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

$$\binom{64}{8} = \frac{64!}{8!56!} \approx 4.43 \times 10^{15}$$
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.

```
# import bohrium as np
import numpy as np
from itertools import permutations

def nqueens(n):
    # Generate all permutations of an identity boards
    boards = list(permutations(range(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([
        np.trace(rotation_boards,
                 axis1=-2, axis2=-1)
     for i in range(n, n)])

    # Count the number of Is in the maximum of the traces
    return np.sum(n > 1)  # solutions for n, "by", n, "board"
```

Doing so yields

$$4 \cdot 4 \cdot 4 = 64$$

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

$$n^2 \cdot n!$$

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, than trace all diagonals from $-n$ to $n$ and max that matrix into a 10 vector. The result is then found by counting is in this 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.