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 $n \times n$-chessboard we find that 8 queens can be positioned in

\[
\binom{n^2}{n} = \frac{n^2!}{n! (n-1)!} \approx 4,26,165,385
\]
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.

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 is greater than 1. If not, then we have 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 $4 \times 4 \times \ldots$-chessboards.

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.

Image 2051x3024 to 2267x3318