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

$$\binom{64}{8} = \frac{64!}{8! \cdot (64-8)!} = 4,261,544,069,683,280$$
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.

![Diagram of backtracking algorithm](Image)

For the four-queens problem there are only two solutions. These are shown in figure 3.

![Diagram of four-queens solutions](Image)

Normally we count two different types of solutions, distinct and fundamental. The distinct solutions do not take reflection and rotation into consideration, but the fundamental solutions do. For $n \times n$ we have 92 distinct solutions, but only 12 fundamental.

**Matrix Representation**

One solution for the four-queens problem can be represented as the following matrix:

$$\begin{bmatrix}
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 \\
\end{bmatrix}$$

To check if this is a solution, we can sum along each column, row, and diagonal, checking if the sum of any of these are 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

$$\begin{bmatrix}0,0,0,1,0,0,1,0,0,0,1,0,0,0,1,0,0,0,0,1\end{bmatrix}$$

Here we see a 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

$$4 \times 4 \times 4 = 64$$

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

$$n^2 \times n^2 \times n^2$$

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 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.