

Local Clustering Decoder as a fast and adaptive hardware decoder for the surface code

Corresponding Author: Mr Abbas Ziad

This file contains all reviewer reports in order by version, followed by all author rebuttals in order by version.

Attachments originally included by the reviewers as part of their assessment can be found at the end of this file.

Version 0:

Reviewer comments:

Reviewer #1

(Remarks to the Author)

The paper presents a new hardware design of real-time QEC decoding that handles leakage errors efficiently. This work achieves a balance between decoding speed and resource usage, and is the first to process leakage errors in real time. The conclusions are well supported by experimental data on FPGAs. The data analysis, interpretation and methodology are sound and meet the expected standards in this field. Although the source code is not provided to immediately reproduce the experimental results, the paper contains sufficient details like architecture design and pseudo code to reimplement the system. However, I do have a couple minor concerns below and suggest a revision of the paper to address these concerns.

1. The authors mentioned in p.3 that "The adjacency graph is then used to autogenerate the SystemVerilog code that defines the neighbours of each PE" which suggests that the SystemVerilog is automatically generated given arbitrary adjacency graph. However, in p.11, the authors mention that "LCD was implemented using the SystemVerilog hardware description language", which suggests a manual implementation. The inconsistency should be fixed and if both are true, e.g., the PE is implemented in SystemVerilog while the connection of PEs is automatically generated, it should be mentioned explicitly.
2. This concern is out of curiosity of the reviewer, and better explained in the paper if there is indeed deliberate design behind it: the PEs in Figure 2 handles multiple stabilizer measurements of a fixed time step; why not let each PE handle a fixed stabilizer measurement for d time steps? The authors mention that they will target streaming decoding on p.6 in future work, which is more natural if each PE handles an indefinite size of time steps.
3. minor typo: on p.12, Figure 5 caption: "textbf(a)"
4. The reviewer appreciates a clear explanation of the difference between leakage error and erasure error. However, it is unclear why p.13 says "we can ignore the new edges because the full depolarization that follows two-qubit gates in the presence of leakage effectively overwrites this modified Pauli error propagation". Is there any reference that supports this argument? From the paper, it seems like the noise model assumes "Two-qubit gates involving leaked qubits introduce additional Pauli errors", in which case it makes sense to only pre-grow edges that are sensitive to a heralded event. However, for other qubit platforms like neutral atoms, a leakage qubit could remove the effect of two-qubit gates, essentially changing the structure of the decoding graph rather than simply updating the error probabilities or edge weights. It would be nice to clarify the conditions behind the approach in this paper.

(Remarks on code availability)

The artifact includes the stim circuit and the processed data in a CSV file, but not source code of the implementation. It is thus NOT reproducible.

Reviewer #2

(Remarks to the Author)

There are two things I need to see included in the manuscript to make an informed decision about publication.

First, make a comparison with an existing decoder using a simple error model to help verify the correctness and performance of the hardware decoder. It would be good to see a plot of logical error rate versus physical error rate for a range of code distances for uniform depolarizing noise and compare this with weighted minimum weight perfect matching using Stim and pymatching.

Second, a discussion of the compatibility of the hardware approach with computation as opposed to just memory. Is this a viable long term approach? When the decoding graph must dynamically change to match the underlying computation, what ideas exist to cope with that?

(Remarks on code availability)

Reviewer #3

(Remarks to the Author)

The article presented for publication elaborates on an implementation of the Union-Find decoding algorithm for surface code under circuit-level noise with (or without) leakage for field programmable gate arrays (FPGAs). While the topic has been researched in the past, the authors introduce a new method which addresses the problems introduced when considering already researched maximum parallelized and serial methods. Their process consists in a coarse-grained parallel scheme in which the decoding graph of the surface code is stored in process elements which themselves are stored in Parts. An intelligent division of the nodes of this process elements allows to be able to decode circuit-level noise below the micro second bound.

Moreover, the authors consider the problem of leakage through an approach named patch wiggling, where the memory of the surface code is preserved while also measuring every qubit in the code every, at maximum, two syndrome extraction rounds. The Results section of the article examines two different scenarios: one with high leakage and one with low leakage, under circuit-level noise. These results show a significant improvement in the performance of LCD when considering leakage adaptivity for the high leakage scenario and more moderate improvement when considering the low leakage one.

The development of decoding algorithms for FPGAs is of capital importance for achieving online decoding with runtimes that avoid any possibility of backlog delays. In this aspect, the LCD method introduces a new approach that provides efficient results at a great decoding time while not being too expensive in terms of FPGA resources. Subsequently, I would like to recommend the publication of the article if the authors answer one small question that the reviewer would like to address.

In Methods A the authors mention that they find the support of the edges in the decoding graph by adding radii. I do not understand how and I do not believe it is easy to understand. Could the authors expand on that? For instance, consider the attached figure, how would the radii be able to find the support of the recovered set of edges? I believe the answer may be given in some cited articles but I also believe it pertinent to be in this article as well.

Plot caption: Four defect clusters merge into one of even parity. The labelled black dots denote defects.

(Remarks on code availability)

The code provided by the authors is useful and provides a tool for any interested researcher in reproducing the claimed results. Nonetheless, I would find it more suitable if it provided a simple script with functions that could generate any stim circuit with the same noise model the authors use in their simulations.

Version 1:

Reviewer comments:

Reviewer #1

(Remarks to the Author)

The authors have addressed all of my previous comments either by answering the questions in the reply PDF or adding clarifications in the paper.

Regarding the reproducibility, I understand that these implementations are proprietary and I agree with the authors that the algorithm itself is clear so if anyone is interested, they can in principle reproduce the results. The DOI link seems to be malfunctioning though. I couldn't find the corresponding resource.

(Remarks on code availability)

Reviewer #2

(Remarks to the Author)

Low decoding latencies are necessary for fast errorcorrected quantum computers needing conditional logic, such as magic-state teleportation. Teleportation requires a measurement across two logical qubits, which in lattice surgery takes $O(d)$ rounds, followed by a reaction conditioned on the outcome of that decoded measurement [39, 40]. Extracting d rounds of syndromes takes around $d \mu s$. This, together with the response time of the QEC system, determines the time that it takes to complete the teleportation of a magic state, setting the logical clock rate.

The above paragraph is inaccurate, the logical clock rate, meaning the minimum time to execute a Clifford circuit plus a layer of non-Clifford gates, is the physical measurement time, plus classical processing time, plus the time to do a feed-forward single qubit rotation (<https://arxiv.org/abs/1210.4626>). The techniques in this paper are QEC independent, and apply to lattice surgery as well. A proposed realization of this paper in lattice surgery can be found in the final slides of (https://docs.google.com/presentation/d/184GHX9jffq9dcwWzbku0K90V9xdA8_25ID8pfeEcRg/edit?usp=sharing).

With this paragraph updated, I recommend publication.

(Remarks on code availability)

Reviewer #3

(Remarks to the Author)

The reviewer finds the revised manuscript satisfactory and is confident in recommending it for publication.

(Remarks on code availability)

Open Access This Peer Review File is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

In cases where reviewers are anonymous, credit should be given to 'Anonymous Referee' and the source.

The images or other third party material in this Peer Review File are included in the article's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

To view a copy of this license, visit <https://creativecommons.org/licenses/by/4.0/>

REVIEWER COMMENTS

Reviewer #1 (Remarks to the Author):

The paper presents a new hardware design of real-time QEC decoding that handles leakage errors efficiently. This work achieves a balance between decoding speed and resource usage, and is the first to process leakage errors in real time. The conclusions are well supported by experimental data on FPGAs. The data analysis, interpretation and methodology are sound and meet the expected standards in this field. Although the source code is not provided to immediately reproduce the experimental results, the paper contains sufficient details like architecture design and pseudo code to reimplement the system. However, I do have a couple minor concerns below and suggest a revision of the paper to address these concerns.

We thank the referee for this summary of our work, and the positive comments on its quality and importance.

1. The authors mentioned in p.3 that "The adjacency graph is then used to autogenerate the SystemVerilog code that defines the neighbours of each PE" which suggests that the SystemVerilog is automatically generated given arbitrary adjacency graph. However, in p.11, the authors mention that "LCD was implemented using the SystemVerilog hardware description language", which suggests a manual implementation. The inconsistency should be fixed and if both are true, e.g., the PE is implemented in SystemVerilog while the connection of PEs is automatically generated, it should be mentioned explicitly.

We thank the referee for pointing out this inconsistency.

The PE, part and controller modules in LCD were manually written, and the inter-PE Network-on-Chip (NoC) module was autogenerated using our in-house design automation tool.

We have rewritten parts of the last paragraph of section III.A, as well as the first paragraph of section Methods C to clarify this.

2. This concern is out of curiosity of the reviewer, and better explained in the paper if there is indeed deliberate design behind it: the PEs in Figure 2 handles multiple stabilizer measurements of a fixed time step; why not let each PE handle a fixed stabilizer measurement for d time steps? The authors mention that they will target streaming decoding on p.6 in future work, which is more natural if each PE handles an indefinite size of time steps.

We thank the referee for highlighting this useful node to PE mapping scheme.

In general, there are three simple rules for assigning nodes to PEs based on their (x, y, t) coordinates in the decoding graph:

1. Assign nodes with the same x and t coordinate to the same PE
2. Assign nodes with the same y and t coordinate to the same PE
3. Assign nodes with the same x and y coordinate to the same PE

Our decoder can in principle support any of these schemes, but schemes (1) and (2) have the feature that no two nodes in the same PE are neighbours, which enables certain hardware-level optimisations, such as hazard-free pipelining.

As mentioned by the referee, it might still be advantageous to use scheme (3), especially in a steaming context, but we would need to assess its effect on hardware-level optimisations, as these directly influence throughput and latency.

3. minor typo: on p.12, Figure 5 caption: " $\text{textbf}(a)$ "

We thank the referee for spotting this, we have fixed it.

4. The reviewer appreciates a clear explanation of the difference between leakage error and erasure error. However, it is unclear why p.13 says "we can ignore the new edges because the full depolarization that follows two-qubit gates in the presence of leakage effectively overwrites this modified Pauli error propagation". Is there any reference that supports this argument? From the paper, it seems like the noise model assumes "Two-qubit gates involving leaked qubits introduce additional Pauli errors", in which case it makes sense to only pre-grow edges that are sensitive to a heralded event. However, for other qubit platforms like neutral atoms, a leakage qubit could remove the effect of two-qubit gates, essentially changing the structure of the decoding graph rather than simply updating the error probabilities or edge weights. It would be nice to clarify the conditions behind the approach in this paper.

We thank the reviewer for their comments about different non-computational noise sources, the terminology in the literature is currently ambiguous and we have attempted to clear some of that within this work.

There are two points to this question which we will address separately: sampling and decoding.

Sampling

In this work we focus on a noise model which is amenable to simulate within a Pauli frame model (which Stim is built on top of). Much of the literature dealing with non-

computational noise currently relies on some level of external Monte-Carlo sampling followed by construction of a stabiliser circuit conditioned upon the sample. This causes prohibitive overheads as one needs to initialise a new stabiliser circuit per shot, unlike being able to sample the same circuit by running Pauli frames in parallel.

In the depolarising model of leakage, if a sealed qubit interacts with a leaked qubit the sealed qubit gets fully depolarised. This is what we mean by *"Two-qubit gates involving leaked qubits introduce additional Pauli errors"*. This full depolarisation turns the sealed qubit into the maximally mixed state. Therefore, any previous Pauli string and the effect of the gate itself get overwritten by this channel. This is why, in this model, we can keep the underlying structure of the noise-free circuit. You effectively erase the gate, but the price you pay is full depolarisation on the partner qubit. This is what we mean by *"the full depolarization that follows two-qubit gates in the presence of leakage effectively overwrites this modified Pauli error propagation"*

Note that this does not work if the partner channel does not effectively erase the gate, which does happen with a maximal depolarisation channel. Pauli strings from the leaked qubit (which is still a qubit from the perspective of the stabiliser simulation, as in our implementation it still carries the register for X and Z errors on top of the leakage register) can propagate through the entangling gates and will remain there. It also does not work if the leakage model just erases the gate with no damage to the partner qubit. As the reviewer points out, this is the case with systems such as ions and neutral atoms. To sample, the literature currently takes one of two approaches:

- Just use the depolarising model, and accept that you are overestimating the damage compared to the actual hardware.
- Construct a new circuit for every shot and accept the overhead.

Both of these are sub-optimal, and it remains an open challenge in the field to simulate more non-computational noise sources in an efficient manner for testing FTQC at scale.

Decoding

When we detect a herald, we know that the qubit must have leaked at some point in its history. Physically, this means two things: (A) errors in the system propagate differently (as 2-qubit gates no longer act between a leaked and sealed qubit), and (B) new noise is introduced in the system.

Re (A) This different propagation of errors would lead to new edges in the decoding graph [14], but as we have argued in the preceding paragraphs, if the new noise is effectively gate erasing, one can effectively ignore these new edges and pretend the two-qubit gates still propagate computational errors as if the qubit were not leaked, at no cost in the accuracy of the approximation. This is a novel observation, and it is hugely beneficial as adding new edges to model changes to error propagation is a complex task in hardware.

Re (B), some subset of the sealed partner qubits will have been fully depolarised, with a probability determined by the leakage event, the structure of the circuit and locations and probabilities of the leakage channels present. We can model this effectively by increasing the probabilities of the errors connected with the leakage herald event. In fact, as we have seen, setting all such probabilities to 0.5 (which is the same as zeroing the weight, or pre-growing the edge) turns out to be an accurate approximation.

The reviewer correctly points out that other noise models, such as the gate erasure in AMO systems, would result in different adaptations. Since the initial publication of this work, there have been some attempts to address this, notably in [60, 61]. However, they both suffer from the sampling constraints detailed above, and it remains an open question as to whether the adaptations they suggest can be implemented in a hardware-efficient way.

We have updated methods and appendices in the manuscript to elaborate on the discussion.

Reviewer #1 (Remarks on code availability):

The artifact includes the stim circuit and the processed data in a CSV file, but not source code of the implementation. It is thus NOT reproducible.

Some of the artifacts used for our simulations are proprietary and we cannot distribute them. However, we believe that the provision of the Stim circuits, together with the detailed explanations of our simulation strategy and the decoding algorithm in the main text, makes our results in principle reproducible.

Reviewer #2 (Remarks to the Author):

There are two things I need to see included in the manuscript to make an informed decision about publication.

First, make a comparison with an existing decoder using a simple error model to help verify the correctness and performance of the hardware decoder. It would be good to see a plot of logical error rate versus physical error rate for a range of code distances for uniform depolarizing noise and compare this with weighted minimum weight perfect matching using Stim and pymatching.

We now include an accuracy comparison between LCD and PyMatching in Fig. 6. The data was generated using X basis quantum memory experiments and a uniform depolarising noise model without leakage (SD6 in [37]), which we simulated using Stim. We found the thresholds for unweighted LCD and PyMatching to be 0.55% and 0.7%, respectively. Adding weights to LCD closes this gap, as we verify in Fig 6 using a software version of the LCD algorithm with weights.

Second, a discussion of the compatibility of the hardware approach with computation as opposed to just memory. Is this a viable long term approach? When the decoding graph must dynamically change to match the underlying computation, what ideas exist to cope with that?

We now include a paragraph in the *Discussions* section on fast reconfiguration by turning vertices ON/OFF between adjacent windows in a streaming logic experiment. In practice, this process takes fewer than 100 FPGA clock cycles (50ns with a clock frequency of 200MHz), making it highly efficient in scenarios requiring high throughput and low latency.

Reviewer #3 (Remarks to the Author):

The article presented for publication elaborates on an implementation of the Union-Find decoding algorithm for surface code under circuit-level noise with (or without) leakage for field programmable gate arrays (FPGAs). While the topic has been researched in the past, the authors introduce a new method which addresses the problems introduced when considering already researched maximum parallelized and serial methods. Their process consists in a coarse-grained parallel scheme in which the decoding graph of the surface code is stored in process elements which themselves are stored in Parts. An intelligent division of the nodes of this process elements allows to be able to decode circuit-level noise below the micro second bound.

Moreover, the authors consider the problem of leakage through an approach named patch wiggling, where the memory of the surface code is preserved while also measuring every qubit in the code every, at maximum, two syndrome extraction rounds. The Results section of the article examines two different scenarios: one with high leakage and one with low leakage, under circuit-level noise. These results show a significant improvement in the performance of LCD when considering leakage adaptivity for the high leakage scenario and more moderate improvement when considering the low leakage one.

The development of decoding algorithms for FPGAs is of capital importance for achieving online decoding with runtimes that avoid any possibility of backlog delays. In this aspect, the LCD method introduces a new approach that provides efficient results at a great decoding time while not being too expensive in terms of FPGA resources. Subsequently, I would like to recommend the publication of the article if the authors answer one small question that the reviewer would like to address.

We confirm that this summary of our work is correct, and we thank the referee for their assessment of its importance and context.

In Methods A the authors mention that they find the support of the edges in the decoding graph by adding radii. I do not understand how and I do not believe it is easy to understand. Could the authors expand on that? For instance, consider the attached figure, how would the radii be able to find the support of the recovered set of edges? I believe the answer may be given in some cited articles but I also believe it pertinent to be in this article as well.

Plot caption: Four defect clusters merge into one of even parity. The labelled black dots denote defects.

The support of an edge, as defined by Delfosse et al in [13], is a variable that determines whether the edge is un-grown (0), half-grown (1) or fully-grown (2). When a cluster grows, the support of each of its boundary edges, i.e., the edges with one endpoint in the cluster and the other endpoint outside the cluster, is incremented by 1. So, un-grown edges become half-grown and half-grown edges become fully-grown. The support of an edge can also transition from un-grown to fully-grown when two active clusters grow from opposite ends before merging into a single cluster.

In our algorithm, we replace the edge-level support variables with vertex-level radius variables. When a cluster grows, the radius of each of its vertices is incremented by 1. Subsequently, the support of an edge can be determined by summing the radii of the vertices on its endpoints:

$$\text{Support}((u, v)) = \min(u.\text{radius} + v.\text{radius}, 2)$$

Reviewer #3 (Remarks on code availability):

The code provided by the authors is useful and provides a tool for any interested researcher in reproducing the claimed results. Nonetheless, I would find it more suitable if it provided a simple script with functions that could generate any stim circuit with the same noise model the authors use in their simulations.

Some of the artifacts used for our simulations are proprietary and we cannot distribute them. However, we believe that the provision of the Stim circuits, together with the detailed explanations of our simulation strategy and the decoding algorithm in the main text, makes our results in principle reproducible.

On top of these suggestions, we have implemented the following improvements to the manuscript:

1. Fixed a typo in Fig 2 (a) where the 16th PE was mislabelled.

2. Added a Data Availability statement at the end of the main text on page 6.
3. Changed the date at the top of the manuscript to August 2025.
4. Updated the email address of the corresponding author in the paper.

REVIEWER COMMENTS

Reviewer #1 (Remarks to the Author):

The authors have addressed all of my previous comments either by answering the questions in the reply PDF or adding clarifications in the paper.

Regarding the reproducibility, I understand that these implementations are proprietary and I agree with the authors that the algorithm itself is clear so if anyone is interested, they can in principle reproduce the results. The DOI link seems to be malfunctioning though. I couldn't find the corresponding resource.

We thank the reviewer for their assessment.

We have checked that the Zenodo link in the Data Availability Statement section is working.

Reviewer #2 (Remarks to the Author):

Low decoding latencies are necessary for fast errorcorrected quantum computers needing conditional logic, such as magic-state teleportation. Teleportation requires a measurement across two logical qubits, which in lattice surgery takes $O(d)$ rounds, followed by a reaction conditioned on the outcome of that decoded measurement [39, 40]. Extracting d rounds of syndromes takes around $d \mu s$. This, together with the response time of the QEC system, determines the time that it takes to complete the teleportation of a magic state, setting the logical clock rate.

The above paragraph is inaccurate, the logical clock rate, meaning the minimum time to execute a Clifford circuit plus a layer of non-Clifford gates, is the physical measurement time, plus classical processing time, plus the time to do a feed-forward single qubit rotation (<https://arxiv.org/abs/1210.4626>). The techniques in this paper are QEC independent, and apply to lattice surgery as well. A proposed realization of this paper in lattice surgery can be found in the final slides of (https://docs.google.com/presentation/d/184GHX9jffq9dcwWzbku0K90V9xdA8_25lD8pfeEcRg/edit?usp=sharing).

We thank the referee for this assessment and this correction. We agree with the remarks regarding the minimum time of circuit execution. We have expanded this paragraph to point to the time-optimal strategies, and to explain that our target of d us for decoder latency is based on a good optimisation in the architecture of [Litinski, Quantum, 2019], and clarify that our usage of “logical clock rate” relates to this architecture, instead of the fastest possible architecture. In the Litinski architecture, our latency target of d us for a window of $O(d)$ rounds is natural, while in other architectures more aggressive latency targets may be more natural.

Reviewer #3 (Remarks to the Author):

The reviewer finds the revised manuscript satisfactory and is confident in recommending it for publication.

We thank the referee for this assessment.

