Supplementary for: A local and scalable lattice renormalization method for ballistic quantum computation

Daniel Herr,[1,2,*] Alexandru Paler,[3] Simon J. Devitt,[4,5] and Franco Nori[1,6]

[1]*Theoretical Quantum Physics Laboratory, RIKEN, Wako-shi, 351–0198, Japan*

[2]*Computational Physics, ETH Zurich, 8093 Zurich, Switzerland*

[3]*Institute for Integrated Circuits, Johannes Kepler University, Linz, 4040, Austria*

[4]*ARC Centre for Engineered Quantum Systems,*
Department of Physics and Astronomy, Macquarie University,
North Ryde, New South Wales 2109, Australia

[5]*Turing Computing, 77 Water St 2201, New York, NY 10005, USA*

[6]*Department of Physics, University of Michigan, Ann Arbor, MI 48109–1040, USA*

[*] daniel.herr@riken.jp

I. SUPPLEMENTARY METHODS: CODE OVERVIEW

The code to this description is open-source and hosted on Github `https://github.com/herr-d/photonic_lattice`. It is written in C++ and was logically divided into three classes of which one implements the lattice of a single box, another class combines all boxes to the larger lattice, and the last class finds paths between different structures.

The lattice implementation for each box is given by the class `Graph`. It implements the graph as a `std::deque`, whose key is a unique identifier for the individual node and the value is a `std::vector` of all neighbors. These neighbours are stored as a `std::pair` where the first value is the id of the box and the second value gives the id of the node inside that box. Further important functions are `Graph::generate_connections` which randomly generates the lattice using the rules for the fusion operations and `Graph::find_structure` which looks for a suitable position for the structure.

The large lattice class, `Parallel`, contains a `std::vector` of the class `Graph`. This vector contains all the information related to the lattice. The class handles all high-level operations, such as output of the purified lattice, and calculations for the statistics. It further determines between which structures a path needs to be found. While our implementation is not yet parallel, the parallelization should be straightforward to implement in this class.

Finally, the class `Astar`, implements the path-finding algorithm A*.