Programming Assignment 2

Han Dongmin, Yuanlai Zhou, Chong Ye

March 7, 2019

1   Problem Statement

In this programming assignment, we need to develop a parallel back-tracking algorithm to place n-queens on a $n*n$ chessboard according to rules. Queens are not allowed to be placed in the same rows/columns or diagonal directions. Both sequential and parallel algorithms need to be designed to find all the correct configurations of the placement. One solution to the problem are represented by an array of n numbers, $sol[0, 1, 2, ..., n-1]$, where $sol[i]$ represents the column where the queen in row i is placed.

2   Algorithm

2.1   seq_solver

We implemented recursive back-tracking algorithm in the sequential calculation. There are two helper functions used in here, *isValid* and *actual_seq_solver*.

Function *isValid* determines if the current placement would threaten the queens already placed. Specifically, if we are planning to place the queen of row i in column j, we need to check if the queen will threaten the queens of row $0, 1, 2, ..., i-2$ which are already placed. If not, the function would return *true*. otherwise, return *false*.

Function *actual_seq_solver* implements the back-tracking algorithm to place the n-queens.The queens are placed row by row, after placing the queen of row i, the function proceeds to place the queen of row $i+1$. After a solution is found, the function proceeds to find another feasible placement until all the correct configurations are found.

2.2   nqueen_master

Master processor (rank 0) implements backtracking to calculate the first k queens' positions, and sends out the tasks of solving the remaining solutions to this sub-problem S_k to the workers. The master dispatches one task at a time to available workers. While waiting for the responses from workers, master prepares the next feasible configuration of placing the first k queens for efficiency. After the worker finishes its work, master dispatches another

partial solution to this available worker till all the partial solutions are found. After all the partial solutions are found and all the workers finish their jobs, master sends out the termination signal to terminate the parallel running.

Two main helper functions are defined here, *next_partial_solution* and *check_worker_idle*. The helper function *next_partial_solution* is used to find the next partial solutions by backtracking algorithm, if a partial solution is found, returns *true*. Then the partial solution would be dispatched to the available worker. The function returns *false* after all the partial solutions are found. The function *check_worker_idle* returns true if all the workers have finished the jobs. Then the master sends out the termination signal to terminate the parallel running.

2.2.1 nqueen_worker

Workers receive two kinds signals from the master, one is partial solution of placing the first k queens, the other is the termination signal. If the received signal is partial solution, then the worker needs to finish placing the remaining $n-k$ queens. The helper function *worker_solver* is defined to find the remaining solutions to the subproblem. The *worker_solver* continues to place the queen of row k to row $n-1$. As soon as one solution is found, the worker sends out the solution to master and then checks if there are other solutions. If all the remaining solutions are found the the subproblem, the worker sends out job finished signal to master and waits for another job.

If the work receives termination signal from master, the function would just return.

3 Optimization

3.1 edge cases

1. $p = 1$
   When we only use one processor, it means sequential computing.

2. $k = n$
   It means the master processor calculates the solutions of placing n queens, which simply represents the full solutions. Thus we use sequential algorithm for this case.

3. $k = 0$

3.2 computation-communication overlap

We use non-blocking **MPI_Irecv** at the master processor side to receive the responses from workers. In this case, the master processor could prepare the next partial solution and have it ready while waiting for the response from workers. As soon as the worker returns solutions and is available, the master hands out the prepared partial solution to the worker, saving the worker's idle waiting time.

4   Implementation and Analysis

1. vary n choose k $\geq$ 8

2. vary p n not too low or too high

3. vary k $n > 10$

5   Results and Conclustion