Effective prime factorization via quantum annealing by modular locally-structured embedding

Jingwen Ding, Giuseppe Spallitta, **Roberto Sebastiani**

Department of Information Engineering and Computer Science (DISI) - University of Trento

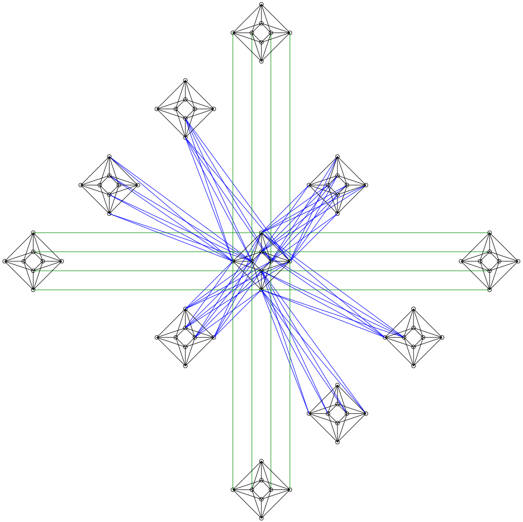
Based on the paper:
Jingwen Ding, Giuseppe Spallitta, Roberto Sebastiani.
"Effective Prime Factorization via Quantum Annealing by Modular Locally-structured Embedding" .
Nature Scientific Reports. Vol. 14, n. 3518. February 2024.
https://doi.org/10.1038/s41598-024-53708-7

Outline

1. The Pegasus Topology
2. Problem and state of the art
3. Encoding
4. Solving
5. Conclusion and future work

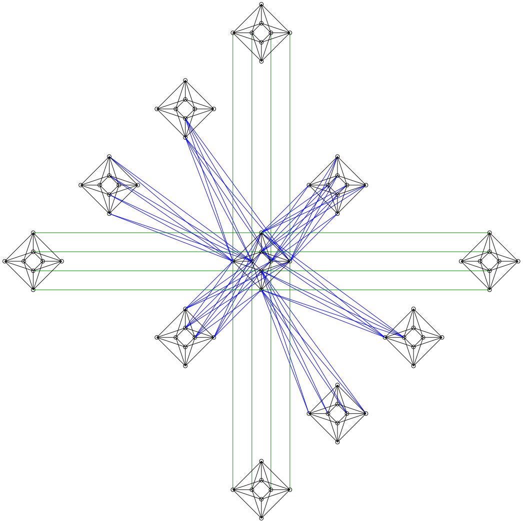
The Pegasus Topology: P_N

- qubits pairwise-linked
  - 3- and 4-cliques
  - qubit duplication



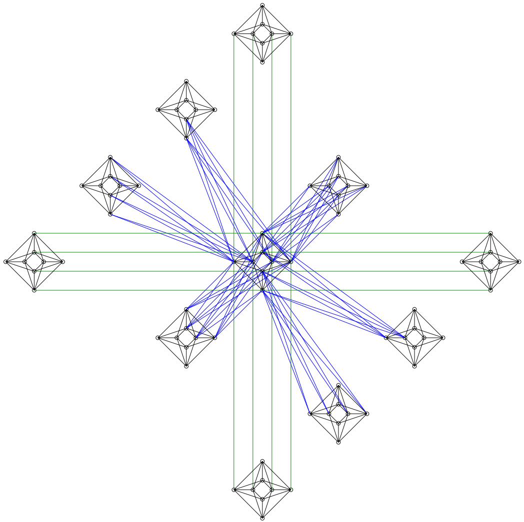
The Pegasus Topology: P_N

- qubits pairwise-linked
  - 3- and 4-cliques
  - qubit duplication
- 15 couplers/qubit



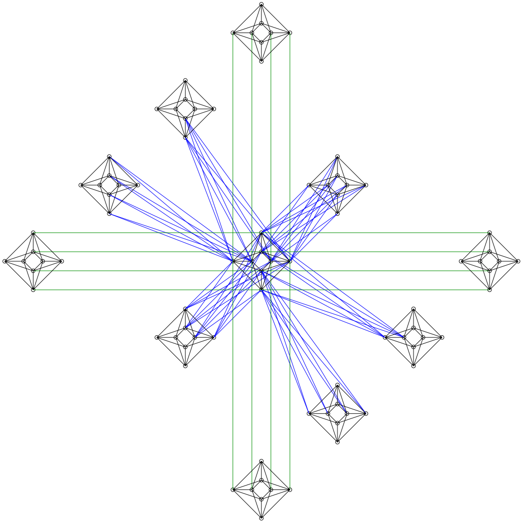
The Pegasus Topology: P_N

- qubits pairwise-linked
  - 3- and 4-cliques
  - qubit duplication
- 15 couplers/qubit
- more interleavings and less modular than Chimera



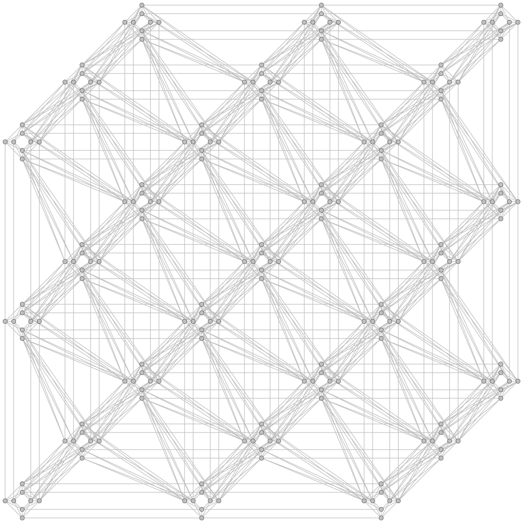
The Pegasus Topology: P_N

- qubits pairwise-linked
  - 3- and 4-cliques
  - qubit duplication
- 15 couplers/qubit
- more interleavings and less modular than Chimera
- wider ranges:
  - $[-4, 4]$ for biases
  - $[-2, 1]$ for couplings

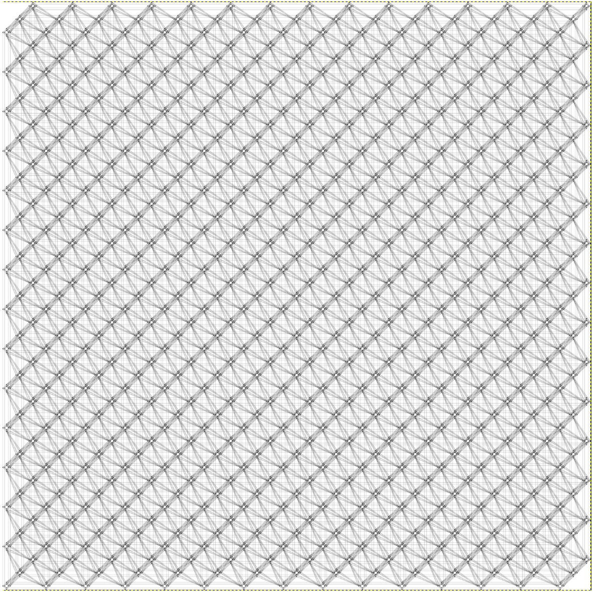


The Pegasus Topology: P_N

- qubits pairwise-linked
  - 3- and 4-cliques
  - qubit duplication
- 15 couplers/qubit
- more interleavings and less modular than Chimera
- wider ranges:
  - $[-4, 4]$ for biases
  - $[-2, 1]$ for couplings
- P_N: $24N(N-1)$ qubits
  - e.g P_4: 288 qubits
  - a few borderline qubits wasted



"Advantage": Current 5760-qubit P_{16} Architecture



Outline

Prime Factorization (PF)

Prime Factorization (PF)

Given a number k, find the two prime numbers n and m such that $k = n \times m$

Example

Given $k = 35$, then $n = 7$ and $m = 5$ is a solution.

- **Computationally hard**: the state-of-the-art classical algorithm to solve PF has sub-exponential time complexity!
- Fundamental problem in many applications, in particular cryptoanalysis.

Solving PF via quantum computing (1)

Shor's Algorithm

- Factors numbers into primes in poly-logarithmic time.
- Implemented on gate-based quantum computers, but **limited to small instances** (order of a few thousand). [1]
- Large-scale simulation on a GPU-based classical supercomputer achieved factorization up to 549,755,813,701. [2]

Hybrid Quantum-Classical Methods

- Hybrid classical-quantum procedures that use parameterized quantum circuits.
- Quantum computation interleaved with external classical search or preprocessing procedures
- Successful factorization attempts on IBMQ hardware for numbers like 91 and bi-primes 3127, 6557, 1,099,551,473,989. [3]
- However, the last numbers can be easily factorized through Fermat's factorization method in linear time.

[1] Amico et al.. Experimental study of Shor's factoring algorithm using the IBM Q Experience. Physical Review A

[2] Willsch et al.. Large-Scale Simulation of Shor's Quantum Factoring Algorithm. Mathematics, 2023

[3] Karamlou et al.. Analyzing the performance of variational quantum factoring on a superconducting quantum processor. Quantum Information

Solving PF via quantum computing (2)

Quantum Annealing (QA)

- High-degree cost functions reduced to quadratic using Groebner bases or equivalent models.
- The largest problem mapped to D-Wave 2000Q is 376,289 [4].
- It was possible to solve all bi-primes up to 200,000 on D-Wave 2X processors relying on QA only [5].
- D-Wave hybrid Classical-QA approaches factored 1,005,973 [6].
- **Important:** all approaches rely on minor embedding, making it difficult to scale.

[4] Dridi, Alghassi Prime factorization using quantum annealing and computational algebraic geometry Nature Scientific Reports, 2017

[5] Jiang, Britt, McCaskey, Humble, Kais. Quantum annealing for prime factorization. Nature Scientific Reports, 2018

[6] Wang et al.. Prime factorization algorithm based on parameter optimization of Ising model. Nature Scientific Reports, 2020

Outline

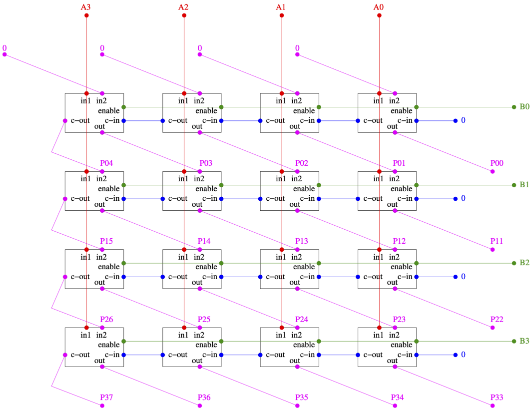
Modularity of a binary multiplier

- Based on the idea of binary multiplier

				0	0	0	0	+
				$A3$	$A2$	$A1$	$A0$	$\times B0$
			$P04$	$P03$	$P02$	$P01$	$P00$	+
			$A3$	$A2$	$A1$	$A0$		$\times B1$
		$P15$	$P14$	$P13$	$P12$	$P11$		+
		$A3$	$A2$	$A1$	$A0$			$\times B2$
	$P26$	$P25$	$P24$	$P23$	$P22$			+
	$A3$	$A2$	$A1$	$A0$				$\times B3$
$P37$	$P36$	$P35$	$P34$	$P33$	[$P22$]	[$P11$]	[$P00$]	

Modularity of a binary multiplier

- Based on the idea of binary multiplier
- A binary multiplier consists of a matrix of interconnected Controlled Full Adders (CFAs);

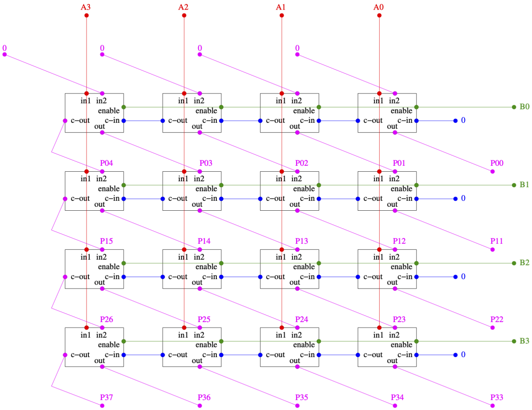


$$CFA(in2, in1, enable, c\_in, c\_out, out) \stackrel{\text{def}}{=} \Big(c\_out \leftrightarrow ((c\_in \wedge ((enable \wedge in1) \vee in2)) \vee ((enable \wedge in1) \wedge in2)\Big)$$

$$\wedge \Big(out \leftrightarrow ((enable \wedge in1) \oplus in2 \oplus c\_in)\Big)$$

Modularity of a binary multiplier

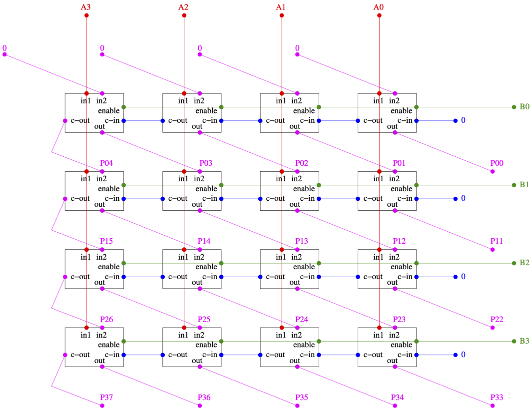
- Based on the idea of binary multiplier
- A binary multiplier consists of a matrix of interconnected Controlled Full Adders (CFAs);
- Modularity can be achieved by providing an efficient encoding for a single CFA that can fit into a single 8-qubit Pegasus tile;



$$CFA(in2, in1, enable, c\_in, c\_out, out) \stackrel{\text{def}}{=} \Big(c\_out \leftrightarrow ((c\_in \wedge ((enable \wedge in1) \vee in2)) \vee ((enable \wedge in1) \wedge in2)\Big)$$
$$\wedge \Big(out \leftrightarrow ((enable \wedge in1) \oplus in2 \oplus c\_in)\Big)$$

Modularity of a binary multiplier

- Based on the idea of binary multiplier
- A binary multiplier consists of a matrix of interconnected Controlled Full Adders (CFAs);
- Modularity can be achieved by providing an efficient encoding for a single CFA that can fit into a single 8-qubit Pegasus tile;
- Trivial to extend for larger architectures when available, obtaining larger multipliers;



$$CFA(in2, in1, enable, c\_in, c\_out, out) \overset{\text{def}}{=} \Big(c\_out \leftrightarrow ((c\_in \wedge ((enable \wedge in1) \vee in2)) \vee ((enable \wedge in1) \wedge in2)\Big)$$
$$\wedge \Big(out \leftrightarrow ((enable \wedge in1) \oplus in2 \oplus c\_in)\Big)$$

Encoding: key idea

How to compute $k = m \times n$

- Compute **offline** via Optimization Modulo Theories (OMT) an Ising model P_{CFA} for a CFA, compatible with Pegasus graph (V,E):

$$P_F(\underbrace{\underline{\mathbf{x}}, \underline{\mathbf{a}}}_{\underline{\mathbf{z}}} | \underline{\boldsymbol{\theta}}) \stackrel{\text{def}}{=} \theta_0 + \sum_{z_i \in V} \theta_i z_i + \sum_{(z_i, z_j) \in E, i<j} \theta_{ij} z_i z_j; \;\; z_i \in \{-1, 1\}; \tag{1}$$

$$\forall \underline{\mathbf{x}} \quad min_{\{\underline{\mathbf{a}}\}} P_F(\underline{\mathbf{x}}, \underline{\mathbf{a}} | \underline{\boldsymbol{\theta}}) \begin{cases} = 0 & \text{if } F(\underline{\mathbf{x}}) = \top \\ \geq g_{min} & \text{if } F(\underline{\mathbf{x}}) = \bot \end{cases} \tag{2}$$

- Encode variable equivalences $(x_k \leftrightarrow x'_k)$ as *chain* penalty functions $2 - 2z_k z'_k$
- Sum P_{CFA}'s and chains into a single penalty function $P_{multiplier}$.
- Force the values of the output qubits to mimic the value of product k

Encoding: topology limitations

- Generating P_{CFA} using an 8-qubit Pegasus tile and embedding them into a Pegasus grid is not as trivial as it seems:
  - the connections among different Pegasus tiles are limited, so it is not always the case a chain among two qubits does exist;
  - Since a CFA has four input bits and two output bits, we can have at most 2 ancillas, s.t. there exists no penalty function which fits into a Pegasus tile

How to address topology limitations?

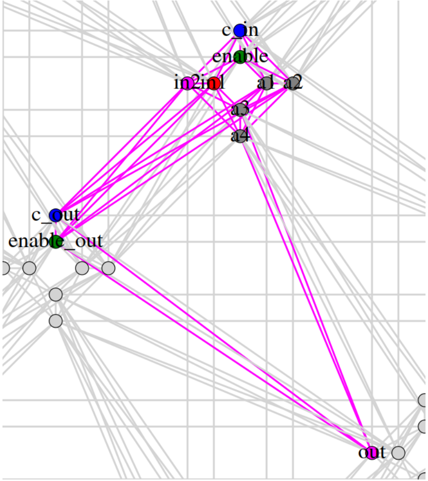
We propose three novel techniques:

- qubit sharing
- alternating CFAs
- virtual chaining

Encoding: qubit sharing

Qubit sharing

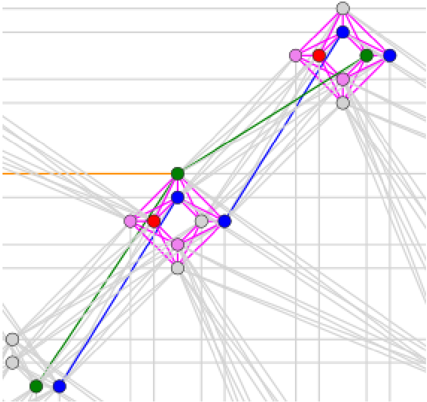
- Rather than connecting two qubits from different CFAs with a chain, we use a single qubit that is *shared* between the two CFAs, partially overlapping them.
- **Example:** The qubit is used for the encoding of one CFA as an output variable (c_out) and as an input variable for the subsequent CFA (c_in).
- **Important:** we must force the sum of the biases of the shared qubits to stay in the range $[-4, 4]$, otherwise automatic rescaling would negatively affect the annealer performances.



Encoding: alternating CFAs

Alternating CFAs

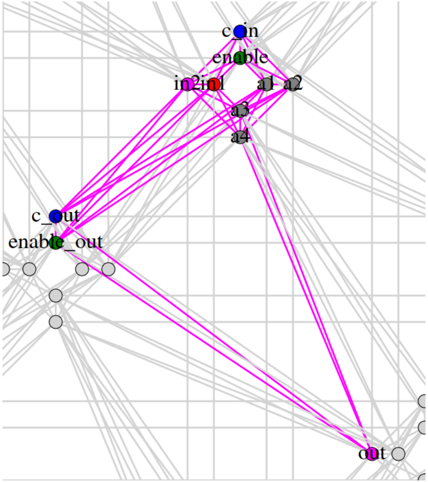
- Useful to deal with missing coupling for 45° direction chains;
- We compute two different CFAS, forcing the qubits that must be chained to have a coupling among them;
- **Example:** the *enable* qubit (green) is placed in the first vertical qubit on the upper tile and the third horizontal qubit in the 45°-degree bottom-left tile.
- **Important:** two different CFA encodings are **not** guaranteed to have the same gap!



Encoding: virtual chaining

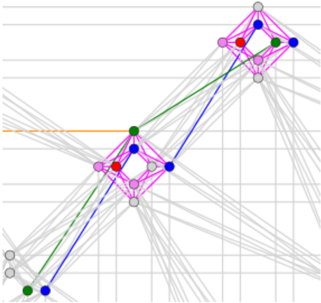
Virtual chaining

- Similar to qubit sharing, but to simulate the connection of qubits with missing couplings.
- **Example:** to chain *enable*, we add *enable_out* in the encoding, add the constraint $(enable \leftrightarrow enable\_out)$ and then generate the CFA by OMT.
- **Important:** we must force the sum of couplings $c\_in - enable$ and $c\_out - enable\_out$ to stay in the range $[-2, 1]$ to avoid rescaling.



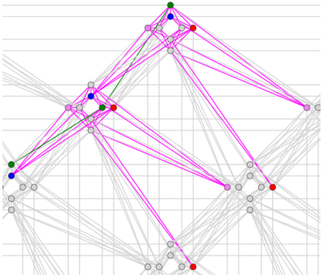
Proposed CFA encoding: V1

Multiplier version	V1
Multiplier Max. Size	22×8
# of ancillae per CFA	2
# of different CFA encodings	2
Gap of CFA penalty functions	$(1, \frac{4}{9})$
Connection $in1(i,j) - in1(i+1, j-1)$	Chain (90°)
Connection $enable(i,j) - enable(i,+1)$	Chain (45°)
Connection $c\_in(i,j) - c\_out(i,j+1)$	Chain (45°)
Connection $out(i,j) - in2(i+1, j-1)$	Chain (45°)



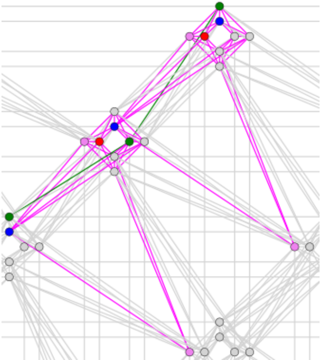
Proposed CFA encoding: V2

Multiplier version	**V2**
Multiplier Max. Size	21×12
# of ancillae per CFA	4
# of different CFA encodings	2
Gap of CFA penalty functions	$(2, \frac{4}{3})$
Connection $in1(i,j) - in1(i+1,j-1)$	Virtual chain (120°)
Connection $enable(i,j) - enable(i,+1)$	Chain (45°)
Connection $c\_in(i,j) - c\_out(i,j+1)$	Qubit sharing
Connection $out(i,j) - in2(i+1,j-1)$	Qubit sharing



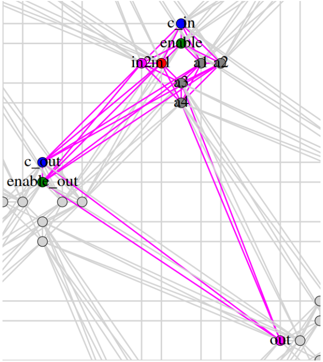
Proposed CFA encoding: V3

Multiplier version	**V3**
Multiplier Max. Size	22×8
# of ancillae per CFA	2
# of different CFA encodings	2
Gap of CFA penalty functions	(2, 2)
Connection $in1(i,j) - in1(i+1, j-1)$	Chain (90°)
Connection $enable(i,j) - enable(i,+1)$	Chain (45°)
Connection $c\_in(i,j) - c\_out(i,j+1)$	Qubit sharing
Connection $out(i,j) - in2(i+1,j-1)$	Qubit sharing

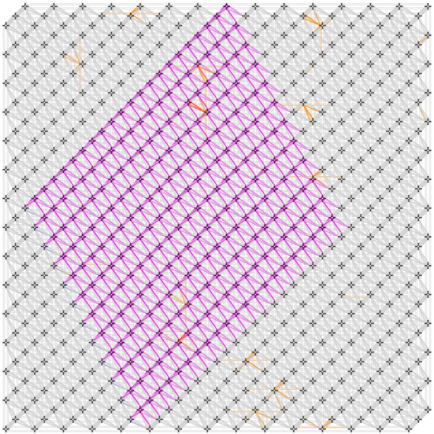


Proposed CFA encoding: V4

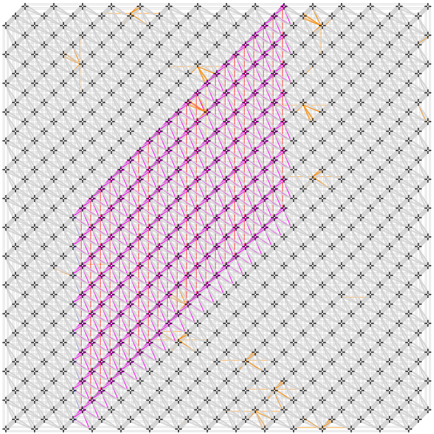
Multiplier version	V4
Multiplier Max. Size	22×8
# of ancillae per CFA	4
# of different CFA encodings	**1**
Gap of CFA penalty functions	2
Connection $in1(i,j) - in1(i+1,j-1)$	Chain (90°)
Connection $enable(i,j) - enable(i,+1)$	Virtual chain (45°)
Connection $c\_in(i,j) - c\_out(i,j+1)$	Qubit sharing
Connection $out(i,j) - in2(i+1,j-1)$	Qubit sharing



Multipliers on Pegasus architecture



21 × 12 multiplier (V2)



22 × 8 multiplier (V1, V3, V4)

To the best of our knowledge, these are the largest factorization problems ever embedded on a quantum annealer!

Which CFA should we use?

All 4 proposed CFAs work to solve prime factorization. However, **V4** is the best option among the four since:

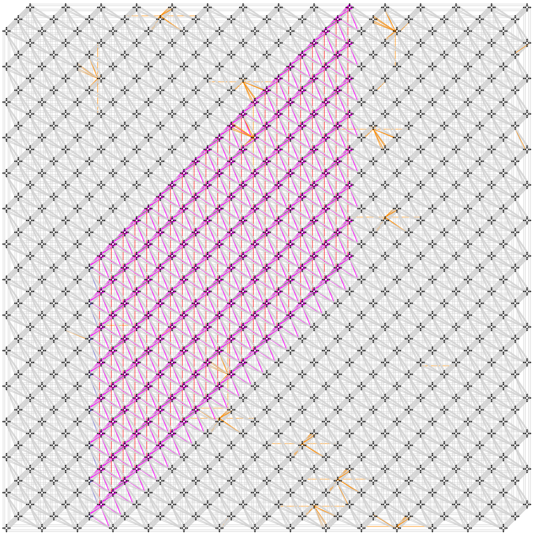
- It only uses one single CFA, so it is easier to scale for bigger multipliers;
- It has a single physical chain, thus optimizing the penalty function by the QA turns out to be more effective.

All the experiments from now on rely on the V4 CFA encoding.

Outline

D-wave Advantage system limitations

- Results in the previous section do not take into account quantum annealer limitations.
- Hardware faults, such as **faulty qubits** and **faulty couplings**, are widespread (marked in orange in the figure on the right).
- After an empirical evaluation we identified a suitable, fault-free area for testing up to 17 $\times$ 8 bits multipliers.



Main results

Size	Input N	#	T_p	S_p	$min(P_F)$	$min(P_F)_{new}$
13×8	$2,055,941$ (8191×251)	1	100	0.38	14.083	6.083
		2	100	0.42	6.083	0[**216**]
14×8	$4,111,631$ (16381×251)	1	200	0.39	16.167	6.083
		2	200	0.44	6.083	0[**467**]
15×8	8,219,999 (32749×251)	1	1	0.4	20.333	12.167
		2	100	0.43	12.167	8.000
		3	200	0.43	8.000	6.000
		4	200	0.44	6.000	4.083
		5	200	0.43	4.083	0[**329**]

We solved biprime factoring problems up to 8,219,999 $= 32{,}749 \times 251$
$\Longrightarrow$ the largest biprime factoring problems ever solved on a quantum device without using hybrid quantum-classical techniques

Outline

1. The Pegasus Topology
2. Problem and state of the art
3. Encoding
4. Solving
5. Conclusion and future work

Main Achievements

We introduced a novel approach for prime factorization via quantum annealing

- **Modularity**: our approach easily scales for bigger multipliers, bounded to current quantum technologies
- We encoded the largest biprime factoring problem ever embedded on a QA (1,052,769,551 $=$ 4,194,301 $\times$ 251, 22 $\times$ 8 bits multiplier)
- We solved the largest biprime factoring problems ever solved on a quantum device without using hybrid quantum-classical techniques (8,219,999 $=$ 32,749 $\times$ 251)

Possible future directions

- Explore and empirically investigate other solving strategies within the annealing process.
- Test encoding algorithms on D-Wave's upcoming Zephyr processors, with enhanced connectivity, for better penalty functions and solving capabilities.
- Conceive more efficient techniques to produce monolithic encoding of sub-circuits.
- Investigate the encoding of other circuits of interest


"That's all Folks!"