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.

Using Permutations
Instead of looking for the right combinations, we can instead compute permutations
of rows, with a queen in each column. There are
permutations of a $8 \times 8$-chessboard’s rows.
Because we construct the rows with a
queen in each column, we now know we only have to check the diagonals, to see
if the current permutation is a solution.

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

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.

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 $n$.

Future Work
Unfortunately we do not currently gain any performance boost using Bohrium, however this is largely due to the cre-
ation of the permutation boards.
We are planning to implement a permu-
tation generator into the Bohrium run-
time as a streaming generator for the
GPU, and will hopefully get a perform-
ce speed-up doing so.

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.

Using Permutations
Instead of looking for the right combinations, we can instead compute permutations
of rows, with a queen in each column. There are
permutations of a $8 \times 8$-chessboard’s rows.
Because we construct the rows with a
queen in each column, we now know we only have to check the diagonals, to see
if the current permutation is a solution.

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

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.

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 $n$.

Future Work
Unfortunately we do not currently gain any performance boost using Bohrium, however this is largely due to the cre-
ation of the permutation boards.
We are planning to implement a permu-
tation generator into the Bohrium run-
time as a streaming generator for the
GPU, and will hopefully get a perform-
ce speed-up doing so.

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.

Using Permutations
Instead of looking for the right combinations, we can instead compute permutations
of rows, with a queen in each column. There are
permutations of a $8 \times 8$-chessboard’s rows.
Because we construct the rows with a
queen in each column, we now know we only have to check the diagonals, to see
if the current permutation is a solution.

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

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.

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 $n$.

Future Work
Unfortunately we do not currently gain any performance boost using Bohrium, however this is largely due to the cre-
ation of the permutation boards.
We are planning to implement a permu-
tation generator into the Bohrium run-
time as a streaming generator for the
GPU, and will hopefully get a perform-
ce speed-up doing so.