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

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]

Variational 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 Scientific Reports, 2017

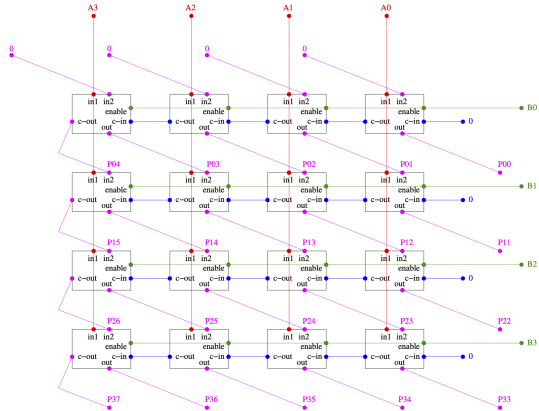
[5] Jiang, Britt, McCaskey, Humble, Kais. Quantum annealing for prime factorization. Scientific Reports, 2018

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

Outline

Modularity of a 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;



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

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):

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

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

- 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, the chances of getting a valid penalty function are quite low.

How to address topology limitations?

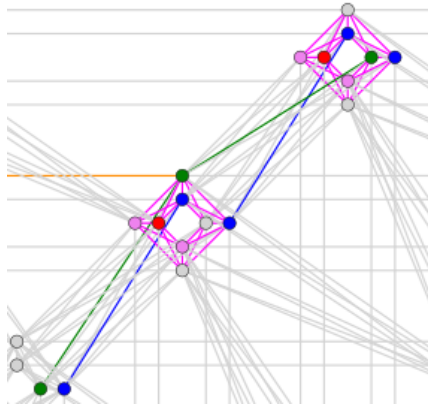
We propose three novel techniques:

- alternating CFAs
- qubit sharing
- virtual chaining

Encoding: alternating CFAs

Alternating CFAs

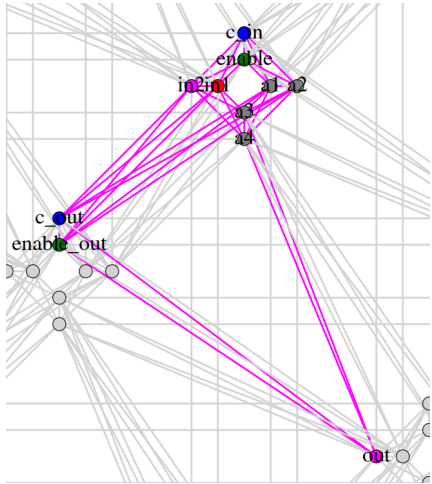
- Useful to deal with missing coupling for 45° direction chains;
- We compute two different CFAS, forcing the qubits that must be chained to have a coupling among them;
- **Example:** the *enable* qubit (green) is placed in the first vertical qubit on the upper tile and the third horizontal qubit in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



Encoding: qubit sharing

Qubit sharing

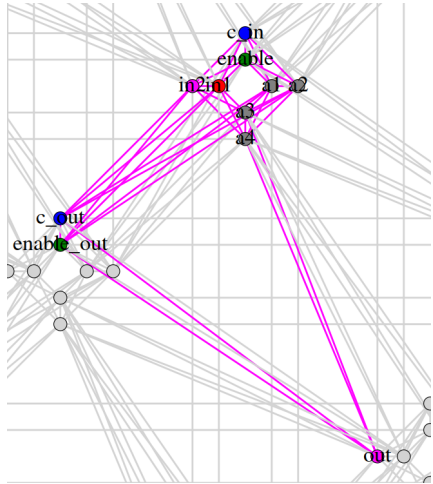
- Rather than connecting two qubits from different CFAs with a chain, we use a single qubit that is *shared* between the two CFAs, partially overlapping them.
- **Example:** The qubit is used for the encoding of one CFA as an output variable (c_out) and as an input variable for the subsequent CFA (c_in).
- **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: virtual chaining

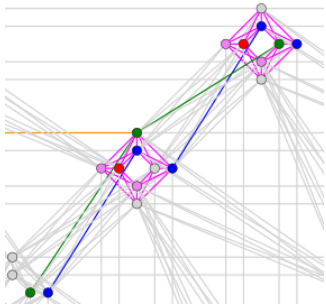
Virtual chaining

- 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.



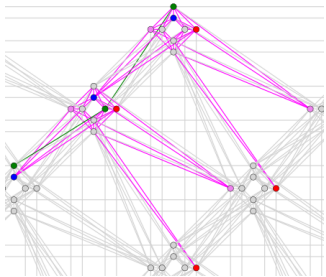
Proposed CFA encoding: V1

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°)



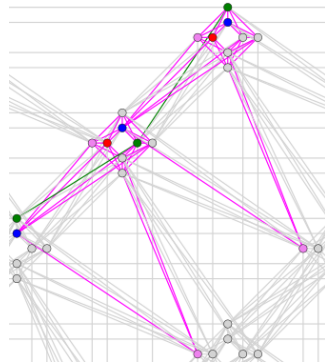
Proposed CFA encoding: V2

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



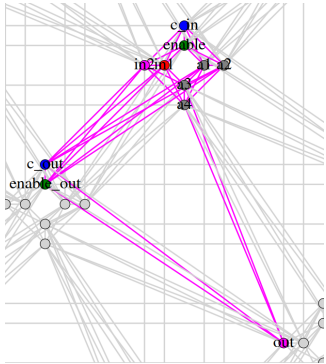
Proposed CFA encoding: V3

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

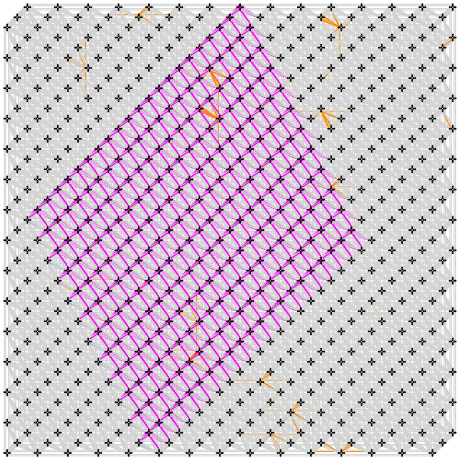


Proposed CFA encoding: V4

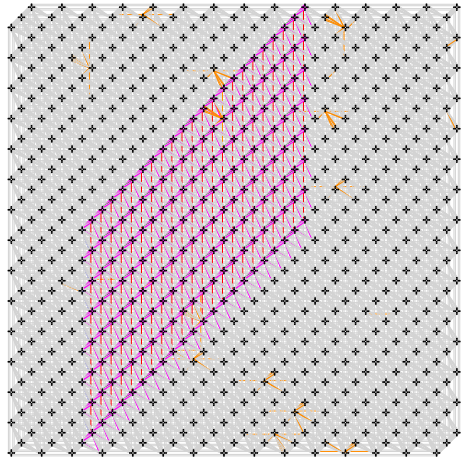
Multiplier version	**V4**
Multiplier Max. Size	22×8
# of ancillae per CFA	4
# of different CFA encodings	**1**
Gap of CFA penalty functions	2
Connection $in1(i,j) - in1(i+1,j-1)$	Chain (90°)
Connection $enable(i,j) - enable(i,+1)$	Virtual chain (45°)
Connection $c\_in(i,j) - c\_out(i,j+1)$	Qubit sharing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

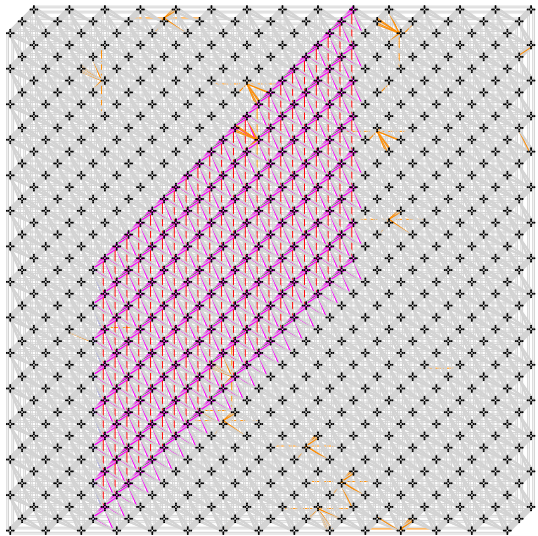
- It only uses one single CFA, so it is easier to scale for bigger multipliers;
- It has a single physical chain, thus optimizing the penalty function by the QA turns out to be more effective.

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 17×8 bits multipliers.



How to initialize qubits (1)

- Initialization of qubits within the multiplier embedding is essential for factoring a specific integer.
- D-Wave Advantage interface provides the *fix_variables*() function, allowing imposed values on qubits (CFA0).
- Substituting values into the penalty function and rescaling **may reduce** g_{min} and worsen performances .
- An alternative method proposes using *flux biases* for qubit initialization, addressing rescaling issues (CFA1).

Size	Input N	CFA0 $\#(P_F = 0)$	CFA1 $\#(P_F = 0)$
3×3	25 (5×5)	**161**	136
	35 (5×7)	389	**951**
	49 (7×7)	450	**997**
4×4	121 (11×11)	**17**	0
	143 (11×13)	40	**67**
	169 (13×13)	**31**	5
5×5	289 (17×17)	**5**	0
	323 (17×19)	**2**	0
	361 (19×19)	1	**3**
	391 (17×23)	6	**9**
	437 (19×23)	**17**	0
	493 (17×29)	**3**	2
	527 (17×31)	21	**37**
	529 (23×23)	5	**8**
	551 (19×29)	0	**4**
	589 (19×31)	16	**52**
	667 (23×29)	0	**105**
	713 (23×31)	11	**138**
	841 (29×29)	5	**7**
	899 (29×31)	17	**343**
	961 (31×31)	1	**338**

How to initialize qubits (2)

Size	Input N	$\#(P_F = 0)$	Size	Input N	$\#(P_F = 0)$
7×7	10,033 (127×79)	0	8×8	49,447 (251×197)	0
	10,541 (127×83)	**1**		49,949 (251×199)	0
	11,303 (127×89)	0		52,961 (251×211)	0
	12,319 (127×97)	0		55,973 (251×223)	0
	12,827 (127×101)	**1**		56,977 (251×227)	0
	13,081 (127×103)	**2**		57,479 (251×229)	0
	13,589 (127×107)	**10**		58,483 (251×233)	0
	13,843 (127×109)	0		59,989 (251×239)	**2**
	14,351 (127×113)	0		60,491 (251×241)	0
	16,129 (127×127)	**7**		63,001 (251×251)	0

Standard forward annealing still fails to reach the ground state with small problems...
we must rely on other techniques!

Exploiting thermal relaxation (1)

- Increasing annealing time T_a may not significantly boost success probability.
- To enhance solving performance, we explore the effectiveness of *thermal relaxation*.
- Thermal relaxation introduces a *pause* T_p at a specific point S_p during the annealing process ($S_p \in [0, 1]$).

Size	Input N	S_p	$min(P_F)$	$\#(P_F = 0)$
8×8	49,447 (251×197)	0.38	0.000	**1**
	49,949 (251×199)	–	4.083	0
	52,961 (251×211)	–	6.000	0
	55,973 (251×223)	0.33	0.000	**6**
	56,977 (251×227)	0.33	0.000	**1**
	57,479 (251×229)	0.33	0.000	**3**
	58,483 (251×233)	–	6.083	0
	59,989 (251×239)	0.33	0.000	**43**
	60,491 (251×241)	0.38	0.000	**1**
	63,001 (251×251)	–	2.000	0

Exploiting thermal relaxation (2)

Size	Input N	S_p	$min(P_F)$	$\#(P_F = 0)$	Size	Input N	S_p	$min(P_F)$	$\#(P_F = 0)$
9×8	100,273 (509×197)	–	4.083	0	10×8	201,137 (1021×197)	–	6.167	0
	101,291 (509×199)	–	8.083	0		203,179 (1021×199)	–	8.000	0
	107,399 (509×211)	–	4.000	0		215,431 (1021×211)	–	6.083	0
	113,507 (509×223)	–	8.083	0		227,683 (1021×223)	0.34	0.000	**1**
	115,543 (509×227)	0.33	0.000	**1**		231,767 (1021×227)	–	8.083	0
	116,561 (509×229)	–	6.000	0		233,809 (1021×229)	–	8.000	0
	118,597 (509×233)	–	4.000	0		237,893 (1021×233)	–	6.000	0
	121,651 (509×239)	0.33	0.000	**1**		244,019 (1021×239)	–	6.250	0
	122,669 (509×241)	–	8.167	0		246,061 (1021×241)	–	6.167	0
	127,759 (509×251)	0.36	0.000	**1**		256,271 (1021×251)	0.35	0.000	**2**

A bit better, but still far from exploiting the potentiality of QAs...

Exploiting quantum local search

- Reverse annealing starts from a local minimum to escape it and reach the global minimum.
- The lowest-energy state from previous experiments serves as the initial state for RV, choosing the one with later pauses if multiple samples exist.
- RV is tested on several pause points until a ground state is found.

Size	Input N	Forward annealing		Reverse annealing		
		S_p	$min(P_F)$	S'_p	P_F	$\#(P_F = 0)$
8×8	49,949 (251×199)	0.50	2.000	0.33	0.000	**7**
	52,961 (251×211)	0.35	2.000	0.41	0.000	**1**
	58,483 (251×233)	0.33	2.083	–	4.000	0
	63,001 (251×251)	0.51	2.000	0.35	0.000	**4**
9×8	100,273 (509×197)	0.36	4.000	–	4.083	0
	101,291 (509×199)	0.44	4.000	–	4.000	0
	107,399 (509×211)	0.51	4.000	–	4.000	0
	113,507 (509×223)	0.38	2.000	–	2.000	0
	116,561 (509×229)	0.36	4.000	0.37	0.000	**35**
	118,597 (509×233)	0.33	2.000	–	4.000	0
	122,669 (509×241)	0.48	4.083	0.36	0.000	**7**
10×8	201,137 (1021×197)	0.38	2.000	–	2.000	0
	203,179 (1021×199)	0.34	4.000	–	4.083	0
	215,431 (1021×211)	0.4	4.000	–	4.000	0
	231,767 (1021×227)	0.33	2.083	–	4.083	0
	233,809 (1021×229)	0.39	4.083	–	6.000	0
	237,893 (1021×233)	0.46	2.083	–	2.083	0
	244,019 (1021×239)	0.48	2.000	–	4.000	0
	246,061 (1021×241)	0.34	4.000	–	2.083	0

Iterative reverse annealing (IRV)

- Starts by running a standard forward annealing process, with thermal relaxation disabled.
- The obtained lowest-energy state is selected as the starting point for the subsequent iterations of the algorithm.
- When we do not escape the local minima, we increase the annealing time.

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

Outline

Conclusions

- We introduced a novel approach for prime factorization via quantum annealing
  - We proposed a modular encoding of a binary multiplier circuit, which scales arbitrarily with the size of a Pegasus chip.
  - We successfully factorized up to 8,219,999 using a D-Wave Advantage 4.1 quantum annealer, marking the largest number without hybrid quantum-classical techniques.

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

Thanks for your attention!