Check-Agnosia based Post-Processor for Message-Passing Decoding of Quantum LDPC Codes

Julien du Crest[1], Francisco Garcia-Herrero[2], Mehdi Mhalla[3], Valentin Savin[4], and Javier Valls[5]

[1]Université Grenoble Alpes, Grenoble INP, LIG, F-38000 Grenoble, France
[2]Department of Computer Architecture and Automatics, Complutense University of Madrid, Madrid, Spain
[3]Université Grenoble Alpes, CNRS, Grenoble INP, LIG, F-38000 Grenoble, France
[4]Université Grenoble Alpes, CEA-Léti, F-38054 Grenoble, France
[5]Instituto de Telecomunicaciones y Aplicaciones Multimedia, Universitat Politecnica de Valencia, Valencia, Spain

arXiv:2310.15000v3 [quant-ph] 29 Apr 2024

The inherent degeneracy of quantum low-density parity-check codes poses a challenge to their decoding, as it significantly degrades the error-correction performance of classical message-passing decoders. To improve their performance, a post-processing algorithm is usually employed. To narrow the gap between algorithmic solutions and hardware limitations, we introduce a new post-processing algorithm with a hardware-friendly orientation, providing error correction performance competitive to the state-of-the-art techniques. The proposed post-processing, referred to as check-agnosia, is inspired by stabilizer-inactivation, while considerably reducing the required hardware resources, and providing enough flexibility to allow different message-passing schedules and hardware architectures. We carry out a detailed analysis for a set of Pareto architectures with different tradeoffs between latency and power consumption, derived from the results of implemented designs on an FPGA board. We show that latency values close to one microsecond can be obtained on the FPGA board, and provide evidence that much lower latency values can be obtained for ASIC implementations. In the process, we also demonstrate the practical implications of the recently introduced t-covering layers and random-order layered scheduling.

1 Introduction

Quantum low-density parity-check (qLDPC) codes [1] have become one of the main candidates to implement the error-correction layer of a large-scale quantum computer architecture [2–4]. Compared to other families of quantum error correction codes, qLDPC codes may reduce the physical qubit overhead, while protecting a larger number of logical qubits, so higher code rates can be obtained with similar or better error-correction performance [5–9]. Yet, for qLDPC codes to work on a real system, a larger number of physical qubits than those available in today's noisy intermediate-scale quantum systems is required [3], [10]. Nonetheless, before large-scale quantum technology becomes available, two important problems need to be addressed from the qLDPC decoding perspective: i) devising new decoding algorithms that overcome or mitigate the effect of degeneracy [11], thus providing increased error correction capabilities, and ii) developing hardware designs that meet latency and power constraints imposed by the quantum system (*e.g.*, latency values within the decoherence time of the qubits to be protected, or power limitations for qubit technologies requiring cryogenic cooling, when the decoder is implemented within the low-temperature layers), a topic that only got attention recently [12–14].

To achieve the first objective, several approaches building upon classical message-passing (MP) decoding algorithms have been recently proposed in the literature, where the degeneracy issue is dealt with by either incorporating neural network techniques in the MP decoder [15], or adding a post-processing step, taking advantage of the soft information delivered by the MP decoder [5, 16].

Neural-network-based decoders are bound to the noise models used to train them and do not scale well with the number of qubits [17]. Moreover, as shown in [18], there are not only different sources and noise models, but also the noise may be different depending on the area of the layout of the quantum processor, the environmental conditions, and the evolution of errors with time since the last calibration (space and time drift of the errors [19]). In that sense, more generalized solutions are required, at least at the moment of writing these lines, when there is no standardized or predominant technology or architecture for future large-scale quantum devices. Hence, post-processing techniques may become an interesting choice. A first

Julien du Crest: julien.du-crest@univ-grenoble-alpes.fr
Francisco Garcia-Herrero: francg18@ucm.es
Mehdi Mhalla: mehdi.mhalla@univ-grenoble-alpes.fr
Valentin Savin: valentin.savin@cea.fr
Javier Valls: jvalls@upv.es

post-processing technique based on ordered statistics decoding (OSD) was proposed in [5, 20]. Although the improvement in terms of coding gain is significant, the complexity is too high, and hence, it becomes unpractical for real-time hardware implementations [21]. Recently, some of us proposed a new post-processing technique for Calderbank-Shor-Steane (CSS) qLDPC codes, called stabilizer inactivation [16]. The post-processing consists in *inactivating* a set of unreliable qubits supporting a check in the dual code (a stabilizer generator of the same type as the decoded error). Then the MP decoding is run again, while taking out of the decoding process the inactivated qubits and their neighbor check nodes. The remaining qubits and check nodes that participate in the MP decoding are called active. Several stabilizer generators may be inactivated, one at a time (which can be implemented either sequentially or in parallel), until one MP decoding meets the syndrome constraints on the active check nodes. Inactivated qubits are then determined by solving a small linear system, defined by the inactive check-nodes. Stabilizer inactivation shows a non-negligible error correction improvement and increased flexibility (with regard to the MP decoding schedule) compared to OSD, with a considerable reduction of complexity. However, as discussed later in this paper, additional hardware-oriented analysis and optimization are required to ensure the hardware design meets the constraints required to provide real-time support to a quantum processor.

The main contributions of the paper are as follows. First, inspired by the stabilizer inactivation, we introduce a new post-processing algorithm for MP decoders. The algorithm takes into account the architectural properties of MP decoders in order to reduce the computational load and the required hardware resources. It also limits the amount of information required from the code, eliminating the need to know the stabilizer structure (dual code) and just treating both parity-check matrices as independent. Similar to the stabilizer inactivation, the algorithm identifies a small set of unreliable qubits (which are however not inactivated, in the sense described above). The information considered to identify such a set of qubits is based on the check-node reliability. When the MP decoder fails, the a priori information for the qubits connected to the least reliable check nodes is erased, and the post-processor will then try to learn again the reliability of these qubits based on the information from the rest. For this reason, we call the post-processing technique check-agnosia. We also suggest several approaches to perform the selection of unreliable check-nodes, to reduce power consumption and latency, which are the constraints that limit the implementation of decoders in real systems [22, 23].

Second, along the document, a non-agnostic hardware perspective is described to help to meet the constraints of future large-scale quantum devices. Aligned with this, a functional description in terms of performance and hardware results of the proposed solution for the two main schedules employed for MP decoders (flooded and layered [24], [25]) is introduced. We carry out a detailed analysis of different corner cases, which is then illustrated for a specific qLPDC code, by providing latency and power consumption values of the check-agnosia solution implemented on an FPGA board.

The rest of the paper is organized as follows. Section 2 introduces the relevant notation and the algorithmic background. Section 3 introduces the check-agnosia post-processing, and discusses the check-node reliability metric along with several hardware-oriented optimizations. Section 4 analyzes the impact of the post-processing algorithm on the hardware implementation, considering architectures with different schedules and varying degrees of parallelism. Latency and power consumption results are also provided here. Section 5 evaluates the error-correction performance of the proposed check-agnosia decoder for different qLDPC codes, and compares it to other existing solutions. Finally, Section 6 provides the main conclusions of this work.

2 Algorithmic Background

We consider qLDPC codes of CSS type, defined by two parity check matrices $\mathbf{H}_x$ and $\mathbf{H}_z$, corresponding respectively to X-type and Z-type generators. In the following, we will consider decoding of one type of error (since similar considerations apply to the other type), and will denote by $\mathbf{H}$ the corresponding decoding matrix (*e.g.*, $\mathbf{H} = \mathbf{H}_z$ for decoding X-type errors). We also consider the Tanner graph associated with $\mathbf{H}$, and denote by $\mathcal{Q}$ the set of qubit-nodes[1] and $\mathcal{C}$ is the set of check-nodes (stabilizer generators). We denote by $\mathcal{N}(q) \subset \mathcal{C}$ the set of neighboring check-nodes of a qubit-node $q \in \mathcal{Q}$, and similarly, by $\mathcal{N}(c) \subset \mathcal{Q}$ the set of neighboring qubit-nodes of a check-node $c \in \mathcal{C}$.

We denote by p the probability that an error of the considered type occurs, and by $e \in \{0,1\}^{|\mathcal{Q}|}$ the error indicator vector[2]. We assume errors happen independently on the qubits, hence $P(e_q = 1) = p$. Information about the error $\mathbf{e}$ is revealed through the measurement of stabilizer generators, in the form of an error syndrome $\mathbf{s} := \mathbf{H} \cdot \mathbf{e}$. Throughout this work, we assume ideal syndrome extraction, *i.e.*, we only consider errors occurring on qubits, not on the extracted syndrome. The decoding problem is determining the

[1] Usually referred to as variable-nodes or bit-nodes, in classical LDPC coding.

[2] For a Pauli noise model in which Pauli errors X, Y, and Z occur with probabilities p_x, p_y, and p_z, respectively, and considering the decoding of the X-type error, we have $p = p_x + p_y$, and $e_q = 1$ iff either an X or Y error occurred on the corresponding qubit.

most likely error $\hat{\mathbf{e}}$, such that $\mathbf{H} \cdot \hat{\mathbf{e}} = \mathbf{s}$.

Although maximum likelihood decoding is optimal, it is computationally prohibitive. Instead, classical LDPC codes are efficiently decoded by MP algorithms. For a qubit $q \in \mathcal{Q}$, we denote by γ_q the a priori log-likelihood ratio (LLR) of an error happening on qubit q, which is defined[3] as $\gamma_q = \log\left(P(e_q = 0)/P(e_q = 1)\right) = \log((1-p)/p)$. The set of LLR values $\{\gamma_q \mid q \in \mathcal{Q}\}$ constitutes the input of the MP decoder and is used to initialize an iterative exchange of messages between qubit and check-nodes. We denote these messages by either $\mu_{q\to c}$ or $\mu_{c\to q}$, the arrow in the notation indicating whether the message is sent from a qubit-node q to a check-node c, or in the opposite direction. At each iteration, exchanged messages are used to compute a posteriori LLR values $\hat{\gamma}_q$, for each qubit-node q, used to provide an estimate $\hat{\mathbf{e}}_q$ of the corresponding qubit error. The iterative message passing process stops when either the estimated error satisfies the syndrome (*i.e.*, $\mathbf{H} \cdot \hat{\mathbf{e}} = \mathbf{s}$), or a maximum number of iterations is reached. In the following, we shall also refer to (a priori/a posteriori) LLR values as *qubit reliabilities*.

Throughout this work, we shall consider the Min-Sum (MS) decoding, or its normalized variant (NMS), which represent the option of choice for hardware implementations for several reasons: reduced computational complexity, reduced memory requirements (by adopting the first and second minimum compression method for check-node messages [26]), and its insensitivity to input LLR values up to a constant scaling factor (see Section 5 for the input LLRs of the finite-precision MS decoder). For details regarding the MS and other MP decoding algorithms we refer to [27] (see also the discussion in [16, Section 2]).

A key attribute of MP decoding algorithms is the underlying *scheduling*, indicating the order in which qubit - and check-node messages are updated [27]. Flooded, layered, or serial schedules[4] are usually implemented through fully-parallel, partially-parallel, or serial hardware architectures, respectively, yielding designs with different performances in terms of latency, area, or power consumption.

Compared to the flooded schedule, serial and layered schedules are also known to propagate information twice faster in the Tanner graph [28] for classical (non-degenerate) LDPC codes. This directly translates into a faster convergence speed. However, the flooded schedule provides decoding performance similar to the serial and layered ones, at the cost of doubling the number of decoding iterations. This is no longer true for qLDPC codes, presumably due to the code degeneracy. As observed in [16], not only the flooded schedule may not be able to approach the decoding performance of the serial or layered schedules, even at the cost of an increased number of iterations, but in some cases it may also penalize the performance of the post-processing algorithm. Layered MP decoding of qLDPC codes has been recently investigated by the authors in [29], where it has also been observed that processing the layers in a random order (at each decoding iteration) may significantly improve the performance of the MP decoder. We will use these results in Section 5 of this paper.

[3] For a Pauli noise model, correlations between X and Z errors (due to the Y errors) can be taken into account by decoding X and Z errors sequentially, say first X and then the Z error, and computing the a priori LLRs for the Z error, conditional on the decoded X error.

[4] Through this work, all schedules are considered to be horizontal, *i.e.*, defined with respect to check-node processing, as opposed to vertical schedules, defined with respect to qubit-node processing.

3 Check-Agnosia Decoder

We introduce in this section the Check-Agnosia (CA) post-processing. We first describe the generic post-processing technique (Algorithm 1) and then discuss possible modifications.

3.1 Generic Check-Agnosia Decoder

The error vector $\hat{\mathbf{e}}$ is estimated first using a soft-output MP decoding algorithm. If the error estimate $\hat{\mathbf{e}}$ satisfies the syndrome, *i.e.*, $\mathbf{H} \cdot \hat{\mathbf{e}} = \mathbf{s}$, then no post-processing is applied.

If the initial MP decoding fails, a metric on the exchanged soft information is used to find the λ checks $\{c_k\}_{k\in[\lambda]}$ whose supports are the most likely to be involved in the decoding failure (this metric will be discussed later). The post-processing will consist of rerunning the MP decoder at most λ times with new a priori qubit reliabilities[5] $\{\gamma'_q\}$ and a modified stopping criterion.

For the k-th decoder, the input reliability will be set to $\gamma'_q = 0$ for qubits $q \in \mathcal{N}(c_k)$, considered unreliable, and $\gamma'_q = \gamma_q$ for the rest of the qubits. Putting the input reliability to 0 can be considered as an erasure in the MP decoder [30], ensuring that these qubits are deprived of any a priori information that may interfere in the decoding attempt of the more reliable qubits. We further define $\mathfrak{N}_k = \cup_{q\in\mathcal{N}(c_k)}\mathcal{N}(q)$, the set of checks that share a neighbor qubit-node with c_k (note that $c_k \in \mathfrak{N}_k$). This allows us to define $\mathbf{s}_{|\overline{\mathfrak{N}_k}}$ the *partial syndrome* vector containing only the checks that have no neighbor qubit-node in $\mathcal{N}(c_k)$, and $\mathbf{s}_{|\mathfrak{N}_k}$ the *residual syndrome*. We then run a MP decoder with a modified stopping criterion that only tries to match the partial syndrome $\mathbf{s}_{|\overline{\mathfrak{N}_k}}$. In Algorithm 1, we denote this decoder by $\mathrm{MP}^\star(\mathbf{H}, \mathbf{s}, \{\gamma'_q\}, \overline{\mathfrak{N}_k})$. We emphasize that MP* applies exactly the same decoding rules on the same Tanner graph as MP, except that MP* is initialized with qubit reliabilities $\{\gamma'_q\}$, and it

[5] Here, we prefer the terminology "qubit reliabilities" rather than "input LLRs" since we modify the actual LLR values.

stops when the partial syndrome $\mathbf{s}_{|\overline{\mathfrak{N}_k}}$ is satisfied (no matter whether the residual syndrome $\mathbf{s}_{|\mathfrak{N}_k}$ is satisfied or not). If the MP* succeeds in matching the partial syndrome, the decoder then attempts to match the residual syndrome by brute-forcing the error pattern on $\mathcal{N}(c_k)$. Note that sometimes the MP* can actually match the full syndrome, in which case no brute-forcing is needed (will be discussed in more detail later). In Algorithm 1, $\mathbf{H}_{|\overline{\mathfrak{N}_k}}$ denotes the submatrix of $\mathbf{H}$ whose rows correspond to check-nodes $c \notin \mathfrak{N}_k$. Consequently, $\mathbf{H}_{|\overline{\mathfrak{N}_k}}(c, q) = 0$ for any $q \in \mathcal{N}(c_k)$, and thus $\mathbf{H}_{|\overline{\mathfrak{N}_k}} \cdot \hat{\mathbf{e}}$ only depends on $\hat{\mathbf{e}}_{|\overline{\mathcal{N}(c_k)}}$ (which explains the slight abuse of notation $\mathbf{H}_{|\overline{\mathfrak{N}_k}} \cdot \hat{\mathbf{e}}_{|\overline{\mathcal{N}(c_k)}}$). Likewise, $\mathbf{H}_{|\mathfrak{N}_k}$ denotes the submatrix of $\mathbf{H}$ whose rows correspond to check-nodes $c \in \mathfrak{N}_k$. If $\hat{\mathbf{e}}_{|\overline{\mathcal{N}(c_k)}}$ matches the partial syndrome, we keep its value and bruteforce $\hat{\mathbf{e}}_{|\mathcal{N}(c_k)}$ to match also the residual syndrome.

The intuition behind the post-processing is that the presence of quantum trapping sets [11] in the Tanner graph causes the a posteriori reliability values of trapped qubit-nodes to oscillate. This prevents the decoder from converging, regardless of the number of decoding iterations (for oscillating trapping sets see also [31]). Taking into account the oscillation effect, it is reasonable to think that the messages associated with the untrapped qubits will grow with each iteration while the trapped ones will keep relatively low reliability. This effect will help to identify possible trapped qubits. To this end, we define a reliability metric on checks to decide (the support of) which checks should be *erased*. A natural approach to define such a reliability metric is to consider the reliability (*i.e.*, absolute value) of either incoming messages $\{\mu_{q\to c} \mid q \in \mathcal{N}(c)\}$ or outgoing messages $\{\mu_{c\to q} \mid q \in \mathcal{N}(c)\}$. However, for MS-based decoders, the absolute value of outgoing messages is equal to either the first or the second minimum of the absolute values of incoming messages, denoted by $\min_{q\in\mathcal{N}(c)} |\mu_{q\to c}|$ and $\min_{2\, q\in\mathcal{N}(c)} |\mu_{q\to c}|$, respectively. This motivates the reliability metric[6] δ_c considered in Algorithm 1. The cost of computing this metric is nearly none, as the two minima are already computed by the MS decoder. Also, this metric is computed for all the check-nodes, based on the $\{\mu_{q\to c}\}$ messages at some specific (predetermined) iteration of the MS decoder (as discussed below). This allows the sorting of all the checks according to the proposed metric, after which the post-processing can be applied to the λ most unreliable checks.

To reduce the overall latency (initial MP decoding and post-processing), one may compute the check reliability values δ_c at an early iteration, *i.e.*, before the initial MP reaches the maximum number of decoding iterations. This allows the post-processing to start running in parallel before the initial MP has ended. If the initial MP succeeds later on, the post-processing will stop and the decoder will output the error found by the initial decoder. However, if the initial MP decoder fails, the post-processing will have already started, reducing the total latency. As it will be shown in Section 5, the error correction performance obtained by determining the list of least reliable checks using the soft information from either the last or an early iteration is almost the same, but the speedup is considerably higher in the latter case. Moreover, the reliability metric computed after a few iterations may be more accurate than the one computed at the last iteration, as the oscillation effects (also combined with saturation effects of the finite precision arithmetic) might alter quite considerably the accuracy of the reliability metric computed after a large number of iterations.

Algorithm 1: Generic Check-Agnosia Decoder

$\hat{\mathbf{e}} \leftarrow \mathrm{MP}(\mathbf{H}, \mathbf{s}, \{\gamma_q\})$
if $(\mathbf{H} \cdot \hat{\mathbf{e}} = \mathbf{s})$ **then**
  return $\hat{\mathbf{e}}$
else
  Compute the check reliability values:
  $\delta_c = \min_{q\in\mathcal{N}(c)} |\mu_{q\to c}| + \min_{2\, q\in\mathcal{N}(c)} |\mu_{q\to c}|,\ \forall c \in \mathcal{C}$
  Sort checks in increasing order of reliability,
  Extract $\{c_k\}_{k\in[\lambda]}$ the least reliable checks.
  for k in $1, \ldots, \lambda$ **do**
    $\forall q \in \mathcal{Q}$, set $\gamma'_q = \begin{cases} 0, & \text{if } q \in \mathcal{N}(c_k), \\ \gamma_q, & \text{otherwise.} \end{cases}$
    Determine $\mathfrak{N}_k = \cup_{q\in\mathcal{N}(c_k)} \mathcal{N}(q)$
    $\hat{\mathbf{e}} \leftarrow \mathrm{MP}^{\star}(\mathbf{H}, \mathbf{s}, \{\gamma'_q\}, \overline{\mathfrak{N}_k})$
    if $(\mathbf{H}_{|\overline{\mathfrak{N}_k}} \cdot \hat{\mathbf{e}}_{|\overline{\mathcal{N}(c_k)}} \neq \mathbf{s}_{|\overline{\mathfrak{N}_k}})$ **then**
      continue
    else
      Try to solve $\mathbf{H}_{|\mathfrak{N}_k} \cdot \hat{\mathbf{e}} = \mathbf{s}_{|\mathfrak{N}_k}$, while keeping $\hat{\mathbf{e}}_{|\overline{\mathcal{N}(c_k)}}$ unchanged, and bruteforcing $\hat{\mathbf{e}}_{|\mathcal{N}(c_k)}$
      if successful **then**
        return $\hat{\mathbf{e}}$
return decoding failure

We discuss now the brute-forcing of $\hat{\mathbf{e}}_{|\mathcal{N}(c_k)}$ in Algorithm 1. To solve the system $\mathbf{H}_{|\mathfrak{N}_k} \cdot \hat{\mathbf{e}} = \mathbf{s}_{|\mathfrak{N}_k}$ there are several possible methods, including Gaussian elimination. However, since the system to solve is small, brute-forcing, *i.e.*, trying all the possible combinations, hopefully finding one that satisfies the system[7], is a more efficient solution for hardware implemen-

[6] While different variations of this metric are possible (*e.g.*, the sum of the absolute values of all incoming messages) we have not observed any significant difference in terms of error correction performance. Also, similar reliability metrics can be obtained for other MP decoding algorithms, *e.g.*, sum-product.

[7] Note that there is not guaranteed that the system has a solution, as such, the algorithm can fail at this step.

tation. Moreover, it is not too difficult to see that the brute force approach can be simplified by taking into account the local structure of the code, eliminating a lot of computation. For instance, a check-node $c \in \mathfrak{N}_k \setminus \{c_k\}$ that has exactly one qubit-node in common with c_k, uniquely determines the value of that qubit.

3.2 Check-Agnosia Decoder Without System Solver

One alternative to determine $\hat{\mathbf{e}}_{|\mathcal{N}(c_k)}$, described in Algorithm 2, is to use a regular MP decoder that stops only if the full syndrome is matched. Precisely, the $\text{MP}^\star(\mathbf{H}, \mathbf{s}, \{\gamma'_q\})$ in Algorithm 2 is a regular MP decoder, initialized with qubit reliabilities $\{\gamma'_q\}$, and which stops when the full syndrome is satisfied. We keep the $\text{MP}^\star$ notation in the post-processing step only to distinguish it from the initial MP decoder (will be needed later on Section 4). To justify Algorithm 2, let us consider the case when the graph induced by any subset $\mathcal{S} \subseteq \mathcal{N}(c_k)$ contains at least a check-node of degree one. Then, assuming the MP decoder has converged on $\hat{\mathbf{e}}_{|\overline{\mathcal{N}(c_k)}}$, it will converge on the remaining $\hat{\mathbf{e}}_{|\mathcal{N}(c_k)}$ at the cost of a few more iterations. The above condition is the same as requiring $\mathcal{N}(c_k)$ contains no stopping subset[8], and running the MP for a few more iterations amounts to running a peeling decoding [32] on the erased qubits. For instance, if the Tanner graph contains no cycles of length four, then $\mathcal{N}(c_k)$ satisfies the no-stopping subset condition, and one extra iteration is enough to determine $\hat{\mathbf{e}}_{|\mathcal{N}(c_k)}$. The no-stopping subset condition may also be satisfied for graphs containing cycles of length four, but in such a case more than one extra iteration may be needed.

For a given Tanner graph the above no-stopping subset condition can easily be verified, and then we may use Algorithm 2 instead of Algorithm 1 (numerical simulations also confirmed that those two approaches give similar performance). For the simulation results shown later in this paper (Section 5), we always use Algorithm 2.

The presumably only meaningful case in which the no-stopping subset condition is not verified is when the code is auto-dual (*i.e.*, $H_x = H_z$), since in such a case $\hat{\mathbf{e}}_{|\mathcal{N}(c_k)}$ is the support of a codeword, hence a stopping set. It is worth noticing that for auto-dual codes, the check-agnosia (Algorithm 1) and stabilizer-inactivation [16] decoders are the same, up to the reliability metric used to select the λ least reliable check-nodes. However, for codes that are not auto-dual, the check-agnosia decoder, implemented as in Algorithm 2, presents several advantages, including the use of a simpler, hardware-friendly check-node reliability metric (and not requiring the use of the dual matrix), as well as the fact that it relies solely on MP decoding, eliminating the need of brute-forcing or other system solving methods.

A final remark is that all MP and $\text{MP}^\star$ decoders can implement a flooded or a layered schedule, as discussed in Section 2, to cope with the hardware constraints.

[8] A set of qubit-nodes is said to be a stopping set, if the induced subgraph contains no check-nodes of degree 1. If the qubit-nodes in a stopping set are erased, they can get no information during the MP decoding, that is, incoming and outgoing messages to and from these qubit-nodes remain equal to zero during the entire iterative decoding process.

Algorithm 2: Check-Agnosia Decoder Without System Solver

$\hat{\mathbf{e}} \leftarrow \text{MP}(\mathbf{H}, \mathbf{s}, \{\gamma_q\})$
if $(\mathbf{H} \cdot \hat{\mathbf{e}} = \mathbf{s})$ **then**
  return $\hat{\mathbf{e}}$
else
  Compute the check reliability values:
  $\delta_c = \min_{q \in \mathcal{N}(c)} |\mu_{q \to c}| + \min_{q \in \mathcal{N}(c)}{}_2 |\mu_{q \to c}|, \ \forall c \in \mathcal{C}$
  Sort checks in increasing order of reliability,
  Extract $\{c_k\}_{k \in [\lambda]}$ the least reliable checks.
  for k in $1, \ldots, \lambda$ **do**
    $\forall q \in \mathcal{Q}$, set $\gamma'_q = \begin{cases} 0, & \text{if } q \in \mathcal{N}(c_k), \\ \gamma_q, & \text{otherwise.} \end{cases}$
    $\hat{\mathbf{e}} \leftarrow \text{MP}^\star(\mathbf{H}, \mathbf{s}, \{\gamma'_q\})$
    if $(\mathbf{H} \cdot \hat{\mathbf{e}} = \mathbf{s})$ **then**
      return $\hat{\mathbf{e}}$
return decoding failure

4 Hardware Architectures

This section aims to analyze the impact of the post-processing algorithm on the hardware implementation, considering architectures with different schedules and varying degrees of parallelism. We carry out a detailed analysis of different corner cases, providing latency and power bounds to assist future hardware decoder designers.

4.1 MP Decoder Architecture

We consider first a single MP decoder, without any post-processing. To implement the MP decoder in hardware [9], one can use a fully parallel architecture, implementing a flooded schedule, referred to as flooded decoder, or a partly parallel architecture, implementing a layered schedule, referred to as layered decoder.

We will make standard assumptions[10] regarding

[9] Serial schedule is not considered due to its extremely large latency, not suitable for real-time implementations.

[10] The hardware implementation reported later on Section 4.4 is consistent with the assumptions made here.

the two above architectures [26]. For the flooded decoder, the Tanner graph is instantiated in hardware, where messages are exchanged through wires between processing units, corresponding to qubit- and check nodes. Each decoding iteration is performed in two clock cycles, with one clock cycle for qubit-node messages and a posteriori LLRs, and a second one for check-node messages. Thus, the worst case (maximum) latency of the flooded decoder is equal to $(1 + 2I_F)/f_F$ (s), where we count one clock-cycle for data loading, I_F is the maximum number of decoding iterations of the flooded decoder, and f_F is the clock frequency.

For the layered decoder, the number of processing units instantiated in hardware is given the size of the largest layer[11], messages are exchanged through shared memory, and each processing unit is reused η_L times for each decoding iteration, where η_L denotes the number of layers per iteration. The worst case latency of the layered decoder is equal to $(1+\eta_L I_L)/f_L$ (s), where we count again one clock-cycle for data loading, I_L is the maximum number of decoding iterations of the layered decoder, and f_L is the clock frequency.

Two observations are in place here. First, the flooded architecture may lead to a large number of connections among processing units, causing routing congestion in case of large codes. Due to the large interconnect network, the operating clock frequency of the flooded architecture (f_F) is usually smaller than twice[12] that of the layered architecture (f_L). Second, as discussed in Section 2, the layered schedule propagates information about twice faster than the flooded one, thus the maximum number of iterations of the layered architecture (I_L) is usually smaller than that of the flooded architecture (I_F). Overall, this can make the layered architecture comparably fast to the flooded one, despite the fact that it employs a reduced degree of parallelism (of course, the number of layers per iteration has to be sufficiently small).

Finally, one possible approach to further increase the clock frequency of the layered decoder is to pipeline the design (*i.e.*, perform each layer in a number of pipelined clock cycles). However, this may lead to delayed message write-backs in memories, and thus, to pipeline related hazards [33]. Solving such hazards (without relying on pipeline stalls, introducing extra latency) can be done for classical LDPC codes at the code construction stage [34]. However such solutions are not generic (need a specific code construction) and may not apply to qLDPC codes. Therefore, to keep the analysis as generic as possible, we do not consider pipelined designs in this work.

[11] Usually all layers have the same size, although this condition is more difficult to satisfy for qLDPC codes [29].

[12] Note that in the flooded architecture, qubit and check-node messages are computed in two different clock cycles, by different processing units, while in the layered architecture they are computed in the same clock cycle, by a processing unit that merges the qubit and check node processing.

4.2 Post-Processing Elements

For the check-agnosia scheme, the first step after the MP decoder is the computation of the check reliability values, as outlined in Algorithms 1 and 2. The metric used to calculate the check reliability, denoted as δ_c, involves adding the two least reliable messages. These values are computed during the tree finder process employed to calculate check-node messages in the min-sum decoder. Thus, the only additional hardware required is an adder per check-node to compute δ_c. These values are updated on-the-fly during each iteration, eliminating the need for extra clock cycles after the MP decoder. As described earlier in Section 3, one does not have to wait until the end of the initial MP decoder to start the post-processing (the impact of utilizing the δ_c information from early iterations will be evaluated in Section 5). In the proposed architecture, the δ_c values can be stored in the registers of the sorting unit (see below) before the first MP decoder completes, without any additional hardware. This allows absorbing some additional latency and initiating the post-processing MP* decoders before the initial MP decoder completes.

After the δ_c values are available, a sort of the checks in order of reliability is computed. To sort the checks in order of increasing reliability $|\mathcal{C}|$-1 comparators are required to implement a tree structure, which should be pipelined to avoid increasing the critical path of the decoder. The number of clock cycles needed to obtain the complete sorted list is $\lceil \lambda/2 \rceil \times \lceil \log_2 |\mathcal{C}| \rceil$.

4.3 Overall Check-Agnosia Architecture

In this section, we detail the check-agnosia architecture corresponding to Algorithm 2 (that relies on MP decoding only, without brute-forcing). After the list of λ least reliable checks is obtained, the λ MP* decoders are performed. Depending on the time constraints and/or power budget, we may consider two different approaches, illustrated in Figure 1.

The first approach consists of performing the λ MP* decoders sequentially, reusing the same hardware as the one used for MP. Only $|\mathcal{Q}|$ extra multiplexors are required to choose between $\gamma'_q = \gamma_q$ or $\gamma'_q = 0$, and $|\mathcal{C}|$ extra multiplexors are required to decide which syndromes belong to $\mathbf{s}_{|\mathfrak{N}_k}$, depending on the check c_k. This approach of reusing hardware yields higher latency, but maybe interesting for a quantum computer with time constraints close to microseconds, *e.g.*, based on trapped ion technology [3].

For the second approach, the λ MP* decoders are performed in parallel, by using dedicated hardware. Moreover, the λ MP* decoders may start before the initial MP completes, using check-reliability values computed at an early iteration, that we will denote

in the sequel by I_{δ_c}. This approach may be interesting for quantum technologies with more restrictive latency constraints, but having in mind that power can be also a limitation, as happens with superconducting qubits in which the decoder needs to reduce its power budget when it operates close to the quantum chip at cryogenic temperatures.

To illustrate the degree of complexity in hardware implementations and measure the gap between the proposed solutions to latency/power constraints, we analyze below the Pareto designs for the two approaches above, where the MP decoder uses either a flooded or a layered schedule. We provide the worst-case latency (simply referred to as latency), as well as the power consumption as a function of the nominal power consumption of the MP decoder, denoted by P_F or P_L, with a subscript indicating the flooded or layered architecture (we may reasonably assume that MP and MP* yield the same power consumption). For the latency value, we take into account the latency induced by sorting the check nodes according to their reliability (Section 4.2). The corresponding power consumption is not accounted for, we will assume it is negligible with respect to the power consumption of the MP decoder.

Flooded MP/MP* decoders:

1. Hardware reuse (sequential post-processing)

   MP flooded decoder + check reliability unit + one MP* flooded decoder running λ rounds:

   - Latency: $\big[(1+2I_F) + \lceil\lambda/2\rceil\lceil\log_2|\mathcal{C}|\rceil + \lambda(1+2I_F)\big]/f_F$
   - Power: P_F

2. Dedicated hardware (parallel post-processing)

   MP flooded decoder + check reliability unit starting after iteration $I_{\delta_c} + \lambda$ MP* flooded decoders running in parallel

   - Latency: $\big[(1+2I_{\delta_c}) + \lceil\lambda/2\rceil\lceil\log_2|\mathcal{C}|\rceil + (1+2I_F)\big]/f_F$
   - Power: $(\lambda+1)P_F$

Layered MP/MP* decoders:

1. Hardware reuse (sequential post-processing)

   MP layered decoder + check reliability unit + one MP* layered decoder running λ rounds:

   - Latency: $\big[(1+\eta_L I_L) + \lceil\lambda/2\rceil\lceil\log_2|\mathcal{C}|\rceil + \lambda(1+\eta_L I_L)\big]/f_L$
   - Power: P_L

2. Dedicated hardware (parallel post-processing)

   MP layered decoder + check reliability unit starting after iteration $I_{\delta_c} + \lambda$ MP* layered decoders running in parallel

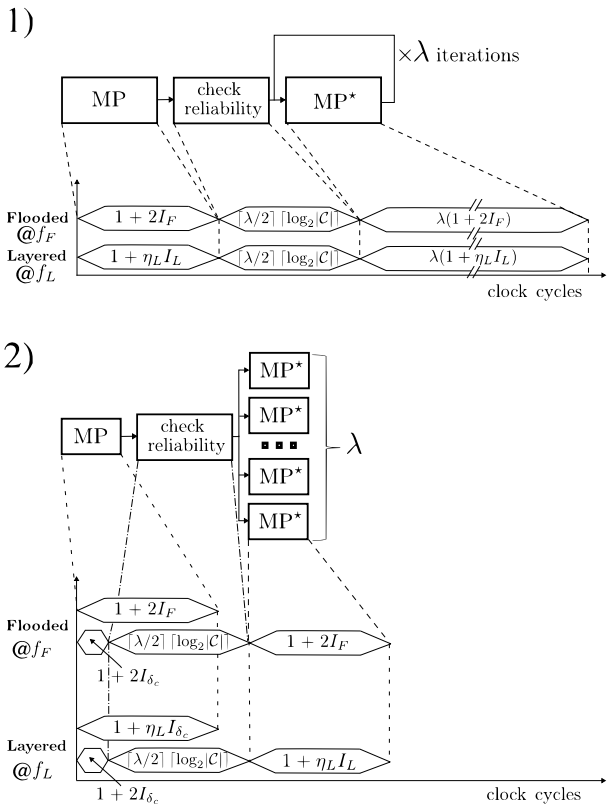


Figure 1: Comparison of different architectures for the check-agnosia decoder. The clock cycle diagram is included for the different proposals (Warning: drawing is not to scale). In case 1), MP and MP* use the same hardware.

   - Latency: $\big[(1+\eta_L I_{\delta_c}) + \lceil\lambda/2\rceil\lceil\log_2|\mathcal{C}|\rceil + (1+\eta_L I_L)\big]/f_L$
   - Power: $(\lambda+1)P_L$

4.4 Implementation Results

To illustrate the analysis from the previous section, we have implemented both flooded and layered NMS decoders on a Xilinx FPGA xcv095 board, for the B1[[882, 24]] code from [5]. The implemented decoders use finite precision arithmetic, with exchanged messages quantized on 6 bits, and a posteriori LLR values quantized on 8 bits. The parity-check matrix (for both X and Z errors) is of size 441×882 (check-nodes $\times$ qubit-nodes) and has no four-cycles (thus, it satisfies the no-stopping subset condition, and we may safely apply Algorithm 2).

The flooded NMS / NMS* decoders achieve a maximum operating frequency $f_F = 100\,\text{MHz}$ (corresponding to a critical path of 10 ns), with 62% of the hardware resources of the device utilized, and a total power consumption $P_F = 5.5\,\text{W}$.

To implement the layered decoders, we use the 2-covering approach from [29], where 7 overlapping layers are used to cover 2 iterations, yielding a fractional

Table 1: Latency (L) and power consumption (P) values for the Pareto designs in Section 4.3 ($I_{\max}$ is the table stands for I_F for flooded architectures, or for I_L for layered ones).

	Flooded	Layered
HW reuse	$L = 7.2\,\mu$s $P = 5.5\,$W	$L = 7.9\,\mu$s $P = 2.03\,$W
Dedicated HW $I_{\delta_c} = I_{\max}$	$L = 1.7\,\mu$s $P = 60.5\,$W	$L = 1.9\,\mu$s $P = 22.3\,$W
Dedicated HW $I_{\delta_c} = 3$	$L = 1.1\,\mu$s $P = 60.5\,$W	$L = 1.4\,\mu$s $P = 22.3\,$W

number of layers per iteration $\eta_L = 3.5$. The layered NMS / NMS* decoders achieve a maximum operating frequency $F_L = 80\,$MHz (corresponding to a critical path of 12.5 ns), where about 25% is due to the logic depth of the operations and 75% is due to the routing limitations of the FPGA device. The decoder uses only 13% of the hardware resources of the device, and the total power consumption is around $P_L = 2.03\,$W.

We consider a maximum number of decoding iterations $I_F = 30$ for the flooded decoders, and $I_L = 15$ for the layered decoders (due to faster convergence). For the post-processing step, we consider a list of $\lambda = 10$ least reliable checks (these parameters will be evaluated from the error correction perspective in Section 5). Latency and power consumption values are summarized in Table 1, for the Pareto designs considered in the previous section. Note that we consider two cases for the dedicated hardware scenario, in which the iteration I_{δ_c} (used to compute the check-node reliability values) is chosen to be either the last or the third iteration of the NMS decoder.

It can be observed that the layered architecture achieves latency values close to the flooded one, despite the fact it employs a degree of parallelism 3.5 times lower, while considerably reducing the power consumption. It is also worth noticing that the part of the latency due to the sorting unit is 0.45 μs for the flooded architectures, and 0.56 μs for the layered ones. To reduce the latency of the sorting unit further optimizations are possible (*i.e.*, carefully balancing the pipeline stages of the sorting unit by taking into account the maximum critical path latency of the MP decoder, splitting the sorting unit into layers in case of a layered schedule, or using a different clock domain for the sorting unit), which are however behind the scope of this work. We mention that the maximum frequency that can be reached for the sorting unit (implemented alone) is 230 MHz, which gives a lower bound on the achievable latency of 0.2 μs[13].

Moreover, as will be shown in Section 5, because of the highly degenerate structure of the codes, the layered schedule provides better error correction performance than the flooded one, even if the number of decoding iterations of the latter exceeds significantly the number of decoding iterations of the former (in fact, to get a flooded decoder that approaches the layered decoder, albeit not closely, one would have to go for at least 60 iterations, see Section 5). One last advantage of the layered architecture, reported in [29], is that the logical error rate can be considerably improved by processing layers in random order at each iteration. Such a random layer order can be implemented at a very low cost, as it only requires modifying the ROM memory that stores the layers' control sequence and including a deeper memory with a pseudo-random sequence of layers.

From the results presented before, it can be concluded that timing constraints can be in the range of the requirements reported in [12] for transmons and ion trap technology, between microseconds and milliseconds. However, these implementations do not meet the highly restrictive conditions of superconducting qubits in both time and power which are around 400 ns and 1W, see [22]. The difference compared to the fastest solution in Table 1 exceeds 3 times the time budget and it is more than one order of magnitude far in terms of power consumption. For these scenarios, it is important to remark that other approaches to implementation like ASICs or more advanced FPGA devices based on 16nm CMOS process or below (note that the xcvu095 belongs to the previous generation of 20nm) need to be explored in future work. Moreover, exploiting a ping-pong architecture that takes benefit of the pipeline registers to reduce the number of MP* decoders to half for the parallel implementation of flooded schedule can be a good proposal to reduce power consumption to almost half.

Extrapolating from state-of-the-art ASIC implementations of classical LDPC decoders, a clock frequency of 151 MHz is reported in [35] for a 65-nm CMOS ASIC implementation of a min-sum decoder using the layered architecture described in Section 4.1, for a regular LDPC code with characteristics similar to those of the B1 code investigated here[14]. The operating frequency is expected to further increase for the B1 code, given that both the parity check matrix and the layer size are smaller than that of the LDPC code in [35]. For $f_L = 151\,$MHz, the latency of the layered architecture with parallel post-processing (dedicated hardware) is equal to 1 μs if $I_{\delta_c} = I_L$ (last iteration), and 0.73 μs if $I_{\delta_c} = 3$. Since the operating frequency increases with decreasing technology node, and assuming an inverse-linear frequency scaling [36], we may conclude that a latency constraint around 400 ns or below can be easily achieved for more advanced technology nodes, *e.g.*, below 22 nm (today

[13] At an operating frequency of 230 MHz, the power consumption for the sorting unit is 0.66 W, versus 0.26 W at at 100 MHz.

[14] In [35], the parity check matrix is of size 648×1296, with column weight 3 and rows weight 6. Each layer consists of 216 checks (referred to as full-layer therein). The parity check matrix of the B1 code is of size 441×882, with column weight 3 and rows weight 6, and each layer consists of 126 checks.

technology scaling is actually much lower).

Finally, we note that all the previous results assume that the check-agnosia decoder is implemented as in Algorithm 2. For the codes where Algorithm 2 cannot be applied, the latency will be a little bit worse than what was computed here, due to the brute-forcing step in Algorithm 1. We also provide in Appendix B arguments for why OSD post-processing (widely used today in the community for decoding of small to medium LDPC codes) is not a viable solution going forward, if trying to cope with the hardware implementation constraints.

5 Error Correction Performance

In this section, we evaluate the error correction performance of the proposed check-agnosia post-processing. The codes used are B1[[882, 24]] and C2[[1922, 50]] from [5]. For both codes, the no-stopping subset condition from Section 3.2 is satisfied, hence in the following all simulations are performed using Algorithm 2 (without brute-forcing the system).

As our post-processing is targeted at decoding X and Z errors separately, we use an X noise model, and thus "physical error rate" does actually refer to the physical X error rate.

Our numerical simulations are consistent with the parameters used in Section 4.4. Precisely, we consider a finite-precision NMS decoder, using 6 bits for the exchanged messages, and 8 bits for the a posteriori LLRs. Although in floating point precision the initial (a priori) LLRs of the NMS decoder can be scaled to 1, in finite precision the initial LLR values have a non-negligible impact on convergence. In the simulations, we use the following parameters, optimized by extensive search. For the flooded decoder we set the initial LLR values to $\mathrm{LLR}_{\mathrm{init}} = 12$ and the NMS scaling factor is set to $s_{\mathrm{NMS}} = 0.875$. For the layered decoder we use $\mathrm{LLR}_{\mathrm{init}} = 8$ and NMS scaling factor $s_{\mathrm{NMS}} = 0.9375$. Note that scaling factors are a sum of powers of 2 and as such the scaling operation can be implemented efficiently in hardware, using only SHIFT and ADD operations.

For the maximum number of decoding iterations, we use $I_F = 60$ for the flooded decoder, and $I_L = 15$ for the layered decoder. The maximum number of decoding iterations for the flooded decoder is the only deviation with respect to the parameters used in Section 4.4 (where $I_F = 30$ was used). In fact, our goal here is to demonstrate the advantage in terms of error correction performance of the layered architecture as compared to the flooded one, even when the latter employs a significantly higher number of decoding iterations. For the post-processing part, we use $\lambda = 10$.

Whenever the layered decoding is used, we add the random ordering perturbation introduced in [29] that was shown to significantly improve the decoding convergence.

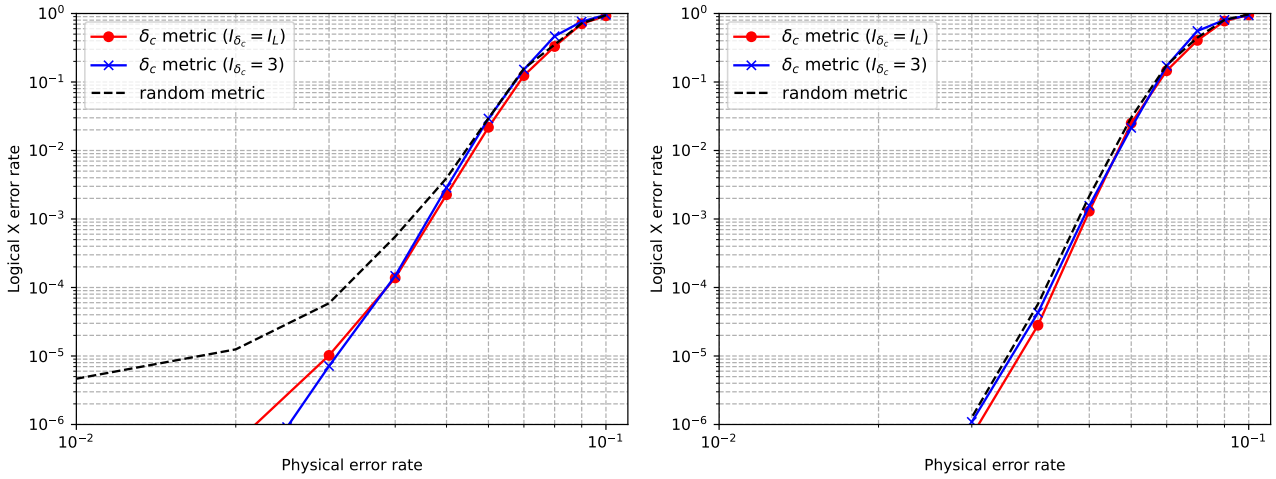
Figure 2 shows the impact of the iteration I_{δ_c} used to select the checks in the post-processing, all simulations are done on the B1 code. For flooded simulations (Figure 2(a)), it is actually beneficial to use the 3rd iteration for the metric instead of the last (60th iteration). The most probable explanation is that the relatively high number of iterations combined with the finite precision algorithm makes the metric less reliable after a larger number of iterations. For comparison purposes, we also consider a random metric, corresponding to a random choice of the λ checks in the post-processing. As it can be seen, the random metric exhibits a bad error floor, validating the metric used in that case.

For layered simulation (Figure 2(b)), all three curves are close by, since the layered NMS decoder with random layer ordering performs already very well.

In fact, the three metrics yield virtually the same performance of the check-agnosia decoder, but which is better than the layered NMS with 30 iterations and without post-processing in Figure 3(a). Although the metric is less important in this case, this shows that the perturbation introduced by the post-processing step in the input reliabilities has an impact on the decoding, and that it is better to run multiple decoders in parallel with perturbed inputs and fewer iterations rather than running a single decoder for a long time. As a whole, this validates the fact that the post-processing can be done efficiently using the dedicated hardware approach, increasing the post-processing parallelism and improving the latency.

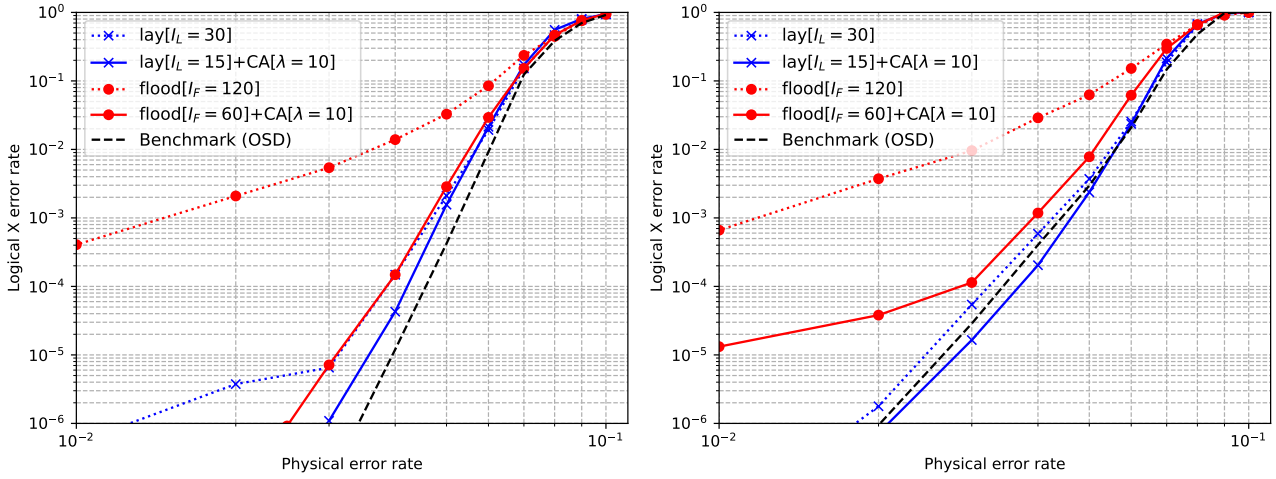
We would also like to make a case for the choice $I_{\delta_c} = 3$. This hyperparameter can be optimized to get the best numerical results for a given code. However, the value 3 here was chosen for a different reason. Since both codes have girth 6, choosing I_{δ_c} to be equal to 3 guarantees that when the aposteriories are extracted, the decoder got access to the information of the biggest neighbourhood of each variable nodes *without loopy information*. This ensures that although it is very local, this information is also less noisy than information coming from later rounds.

In Figure 3, the post-processing is applied to the codes B1 and C2, with both flooded and layered schedules. For a comparison with the state of the art, in both figures, we added a dashed black curve of an optimized NMS-OSD decoder using 100 iterations, floating point NMS with a scaling factor of 0.625 [5]. Keep in mind that this decoder is not at all hardware-friendly, in terms of complexity, latency and power consumption, and it only serves as a reference. As it can be seen from both simulations, our results are matching closely the performance of the OSD post-processing, concretely showing the effectiveness of our hardware-friendly approach. In both figures, the results in red are the curves for flooded and in blue for layered. Each time, the dotted curves show the per-



(a) $B1[[882, 24]]$ flooded (b) $B1[[882, 24]]$ layered

Figure 2: Performance of the check-agnosia decoder with different reliability metrics (B1 code)



(a) $B1[[882, 24]]$ (b) $C2[[1922, 50, 16]]$

Figure 3: Analysis of the check-agnosia post-processing on codes B1 and C2 (δ_c metric, with $I_{\delta_c} = 3$).

formance of the decoder without post-processing.

On the B1-code for the flooded schedule, the impact of the post-processing is clear, and the check-agnosia flooded decoder exhibits good performance while keeping a latency around 1.7 μs (taking into account $I_F = 60$). For the layered schedule, the use of the post-processing increases the steepness of the waterfall. The check-agnosia layered decoder keeps the latency at around 1.4 μs. (Latency values above correspond to our FPGA implementation from Section 4.4.)

On the C2 Code, the performance gains for flooded are clear even if the post-processing suffers from a relatively high error floor. For layered scheduling, once again check-agnosia achieves better results in the error floor compared to no post-processing, closely matching the NMS-OSD curve.

Further numerical results are provided in Appendix A, where we evaluate the error correction performance of the check-agnosia decoder on the family of T-codes from [20], showing a threshold phenomenon with hyperparameter $\lambda = 0.02 \times |C|$.

6 Conclusions

This work introduced the check-agnosia algorithm, a new post-processing method improving on the syndrome-inactivation algorithm from a hardware-oriented viewpoint. Interestingly, although in the general case brute-forcing a small linear system may still be needed, for a large class of qLDPC codes the check-agnosia post-processing relies only on MP decoding, eliminating the need for any system solver. The proposed solution is flexible and it allows devising different hardware architectures, in order to meet the latency or the power constraints of the quantum system. The analysis carried out in the document, along with the hardware implementation results for MP decoders (our own implementation on an FPGA board, or results extrapolated from state-of-the-art ASIC implementations), showed that our solution can meet latency constraints of a wide range of quantum technologies, while providing state of the art error-correction performance, with hardware-accurate, finite-precision arithmetic. To the best of our knowledge, there is no prior work on the hardware architecture and implementation of a post-processing enhanced MP decoder for qLDPC codes. An interesting open question going forward would be to look at space-time decoding and see if the underlying graph structure lends itself well to the use of the check-Agnosia post-processing without system-solving.

7 Acknowledgment

This work is supported by the QuantERA grant EQUIP (French ANR-22-QUA2-0005-01 and Spain MCIN/AEI/10.13039/501100011033), by the Plan France 2030 (ANR-22-PETQ-0006), the grant PCI2022-132922 funded by Agencia Estatal de Investigación, Ministerio de Ciencia e Innovación, Gobierno de España and by the European Union "NextGenerationEU/PRTR".

A Hyperparameter λ and decoding threshold

In Figure 4, we include additional numerical results giving a threshold for a constant rate family of LDPC codes, namely the T codes family from [20]. Since we are not aware of a simple way to build layers for this family of codes, we ran the simulations using a serial decoder, which is a fair approximation of the numerical results one would get with layered decoding. We use check-agnosia with hyperparameter $\lambda = 0.02 \times |\mathcal{C}| = 0.01 \times |\mathcal{Q}|$, since for all the codes of the T family, $|\mathcal{C}| = |\mathcal{Q}|/2$. The computational complexity of the algorithm hence is $(0.01\times)n^2 \times \log n$, where $n = |\mathcal{Q}|$ is the number of qubits, and this complexity can be spread between time and energy consumption depending on the architecture needs (see Fig 1). Furthermore, we make a case that the average complexity of the decoder is actually much better than that. On the figure, we also included the average number of inactivations (denoted λ_{avg}) for physical error rates 0.6 and 0.5, where the average values get very close to one (meaning only one inactivation might usually be necessary). Since this number goes close to one for low error-rates, it means that in practice the cost of the post-processing could only add a constant mul-

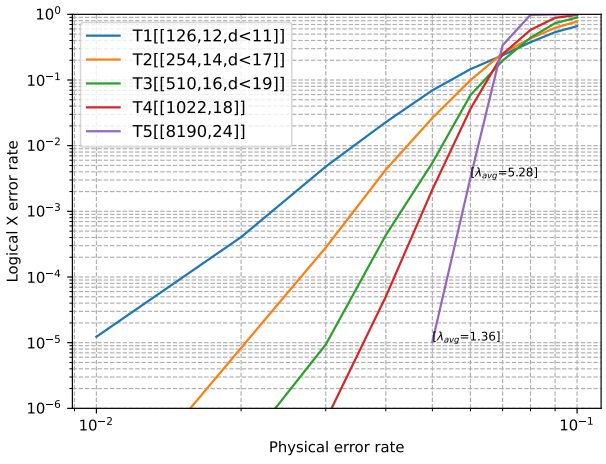


Figure 4: Check-agnosia threshold for the T codes family from [20]. Serial scheduling with random ordering and 15 iterations. Normalized min-sum with scaling factor 0.9375 and finite precision arithmetic, with 6-bit quantization for the input LLRs and exchanged messages, and 8-bit quantization for the a posteriori LLRs. For the post-processing, check-agnosia is used with $I_{\delta_c} = 3$ and $\lambda = 0.02 \times |\mathcal{C}|$.

tiplicative overhead. This lambda average is particularly meaningful in a sequential architecture where we stop the post-processing as soon as the first post-processing converges. In the parallel architecture, it should still be possible to optimize the actual number of parallel runs if we have access to some prior information on the noise level, *e.g.*, by considering a pool of MP* decoders that serve for the post-processing of several logical qubits and are dynamically allocated between them.

B Latency comparison of CA and OSD

We provide below a comparison, in terms of latency, between the check agnosia proposal and the OSD post-processing solution. This comparison is similar to the method presented in [21], and is intended to clarify the differences between the two solutions with respect to hardware VLSI implementations.

We consider a layered MP decoder, with check-agnosia implemented through "Dedicated hardware" (that is, the λ MP* decoders are executed in parallel). Since the layered MP* decoders achieve a maximum operating frequency $F_L = 80\,\text{MHz}$ (corresponding to a critical path of 12.5 ns), the total latency of the check-agnosia post-processing is $(12.5{\times}3.5{\times}15) = 656.25\,\text{ns}$. We have omitted here the latency of the sorting unit, required to sort the check-nodes according to their reliability.

Considering now the OSD post-processing, we will omit again the latency of the sorting unit, required this time to sort the qubit-nodes according to their reliability. We will actually consider only the latency of the Gaussian elimination step required by OSD post-processing (and omit the latency of any other steps). Refs. [37, 38] below provide the two main highly parallel architectures known in the literature to perform Gaussian elimination over finite fields. However, in both cases, the number of clock cycles required to perform Gaussian elimination is equal to $(M^2 + M)/2$, where M is the number of rows of the parity-check matrix. Thus, the latency of the Gaussian elimination implementation is determined by $T_{\text{OSD}} = (M^2 + M)/2/f_{\text{OSD}}$, where f_{OSD} is the operating frequency. So the frequency required to achieve the same time budget as our proposal is $f_{\text{OSD}} = (441^2 + 441)/2/656.25\,\text{ns} = 148.5\,\text{GHz}$. Such a frequency is completely unrealistic, and would certainly lead to timing violations in the design (note that it is $148.5/0.08 = 1856$ times larger than the one of the layered MP decoder). Besides, it would also translate into an extremely large power consumption, which typically increases linearly with the operating frequency.

References

[1] Z. Babar, P. Botsinis, D. Alanis, S. X. Ng, and L. Hanzo, "Fifteen Years of Quantum LDPC Coding and Improved Decoding Strategies," *IEEE Access*, vol. 3, pp. 2492–2519, 2015. [Online]. Available: https://doi.org/10.1109/ACCESS.2015.2503267

[2] D. Gottesman, "Fault-tolerant quantum computation with constant overhead," *Quantum Information & Computation*, vol. 14, no. 15-16, pp. 1338–1372, 2014. [Online]. Available: https://doi.org/10.48550/arXiv.1310.2984

[3] N. P. Breuckmann and J. N. Eberhardt, "Quantum Low-Density Parity-Check Codes," *PRX Quantum*, vol. 2, p. 040101, Oct 2021. [Online]. Available: https://doi.org/10.1103/PRXQuantum.2.040101

[4] S. Bravyi, O. Dial, J. M. Gambetta, D. Gil, and Z. Nazario, "The future of quantum computing with superconducting qubits," *Journal of Applied Physics*, vol. 132, no. 16, 2022. [Online]. Available: https://doi.org/10.1063/5.0082975

[5] P. Panteleev and G. Kalachev, "Degenerate Quantum LDPC Codes With Good Finite Length Performance," *Quantum*, vol. 5, p. 585, nov 2021. [Online]. Available: https://doi.org/10.22331%2Fq-2021-11-22-585

[6] N. P. Breuckmann and J. N. Eberhardt, "Balanced Product Quantum Codes," *IEEE Transactions on Information Theory*, vol. 67, no. 10, pp. 6653–6674, oct 2021. [Online]. Available: https://doi.org/10.1109%2Ftit.2021.3097347

[7] A. Leverrier and G. Zémor, "Quantum Tanner codes," in *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2022, pp. 872–883. [Online]. Available: https://doi.org/10.1109/FOCS54457.2022.00117

[8] N. P. Breuckmann and V. Londe, "Single-Shot Decoding of Linear Rate LDPC Quantum Codes With High Performance," *IEEE Transactions on Information Theory*, vol. 68, no. 1, pp. 272–286, 2022. [Online]. Available: https://doi.org/10.1109/TIT.2021.3122352

[9] S. Bravyi, A. W. Cross, J. M. Gambetta, D. Maslov, P. Rall, and T. J. Yoder, "High-threshold and low-overhead fault-tolerant quantum memory," *preprint arXiv:2308.07915*, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2308.07915

[10] J. Roffe, "Towards practical quantum LDPC codes," *Quantum Views*, vol. 5, p. 63, 2021. [Online]. Available: https://doi.org/10.22331/qv-2021-11-30-63

[11] N. Raveendran and B. Vasić , "Trapping Sets of Quantum LDPC Codes," *Quantum*, vol. 5, p. 562, oct 2021. [Online]. Available: https://doi.org/10.22331%2Fq-2021-10-14-562

[12] F. Battistel, C. Chamberland, K. Johar, R. W. J. Overwater, F. Sebastiano, L. Skoric, Y. Ueno, and M. Usman, "Real-Time Decoding for Fault-Tolerant Quantum Computing: Progress, Challenges and Outlook," 2023. [Online]. Available: https://doi.org/10.1088/2399-1984/aceba6

[13] N. Raveendran, E. Boutillon, and B. Vasic, "Turbo-XZ Algorithm: Low-Latency Decoders for Quantum LDPC Codes," in *International Symposium on Topics in Coding*, 2023. [Online]. Available: https://doi.org/10.1109/ISTC57237.2023.10273490

[14] N. Delfosse and N. H. Nickerson, "Almost-linear time decoding algorithm for topological codes," *Quantum*, vol. 5, p. 595, Dec. 2021. [Online]. Available: https://doi.org/10.22331/q-2021-12-02-595

[15] Y.-H. Liu and D. Poulin, "Neural Belief-Propagation Decoders for Quantum Error-Correcting Codes," *Phys. Rev. Lett.*, vol. 122, p. 200501, May 2019. [Online]. Available: https://doi.org/10.1103/PhysRevLett.122.200501

[16] J. D. Crest, M. Mhalla, and V. Savin, "Stabilizer Inactivation for Message-Passing Decoding of Quantum LDPC Codes," in *2022 IEEE Information Theory Workshop (ITW)*, 2022, pp. 488–493. [Online]. Available: https://doi.org/10.1109/ITW54588.2022.9965902

[17] X. Ni, "Neural Network Decoders for Large-Distance 2D Toric Codes," *Quantum*, vol. 4, p. 310, Aug 2020. [Online]. Available: http://doi.org/10.22331/q-2020-08-24-310

[18] C. Chamberland and P. Ronagh, "Deep neural decoders for near term fault-tolerant experiments," *Quantum Science and Technology*, vol. 3, no. 4, p. 044002, jul 2018. [Online]. Available: https://doi.org/10.1088/2058-9565/aad1f7

[19] P. Murali, N. M. Linke, M. Martonosi, A. J. Abhari, N. H. Nguyen, and C. H. Alderete, "Full-Stack, Real-System Quantum Computer Studies: Architectural Comparisons and Design Insights," in *Proceedings of the 46th International Symposium on Computer Architecture*, ser. ISCA '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 527–540. [Online]. Available: https://doi.org/10.1145/3307650.3322273

[20] J. Roffe, D. R. White, S. Burton, and E. Campbell, "Decoding across the quantum low-density parity-check code landscape," *Physical Review Research*, vol. 2, no. 4, dec 2020. [Online]. Available: https://doi.org/10.1103%2Fphysrevresearch.2.043423

[21] J. Valls, F. Garcia-Herrero, N. Raveendran, and B. Vasić, "Syndrome-Based Min-Sum vs OSD-0 Decoders: FPGA Implementation and Analysis for Quantum LDPC Codes," *IEEE Access*, vol. 9, pp. 138 734–138 743, 2021. [Online]. Available: https://doi.org/10.1109/ACCESS.2021.3118544

[22] Y. Ueno, M. Kondo, M. Tanaka, Y. Suzuki, and Y. Tabuchi, "QECOOL: On-line quantum error correction with a superconducting decoder for surface code," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, dec 2021. [Online]. Available: https://doi.org/10.1109/dac18074.2021.9586326

[23] S. S. Tannu, Z. A. Myers, P. J. Nair, D. M. Carmean, and M. K. Qureshi, "Taming the Instruction Bandwidth of Quantum Computers via Hardware-Managed Error Correction," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-50 '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 679–691. [Online]. Available: https://doi.org/10.1145/3123939.3123940

[24] E. Sharon, S. Litsyn, and J. Goldberger, "An efficient message-passing schedule for LDPC decoding," in *2004 23rd IEEE Convention of Electrical and Electronics Engineers in Israel*, 2004, pp. 223–226. [Online]. Available: https://doi.org/10.1109/EEEI.2004.1361130

[25] M. Mansour and N. Shanbhag, "VLSI architectures for SISO-APP decoders," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 4, pp. 627–650, 2003. [Online]. Available: https://doi.org/10.1109/TVLSI.2003.816136

[26] E. Boutillon and G. Masera, "Hardware design and realization for iteratively decodable codes," in *Channel coding: Theory, algorithms, and applications*, D. Declercq, M. Fossorier, and E. Biglieri, Eds. Elsevier, 2014, pp. 583–642. [Online]. Available: https://doi.org/10.1016/B978-0-12-396499-1.00013-3

[27] V. Savin, "LDPC decoders," in *Channel coding: Theory, algorithms, and applications*, D. Declercq, M. Fossorier, and E. Biglieri, Eds. Elsevier, 2014, pp. 211–260. [Online]. Available: https://doi.org/10.1016/C2011-0-07211-3

[28] J. Zhang, Y. Wang, M. P. C. Fossorier, and J. S. Yedidia, "Iterative decoding with replicas," *IEEE Transactions on Information Theory*, vol. 53, no. 5, pp. 1644–1663, 2007. [Online]. Available: https://doi.org/10.1109/TIT.2007.894683

[29] J. du Crest, F. Garcia-Herrero, M. Mhalla, V. Savin, and J. Valls, "Layered decoding of quantum LDPC codes," in *International Symposium on Topics in Coding*, 2023, arXiv:2308.13377. [Online]. Available: https://doi.org/10.1109/ISTC57237.2023.10273477

[30] G. Liva, E. Paolini, B. Matuz, and M. Chiani, "A decoding algorithm for LDPC codes over erasure channels with sporadic errors," in *2010 48th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2010, pp. 458–465. [Online]. Available: https://doi.org/10.1109/ALLERTON.2010.5706942

[31] A. D. Kumar and A. Dukkipati, "A two stage selective averaging LDPC decoding," in *2012 IEEE International Symposium on Information Theory Proceedings*, 2012, pp. 2866–2870. [Online]. Available: https://doi.org/10.1109/ISIT.2012.6284048

[32] M. Luby, M. Mitzenmacher, M. Shokrollahi, and D. Spielman, "Efficient erasure correcting codes," *IEEE Trans. on Information Theory*, vol. 47, no. 2, pp. 569–584, 2001. [Online]. Available: https://doi.org/10.1109/18.910575

[33] O. Boncalo, G. Kolumban-Antal, A. Amaricai, V. Savin, and D. Declercq, "Layered LDPC decoders with efficient memory access scheduling and mapping and built-in support for pipeline hazards mitigation," *IEEE Transactions on Circuits and Systems I: Regular Papers*, pp. 1643–1656, December 2018. [Online]. Available: https://doi.org/10.1109/TCSI.2018.2884252

[34] O. Boncalo, G. Kolumban-Antal, D. Declercq, and V. Savin, "Code-design for efficient pipelined layered LDPC decoders with bank memory organization," *Microprocessors and Microsystems*, vol. 63, pp. 216–225, September 2018. [Online]. Available: https://doi.org/10.1016/j.micpro.2018.09.011

[35] T. T. Nguyen-Ly, V. Savin, K. Le, D. Declercq, F. Ghaffari, and O. Boncalo, "Analysis and design of cost-effective, high-throughput LDPC decoders," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 3, pp. 508–521, March 2018. [Online]. Available: https://doi.org/10.1109/TVLSI.2017.2776561

[36] J. R. Hauser, "MOSFET device scaling," in *Handbook of Semiconductor Manufacturing Technology*. Boca Raton, FL: CRC Press, 2008. [Online]. Available: https://doi.org/10.1201/9781420017663

[37] A. Rupp, J. Pelzl, C. Paar, M. Mertens, and A. Bogdanov, "A parallel hardware architecture for fast gaussian elimination over gf(2)," in *2006 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2006, pp. 237–248. [Online]. Available: https://doi.org/10.1109/FCCM.2006.12

[38] B. Hochet, P. Quinton, and Y. Robert, "Systolic solution of linear systems over gf(p) with partial pivoting," in *1987 IEEE 8th Symposium on Computer Arithmetic (ARITH)*, 1987, pp. 161–168. [Online]. Available: https://doi.org/10.1109/ARITH.1987.6158700