A Brute-Force Solution to the 27-Queens Puzzle
Using a Distributed Computation

Dr.-Ing. Thomas B. Preuß
thomas.preusser@utexas.edu
Accemic Technologies
CMC19, Sep 2018, Dresden
Itinerary

- The Puzzle - An Overview
- Algorithmic Approaches
- Programmable Hardware - FPGAs
- Implementing the Q27 Computation
- Some Take-Aways / Discussion
The $N$-Queens Puzzle

- Place $N$ non-attacking Queens on an $N \times N$ chessboard:

- Construction of a solution?
- Completion of a solution?

→ How many (fundamental) solutions are there in total?
Some Background

- Postulated by *Max Friedrich Wilhelm Bezzel* in 1848.
- Solution count of the $8 \times 8$ puzzle by 1850.
- Up to $11 \times 11$ solved *manually* by 1890.
- Becomes a standard computer science problem in the 1970s:
  - Benchmarking advocating platforms, concurrency, AI algorithms, ...
- Conjecture by *Benoit Cloitre*:

  \[ Q(n) \rightarrow \frac{n!}{c^n} \quad \text{for} \quad n \rightarrow \infty, \ c \approx 2.54 \]
Sources to Delve Into

- Matthias Engelhardt: http://nqueens.de/ - Overview, Variations, ...
- N. J. A. Sloane: https://oeis.org/A000170 - The Known Sequence.
Motivation

The exploration of an $N$-Queens Puzzle is:
- an embarrassingly parallel,
- easily scalable,
- fiercely compute-bounded workload.

It serves for:
- algorithmic illustration,
- training target for efficient algorithms and implementations, and
- demonstrating the performance of compute equipment.

And yes: *We just can!*
Known Solution Counts

<table>
<thead>
<tr>
<th>$N$</th>
<th>Solutions</th>
<th>$N$</th>
<th>Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>14</td>
<td>365596</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>15</td>
<td>2279184</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>16</td>
<td>14772512</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>17</td>
<td>95815104</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>18</td>
<td>666090624</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>19</td>
<td>4968057848</td>
</tr>
<tr>
<td>7</td>
<td>40</td>
<td>20</td>
<td>3902918884</td>
</tr>
<tr>
<td>8</td>
<td>92</td>
<td>21</td>
<td>31466622712</td>
</tr>
<tr>
<td>9</td>
<td>352</td>
<td>22</td>
<td>2691008701644</td>
</tr>
<tr>
<td>10</td>
<td>724</td>
<td>23</td>
<td>24233937684440</td>
</tr>
<tr>
<td>11</td>
<td>2680</td>
<td>24</td>
<td>227514171973736</td>
</tr>
<tr>
<td>12</td>
<td>14200</td>
<td>25</td>
<td>2207893435808352</td>
</tr>
<tr>
<td>13</td>
<td>73712</td>
<td>26</td>
<td>22317699616364044</td>
</tr>
<tr>
<td></td>
<td></td>
<td>27</td>
<td>234907967154122528</td>
</tr>
</tbody>
</table>

Sep 19, 2016:

Exhaustive backtracking of solution space requires factorial time $O(N!)$. Very hard beyond $N = 20$.

$N = 25$:

- Java grid computation by INRIA, France.
- Runtime:
  - Real $>6$ Months
  - Sequential $>53$ Years
### Known Solution Counts

<table>
<thead>
<tr>
<th>$N$</th>
<th>Solutions</th>
<th>$N$</th>
<th>Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>14</td>
<td>365596</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>15</td>
<td>2279184</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>16</td>
<td>14772512</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>17</td>
<td>95815104</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>18</td>
<td>666090624</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>19</td>
<td>4968057848</td>
</tr>
<tr>
<td>7</td>
<td>40</td>
<td>20</td>
<td>3902918884</td>
</tr>
<tr>
<td>8</td>
<td>92</td>
<td>21</td>
<td>314666222712</td>
</tr>
<tr>
<td>9</td>
<td>352</td>
<td>22</td>
<td>2691008701644</td>
</tr>
<tr>
<td>10</td>
<td>724</td>
<td>23</td>
<td>24233937684440</td>
</tr>
<tr>
<td>11</td>
<td>2680</td>
<td>24</td>
<td>227514171973736</td>
</tr>
<tr>
<td>12</td>
<td>14200</td>
<td>25</td>
<td>2207893435808352</td>
</tr>
<tr>
<td>13</td>
<td>73712</td>
<td>26</td>
<td>22317699616364044</td>
</tr>
</tbody>
</table>

**Sep 19, 2016:**

<table>
<thead>
<tr>
<th>$N$</th>
<th>Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>234907967154122528</td>
</tr>
</tbody>
</table>

Exhaustive backtracking of solution space requires factorial time $O(N!)$. Very hard beyond $N = 20$. **$N = 26$:**

- 9-month computation on FPGAs completing Jul 11, 2009.
- Result confirmed by Russian MC# super computing project on Aug 30, 2009.
### Known Solution Counts

<table>
<thead>
<tr>
<th>$N$</th>
<th>Solutions</th>
<th>$N$</th>
<th>Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>14</td>
<td>365596</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>15</td>
<td>2279184</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>16</td>
<td>14772512</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>17</td>
<td>95815104</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>18</td>
<td>666090624</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>19</td>
<td>4968057848</td>
</tr>
<tr>
<td>7</td>
<td>40</td>
<td>20</td>
<td>3902918884</td>
</tr>
<tr>
<td>8</td>
<td>92</td>
<td>21</td>
<td>31466622712</td>
</tr>
<tr>
<td>9</td>
<td>352</td>
<td>22</td>
<td>2691008701644</td>
</tr>
<tr>
<td>10</td>
<td>724</td>
<td>23</td>
<td>2423393768440</td>
</tr>
<tr>
<td>11</td>
<td>2680</td>
<td>24</td>
<td>227514171973736</td>
</tr>
<tr>
<td>12</td>
<td>14200</td>
<td>25</td>
<td>2207893435808352</td>
</tr>
<tr>
<td>13</td>
<td>73712</td>
<td>26</td>
<td>22317699616364044</td>
</tr>
</tbody>
</table>

Sep 19, 2016: 27 234907967154122528

Exhaustive backtracking of solution space requires factorial time $O(N!)$.

Very hard beyond $N = 20$.

$N = 27$:

- More than 12 months of a distributed FPGA computation.
- Thorough symmetry exploitation.
- Result not yet confirmed by others.
Tackling $N = 26$

- Embarrassingly parallel workload:
  1. Preplace $L \ll N$ columns.
  2. Explore subboards **independently**.
  3. Collect and add up subtotals.

- Ideally suited for distributed computation:
  - Internet (BOINC) $\rightarrow$ NQueens@Home
  - FPGA! $\rightarrow$ Queens@TUD

- Challenged the power of a world-wide distributed computation effort by an intelligent FPGA implementation.
- Identified and reported a 32-bit integer overflow bug in Nov 2008.
- Thereby, resolved an open dispute on the solution for $N = 24$ *without* any own computation.
Partial Solution Space

Problem Size $N$

- 7
- 11
- 15
- 19

Partial Solutions (scaled to max = 1) vs. $L/N$
Algorithmic Overview

Exhaustive backtracking solution exploration.

Yes 1. Mark explored.
2. Update blocking vectors.
3. Advance to next column / Count solution.

No 1. Clear markings.
2. Retreat to previous column / Done.

valid placement yet to explore?

Computing Blocking Vectors avoids frequent constraint validation.
What are Those FPGAs?

Meta-circuits for implementing digital logic \textit{structurally}.

1. The circuit grid:
What are Those FPGAs?

Meta-circuits for implementing digital logic *structurally*.

2. is configurable *in the field*:

![Diagram of a LUT with configuration bit (SRAM cell)]
Carry Chain Structures

Spartan / Virtex Manchester Carry Chain

Stratix II+ Designated Ripple Carry Adder

Carry chains are implemented to speed up binary word addition.
FPGA Mapping

Using carry chains to process one column in one *fast* clock cycle.

Thomas Preußer
27-Queens Puzzle by Brute-Force
Generic Carry-Chain Mapping through Addition

1. **Derive Carry / Token Propagation**

<table>
<thead>
<tr>
<th>Case</th>
<th>$c_{i+1}$</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_i$: Kill</td>
<td>0</td>
<td>never a carry: holding no queen and not blocked</td>
</tr>
<tr>
<td>$p_i$: Propagate</td>
<td>$c_i$</td>
<td>pass a carry: holding no queen but blocked</td>
</tr>
<tr>
<td>$g_i$: Generate</td>
<td>1</td>
<td>always a carry: holding current queen placement</td>
</tr>
</tbody>
</table>

2. **Determine Addends**

   
   \[ a_i = g_i + p_i \]
   
   \[ b_i = g_i \]

3. **Infer Token from Sum** $s \leq a + b$

   In equations dependent on the incoming carry / token use:

   \[ c_i = s_i \oplus p_i \]

   Shown mapping to Xilinx devices uses optimized implementation.
Concrete Carry-Chain Mapping

<table>
<thead>
<tr>
<th>( blocked_i )</th>
<th>( Q_i )</th>
<th>Target Function</th>
<th>Chosen Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>( t_i )</td>
<td>( t_i )</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>( t_i )</td>
<td>( t_i )</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>( t_i )</td>
<td>( t_i )</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>
Optimization for a small size and a high clock frequency:

- Maintain a single active column.
- Keep placed columns within a plain array of shifted registers.
- Use global blocking vectors for all rows and diagonals setting and unsetting placements and retreats, respectively.

Note that this is quite expensive in software!
Solver Overview
Designing $N = 27$

- Column-based pre-placement may exploit line symmetry to cut search space in half.
Designing $N = 27$

- Column-based pre-placement may exploit line symmetry to cut search space in half.
- The dihedral group $D_4$ of the symmetry of the square offers more:

![Diagram of dihedral group $D_4$]

Thomas Preußer
27-Queens Puzzle by Brute-Force
Coronal Pre-Placements

...can express this symmetry.
Coronal Pre-Placements

...can express this symmetry.
Coronal Pre-Placements

...can express this symmetry.

Advantage:
- Search space reduced to an eighth.

Challenges (solved):
- Define *canonical* representative.
- Count solutions of self-symmetric pre-placements correctly.

2.024.110.796 coronal pre-placements for \( N = 27 \).
Solve one per second: 64 years of sequential computation time.
Pre-Placement: Canonical Representative

Minimum as determined by lexicographic order of \textit{traits}.
Project Infrastructure

- **Host**
- **Server**
- **FPGA**
- **UART**
- **FIFOs**
- **Solvers #0, #1, ...**
- **DB**
Scalability

Inherently computation bounded: subproblem solution is encoded in 21 bytes only.
Current peaks at 120 solutions per second, i.e. 25kBit/s of net payload.
Assuming a 100% protocol overhead, exhausting a mature 100 MBit/s interface at the server side would imply that we are completely done in 2.5 hours.
## Contributing Devices

<table>
<thead>
<tr>
<th>Board</th>
<th>Device</th>
<th>Solvers</th>
<th>Clock</th>
<th>SE</th>
</tr>
</thead>
<tbody>
<tr>
<td>VC707</td>
<td>XC7VX485T-2</td>
<td>325</td>
<td>250.0 MHz</td>
<td>812</td>
</tr>
<tr>
<td>KC705</td>
<td>XC7K325T-2</td>
<td>241</td>
<td>290.4 MHz</td>
<td>700</td>
</tr>
<tr>
<td>ML605</td>
<td>XC6VLX240T-1</td>
<td>125</td>
<td>200.0 MHz</td>
<td>250</td>
</tr>
<tr>
<td>DE4</td>
<td>EP4SGX230KF40C2</td>
<td>125</td>
<td>250.0 MHz</td>
<td>312</td>
</tr>
<tr>
<td>DNK7_F5_PCIE</td>
<td>5× XC7K325T-1</td>
<td>5× 240</td>
<td>220.0 MHz</td>
<td>2640</td>
</tr>
</tbody>
</table>

SE (Solver Equivalent): one solver unit running at 100 MHz
Emergent Patterns: Solution Counts

(First 1000 lexicographically ordered subproblems)
Computational Snapshot

![Graph showing solutions per second over time from 2015-12-22 to 2015-12-26. The x-axis represents time, and the y-axis represents solutions per second. The graph shows fluctuations in the solutions per second over the period.]
Other Great Algorithmic Fits

- Stream-based computations:
  DSP, online data recoding & analysis.
- Reduced-precision acceleration: QNNs.

- FPGAs enable concurrency one or two orders of magnitude higher than GPUs.
- Make your algorithms fit!
Take Aways

We are in the lucky position to be able to handle great computational challenges. Knowing your options helps:

- Embrace concurrency: multi-core, multi-processor, clusters, cloud.
- Embrace heterogeneity: CPU, GPU, FPGA, CPU+FPGA (Zynq), ...
- Forge your algorithms: latency independence, data locality.
- Know your arithmetic: bits, ints, floats.

Platforms are easily available, even FPGAs:
  - AWS EC2 F1 with very powerful Virtex UltraScale+ (VU9P) FPGAs.
Thank you!

The complete Q27 infrastructure implementation is available as open source:

https://github.com/preusser/q27