UNIVERSITY OF CAPE TOWN
Department of Computer Science

PARALLEL AND CONCURRENT PROGRAMMING ASSIGNMENT 1

Tracey Letlape

UIVERSITY OF CAPE TOWN

Serial Program Profiling

The following measured time taken is the average of the first ten runs (rounded off to the nearest digit), adjusted for outliers (extremely high or low values)

Increasing dungeon size with fixed search density

Table 1: Increasing dungeon size

Input	Time taken on Laptop (ms)	Time taken on Server (ms)
100 0.01 4	21	97
200 0.01 4	63	324
300 0.01 4	133	635
400 0.01 4	228	785
500 0.01 4	365	1 340

Increasing search density with fixed dungeon size

Table 2: Increasing search density

Input	Time taken on Laptop (ms)	Time taken on Server (ms)
200 0.02 4	117	342
200 0.04 4	203	514
200 0.06 4	285	791
200 0.08 4	361	958
200 0.10 4	412	1 116

Decreasing search density while increasing dungeon size

Table 3: Increasing dungeon size and search density

Input	Time taken on Laptop (ms)	Time taken on Server (ms)
100 0.10 4	213	289
200 0.08 4	573	958
300 0.06 4	1 100	1 820
400 0.04 4	1 340	2 441
500 0.02 4	1 273	2 231

Fixed dungeon size and search density with changing seeds

Table 4: Changing seeds

Input	Time taken on Laptop (ms)	Time taken on Server (ms)
200 0.05 4	246	689
200 0.05 13	226	603
200 0.05 7	228	729
200 0.05 11	233	730
200 0.05 17	225	704

Comments

- Increase in dungeon size has significant impact on the time taken. Increasing the dungeon size with other input parameters fixed will increase the time taken by the serial program.

- Increase in the search density has little impact on the time taken. Increasing the search density with other parameters fixed will increase the time taken by the program, but not greatly.
- Changing seeds has little and indeterminate impact on the time taken.
- Increasing both the dungeon size and the search density will increase the time taken by the program, with the dungeon size having great impact.

Methods section

Validation

In order to verify that the parallel algorithm was correctly working, I ran each set of input ten times on my local machine and on the server for both the parallel and the serial programs. Each time, I would take the images produced by both programs, used the BeyondCompare software to compare the pixel details of both the images to ensure both the images are exactly identical. The software has a summary info that details how many pixels are the same, how many pixels are different and so on. I would also compare the console outputs to ensure that the position of the boss is the same.

Optimum Search Density

To determine the optimum search density, I first wanted each hunter to process at least 100 elements for a large enough dungeon area. I tested with a dungeon size of 100 which yields a dungeon area of 1 000 000. So if I have 1 000 000 elements and I want each hunter to process at least 100 elements, how many hunters will be optimal. I used a mathematical formula by diving the dungeon area with x (the unknown number of hunters) and equating the quotient to 100. Then I tried to solve for x and got x to be 10 000. So if I got x to be 10 000, then using the formula in the code to compute numSearches, I substituted numSearches to 10 000, gateSize to 100 and worked backwards to get args[1], which I got to be around 0.01. Since 0.01 seemed optimal, I wanted to see if anything higher would have been better, so I tested on different search density value in the range of [0.02, 0.10] in steps of 0.02. However, for benchmarking, I used 0.01.

Benchmarking

To benchmark my parallel algorithm to ensure that it is both correct and faster, I ran the serial and parallel algorithms for each set of inputs multiples times on both my local machine () and on the nightmare UCT server (), recorded the results and took the average first ten runs adjusted for outliers. The range of inputs I tested on are exactly the same as the ones used for **SERIAL PROGRAM PROFILING**.

In order to determine the sequential cutoff, I decided on how I wanted the tasks to be distributed among the cores. It seemed logical to have each core have a single task at a time. However, due to some CPU delays we don't have control over, we don't want to have a core not doing anything and just waiting for others to finish. So, we must give each thread just enough work, so how about 4 tasks per core, 6 or even 8? So, I set the threshold to the number of hunters divided by the product of available processors and tasks per core. Then tested with tasks per core in {4, 6, 8}. From these results () I choose

tasks per core to be 6 since its curve is more logarithmic, indicating faster time complexity and better speedup.

Result section

Validation

https://traceyomphile.github.io/Validation.html

Benchmarking

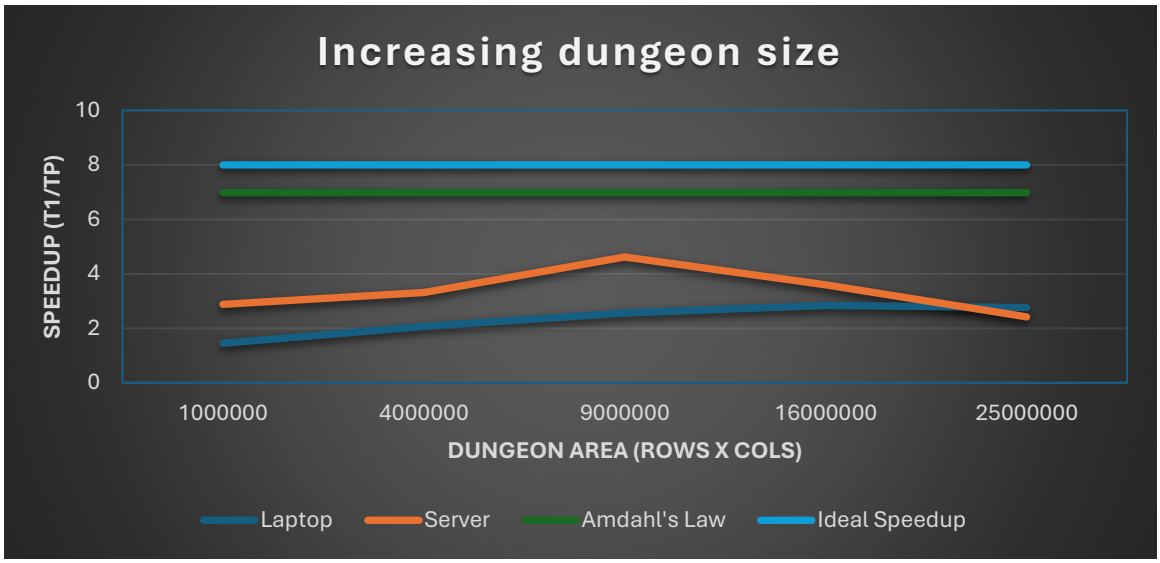

Figure 1: Increasing dungeon size with fixed search density of 0.01

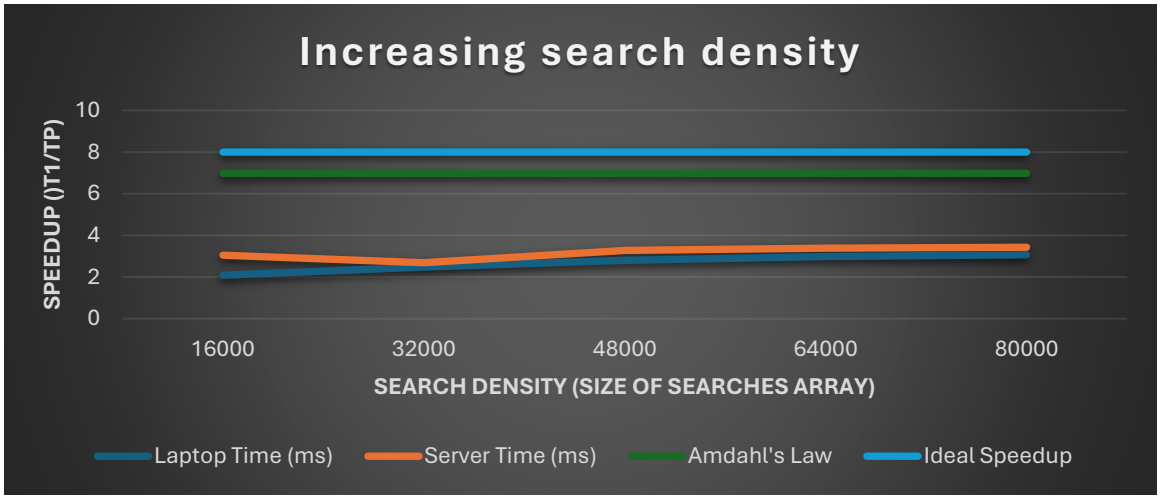

Figure 2: Increasing search density with fixed dungeon size of 200

- Dungeon area = ((x + x) * 5)^2 where $x \in \{100, 200, 300, 400, 500\}$
- Search Density = (y)(200 * 2)(200 * 2)(5) where $y \in \{0.02, 0.04, 0.06, 0.08, 0.10\}$

Brief Discussion

1. For what input does parallel program perform well?
    - For input ARGS=”300 0.01 4”, the parallel program seems perform well as seen in **Figure 1**. The program performs well on both the server and the local machine, and it is closer to the maximum speedup achievable by Amdahl’s law.
2. What is the maximum speedup you obtained and how close is it to the ideal expected?
    - The maximum speedup achieved by was 4.62 which is very far from the ideal expected of 8.
3. How reliable are your measurements?
    - About 85% reliable. This is because of nondeterminism in scheduling, hardware behaviour (caches, clocks), thread overhead and runtime effects.
    - The reliability is a bit high because the measurements are an average of about 10 runs adjusted for outliers.
4. Are there any anomalies and can you explain how they occur?
    - Yes. While running the program multiple of times, it sometimes occurs that the time taken by the program is negative.
    - Also, the number of dungeon points evaluated by the parallel program is not the same as the number of dungeon points evaluated by the serial program. Well, this is because of the race condition in the code as there is not much synchronization.

Conclusion

Is it worth parallelising this problem? Well, considering that the dungeon map is very huge, why would anyone want to search the whole forest alone? It is much more effective and quicker to parallelise this problem. Though the race condition issue still exists, it is too insignificant in this case as the number of points evaluated is not actually the problem at hand, but finding the dungeon master. So long as the dungeon master can be found quickly, who would be bothered about how many places were actually searched? So, yes, it is worth parallelizing this problem.

Use of Artificial Intelligence

- I used Google Gemini to help create html files linked in this file. This was to allow myself to be able to stick to the assignment requirements of having a report with no more than 5 pages while also providing sufficient information. I converted the word files I generated into html files, made them available on GitHub so that they can be viewed on any machine.
- I also used ChatGPT to help explain and break down the assignment.
- I also used Grok to help debug my code, and to validate whether my method of determining the sequential cutoff was right.