Assignment 1 ID5130

Anurag Manoj ee22b003

March 2024

1 Q1

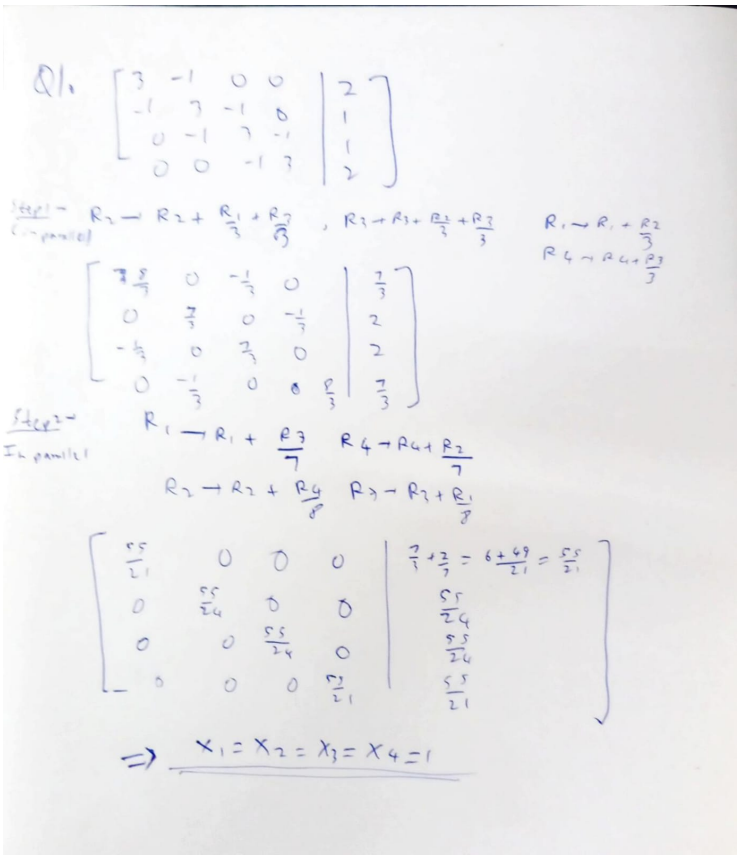


Figure 1: Hand calculations

2 Q2

2.1 part a

Thomas algorithm is used to solve the tridiagonal system that has to be solved to differentiate a function using 4th order accurate pade scheme. The memory usage is 3 n-element arrays to store the tridiagonal matrix, an n-element array to store output derivative at each point, and an n-element array to store RHS. The time complexity is O(n). The L and U matrices have not been explicitly created to save on memory and time, but the algorithm, which consists of 2 passes through the arrays, is effectively the same. Time complexity is O(n). Output with very small element size is shown below. This differentiation method suffers from spikes near the ends of the interval of differentiation.

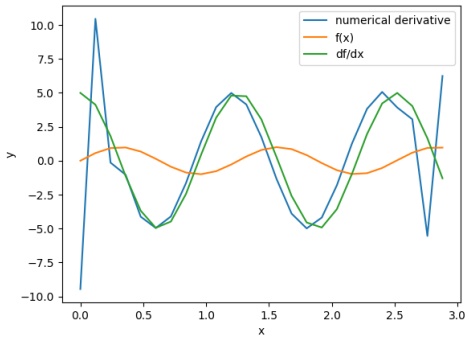
(a) Accurate derivative

(b) n = 25, LU-decomposition

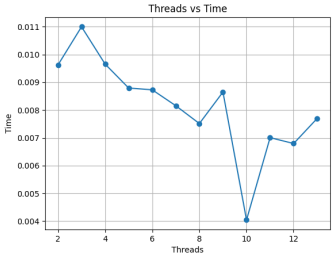
Figure 2: Q2

2.2 part b

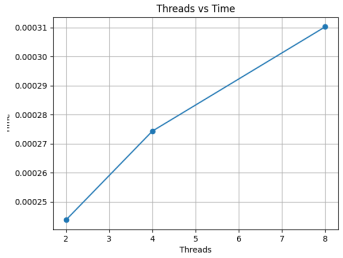
Recursive doubling is used to solve the same system as in part a. The algorithm is shown in the hand calculations. The memory requirement is more, as a copy of the matrix as well as vectors have to be created. The time complexity is O(nlog(n)/p), where p is the number of processors. The parallelization of the main for loop in recursive doubling is found to be best when using dynamic scheduling with a chunk size of 128. This is as compared with auto, various static scheduling, and other possible chunk sizes of dynamic scheduling. A reason for dynamic scheduling outperforming static scheduling could be that there are conditionals within the for loop, and the number of instructions to be executed varies, depending on the row number in the tridiagonal system. The improvement in speed due to parallelism is plotted for n = 100000. At n = 1000 though the benefits of parallelism are not seen, and the time taken to allocate memory and initialize the solver dominate.



(a) n=100, p=2 recursive doubling



(b) n=100000



(c) n = 25, n=1000

Figure 3: Recursive doubling timings

3 Q3

3.1 Intro

One serial and 2 parallel algorithms have been used to solve Poisson's equation on the square of side length 2 centred at the origin. The convergence of the algorithms are evaluated as error = (max(absolute difference between analytical and numerical solution) / maximum of solution (which is 1)) x 100 percent. The algorithm is sampled every 5*(side length of grid) iterations to check if the error has dropped below 1 percent.

3.2 serial

The algorithm used for the serial solution is Gauss seidel. The memory usage is O(n^2), where n is the length of the side of the solution grid, and time complexity is O(n^2log(n)), due to having to traverse each point in the grid, as well as converging as O(log(n)) number of iterations.

3.3 red black coloring approach

The grid is alternately colored red and black, and the 5 point stencil is applied parallelly to each cell for each color. To speed up the code, evaluation of a single color is done in 2 separate for loops, which can be collapsed using omp for collapse(2) feature for more parallelism, and allowing the compiler to vectorize it. Auto scheduling of the for loops is the fastest.

3.4 diagonal approach

The 5 point stencil is evaluated for cells on a grid diagonal by diagonal. The evaluation for each diagonal is parallelized. Due to the small size of the grid, trying to improve cache efficiency through reordering the grid as well as increasing the thread count in this code is useless and only increases execution time due to the overhead involved. For 16 threads, execution time becomes 25 times, as my laptop has only 10 available cores, the overhead of managing more threads than cores is too much.

3.5 part a

For delta = 0.1, it took 210 iterations to converge.

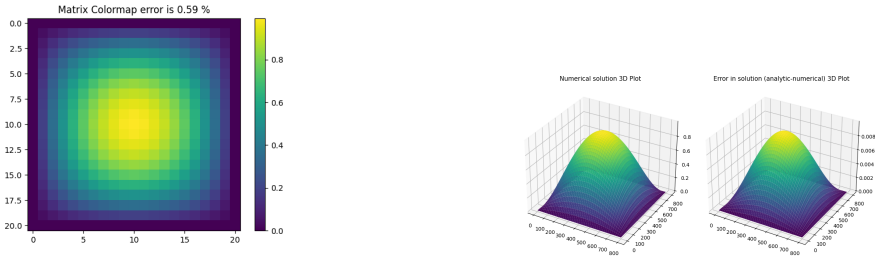
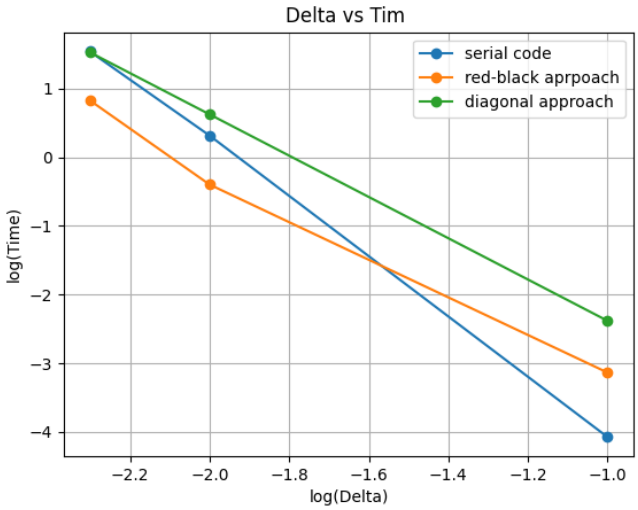
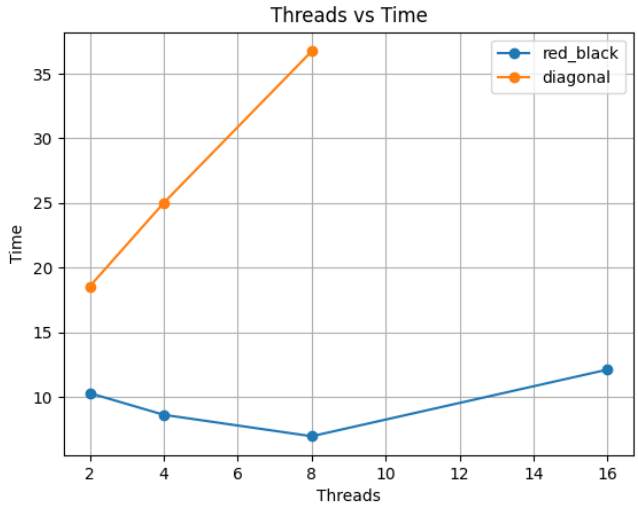


Figure 4: plot of error and solution

3.6 part c,d

Graphs below show various timings of the solver


Delta vs Tim
serial code
red-black aprpoach
diagonal approach
log(Time)
log(Delta)
1
0
−1
−2
−3
−4
−2.2
−2.0
−1.8
−1.6
−1.4
−1.2
−1.0


Threads vs Time
red_black
diagonal
Time
Threads
35
30
25
20
15
10
2
4
6
8
10
12
14
16