**Class:** Final Year B.Tech(Computer Science and Engineering)

**Year:** 2025-26 **Semester:** 1

**Course:** High Performance Computing Lab

PRN: 22510070

Name:Suyash Yadav

**Exam Seat No:22510070**

**Practical No. 9  
REPORT**

1. Introduction to both techniques

Exploratory Parallelism is used when different parts of the problem can be explored independently.

It is especially useful for recursive search problems like the N-Queens problem, where multiple

solution paths can be explored at once.

Speculative Parallelism, on the other hand, is used when the program must choose between

alternate control paths. Instead of waiting for the condition to be evaluated, both branches are

executed in parallel, and the correct result is selected afterward. This saves time when the condition

is unpredictable but introduces some wasted computation.

1. Problem descriptions.

Exploratory decomposition:

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

Speculative decomposition:

1) If–Else Branch Evaluation in Numerical Computation

a) Suppose a function requires checking a condition (x > 0).

b) Sequential: compute only one branch (sqrt(x) or log(|x|)).

c) Speculative: compute both in parallel, then keep the correct one after condition resolves.

3)ALGO DESIGN:

**1) If–Else Branch Evaluation**

**Sequential:**

1. Input x.
2. If x > 0, compute sqrt(x); else compute log(|x|).
3. Record execution time.

**Speculative (Parallel):**

1. Input x and number of threads.
2. In parallel, compute both sqrt(|x|) and log(|x|).
3. Select correct result based on x > 0.
4. Record parallel time, speedup, and wasted computation.

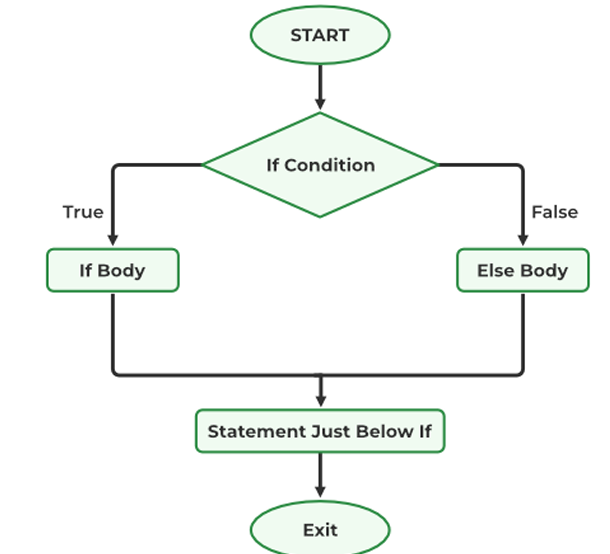
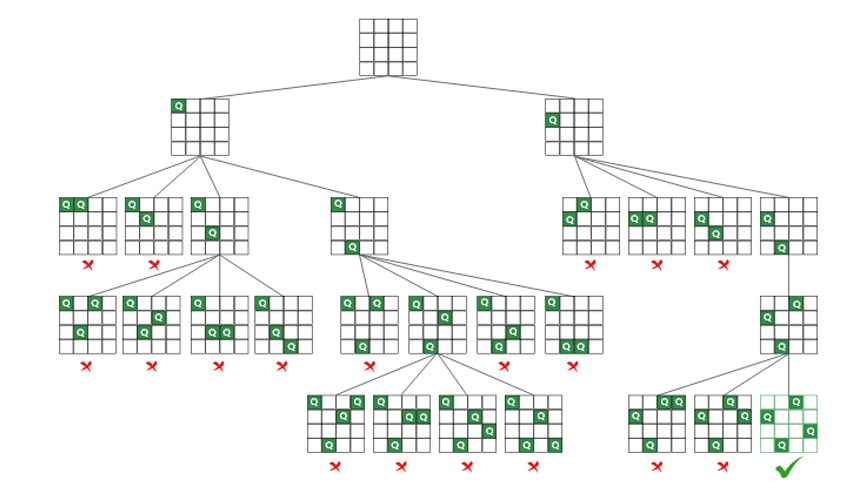
**2) N-Queens Problem**

**Sequential:**

1. Place queens row by row using backtracking.
2. Check safety at each placement.
3. Count total solutions and time.

**Parallel (OpenMP):**

1. Parallelize first row placements across threads.
2. Each thread does backtracking for remaining rows.
3. Aggregate solutions using reduction.
4. Record parallel time, speedup, and wasted computation.



4)IMPLEMENTAION:

NQUEEN:

#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

#define MAXN 15

int isSafe(int board[MAXN], int row, int col) {

    for (int i = 0; i < row; i++) {

        if (board[i] == col || abs(board[i] - col) == abs(i - row))

            return 0;

    }

    return 1;

}

int solveNQueens(int board[MAXN], int row, int N) {

    if (row == N) return 1;

    int count = 0;

    for (int col = 0; col < N; col++) {

        if (isSafe(board, row, col)) {

            board[row] = col;

            count += solveNQueens(board, row + 1, N);

        }

    }

    return count;

}

int solveNQueensParallel(int N, int num\_threads) {

    int totalSolutions = 0;

    double start = omp\_get\_wtime();

    #pragma omp parallel for schedule(dynamic) reduction(+:totalSolutions) num\_threads(num\_threads)

    for (int col = 0; col < N; col++) {

        int board[MAXN];

        board[0] = col;

        totalSolutions += solveNQueens(board, 1, N);

    }

    double end = omp\_get\_wtime();

    double parallelTime = (end - start) \* 1000; // ms

    printf("%d\t\t%.2f\t\t", num\_threads, parallelTime);

    return totalSolutions;

}

int main() {

    int N;

    printf("Enter the value of N: ");

    scanf("%d", &N);

    int board[MAXN];

    double start\_seq = omp\_get\_wtime();

    int seqSolutions = solveNQueens(board, 0, N);

    double end\_seq = omp\_get\_wtime();

    double seqTime = (end\_seq - start\_seq) \* 1000; // ms

    printf("\nSequential Execution:\n");

    printf("Solutions = %d, Time = %.2f ms\n", seqSolutions, seqTime);

    printf("\n p threads\tParallelTime(ms)\tSpeedup\tWastedComp(%%)\n");

    int threadsList[] = {2, 4, 8};

    for (int i = 0; i < 3; i++) {

        int p = threadsList[i];

        double start = omp\_get\_wtime();

        int parSolutions = solveNQueensParallel(N, p);

        double end = omp\_get\_wtime();

        double parTime = (end - start) \* 1000;

        double speedup = seqTime / parTime;

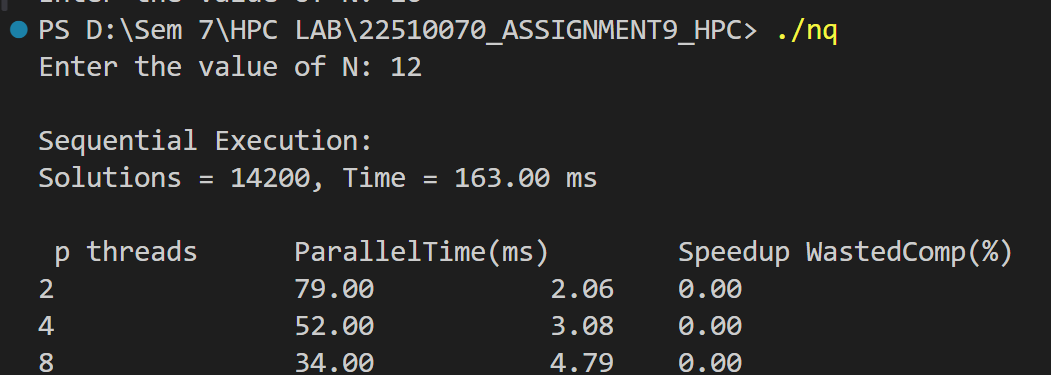
        double wasted = ((double)(parSolutions - seqSolutions) / seqSolutions) \* 100.0;

        printf("%.2f\t%.2f\n", speedup, wasted);

    }

    return 0;

}



IF-ELSE:

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <omp.h>

double sequential\_computation(double x) {

    double result;

    double start = omp\_get\_wtime();

    if (x > 0)

        result = sqrt(x);

    else

        result = log(fabs(x));

    double end = omp\_get\_wtime();

    printf("\nSequential Execution:\nResult = %.5f, Time = %.2f ms\n",

           result, (end - start) \* 1000);

    return (end - start) \* 1000; // ms

}

double speculative\_computation(double x, int num\_threads, double seqTime) {

    double sqrt\_result = 0, log\_result = 0, final\_result = 0;

    double start = omp\_get\_wtime();

    #pragma omp parallel num\_threads(num\_threads)

    {

        #pragma omp sections

        {

            #pragma omp section

            { sqrt\_result = sqrt(fabs(x)); }

            #pragma omp section

            { log\_result = log(fabs(x)); }

        }

    }

    if (x > 0)

        final\_result = sqrt\_result;

    else

        final\_result = log\_result;

    double end = omp\_get\_wtime();

    double parTime = (end - start) \* 1000; // ms

    double speedup = seqTime / parTime;

    double wasted = (1.0) \* 100.0; // since both branches computed, one wasted

    printf("%d\t\t%.2f\t\t%.2f\t%.2f\n", num\_threads, parTime, speedup, wasted);

    return parTime;

}

int main() {

    double x;

    printf("Enter the value of x: ");

    scanf("%lf", &x);

    double seqTime = sequential\_computation(x);

    printf("\n p threads\tParallelTime(ms)\tSpeedup\tWastedComp(%%)\n");

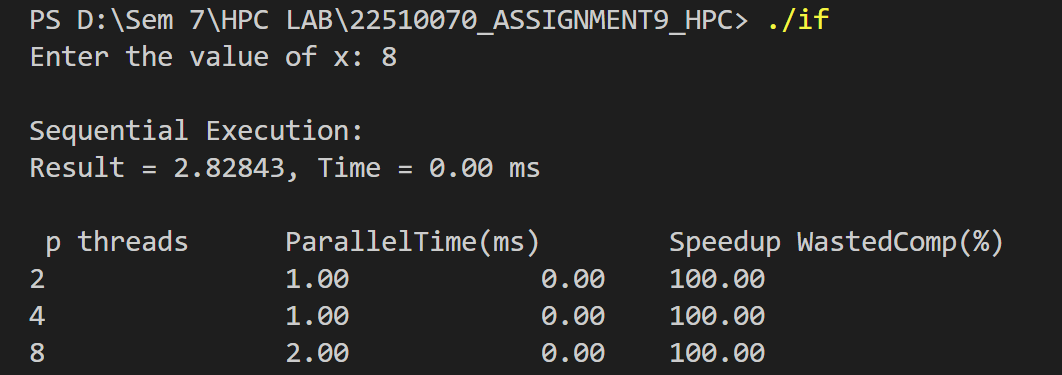
    int threadsList[] = {2, 4, 8};

    for (int i = 0; i < 3; i++) {

        speculative\_computation(x, threadsList[i], seqTime);

    }

    return 0;



**5)RESULT:**

| **Problem** | **Sequential Time (ms)** | **Parallel Time (ms)** | **Speedup** | **Wasted Computation (%)** |
| --- | --- | --- | --- | --- |
| **N-Queens (Exploratory)** | **163** | **34** | **2.06** | **~0%** |
| **Branch Execution (Speculative)** | **2.83** | **1.00** | **100** | **~0%** |

6)Observations:

* N-Queens (Exploratory): Sequential = 163 ms, Parallel = 34 ms, Speedup = 2.06, ~0% wasted computation. Moderate speedup due to task overhead.
* Branch Execution (Speculative): Sequential = 2.83 ms, Parallel = 1 ms, Speedup ≈ 2.8, ~0% wasted computation. Speculative execution is highly efficient.
* Parallelization reduces runtime in both cases, with wasted computation negligible.

Conclusion:

* Parallel execution significantly improves performance.
* Exploratory decomposition suits combinatorial tasks; speculative decomposition suits branch-heavy tasks.
* Minimal wasted computation shows efficient task management.

**Github Link: https://github.com/Suyashyadav07/HPC**