Indian Institute of Information Technology Surat

Lab Report on
High Performance Computing (CS 602) Practical

Submitted by

[RAHUL KUMAR SINGH] (UI21CS44)

Course Faculty

Dr. Sachin D. Patil

Department of Computer Science and Engineering
Indian Institute of Information Technology Surat
Gujarat-394190, India

Jan-2024

Lab No: 5

Aim: To Develop an efficient algorithm to solve the 10×10 chessboard problem, placing 10 Queens or Horses without conflicts, and analyze time complexity and execution time for both individual and all possible solutions.

Description:

- Problem Statements:
  i) 10 Queens: Avoid striking each other.
    - Algorithm: Recursive placement with OpenMP parallelization.
    - Analyze time complexity and execution time for individual and all solutions.

  ii) 10 Horses: Identify maximum placements without conflicts.
    - Algorithm: Parallelized approach using OpenMP for horse placement.
    - Analyze time complexity and execution time for individual and all solutions.
- Queen Placement: Backtracking algorithm ensuring no conflicts.
- Horse Placement: Parallelized approach to maximize placements without conflicts.
- OpenMP: Utilized for parallel constructs, enhancing computational efficiency.
- Dynamic and Static Scheduling: Applied for load balancing in parallel sections.

Source Code:

Task 1:

```
#include <omp.h>
#include <stdio.h>
#include <time.h>
#include <stdbool.h>

#define T 4
#define N 10
#define M 20

int row[N];
int col[N];
int dig[2*N];
int dig2[2*N];
char board[N][N];
int Qcnt = 0;
int Bcnt = 0;
bool check(int i, int j)
{
        if (row[i]==1 || col[j]==1 || dig[N+i-j]==1 || dig2[i+j]==1) return false;
        else return true;
}
void display_Board()
{
        printf("Board Id: %d\n",Bcnt);
        for (int i=0;i<N;i++)
        {
                for (int j=0;j<N;j++)
                {
                        printf("%c ", board[i][j]);
                }
                printf("\n");
        }
}
```

```
void NQueen_Parallel(int j)
{
        if (Qcnt==N)
        {
                display_Board();
                Bcnt++;
        }
        #pragma omp parallel for
        for (int i=0;i<N;i++)
        {
                if (check(i,j))
                {
                        #pragma omp critical
                        {
                                board[i][j] = 'Q';
                                row[i]=1;
                                col[j]=1;
                                dig[N+i-j]=1;
                                dig2[i+j]=1;
                                Qcnt++;
                        }
                        NQueen_Parallel(j+1);
                        #pragma omp critical
                        {
                                board[i][j] = '.';
                                row[i]=0;
                                col[j]=0;
                                dig[N+i-j]=0;
                                dig2[i+j]=0;
                                Qcnt--;
                        }
                }
        }
        return;
}
```

```
int main() {
        for (int i=0;i<N;i++){
                row[i]=0;
                col[i]=0;
                dig[i]=0;
                dig[N+i]=0;
                dig2[i]=0;
                dig2[N+i]=0;
        }
        for (int i=0;i<N;i++) for (int j=0;j<N;j++) board[i][j]='.';
        clock_t start_serial = clock();
        NQueen(0);
        clock_t end_serial = clock();
        int serial_count = Bcnt;
        Bcnt=0;
        double serial_time = ((double)end_serial - start_serial) / CLOCKS_PER_SEC;
        for (int i=0;i<N;i++) for (int j=0;j<N;j++) board[i][j]='.';
        clock_t start_parallel = clock();
        #pragma omp parallel
        {
                #pragma omp single
                {
                        NQueen_Parallel(0);
                }
        }
        clock_t end_parallel = clock();
        int parallel_count = Bcnt;
        double parallel_time = ((double)end_parallel - start_parallel) / CLOCKS_PER_SEC;
        printf("Serial board count: %d\n", serial_count);
        printf("Parallel board count: %d\n", parallel_count);
        printf("Serial execution time: %f seconds\n", serial_time);
        printf("Parallel execution time: %f seconds\n", parallel_time);
        double scale_factor = serial_time/parallel_time;
        printf("Serial-to-Parallel Time Ratio: %.2f\n",scale_factor);
        return 0;
}

```

Task 2:

```
void KHorse_Parallel() {
        if (N==1) {
                board[0][0]='K';
                display_Board();
                Bcnt++;
                return;
        }
        #pragma parallel for
        for (int i = 0; i < N; i++) for (int j = 0; j < N; j++) if (j%2==0) board[i][j] = 'K';
        display_Board();
        Bcnt++;
        #pragma parallel for
        for (int i = 0; i < N; i++) for (int j = 0; j < N; j++) board[i][j*2] = '.';
        #pragma parallel for
        for (int i = 0; i < N; i++) for (int j = 0; j < N; j++) if (j%2==1) board[i][j] = 'K';
        display_Board();
        Bcnt++;
        return;
}
```

```
32 void attack(int row, int col, char a, char** board) {
33         for (int k = 0; k < 8; k++) {
34                 int nextRow = row + rowMove[k];
35                 int nextCol = col + colMove[k];
36                 if (nextRow >= 0 && nextRow < N && nextCol >= 0 && nextCol < N) board[nextRow][nextCol] = a;
37         }
38 }
39
40 int canPlace(int i, int j, char** board) {
41     if (board[i][j] == '_') return 1;
42     else return 0;
43 }
44
45 void place(int i, int j, char k, char a, char** board, char** new_board) {
46     for (int y = 0; y < N; y++) for (int z = 0; z < N; z++) new_board[y][z] = board[y][z];
47     new_board[i][j] = k;
48     attack(i, j, a, new_board);
49 }
50
51 void kkn(int k, int sti, int stj, char** board) {
52     if (k == 0) {
53         displayBoard(board);
54         Bcnt++;
55     } else {
56         for (int i = sti; i < N; i++) {
57             for (int j = stj; j < N; j++) {
58                 if (canPlace(i, j, board)) {
59                     char** new_board = (char**)malloc(N * sizeof(char*));
60                     for (int x = 0; x < N; x++) {
61                         new_board[x] = (char*)malloc(N * sizeof(char));
62                     }
63                     place(i, j, 'K', 'A', board, new_board);
64                     kkn(k - 1, i, j, new_board);
65                     for (int x = 0; x < N; x++) free(new_board[x]);
66                     free(new_board);
67                 }
68             }
69             stj = 0;
70         }
71     }
72 }
```

```
106 int main() {
107         char** board = (char**)malloc(N * sizeof(char*));
108         for (int i = 0; i < N; i++) {
109                 board[i] = (char*)malloc(N * sizeof(char));
110         }
111         makeBoard(board);
112         clock_t start_serial = clock();
113         for (int i=1;i<=N*N;i++){
114                 kkn(i, 0, 0, board);
115                 makeBoard(board);
116         }
117         clock_t end_serial = clock();
118         int serial_count = Bcnt;
119         Bcnt = 0;
120         double serial_time = ((double)end_serial - start_serial) / CLOCKS_PER_SEC;
121         makeBoard(board);
122         clock_t start_parallel = clock();
123         #pragma omp parallel num_threads(T)
124         {
125                 #pragma omp for schedule(dynamic)
126                 for (int i=1;i<=N*N;i++){
127                         // #pragma omp parallel
128                         {
129                                 // #pragma omp task
130                                 {
131                                         kkn_parallel(i, 0, 0, board);
132                                         makeBoard(board);
133                                 }
134                         }
135         }
```

Output:

Task 1:

```
Board Id: 723
. . . . Q . . . . .
. . . . . . Q . . .
. . . Q . . . . . .
. . . . . . . . . Q
. . Q . . . . . . .
. . . . . Q . . . .
. . . . . . . . Q .
. Q . . . . . . . .
. . . . . . . Q . .
Q . . . . . . . . .
Serial execution time: 0.012474 seconds
Parallel execution time: 0.000622 seconds
Serial-to-Parallel Time Ratio: 20.05
```

Task 2:
For Maximum:

```
Board Id: 0
K K K K K
K K K K K
K K K K K
K K K K K
K K K K K
Board Id: 1
. K . K .
. K . K .
. K . K .
. K . K .
. K . K .
Serial board count: 2
Parallel board count: 2
Serial execution time: 0.000082 seconds
Parallel execution time: 0.000013 seconds
Serial-to-Parallel Time Ratio: 6.31
```

For All Possible Combination:

```
Board Id: 1363
 .  K  .  K
 K  .  K  .
 .  K  .  K
 K  .  K  .

Serial board count: 1364
Parallel board count: 1364
Serial execution time: 0.012815 seconds
Parallel execution time: 0.003038 seconds
Serial-to-Parallel Time Ratio: 4.22
```

Conclusion:

- Utilized OpenMP for parallelizing Queen placement on a 10×10 chessboard.
- Explored static, dynamic, guided, auto, and runtime scheduling options for load balancing.
- Investigated the maximum number of non-conflicting Horse placements (2 for N>1 and 1 for N=1).
- Analyzed time complexity and execution time for solutions.
- Implemented dynamic memory allocation for efficient array usage.
- Ensured proper memory deallocation to prevent memory leaks.