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

Dr.-Ing. Thomas B. Preußer
thomas.preusser@utexas.edu

Accemic Technologies

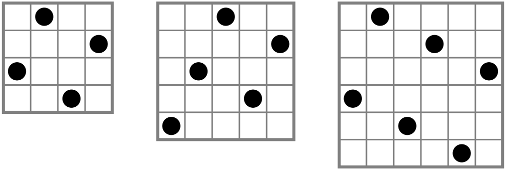
CMC19, Sep 2018, Dresden

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 8×8 puzzle by 1850.
- Up to 11×11 solved *manually* by 1890.
- Becomes a standard computer science problem in the 1970s:
  - *Edgar W. Dijkstra*: Backtracking.
  - Benchmarking advocating platforms, concurrency, AI algorithms, ...
- Conjecture by *Benoit Cloitre*:

$$Q(n) \rightarrow \frac{n!}{c^n} \quad \text{for } n \rightarrow \infty, c \approx 2.54$$

Sources to Delve Into

- Ian P. Gent et al.: Complexity of n-Queens Completion, 2017.
- Matthias Engelhardt: `http://nqueens.de/` - Overview, Variations, . . .
- Walter Kosters: `http://liacs.leidenuniv.nl/~kosterswa/nqueens/` - Bibliography.
- 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,
- fiercly 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

N	Solutions	N	Solutions
1	1	14	365596
2	0	15	2279184
3	0	16	14772512
4	2	17	95815104
5	10	18	666090624
6	4	19	4968057848
7	40	20	39029188884
8	92	21	314666222712
9	352	22	2691008701644
10	724	23	24233937684440
11	2680	24	227514171973736
12	14200	25	2207893435808352
13	73712	26	22317699616364044
Sep 19, 2016:		27	234907967154122528

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

ACCEMIC TECHNOLOGIES

Known Solution Counts

N	Solutions	N	Solutions
1	1	14	365596
2	0	15	2279184
3	0	16	14772512
4	2	17	95815104
5	10	18	666090624
6	4	19	4968057848
7	40	20	39029188884
8	92	21	314666222712
9	352	22	2691008701644
10	724	23	24233937684440
11	2680	24	227514171973736
12	14200	25	2207893435808352
13	73712	26	22317699616364044
Sep 19, 2016:		27	234907967154122528

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

Very hard beyond $N = 20$.

<u>$N = 26$:</u>

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

Known Solution Counts

N	Solutions	N	Solutions
1	1	14	365596
2	0	15	2279184
3	0	16	14772512
4	2	17	95815104
5	10	18	666090624
6	4	19	4968057848
7	40	20	39029188884
8	92	21	314666222712
9	352	22	2691008701644
10	724	23	24233937684440
11	2680	24	227514171973736
12	14200	25	2207893435808352
13	73712	26	22317699616364044
Sep 19, 2016:		27	234907967154122528

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

Very hard beyond $N = 20$.

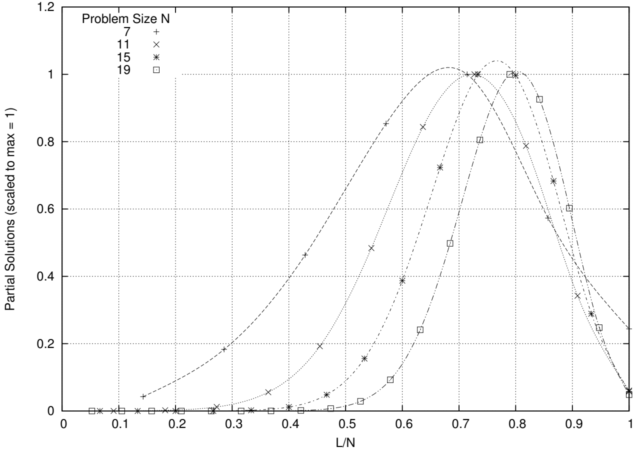
$N = 27$:

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

Tackling $N = 26$

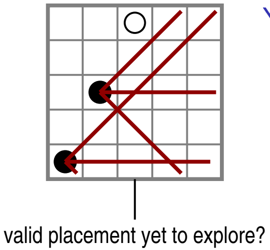
- Embarrassingly parallel workload:
  1. Preplace $L \ll N$ columns.
  2. Explore subboards **independently**.
  3. Collect and add up subtotals.
- Ideally suited for distributed computation:
  - Internet (BOINC) $\rightarrow$ NQueens@Home
  - FPGA! $\rightarrow$ Queens@TUD
    - Challenged the power 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



Algorithmic Overview

Exhaustive backtracking solution exploration.



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

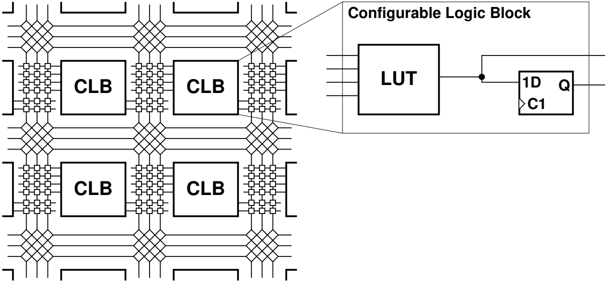
No
1. Clear markings.
2. Retreat to previous column / Done.

Computing *Blocking Vectors* avoids frequent constraint validation.

What are Those FPGAs?

Meta-circuits for implementing digital logic *structurally*.

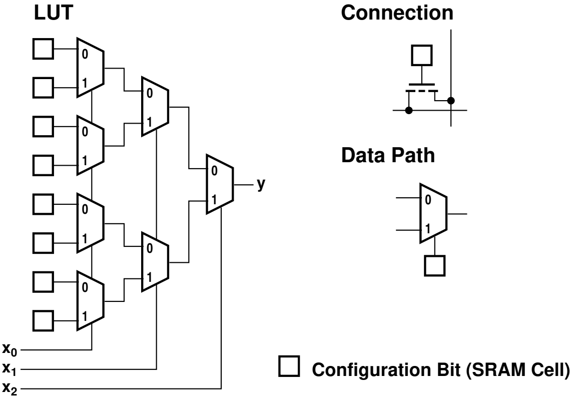
1. The circuit grid:



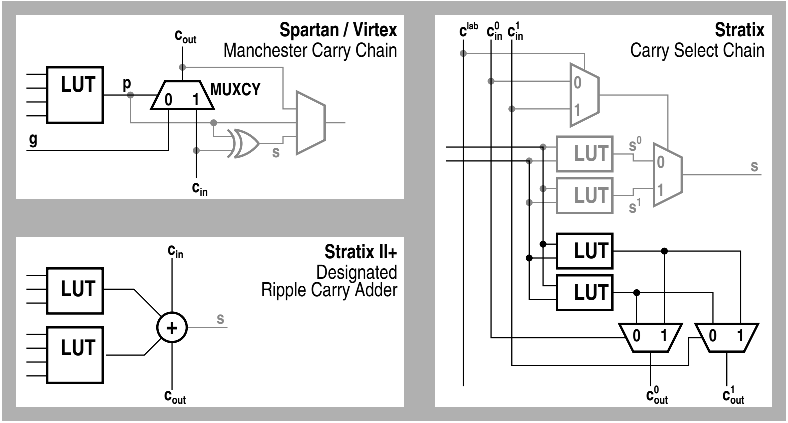
What are Those FPGAs?

Meta-circuits for implementing digital logic *structurally*.

2. is configurable *in the field*:

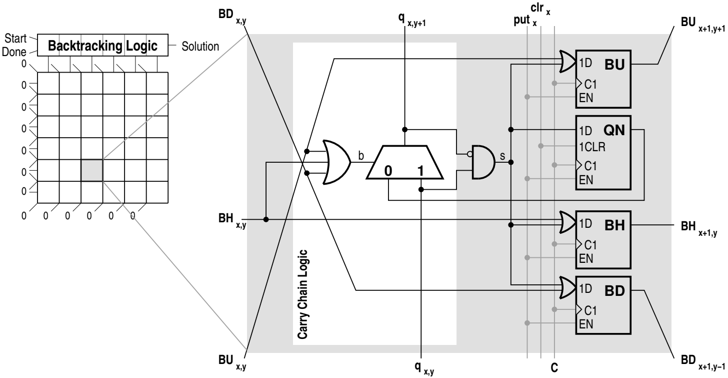


Carry Chain Structures



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

FPGA Mapping



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

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**

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

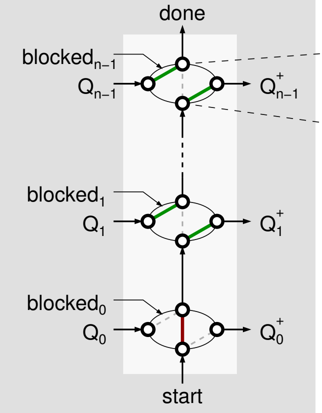
3. **Infer Token from Sum** $s <= a + b$

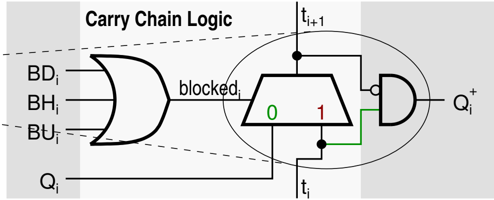
   In equations dependent on the incoming carry / token use:

$$c_i = s_i \oplus p_i$$

Shown mapping to Xilinx devices uses optimized implementation.

Concrete Carry-Chain Mapping





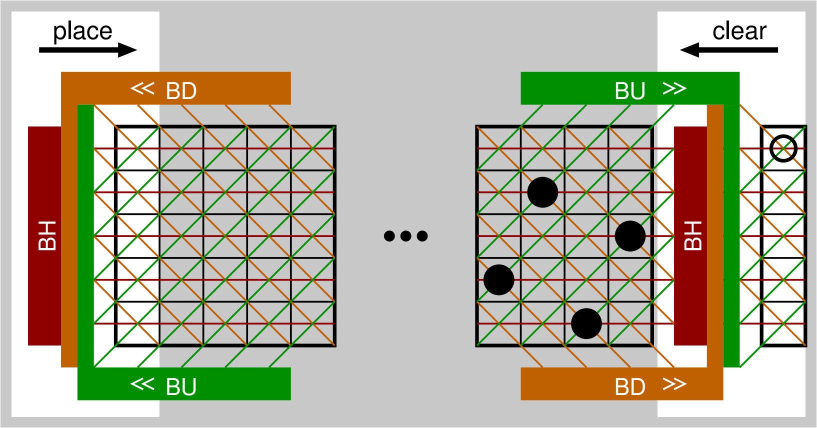
		Target Function		Chosen Implementation	
$blocked_i$	Q_i	t_{i+1}	Q_i^+	t_{i+1}	Q_i^+
0	0	K : 0	t_i	K : 0	t_i
0	1	G : 1	0	G : 1	0
1	0	P : t_i	0	P : t_i	0
1	1	$*$: $*$	$*$	P : t_i	0

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!

Solver Overview

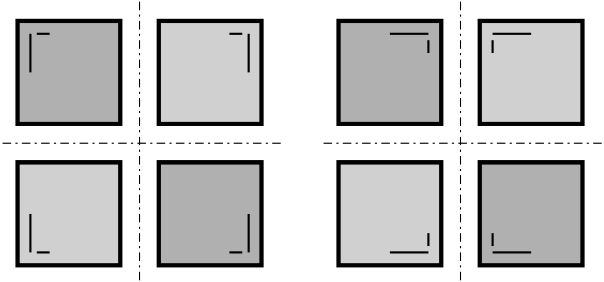


Designing $N = 27$

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

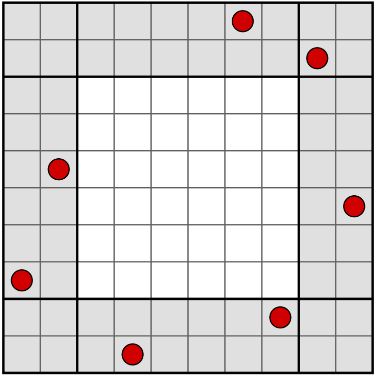
Designing $N = 27$

- 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:



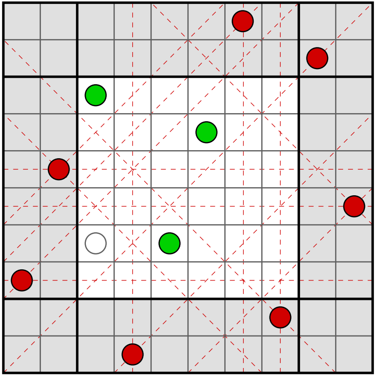
Coronal Pre-Placements

. . . can express this symmetry.



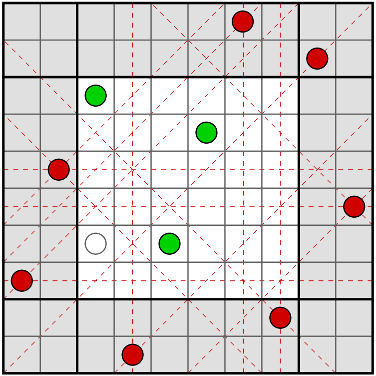
Coronal Pre-Placements

. . . can express this symmetry.



Coronal Pre-Placements

. . . can express this symmetry.



Advantage:

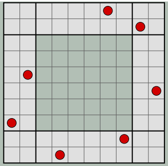
- Search space reduced to an eighth.

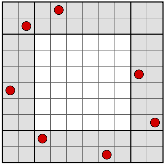
Challenges (solved):

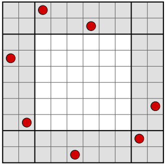
- Define *canonical* representative.
- Count solutions of self-symmetric pre-placements correctly.

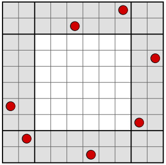
2.024.110.796 coronal pre-placements for $N = 27$.
Solve one per second: 64 years of sequential computation time.

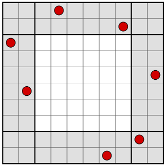
Pre-Placement: Canonical Representative

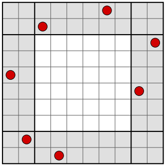

(2, 5; 6, 8; 5, 1; 6, 2)

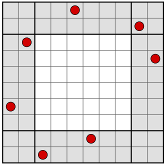

(4, 8; 3, 1; 7, 4; 3, 7)

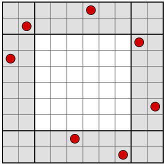

(6, 2; 2, 5; 6, 8; 5, 1)


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


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

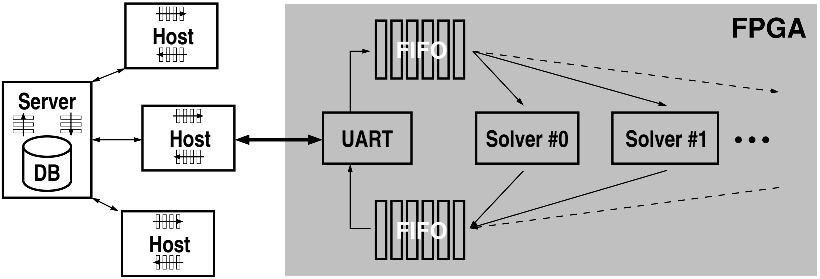

(5, 1; 6, 2; 2, 5; 6, 8)


(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 120 solutions per second, i.e. 25 kBit/s of net payload.

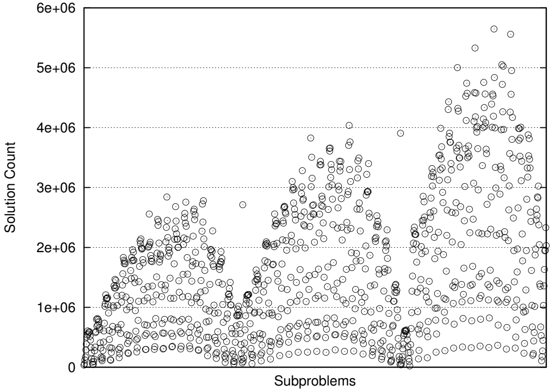
Assuming a 100% protocol overhead, exhausting a mature 100 MBit/s interface at the server side would imply that we are completely done in 2.5 hours.

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× XC7K325T-1	5× 240	220.0 MHz	2640

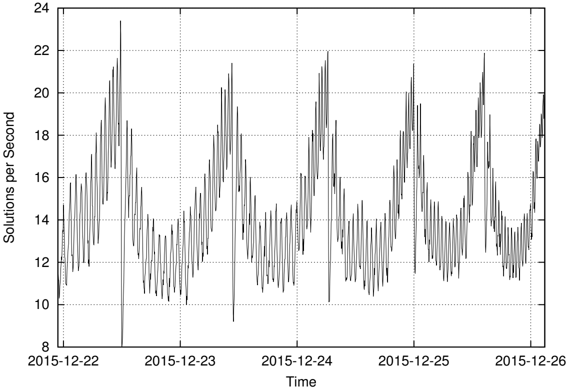
SE (Solver Equivalent): one solver unit running at 100 MHz

Emergent Patterns: Solution Counts



(First 1000 *lexicographically ordered* subproblems)

Computational Snapshot



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`