TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science Institute for Computer Engineering

Putting Queens in Carry Chains – No. 27 –

Dr.-Ing. Thomas B. Preußer

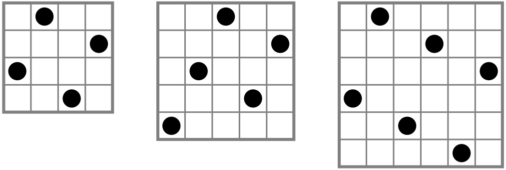
thomas.preusser@tu-dresden.de

Itinerary

- Problem and Complexity Overview
- Solution Approach
- Hardware Mapping and Optimization
- Ongoing Computation for $N = 27$

The N-Queens Puzzle

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



- Generic solution templates?

$\rightarrow$ How many (fundamental) solutions?

Motivation

The exploration of an N-Queens Puzzle is:

- an embarrassingly parallel,
- easily scalable,
- computation-bounded

workload.

It serves as:

- a training object for working with cramped designs:
  - efficient coding and resource utilization,
  - tooling and parameter exploration.
- a tool and device benchmark.

And yes: *We just can!*

Known Solution Counts

N	Solutions
1	1
2	0
3	0
4	2
5	10
6	4
7	40
8	92
9	352
10	724
11	2680
12	14200
13	73712

N	Solutions
14	365596
15	2279184
16	14772512
17	95815104
18	666090624
19	4968057848
20	39029188884
21	314666222712
22	2691008701644
23	24233937684440
24	227514171973736
25	2207893435808352
26	22317699616364044

Exhaustive backtracking solution exploration 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

N	Solutions
1	1
2	0
3	0
4	2
5	10
6	4
7	40
8	92
9	352
10	724
11	2680
12	14200
13	73712

N	Solutions
14	365596
15	2279184
16	14772512
17	95815104
18	666090624
19	4968057848
20	39029188884
21	314666222712
22	2691008701644
23	24233937684440
24	227514171973736
25	2207893435808352
26	22317699616364044

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

Very hard beyond $N = 20$.

$\underline{N = 26\textbf{:}}$

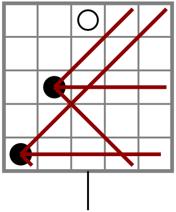
- 9-month computation on FPGAs completing July 11, 2009.
- Result confirmed by Russian MC# super computing project on August 30, 2009.

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
    - Challenge the power of a world-wide distributed computation effort by an intelligent FPGA implementation.
    - Identified and reported an overflow bug on November 7, 2008.
    - Thereby, resolved an open dispute on the solution for $N = 24$ *without* any own computation.

Algorithmic Overview

Exhaustive backtracking solution exploration.



valid placement yet to explore?

Yes

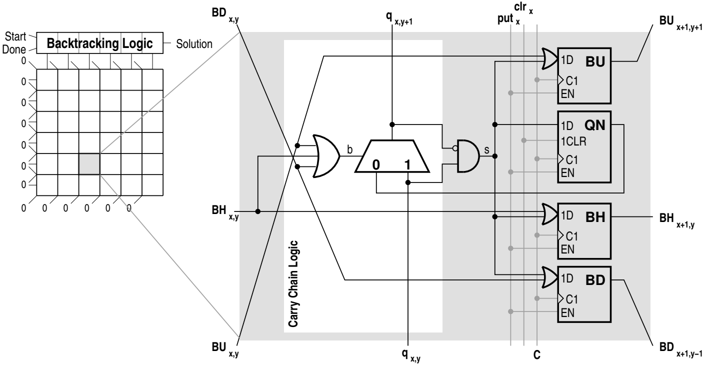
1. Mark *explored.*
2. Update *blocking vectors*.
3. Advance to next column / Count solution.

No

1. Clear markings.
2. Retreat to previous column / Done.

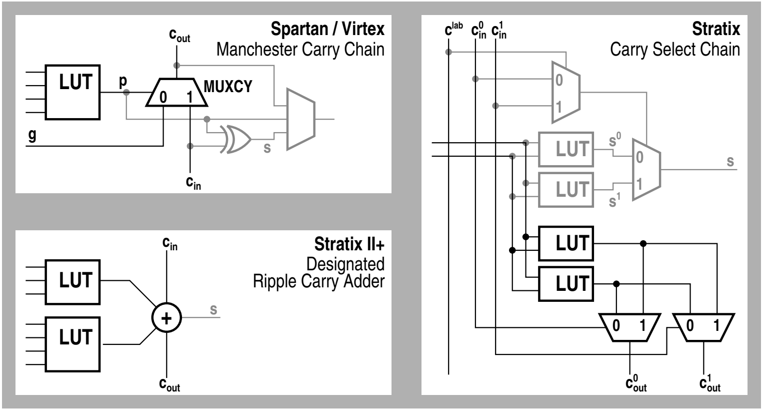
Computing *Blocking Vectors* avoids frequent constraint validation.

FPGA Mapping



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

Carry Chain Structures



Carry chains are implemented to speed up binary word *addition*.

Generic Carry-Chain Mapping through Addition

1. **Derive Carry / Token Propagation**

Case	c_{i+1}	Description
k_i: Kill	0	never a carry: holding no queen and not blocked
p_i: Propagate	c_i	pass a carry: holding no queen but blocked
g_i: Generate	1	always a carry: holding current queen placement

2. **Determine Addends**

$$\begin{aligned} a_i &= g_i + p_i \\ b_i &= g_i \end{aligned}$$

3. **Infer Token from Sum** $s <= a + b$
   In equations dependent on the incoming carry / token use:

$$c_i = s_i \oplus p_i$$

Shown mapping to Xilinx devices uses optimized implementation.

Pushing Performance

Optimization for a small size and a high clock frequency:

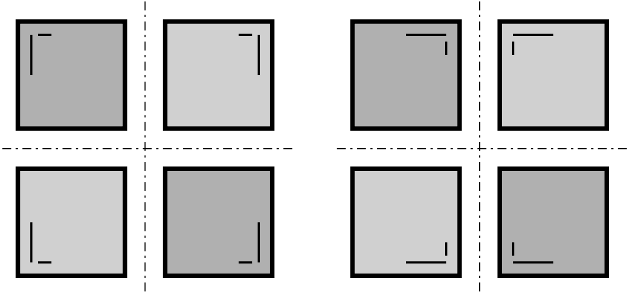
- Maintain a single active column.
- Keep placed columns within a plain array of shifted registers.
- Use global blocking vectors for all rows and diagonals setting and unsetting placements and retreats, respectively.
  Note that this is quite expensive in software!

Designing $N = 27$

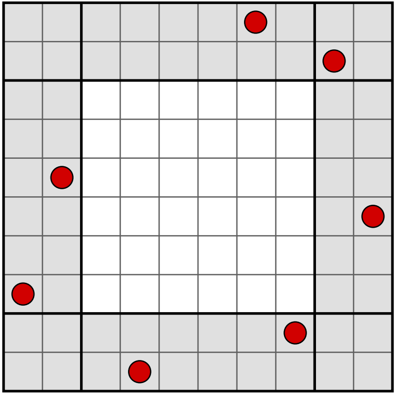
Column-based pre-placement may exploit line symmetry to cut search space in half.

Designing $N = 27$

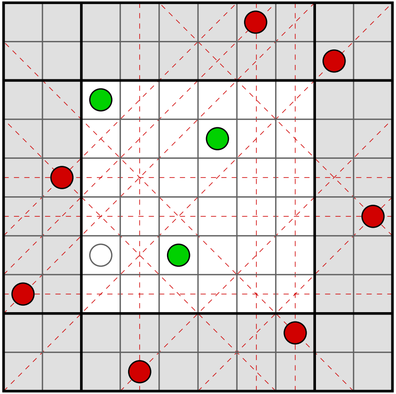
Column-based pre-placement may exploit line symmetry to cut search space in half.
There is more:



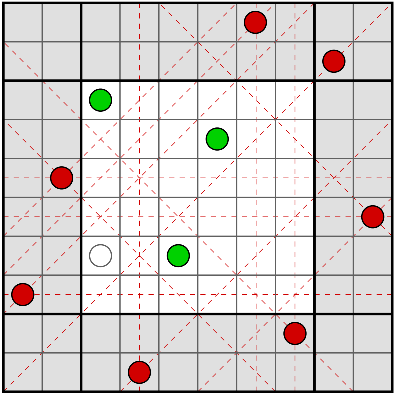
Coronal Pre-Placement



Coronal Pre-Placement



Coronal Pre-Placement



Advantage:

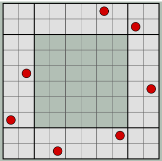
- Search space reduced to an eighth.

Challenges (solved):

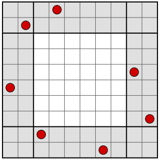
- Define *canonical* representative.
- Count solutions of self-symmetric pre-placements correctly.

2.024.110.796 coronal pre-placements for $N = 27$.
Solve one per second: 64 years of sequential computation time.

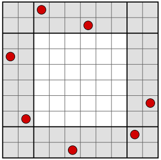
Pre-Placement: Canonical Representative



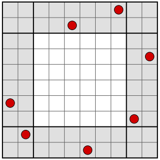
$(2, 5; 6, 8; 5, 1; 6, 2)$



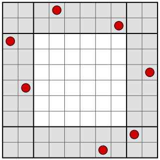
$(4, 8; 3, 1; 7, 4; 3, 7)$



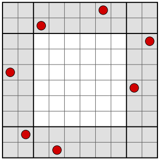
$(6, 2; 2, 5; 6, 8; 5, 1)$



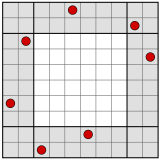
$(3, 1; 7, 4; 3, 7; 4, 8)$



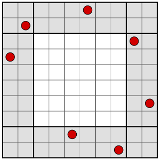
$(7, 4; 3, 7; 4, 8; 3, 1)$



$(5, 1; 6, 2; 2, 5; 6, 8)$



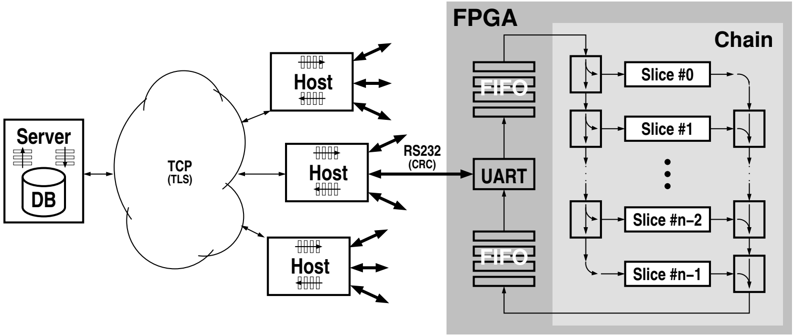
$(3, 7; 4, 8; 3, 1; 7, 4)$



$(6, 8; 5, 1; 6, 2; 2, 5)$

Minimum as determined by lexicographic order of *traits*.

Project Infrastructure



Scalability

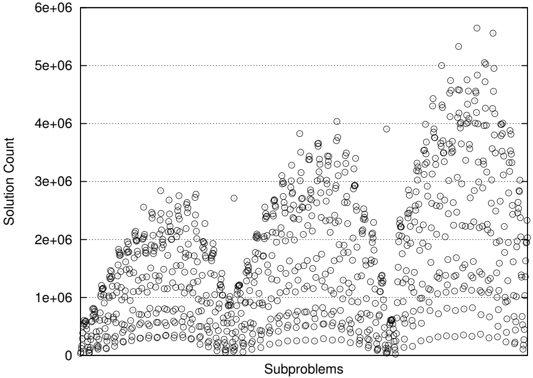
Inherently computation bounded: subproblem solution is encoded in 21 bytes only.
Current peaks at 25 solutions per second, i.e. 4.2 kBit/s of net payload.
Assuming a 100% protocol overhead, exhausting a mature 100 MBit/s interface at the server side would imply that we are completely done in 2.5 hours.

Contributing Devices

Board	Device	Solvers	Clock	SE
VC707	XC7VX485T-2	325	250.0 MHz	812
KC705	XC7K325T-2	241	290.4 MHz	700
ML605	XC6VLX240T-1	125	200.0 MHz	250
DE4	EP4SGX230KF40C2	125	250.0 MHz	312
DNK7_F5_PCIe	$5\times$ XC7K325T-1	$5\times$ 240	220.0 MHz	2640

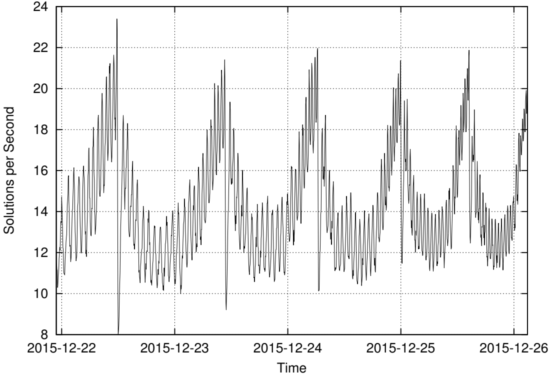
SE (Solver Equivalent): one solver unit running at 100 MHz

Emergent Patterns: Solution Counts



(First 1000 *lexicographically ordered* subproblems)

Computational Snapshot



State of Affairs

Currently:

- an average of 15 solutions per second in undisrupted operation is achieved, and
- 2.7% of the 2,024,110,796 subproblems are solved.

Ongoing Efforts:

- Use of local clock resources (`BUFR`, `BUFH`) is explored to squeeze out more performance.
- A GPU port is under development.

Thank you!

The whole implementation is available as open-source:
`https://github.com/preusser/q27`