermore then tested on a dual core computer architecture to investigate the capacity of the algorithm to effectively utilize an increasing number of cores. The work law which states that the time it takes to do work on “P” number of processors times the number of processors is equal to the time it takes 1 processor as shown in figure1 is used to theoretically gauge the ideal time it will take to run a parallel program compared to sequential.

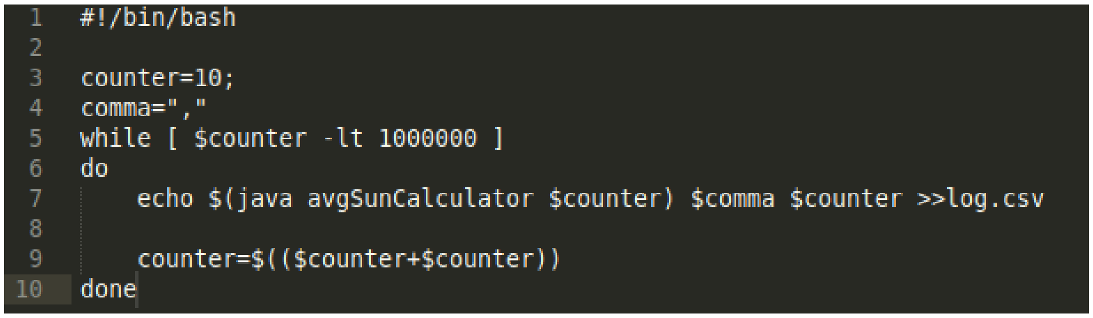
● Work Law: $T_P >= T_1 / P$

Figure 1

Therefore both the algorithms are tested on a significantly large data set and the expected speedup is more than 4 times or equal to the time it will take a sequential algorithm theoretically because they are run on quad core machine.

Both the serial and parallel algorithm are tested with different input sizes using a bash script to compare the performance of both the programs   with change in input size. The input size varied is the amount of trees that have to be summed up for the total exposure. This is to also investigate how the input size impacts and is related to the sequential cutoff.

The parallel algorithm is also run with different sequential cutoffs to find the optimal number of threads for speed up. This testing operation was executed using a bash script, the algorithm in the bash script runs the parallel algorithm with a sequential cutoff that varies from the minimum amount of input to a significantly large input size as shown in figure2.



```
#!/bin/bash

counter=10;
comma=","
while [ $counter -lt 1000000 ]
do
    echo $(java avgSunCalculator $counter) $comma $counter >>log.csv

    counter=$(($counter+$counter))
done
```

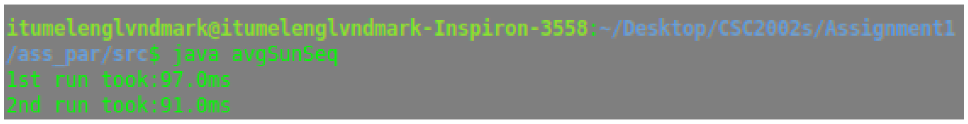
Figure 2

The output is then saved to a csv file to allow analysis to find limits for the sequential threshold. Some of the challenges that occur during testing is that the parallel algorithm does not provide speed up on first run which has to do with how the task scheduler works but on the second run it provides the ideal speed up. The sequential portion of the overall algorithm in the parallel program which is involved in reading the text file is also timed to see the effect it has on the overall program.

Results and Discussion

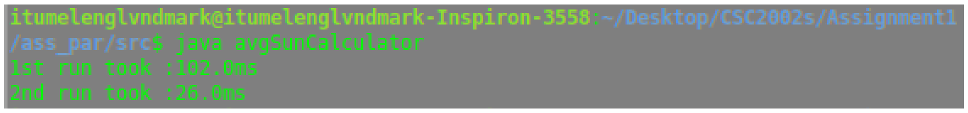
Parallelism

The parallel and sequential algorithms for the problem were implemented and executed and results in figure 3 and figure 4 Were obtained to demonstrate speed up. On the second run the parallel implementation takes almost 4 times the time that the sequential algorithm would take which is the expected speed up according to the work law using a machine with 4 cores.



```
itumelenglvndmark@itumelenglvndmark-Inspiron-3558:~/Desktop/CSC2002s/Assignment1
/ass_par/src$ java avgSunSeq
1st run took:97.0ms
2nd run took:91.0ms
```

Figure 3



```
itumelenglvndmark@itumelenglvndmark-Inspiron-3558:~/Desktop/CSC2002s/Assignment1
/ass_par/src$ java avgSunCalculator
1st run took :102.0ms
2nd run took :26.0ms
```

Figure 4

$$T_P \geq \frac{T_1}{P}$$

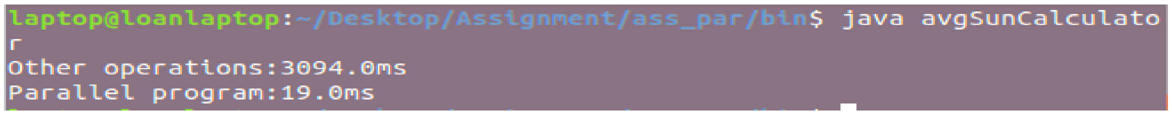
Figure 5:work law

Using the work law equation in figure 5 the expected time it should take the parallel algorithm ideally on a quad core machine is 4 times the time it takes on a sequential algorithm. The sequential algorithm yields 91.0milliseconds while the parallel algorithm yields 26.0 milliseconds. According to figure 6 the speed up is 3 and a half times the time a sequential program would take which is almost equal to the ideal value.

$$speed\ up = \frac{91.0}{26.0}=3.5$$

Figure 6

The sequential part of the program which is involved in reading the text file and creating an array of the terrain amounts to most of the time consumed by the algorithm as shown in figure 7 .



```
laptop@loanlaptop:~/Desktop/Assignment/ass_par/bin$ java avgSunCalculato
r
Other operations:3094.0ms
Parallel program:19.0ms
```

Figure 7

It is evident that using parallel programming approach to solve this problem only reduces time of execution for calculation operations while most of the time is used by the program reading the text file and creating arrays.

Optimal sequential cutoff for the problem

The results were obtained using a multi core machine with different sizes of dataset. Sequential cutoff may vary between problem sizes because it depends on the type of computer architecture and input size. The sequential cutoff investigated in figure 8 is based on a quad core computer and an input size of 1million tree objects .

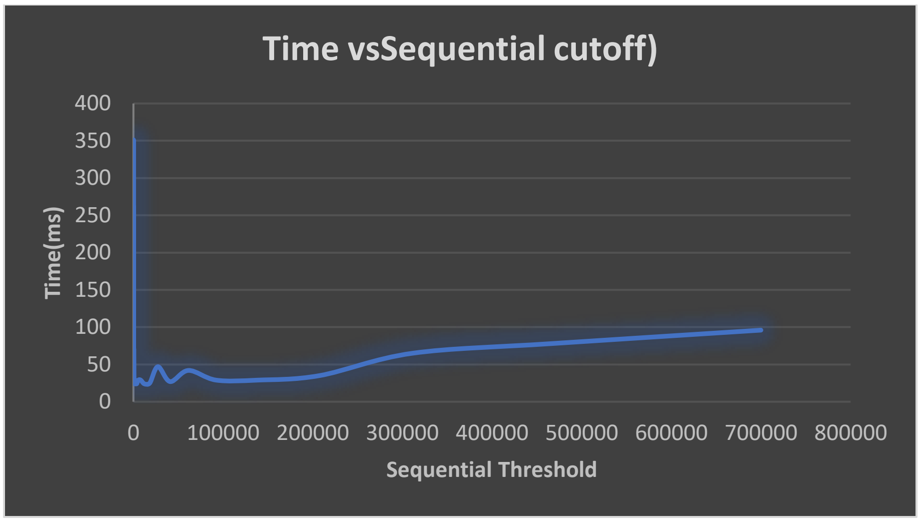


figure 8

From figure 8 it can be deduced that an optimal threshold occurs between the range of 0 – 200 thousand which is approximately 0-20 percent of the input size. Figure 8 shows a more precise values of the time taken by the program along the range of sequential cutoffs.

141	41
211	37
316	63
474	71
711	25
1066	30
1599	24
2398	26

figure 9

The lowest amount of time taken is 24 milliseconds from the test with a sequential cutoff of 1599 which is 0.2 percent of the input size. Therefore the optimal number of threads for a program of this input size is approximately 625 as shown in figure 10.

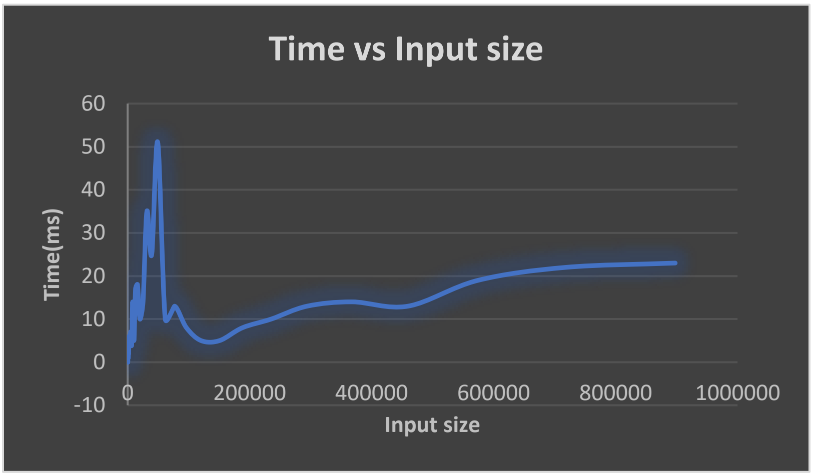
$$\frac{1000000}{1599} = 625.4$$

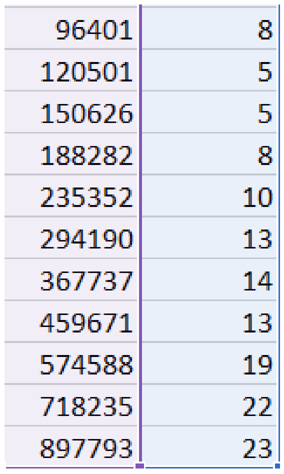
figure 10

Therefore the maximum speedup obtainable with this parallel approach is 3.9 times the time taken by a sequential algorithm on a quad-core computer.

Parallelism with growth of problem size

Theoretically an increase in input size with no change in number of cores must result in slower execution of the program to solve the problem. It is expected that with increase in data size while keeping the number of cores constant will result in more time taken to do parallel operations. Figure 11 illustrates how the time of execution grows with the input size.





96401	8
120501	5
150626	5
188282	8
235352	10
294190	13
367737	14
459671	13
574588	19
718235	22
897793	23

figure 11

It is evident from the above figure that the relationship between time of execution and input size is not linear and there are inconsistencies for small input size. Parallel programs are non-deterministic, implying that they exhibits different behaviors on different runs therefore the time calculations are not as precise as they should be. For small input sizes there are discrepancies because when the input size is less than the sequential cutoff, the program runs sequentially thus it takes a significantly large amount of time as compared to parallel algorithms. However from input size from 200 thousand there is expected results because the data size is above the sequential cutoff, thus it runs on multiple threads.

Scalability

The figure 12 demonstrates the behavior of the parallel algorithm on 2 different computer architectures. It is very crucial for parallel algorithms to perform better with change in architecture type and increase in number of CPU cores.

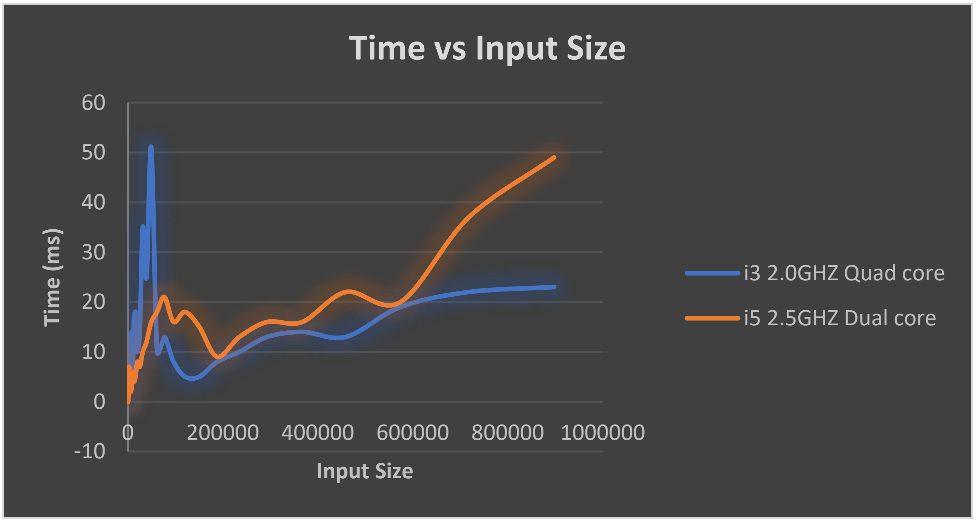


Figure 12

It can be observed from figure 12 that the dual core computer architecture takes more execution time than the quad core as expected and it takes even more time with increase in input size. It is expected that the dual core computer will take double the time the quad core computer takes to prove scalability but the dual core has a faster CPU thus the results are not exactly as expected.

Conclusion

It is evident that the parallel approach to the problem increased the speed up of the overall program although most of the time in the program is consumed by reading the text file and creating objects from it. However it is clearly evident from the results that the parallel algorithm is effective for doing operations on large data sets of input. The parallel program proves to be implemented correctly and performs as intended.

From the scalability section of the results it is evident that parallel algorithms might have to be tuned from one computer architecture to another but it is often desirable to have general solution to a problem that can have the capacity to effectively utilize an increasing number of CPU cores. It is also evident that with increase in speed of CPU cores the program performs better which illustrates that the parallel program is scalable. The results obtained are reliable because an average amount of time execution was used for the results were the program was run at least 5 times for every parameter to account for the fact that parallel algorithms are non-deterministic. Therefore the parallel program is scalable and correct.

Although the parallel portion of the program is scalable and correct, most of the time in the program is consumed by the program reading the text file (input processing) and creating objects and arrays where order is important and parallel implementation would not work. Therefore it is not worth using parallelization for such a problem to achieve overall speed up of algorithm including input processing.