Report

Author: Bilal Hasan Aslan

12.09.2019

Bilalhasanaslan@gmail.com

CONTENTS PAGE

«Introduction»

Aim

Aim of this project is to calculate efficiency of using many cores over just one. We will be able use many cores for our project with parallel algorithms. Our Project is going to input data about winds and we will classify clouds that might form according to their properties. Also we will get global average of wind speeds. After all we will going to test code between sequential and parallel code also between different sequential codes.

An Introduction, comprising a short description of the aim of the project, the parallel algorithms and their expected speedup/performance.

Parallel Algorithms and Their Expected Performance

A parallel Algorithm is algorithm that can execute many threads at the same time on different processors and combines all individual outputs. We expect Parallel Algorithm to be faster than Single Core running Sequential code. From research it is expected to have %80 efficiency over 4 cores while sequential will have 25% efficiency.

Methods

I started coding for Sequential to make sure my Paralleling algorithm is right. After finishing sequential code, I extended RecursiveTask java class and modified compute method for paralleling. I designed my compute code in a way it will return global average in vector while classifying each wind. Then I compared my parallel code output file to the sequential code output file to make sure my parallel algorithm running perfectly. I generated different data sizes to test time on my parallel algorithm then I calculated time taken for each parallel to run. I made it run 200 times automatically and took average of total time to get perfect time taken for each data size.

I measured Speedup by also calculating time taken sequential code and divided it to my parallel algorithm's taken time.

I tested on 2 different computers one with 2 core and other with 4 core. I chose these architectures specially.

I had a problem with heap size when I tried to use big data sizes so I couldn't use very big sizes. I also first tried to paralleling only time steps but that was not enough to test efficiency of parallel algorithm so I changed it in a way to be able to parallel for every cells in given data.

«Results and Discussions»

Results

Data from 2 core Laptop.

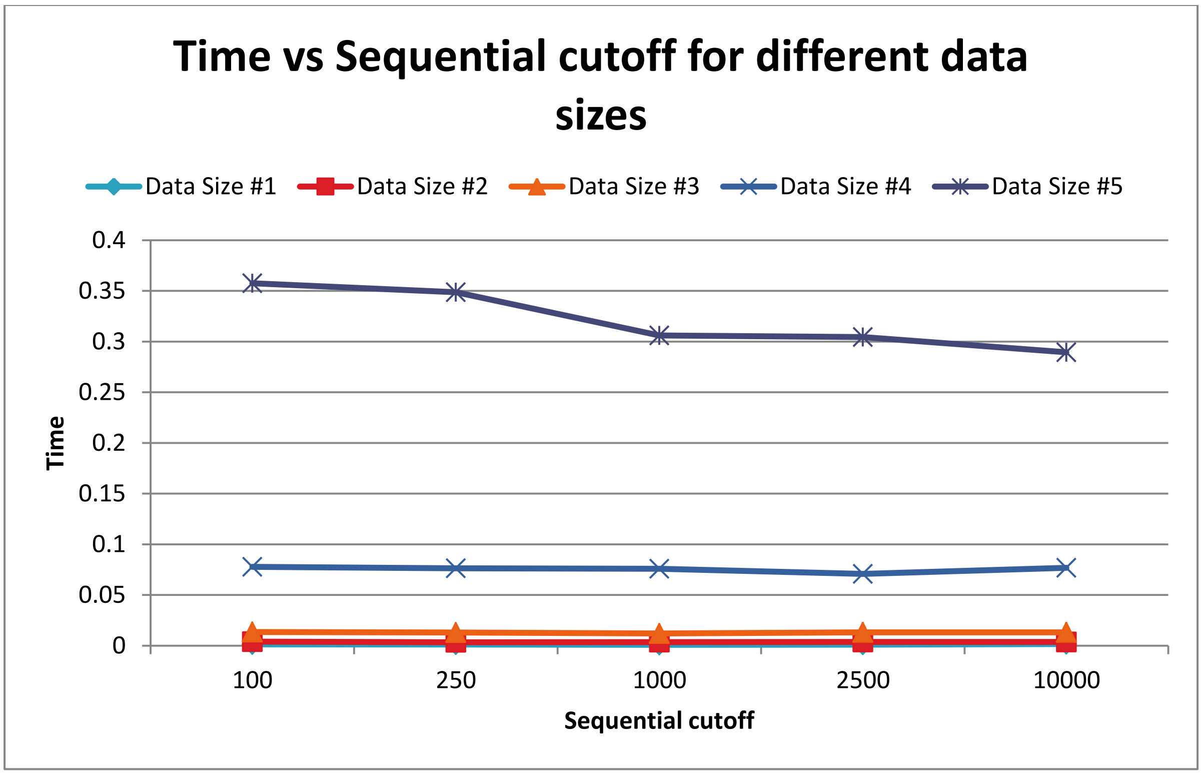
Parallel	Sequential Cutoff				
Data Size	100	250	1000	2500	10000
12500	0.0112	0.012	0.012	0.0112	0.012
5000	0.0128	0.012	0.0128	0.0124	0.0128
200000	0.0256	0.0244	0.0248	0.0244	0.0272
1250000	0.2792	0.2752	0.2744	0.2816	0.2852
5000000	0.7388	0.7284	0.7424	0.7308	0.75

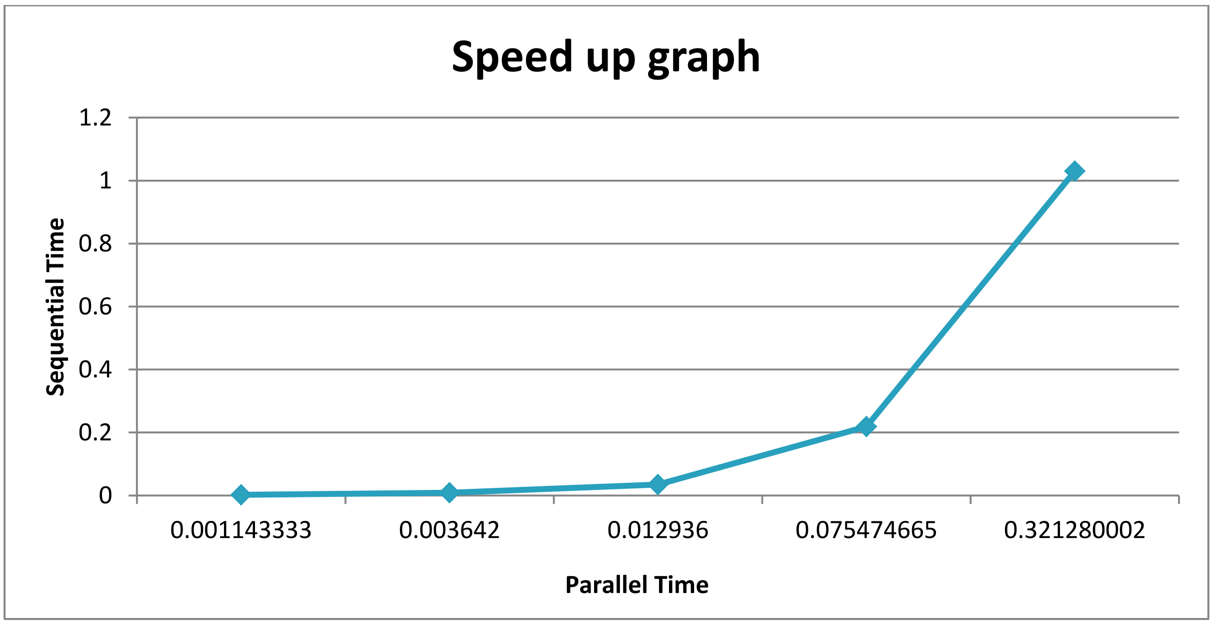
Sequential	
12500	0.002114
5000	0.008349
200000	0.03246
1250000	0.203614
5000000	0.957884

Data from 4 core Laptop.

Parallel	Sequential Cutoff				
Data Size	100	250	1000	2500	10000
12500	0.001277	0.001073	9.03E-04	9.17E-04	0.001547
5000	0.00395	0.00334	0.003517	0.003747	0.003657
200000	0.013427	0.01301	0.011967	0.013197	0.01308
1250000	0.077707	0.076347	0.075827	0.070743	0.07675
5000000	0.357613	0.348747	0.306097	0.304517	0.289427

Sequential	
12500	0.002273
5000	0.008977
200000	0.034903
1250000	0.21894
5000000	1.029983





(Graphs is from 4 core architectures)

Discussion

From Speed up graph we can see as we get larger data size Sequential Time vs. Parallel Time is growing exponential meaning for Sequential it takes much more time to finish the job compare to parallel as data gets larger. So using Parallel Algorithm really shows its efficiency on big data which means when it comes to working on big data parallel algorithms is a must.

Using smaller data sizes Parallel is definitely not performing that well because of code complexity compare to sequential code so data sizes starting around 1250000 cells performs much better.

I obtained 3.55870198283 as my maximum speed up. The ideal speed up is since I am using 4 cores it must be 4.

Optimal number of threads for my 4 core laptop will be 4 threads which will make my optimal Sequential cut off for my biggest data 1250000.

Optimal number of threads for architecture will be its number of cores.

From testing two different architectures I can conclude more cores means less time takes to finish the job for parallel algorithm.

As I increase my Sequential Cut off for fork and join code I get less time taken to complete up until sequential cut off 1250000.

«Conclusion»

According to my test results I can see that using parallel algorithm is much more efficient to reduce time taken for calculations to complete when we dealing with large data sets. Choosing Sequential cut off right is very important since choosing it very low or very high will reduce efficiency of the parallel algorithm. I also concluded that all Data sizes has their own unique ideal Sequential cut offs. Using more cores decreases time taken but sequential cut off should be chosen accordingly to the number of cores.

«REFERENCES»

https://pdfs.semanticscholar.org/43a7/e594c823a98b986febd435d9d92505573238.pdf

https://stackoverflow.com/questions/24251610/fork-join-optimal-threads-count