Vectorized Method for Solving the n-queens Problem using Bohrium

Larsen, Mads Ohm

Publication date: 2016

Document Version
Publisher's PDF, also known as Version of record

Citation for published version (APA):
Vectorized Method for Solving the $n$-queens Problem using Bohrium

Mads Ohm Larsen <ohm@nbi.ku.dk>

Introduction

On a normal $8 \times 8$-chessboard we find that 8 queens can be positioned in
different legal ways. This is too many to brute force, especially as we go for larger $n$.

For the $8 \times 8$-queens problem we only have 92 distinct solutions.

Backtracking Algorithm

Since only one queen are allowed in each row and column, we can skip placing another queen in the same column or row in the backtracking algorithm.

Figure 1 shows the “identity” chessboard, with a queen in each row and column. It is of course not a solution, but permuting the rows will eventually grant the 92 solutions.

Figure 2: The $4 \times 4$ “identity” chessboard.

If we do the same calculations for $n = 4$ as we did for $n = 8$, we get that we have
greater than 1. If not, then we have a solution.

Gathering all the traces (the sums along the diagonals and offset-diagonals) for the $4 \times 4$ identity board we get

Here we see 4, which means that the identity board isn’t a solution to the four-queens problem. In fact, the maximum trace must be equal to 1 to be a solution.

Adding a Dimension

Now that we know how to solve one board, we can create all permutations of that board, to solve the entire $n$-queens problem for some $n$.

We can do this, by adding a dimension to our matrices above, creating a tensor of $4 \times 4$-chessboards.

Doing so yields

boards, which all needs to be checked. Generalizing for the $n$-queens problem, this would be

For the four-queens problem, this yields $4^4 = 256$ traces of which we only need to count the number of 1s. As we have seen already, there is only two solutions, and we only have two max-traces which are 1.

Note that since there is always queens in the diagonals or offset-diagonals, we can never have a max-trace of zero value.

Using NumPy and Bohrium

With our knowledge, we can construct a simple Python program using NumPy or Bohrium to calculate how many distinct solutions are to the $n$-queens problem for a given $n$. In this program we simply list all permutations of the $n \times n$-chessboard, flip it, then trace all diagonals from $-n$ to $n$ and max that matrix into a 10 vector. The result is then found by counting this in vector.

Future Work

Unfortunately we do not currently gain any performance boost using Bohrium, however this is largely due to the creation of the permutation boards.

We are planning to implement a permutation generator into the Bohrium runtime as a streaming generator for the GPU, and will hopefully get a performance speed-up doing so.

```python
import numpy as np

from itertools import permutations

def nqueens(n):
    # Generate all permutations of n-identity boards
    boards = list(permutations(np.eye(n)))

    # Attack the flipped boards as another dimension
    rotation_boards = np.array((boards, np.flip(boards)))

    # Calculate all the traces, from -n to n for all the boards
    n = np.max(n
    np.trace(
        rotation_boards,
        1,
        axis1=0,
        axis2=-1
    ) for i in range(n, n)
    ], axis=0, 1)

    # Count the number of 1s in the maximum of the traces
    solutions = np.sum(np.max(solutions, axis=1) == 1)

    return solutions

nqueens(8)  # => 92 solutions for 8 by 8 board.
```