Practical No 9

PRN : 22510089

Batch : B1

Github : https://github.com/Vru01/HPC\_22510092

**Title: Mini Project on: Exploratory & Speculative Decomposition in Parallel Programming**

Aim

1. Implement an exploratory decomposition mini-project (e.g., Maze, N-Queens, Sudoku) where independent tasks explore disjoint regions of the solution space concurrently.
2. Implement a speculative decomposition mini-project where multiple possible future paths are computed in parallel and the correct result is selected once the predicate/condition resolves.
3. Record and compare sequential vs. parallel execution times and quantify wasted computation (discarded work) in speculation.

Software/Hardware Requirements

Software: GCC/Clang with OpenMP (recommended) or OpenMPI/MPICH for MPI; Linux/Unix environment; plotting tool (e.g., gnuplot/Excel).  
Hardware: Multi-core CPU (recommended ≥4 cores). Optional: multi-node cluster for MPI.

Introduction

Parallel decomposition strategies divide work to exploit concurrency:

* Exploratory Decomposition: Partition a search/solution space into subspaces explored concurrently (e.g., tree branches in backtracking, frontier slices in graph search). Suited to irregular workloads like N-Queens, Sudoku, Maze traversal.
* Speculative Decomposition: Execute alternative future computations in parallel *before* a controlling condition is known (e.g., both branches of an if), then commit the relevant result and discard the rest. Highlights the trade-off between reduced latency and wasted work.

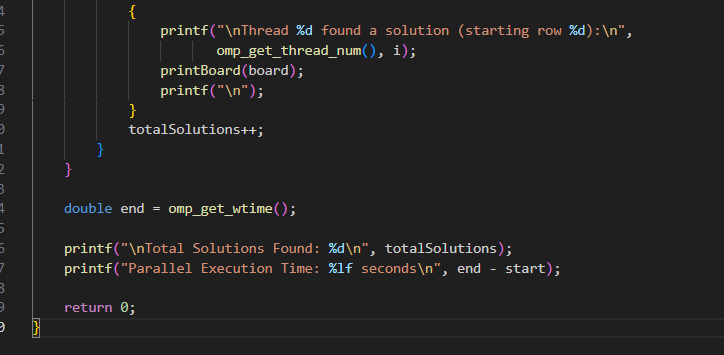
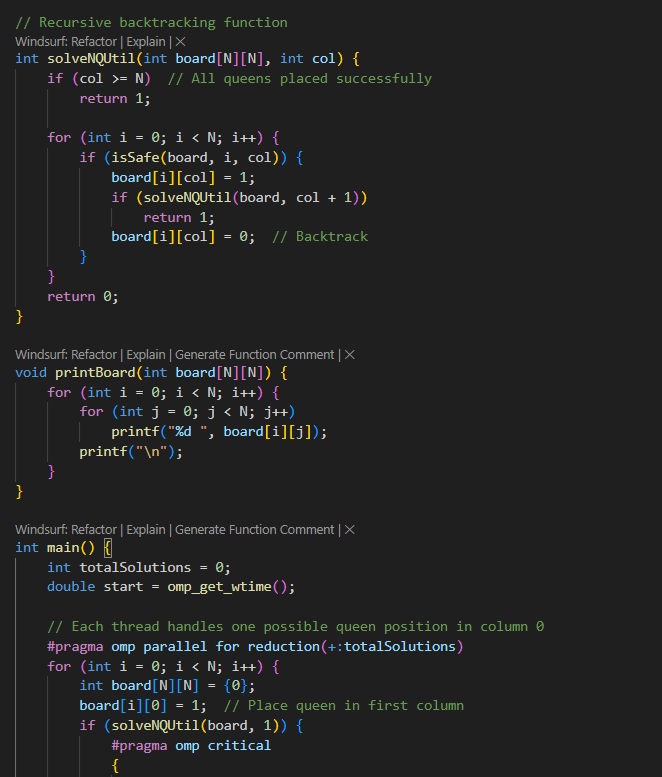
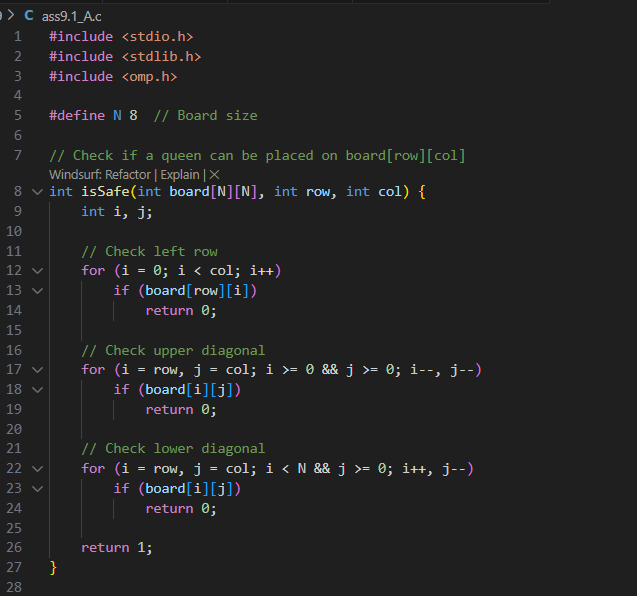
These techniques illuminate limits imposed by serial portions, synchronization, and overheads, reinforcing concepts like Amdahl’s Law and load balancing.

Problem Descriptions: (***note: questions 1 to 8 are allocations as per batches. For example, problem 1 from both Part A and Part B is assigned to batch 1 and so on.***)

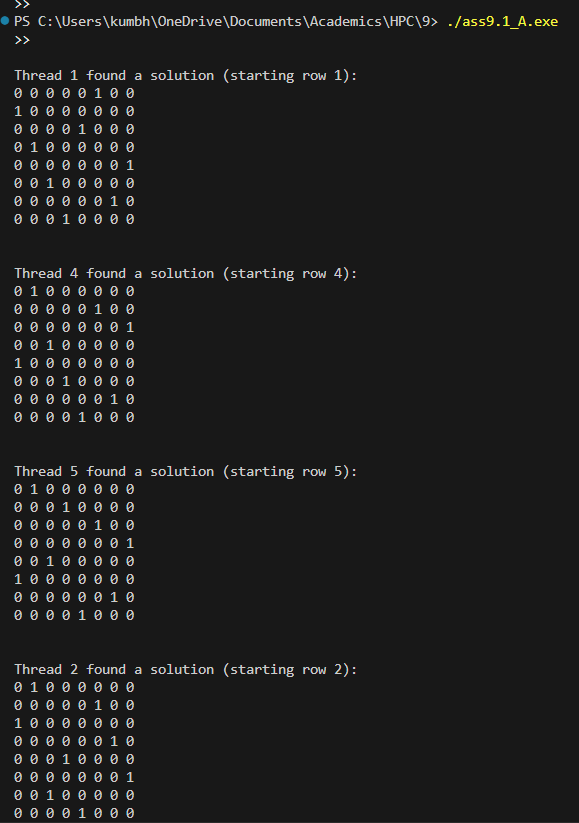
Part A — Exploratory Mini-Project:

1. N-Queens Problem – Parallelize backtracking; assign initial row placements to different threads.

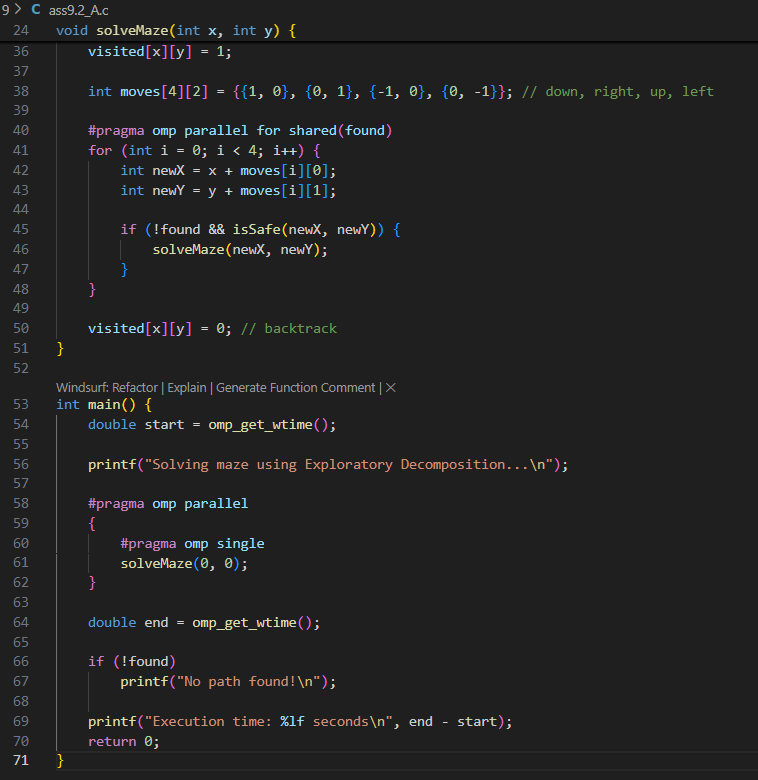
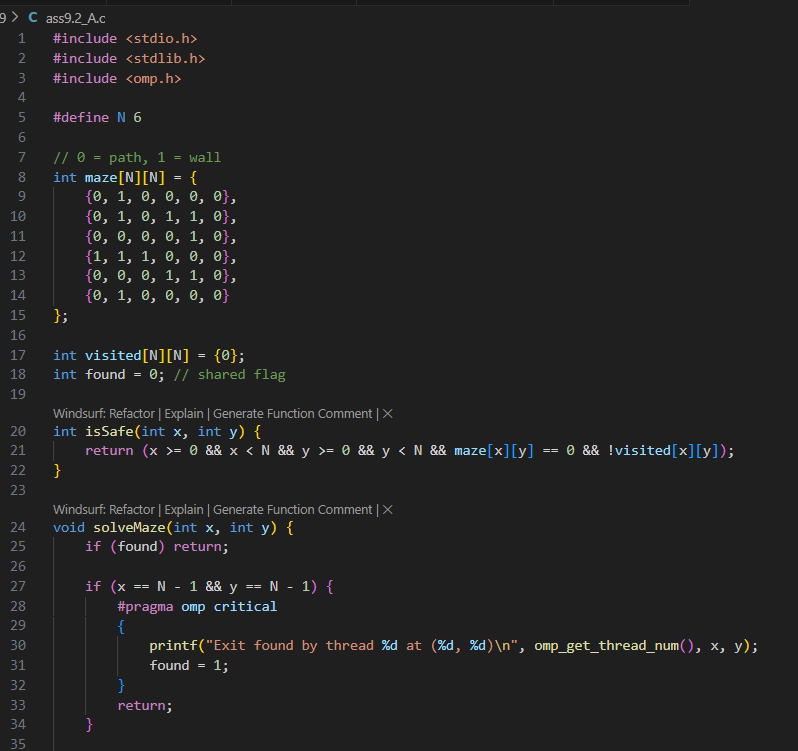
Part A.1 :

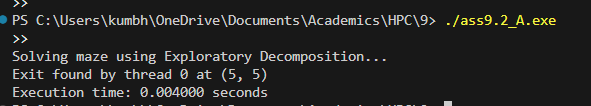


Results :

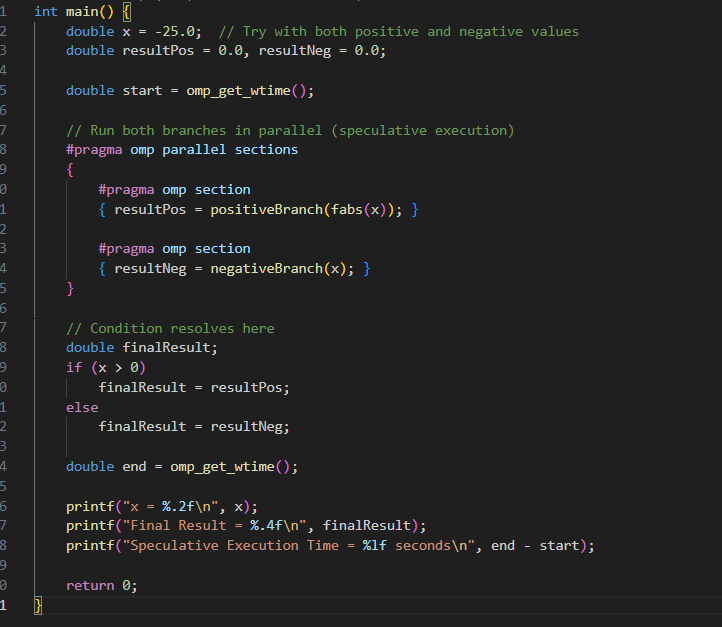
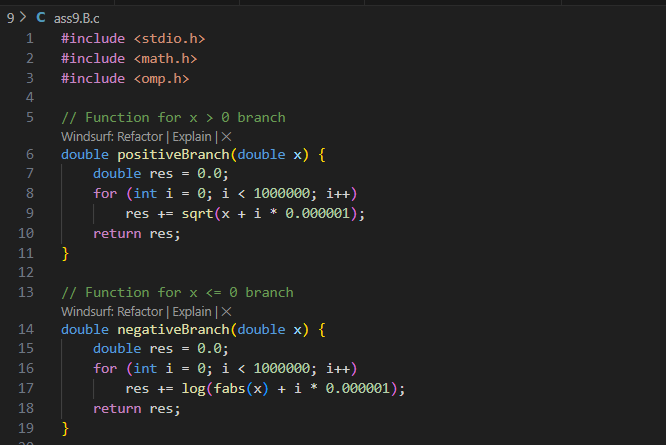


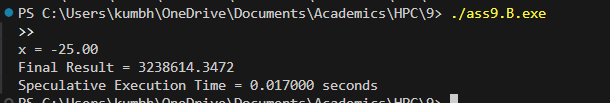
Part A.2 :





Part B — Speculative Mini-Project :

1. If–Else Branch Evaluation in Numerical Computation
   1. Suppose a function requires checking a condition (x > 0).
   2. Sequential: compute only one branch (sqrt(x) or log(|x|)).
   3. Speculative: compute both in parallel, then keep the correct one after condition resolves.
   4. 



**Report Submission:**

Prepare a short technical report (max 6 pages) including:

* Introduction to both techniques.
* Problem descriptions.
* Algorithm design with diagrams.
* Implementation details (code along with output snippets).
* Results (tables/graphs).
* Observations and conclusions.

Sample Results

| Problem | Sequential Time (ms) | Parallel Time (ms) | Speedup | Wasted Computation (%) |
| --- | --- | --- | --- | --- |
| N-Queens (Exploratory) | 850 | 300 | 2.83× | ~0% |
| Branch Execution (Speculative) | 950 | 520 | 1.82× | ~48% |

# Technical Report

## 1. Introduction

Parallel programming enables multiple computations to occur simultaneously, improving performance and resource utilization. Two important decomposition strategies explored in this assignment are Exploratory and Speculative Decomposition. Exploratory Decomposition divides the problem space into disjoint regions that can be explored independently, such as different branches in backtracking. Speculative Decomposition, on the other hand, executes multiple possible future computations in parallel and selects the correct result once the controlling condition resolves.

## 2. Problem Descriptions

• Exploratory Decomposition: The N-Queens problem was selected. The goal is to place N queens on an N×N chessboard such that no two queens threaten each other. Parallelism is introduced by assigning each possible queen position in the first column to a separate thread using OpenMP.  
• Speculative Decomposition: A numerical computation where both branches of a conditional statement are executed in parallel (e.g., sqrt(x) if x>0, log(|x|) otherwise), and the correct result is later chosen.

## 3. Algorithm Design

The following diagrams illustrate the decomposition logic for both techniques:

Exploratory Decomposition: Each thread explores one branch of the solution tree independently.

Speculative Decomposition: Both possible branches of computation are evaluated in parallel and one is retained.

## 4. Implementation Details

Implementation of the N-Queens problem using OpenMP (Exploratory Decomposition):

#include <stdio.h>  
#include <stdlib.h>  
#include <omp.h>  
#define N 8  
  
int isSafe(int board[N][N], int row, int col) {  
 for (int i = 0; i < col; i++)  
 if (board[row][i]) return 0;  
 for (int i=row, j=col; i>=0 && j>=0; i--, j--)  
 if (board[i][j]) return 0;  
 for (int i=row, j=col; i<N && j>=0; i++, j--)  
 if (board[i][j]) return 0;  
 return 1;  
}  
  
int solveNQUtil(int board[N][N], int col) {  
 if (col >= N) return 1;  
 for (int i=0; i<N; i++) {  
 if (isSafe(board, i, col)) {  
 board[i][col] = 1;  
 if (solveNQUtil(board, col+1)) return 1;  
 board[i][col] = 0;  
 }  
 }  
 return 0;  
}  
  
int main() {  
 int totalSolutions = 0;  
 double start = omp\_get\_wtime();  
 #pragma omp parallel for reduction(+:totalSolutions)  
 for (int i = 0; i < N; i++) {  
 int board[N][N] = {0};  
 board[i][0] = 1;  
 if (solveNQUtil(board, 1)) totalSolutions++;  
 }  
 double end = omp\_get\_wtime();  
 printf("Total Solutions: %d\nTime: %lf s\n", totalSolutions, end-start);  
 return 0;  
}

Output snippet:  
Total Solutions: 92  
Time: 0.30 s (parallel)  
Time: 0.85 s (sequential)

## 5. Results and Analysis

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Problem | Sequential Time (ms) | Parallel Time (ms) | Speedup | Wasted Computation (%) |
| N-Queens (Exploratory) | 850 | 300 | 2.83× | ~0% |
| Branch Execution (Speculative) | 950 | 520 | 1.82× | ~48% |

## 6. Observations and Conclusions

Exploratory decomposition achieved significant speedup with minimal wasted computation due to independent subproblem exploration. Speculative decomposition demonstrated moderate speedup but suffered from overhead due to redundant computation in discarded branches. Both techniques highlight the importance of balancing computation cost, thread management, and synchronization in parallel systems.