Distributed Quantum Computing with QMPI

Thomas Häner
Microsoft Quantum
Switzerland

Damian S. Steiger
Microsoft Quantum
Switzerland

Torsten Hoefler
ETH Zürich
Switzerland

Matthias Troyer
Microsoft Quantum
USA

ABSTRACT

Practical applications of quantum computers require millions of physical qubits and it will be challenging for individual quantum processors to reach such qubit numbers. It is therefore timely to investigate the resource requirements of quantum algorithms in a distributed setting, where multiple quantum processors are interconnected by a coherent network. We introduce an extension of the Message Passing Interface (MPI) to enable high-performance implementations of distributed quantum algorithms. In turn, these implementations can be used for testing, debugging, and resource estimation. In addition to a prototype implementation of quantum MPI, we present a performance model for distributed quantum computing, SENDQ. The model is inspired by the classical LogP model, making it useful to inform algorithmic decisions when programming distributed quantum computers. Specifically, we consider several optimizations of two quantum algorithms for problems in physics and chemistry, and we detail their effects on performance in the SENDQ model.

CCS CONCEPTS

• **Hardware → Quantum computation**.

KEYWORDS

distributed quantum computing, QMPI

ACM Reference Format:
Thomas Häner, Damian S. Steiger, Torsten Hoefler, and Matthias Troyer. 2021. Distributed Quantum Computing with QMPI. In *The International Conference for High Performance Computing, Networking, Storage and Analysis (SC '21), November 14–19, 2021, St. Louis, MO, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3458817.3476172

1 INTRODUCTION

Quantum computing promises to solve certain computational tasks exponentially faster than classical computers, with application domains ranging from cryptography [49] to chemistry and material science [33]. A host of case studies investigate the minimal requirements for quantum computers to yield a practical advantage [20, 29, 45, 48, 58]. While the resource requirements seem generally feasible, e.g., for applications in computational catalysis [58] and for breaking RSA [20], such applications require millions of physical qubits. Given current projections [19, 39], it will be challenging for individual quantum processors to achieve such qubit numbers. Consequently, these applications may require that computations are distributed across multiple entangled quantum processors.

In a distributed setting, multiple smaller quantum chips are connected coherently, allowing for inter-node communication of quantum information. For example, IBM's roadmap for large scale devices containing more than 1 million qubit is planned as a set of individual systems with quantum interconnects linking many dilution refrigerators [19] and Google Quantum AI have communicated plans to connect 100 tiles of 10,000 physical qubits each to reach a million physical qubits [39]. For an overview of possible paths to distributed quantum computing, we refer the reader to reviews on the topic [2, 56].

Related Work. There exists a host of software frameworks, programming languages, and compilers for quantum computing [1, 6, 21, 26, 51, 53]. However, to the best of our knowledge, no existing framework for quantum computing allows for development of distributed algorithms.

Moreover, progress has been made on simulations and applications of a quantum internet [12, 13, 61]. Yet, as with today's classical internet applications, these works do not aim to provide a framework for high-performance distributed quantum computing. Instead, typical use cases of a quantum internet are secure communication, clock synchronization and leader election [44, 61].

There exists a large body of theoretical work to estimate the resource requirements of non-local operations [10, 17], of distributed quantum algorithm primitives such as distributed arithmetic [35] and the quantum Fourier transform [65], and of entire applications in cryptography [20, 36, 64] and computational chemistry [15].

Finally, there is related work on theoretical models of distributed quantum computing. Beals *et al.* [3] introduce the Q PRAM model, the shared quantum memory equivalent of the PRAM model with global load/store access as a model for distributed quantum computing. They analyze several quantum algorithms in the Q PRAM model. In contrast to our work, however, their focus is on asymptotic runtimes, and not on performance.

While we consider systems where each node has a sufficient number of physical qubits to encode several logical qubits, we note that there are alternative approaches. For example, Nickerson et al. [41] propose a protocol for distributed quantum computing

SC '21, November 14–19, 2021, St. Louis, MO, USA

ACM ISBN 978-1-4503-8442-1/21/11…$15.00
https://doi.org/10.1145/3458817.3476172

in which small cells of only 5 to 50 physical qubits are connected. In this setting, even a single logical qubit is spread over different nodes, and the distribution is a hardware implementation detail not exposed to the user.

Contributions. In order to bridge the gap between theoretical distributed quantum algorithms and software frameworks for quantum computing, we propose Quantum MPI (QMPI), an extension of the classical MPI standard [59] to quantum computing. The focus of QMPI is to provide the primitives that are necessary to implement high-performance distributed quantum algorithms. To reason about the performance of distributed quantum algorithms, we develop the SENDQ model and we present examples that illustrate how this model may be leveraged to inform algorithmic decisions.

Specifically, our contributions are as follows:

- We define Quantum MPI (QMPI) as an extension to classical MPI. QMPI supports all classical MPI functionality on computational basis states (including their inverses to enable reversibility) as well as general-purpose point-to-point and collective functions that *entangle* and *move* qubits between nodes.
- We present a quantum communication model (SENDQ) that is inspired by the classical LogP model [11] to model the performance of distributed quantum algorithms and foster algorithmic optimizations.
- We implement quantum-specific optimizations, such as using asynchronously pre-established entangled quantum states to optimize point-to-point and collective quantum communication with zero quantum communication depth and purely classical communication.
- We discuss potential applications of quantum MPI to problems from physics and chemistry, and we show how such applications can be optimized using the SENDQ model.

QMPI adds support for quantum message passing to existing quantum programming languages, thus enabling programmers to implement distributed quantum algorithms. In combination with SENDQ, the resulting implementations can be used to investigate typical workloads at application scale. The results of such investigations are crucial for making informed architectural decisions along the road to practical distributed quantum computing.

2 QUANTUM COMPUTING

This section serves as a brief introduction to quantum computing and our notation. For a more detailed treatment of this subject, we refer the reader to the textbook by Nielsen and Chuang [42].

Quantum States and Dirac Notation. A quantum computer consists of multiple quantum bits (qubits) whose quantum state may be represented as a complex superposition over all classical bitstrings. Specifically, the state $|\psi\rangle$ ("ket ψ") of an n-qubit quantum computer may be written as

$$|\psi\rangle = \sum_{i=0}^{2^n-1} \alpha_i \, |i_{n-1} \cdots i_0\rangle \,,$$

where i_k denotes the kth bit of the integer i, and $\alpha_i \in \mathbb{C}$ such that $\sum_i |\alpha_i|^2 = 1$. When measuring all qubits at once, the probability of observing the integer i is given by $|\alpha_i|^2$. This also explains the normalization condition, since the probability of observing any integer should be equal to 1. The "ket" notation $|\cdot\rangle$ denotes column vectors, whereas row vectors are denoted by "bra": $\langle\cdot|$. Therefore, the dot-product between two state vectors $|\psi\rangle, |\phi\rangle$ can be written as $\langle\psi|\phi\rangle$ and the projector onto $|\psi\rangle$ is $P_{|\psi\rangle} := |\psi\rangle\langle\psi|$. By identifying each of the computational basis states $|i_{n-1}\cdots i_0\rangle$ with the corresponding standard basis vector $e_i \in \mathbb{C}^{2^n}$, i.e., $(e_i)_j = \delta_{ij}$ where δ_{ij} denotes the Kronecker delta, $|\psi\rangle$ can be written as a column vector with entries $(|\psi\rangle)_i = \alpha_i$.

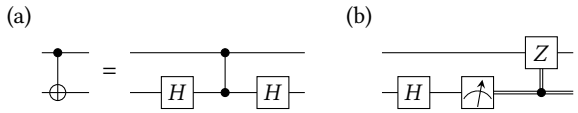


Figure 1: (a) A CNOT may be written in terms of a CZ using Hadamard gates. (b) If the target qubit is known to be reset to $|0\rangle$ by the CNOT, then this reset may be implemented more efficiently using only single-qubit gates and classical control.

Quantum Gates. A computation can be performed by applying a sequence of quantum instructions to the qubits. Such quantum instructions consist of a list of qubit indices that the instruction acts upon, and a so-called quantum gate, akin to classical gates such as AND, XOR, etc.. In the same way that quantum states can be represented as column vectors, quantum gates may be represented as (complex) unitary matrices U of dimension $2^n \times 2^n$. A matrix is said to be unitary iff $U^\dagger U = UU^\dagger = \mathbb{1}$, where $U^\dagger$ denotes the Hermitian conjugate of U. Then, matrix-vector multiplication models the application of a quantum gate.

We will be using the following gates in this paper. The Hadamard gate $H = \frac{1}{\sqrt{2}}\left(\begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix}\right)$, the S gate $S = \left(\begin{smallmatrix} 1 & 0 \\ 0 & i \end{smallmatrix}\right)$, the Pauli gates $X = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$, $Y = \left(\begin{smallmatrix} 0 & -i \\ i & 0 \end{smallmatrix}\right)$, $Z = \left(\begin{smallmatrix} 1 & 0 \\ 0 & -1 \end{smallmatrix}\right)$, the controlled Pauli gates, e.g., controlled X (or controlled NOT, CNOT)

$$CX = |0\rangle\langle 0| \otimes \mathbb{1}_2 + |1\rangle\langle 1| \otimes X = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix},$$

(where $\otimes$ denotes the Kronecker product), and Pauli rotation gates

$$R_P(\theta) = e^{-0.5i\theta P},$$

where P is a single-qubit Pauli gate $P \in \{X, Y, Z\}$.

Quantum Circuits. To illustrate a sequence of quantum gates acting on qubits, we use circuit diagrams such as the one in Fig. 1. Each horizontal line corresponds to a qubit and boxes/symbols on these lines represent gates, with time advancing from left to right. We use double-lines to denote classical information such as measurement outcomes. Controlled gates (such as CNOT) are drawn with as a filled circle • on the control qubit and a line connecting the control qubit to the target qubit gate. Moreover, the Pauli X gate (or NOT gate) is drawn as a $\oplus$-symbol, since it corresponds to addition modulo two, and the controlled Z gate is sometimes drawn as two •-symbols connected by a line (to illustrate its symmetry with respect to control and target qubit).

We will be using a circuit primitive called *fanout*, which can be viewed as copying in a quantum superposition. Let us assume a qubit $q_c = \alpha|0\rangle + \beta|1\rangle$ in a superposition of classical values 0 and 1. Fanout adds auxiliary qubits and transforms the state to

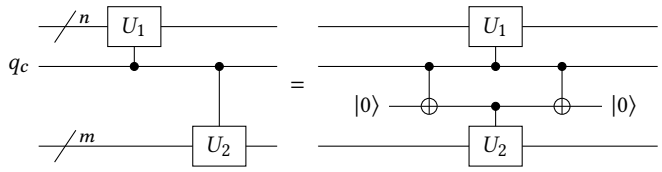


Figure 2: Fanout of control qubit q_c to apply the controlled gates U_1 and U_2 in parallel.

$\alpha|0...0\rangle + \beta|1...1\rangle$., thus it is now in a superposition of multiple copies of the classical values 0 and 1. Note that this is not the same as cloning the qubit. One application of fanout is to to parallelize computations [25] by fanning out control qubits so that gates can be executed in parallel even when they are executed conditionally on the same control qubit(s). Gates that are controlled on q_c and that act on distinct qubits may now be applied in parallel by choosing qubit q_c or any of the auxiliary qubits as a control qubit. After completion of the controlled gates, all auxiliary qubits need to reverted back to 0 by reverse fanout. Locally, this is again done by a simple CNOT gate, see Fig. 2.

Quantum and Reversible Computing. Fanout is just one example of classical computation applied to a superposition of input states. More generally, any classical computation can be applied in superposition after making it *reversible*: Let $f : \{0,1\}^n \to \{0,1\}^m$ denote a classical function. f can be made reversible by replacing all gates in an implementation of f with their reversible counterpart, which may also introduce extra work qubits to store intermediate results that we denote by $g(x)$. For example, AND gates in the implementation of f are mapped to the so-called Toffoli gate, which stores the AND of the two inputs into a fresh output (qu)bit and, crucially, does not discard inputs, making the computation reversible.

The reversible implementation of f acts on three registers as follows

$$|x\rangle|0\rangle|0\rangle \overset{f}{\mapsto} |x\rangle|g(x)\rangle|f(x)\rangle,$$

where the first register $|x\rangle$ is an n-qubit register that represents x, and the last register $|f(x)\rangle$ consists of m qubits that represent $f(x)$. In order to allow for quantum interference (and register re-use), the extra work qubits holding $g(x)$ must be *uncomputed* [4]. This can be achieved by fanning out $f(x)$ to a new register, and then running the reversible implementation of f in reverse, which we denote by $\overline{f}$:

$$|x\rangle|g(x)\rangle|f(x)\rangle|f(x)\rangle \overset{\overline{f}}{\mapsto} |x\rangle|0\rangle|0\rangle|f(x)\rangle.$$

In contrast to classical MPI, QMPI reductions must be reversible and thus such uncomputations of intermediate results are required.

EPR Pairs and Quantum Teleportation. When distributing a quantum algorithm to multiple nodes, some multi-qubit gates act on qubits that are located on different nodes. This situation can be resolved through either gate or qubit teleportation. We will briefly review the latter and we refer the reader to the work by Yimsiriwattana and Lomonaco Jr [65] for a more detailed discussion.

The fundamental resource to enable communication of quantum data are Einstein-Podolsky-Rosen (EPR) pairs [16], which consist

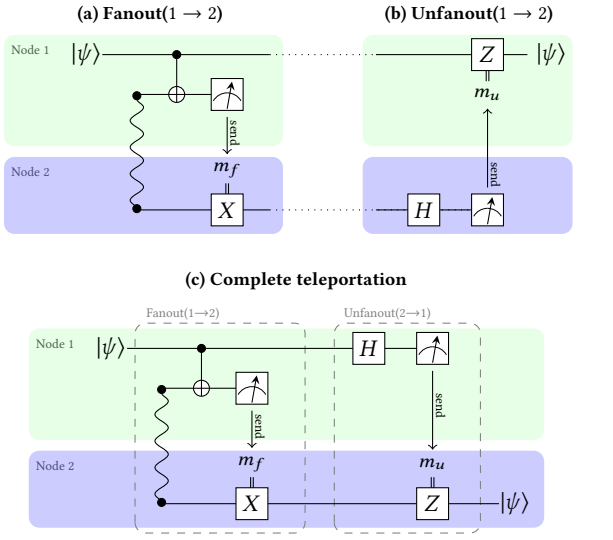


Figure 3: Quantum circuits illustrating fanout/unfanout and teleportation of a qubit in state $|\psi\rangle$ from node 1 to node 2 using an EPR pair. Quantum teleportation can be seen as a fanout to node 2, followed by an unfanout from node 2 to node 1. For teleportation, 1 EPR pair is used and 2 bits of classical information are sent to node 2.

of two qubits in the state

$$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle).$$

In a circuit diagram, we depict EPR pairs as two filled circles that are interconnected with a serpentine line •∿•.

When two nodes share a EPR pair, another qubit may be moved from one node to the other using the EPR pair and classical communication. The basic idea is to first fan out the qubit to the other node and to then remove (using measurement) the qubit from the first node. Fanout may be achieved using a parity measurement between the local EPR pair qubit and the qubit to send, followed by a conditional fix-up operation, as shown in Fig. 3(a). We compute the parity using a CNOT gate between the qubit to send and the local share of the EPR pair, and then we measure the parity. If the outcome is 0, no further action is required. However, if the parity is 1, the other node must fix its "fanout" qubit (its share of the EPR pair) by flipping the qubit using an X gate.

At this point, the qubit has been fanned out successfully to the second node. If both qubits were located on the same node, we could use a CNOT to reset the first qubit to $|0\rangle$, resulting in the qubit having moved from the original position to where the second qubit of the EPR pair was located. It turns out that the same is true in the remote setting: Because the CNOT resets the qubit to $|0\rangle$, we may use the principle of deferred measurement [42] to implement this uncomputing CNOT using just local gates and classical communication, as illustrated in Fig. 1 and Fig. 3(b): All that is needed is a measurement in the X-basis (apply Hadamard, then measure) and, if the outcome is 1, we apply a Z gate to the qubit on the second node. This completes the Unfanout(2⟶1) section of

the quantum circuit for teleportation in Fig. 3(c). Note that only classical communication and no quantum communication is needed for the unfanout, see Fig. 3(b).

3 DISTRIBUTED QUANTUM COMPUTING

In order to run practical quantum applications, a fault tolerant quantum computer is necessary as current error rates for physical two-qubit gates are on the order of $10^{-3} - 10^{-4}$ [30], while practical applications in chemistry and cryptography require around 10^{10} Toffoli (or doubly-controlled NOT) gates [20, 22, 29, 58].

Overhead from Fault Tolerance. In order to store quantum information for the duration of computation without any errors, quantum error correction (QEC) uses many physical qubits to encode each logical qubit such that the error rates are low enough. Additionally, we require to implement quantum gates in a fault tolerant way to execute the quantum program on these logical qubits. Logical qubits and fault tolerant gates require a significant overhead in terms of physical qubits and runtime. A modular quantum computer design might be beneficial to handle the large number of qubits, the cooling requirements for some technologies, and the necessary control electronics and hence we are considering distributed quantum computing in this work. The logical clock cycle is optimistically assumed to be $10\mu s$ for midterm quantum computers in the paper by von Burg et al. [58]. This number heavily depends on the choice of qubit technology and the physical error rates. For example state-of-the-art ion trap physical two-qubit gates take already $1.6\mu s$ [47] so the logical gate times for this technology will be slower than the estimated $10\mu s$. Lekitsch et al. [30] estimate a logical gate time of $235\mu s$ for ion traps which results in a runtime of 110 days to factor a 2048-bit number. Such slow logical gate times at least initially allow us to hide the latency of classical communication in a distributed setting.

A standard set of universal quantum fault-tolerant gates are the single-qubit Pauli, Hadamard, and S gates, the single-qubit $T := \sqrt{S}$ gate, and the two-qubit CNOT gate. Only the CNOT gates will require communication by either teleporting the involved qubits onto the same quantum node or by fanout of the control qubit to the other node. Both of which can be achieved by sharing logical EPR pairs. When using the surface code, which is currently viewed to be the most promising approach to enable fault-tolerant quantum computing, the most costly (local) operation is the T gate. In contrast to other gates, T gates require *distillation*, which is performed in so-called *magic state factories* [8]. The overhead due to these factories limits the parallelism on a fixed-size chip, since such factories are expected to require tens of thousands of physical qubits when assuming physical error rates of 10^{-3} [31].

Quantum-Coherent Interconnects. The physical implementation of connecting different quantum nodes by, e.g., creating a distributed EPR pair, depends on the underlying technology. Using optical photons is a natural choice given their property to travel long distance with little perturbation. There is a large number of theoretical proposals [38, 54, 63] and also first experimental prototypes: optical photons have been used to demonstrate entanglement sharing between ion traps 20m apart from each other [24] or between atomic qubits [37]. In superconducting transmon qubit architectures, it is necessary to convert microwave photons, which are are used to perform two-qubit operations locally, to optical photons [18]. However, such transducers are still challenging to build. Therefore, alternative approaches are also being pursued, e.g., directly coupling two quantum nodes with microwave photons in a cryogenic waveguide [32, 66].

In addition to the physical implementation for entanglement sharing between nodes, a protocol for fault tolerance is required [5, 14, 55, 57].

Inter-Node Communication. With entanglement sharing in place, it is possible to establish EPR pairs between nodes through the quantum-coherent interconnect. In turn, this enables quantum teleportation between nodes, thus allowing for sending and receiving quantum information with *move* semantics.

However, we note that an additional mode of operation is possible: Instead of fully moving a qubit from one node to the other, the qubit may also be fanned out to the other node, thus exposing its value on multiple nodes at once. This is also referred to as an *entangled copy*, which may be used, e.g., to reduce the delay of certain quantum circuits, see Fig. 2 for an example.

In this second mode of operation, one may support all functionality of classical MPI. However, due to reversibility constraints, the inverse of each function must be available as well [4]. For example, reductions must be performed in a reversible manner. To this end, depending on the reduction operation, additional work qubits may be required. These must be stored and managed by the implementation until the inverse of the reduction is applied, allowing to uncompute these work qubits.

4 QUANTUM MPI

To allow programmers to express distributed algorithms in their quantum programming language of choice, we propose Quantum MPI (QMPI) – a quantum extension of the classical message-passing interface (MPI) standard.

4.1 Communicators and Interaction with MPI

QMPI leverages MPI for classical communication. As such, the communication of classical and quantum data is completely separated: The first is handled by MPI, whereas QMPI handles the latter.

While some nodes in `MPI_COMM_WORLD` may be purely classical, such nodes must not be part of any communicator that is passed to a QMPI function. `QMPI_COMM_WORLD`, which is of type `MPI_Comm`, contains all quantum (i.e., not purely classical) nodes. All quantum nodes must support classical MPI since otherwise, teleportation would not be possible, as it requires communicating classical bits.

4.2 Datatypes

Qubits may be allocated using `QMPI_Alloc_qmem(n)`, where n denotes the number of qubits to allocate. `QMPI_Alloc_qmem` returns a `QMPI_QUBIT_PTR qptr`, which points to the first qubit. Qubits may be deallocated using `QMPI_Free_qmem`.

QMPI defines one basic quantum-specific datatype, `QMPI_QUBIT`, which represents a single quantum bit. Given that qubits will be

a scarce resource initially, we leave the construction of more complex data types such as quantum integers and quantum floating-point numbers to the programmer: Such data types may be constructed from `QMPI_QUBIT` using `QMPI_Type_*` functions such as `QMPI_Type_contiguous`, as in classical MPI.

As we do not expect classical communication to be a bottleneck in the near term and in order to keep classical communication separate from quantum communication (the first using MPI, the second using QMPI), we do not allow for mixing of quantum and classical datatypes in the first version of QMPI. However, as protocols for quantum error correction and entanglement sharing are optimized, a tighter integration of QMPI with MPI may become critical to performance, and this restriction could thus be dropped if needed in a future version.

4.3 EPR pairs

The basic building block and most time consuming part for all quantum communication is the creation of EPR pairs between qubits on the sending and receiving nodes. Established EPR pairs allow for higher-level communication primitives such as entangled copying (fanout) or moving (teleportation) of qubits between two nodes that share an EPR pair.

In order to request that an EPR pair be created between two nodes, each node invokes

```
QMPI_Prepare_EPR(qubit, dest, tag, comm),
```

where `qubit` is a fresh qubit in $|0\rangle$, `dest` refers to the rank of a QMPI process running on the other node, `tag` is the message tag, and `comm` is the communicator (e.g., `QMPI_COMM_WORLD`). Upon completion, the quantum state of the two qubits (located on different nodes) is $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$.

As is the case for other communication primitives, asynchronous versions (e.g., `QMPI_Iprepare_EPR`) are available to allow for requesting EPR pairs ahead of time. Asynchronous QMPI functions return regular MPI requests and message progression semantics are similar to MPI's original semantics, i.e., the global progress rule applies.

4.4 General Point to Point Communication

As discussed in Section 3, QMPI supports communication in two modes, one with copy semantics, the other with move semantics. Both modes rely on EPR pairs to move and to fan out qubits to other nodes. Qubits are moved from one node to another using quantum teleportation, whereas fanout exposes their values on multiple nodes simultaneously. QMPI provides functionality for fanning out and sending/receiving qubit (the latter with move semantics) via the two pairs of functions `QMPI_Send / Recv` and `QMPI_Send_move / Recv_move`. In addition, there are the inverses of `QMPI_Send / Recv`, denoted by `QMPI_Unsend / Unrecv`, respectively. The reason for this addition is that the uncomputation can be performed more efficiently: The qubit can simply be measured after applying a Hadamard gate and, if the outcome is nonzero, the other node must apply a Pauli Z gate to its qubit, as shown in Fig. 1(b). Therefore, uncomputing a fanned-out qubit can be achieved by communicating only a single bit of classical information without needing an EPR pair. The resource requirements for entangled copy/fanout, move, and their respective inverses can also be found in Table 1. Table 2 lists all point-to-point primitives and the required resources in terms of the costs for entangled copy and move from Table 1.

4.5 Collective Operations

In addition to general point-to-point communication, QMPI provides collective operations. `QMPI_Bcast` is an example of a simple QMPI collective implementing fanout. Its main purpose is to expose the value of a qubit on multiple nodes (and then uncomputing that value again with its inverse, `QMPI_Unbcast`), similar to copying a classical value with the corresponding MPI routine. In the quantum case, collective communications allow even more optimizations beyond what is possible classically. In particular `QMPI_Bcast` can be implemented with *constant* quantum time. As discussed by Watts et al. [60, Theorem 17], this can be done by first creating EPR pairs on all edges of a spanning tree of the nodes in the communicator as the only quantum communication step, which can be done in parallel in constant time. This is followed by local parity measurements among the entangled qubits at each node, the time for which is logarithmic in the maximum degree of a node in the spanning tree, which is typically a small constant. The last step consists of collective classical communication and computation to identify which qubits need to be changed by a Pauli X gate. The logarithmic complexity of `QMPI_Bcast` is thus due to (fast) classical communication, while the (slow) quantum communication is of constant time.

Another collective operation, `QMPI_Scatter_move` / `QMPI_Gather_move` is an example of a QMPI collective with move semantics. A typical use case for this function is a section in the quantum algorithm where multiple rotation gates are applied to distinct qubits, all of which are located on the same node. In order to increase the number of local rotation factories per rotation qubit, the rotation qubits may be scatter-moved to separate nodes. After all rotations have been applied in parallel, the qubits may be gathered on the original node, allowing the computation to advance.

A QMPI collective with entangled copy semantics that is also worth a quick discussion is `QMPI_Reduce` (and its inverse `QMPI_Unreduce`). It differs from a classical MPI reduction only in that the reduction operation is reversible and that it must be uncomputed eventually (to free scratch space and to allow for interference in the quantum algorithm). In this first version, the QMPI implementation leaves all memory management to the user and `QMPI_Reduce` only accepts reversible operations[1].

An example operation is `QMPI_PARITY`, which computes the parity of all qubits in the reduction. We note that there is a host of different methods that the QMPI implementation may choose from, depending on the situation (available scratch space, size of reduction, etc.). We refer to Section 7 for a selection of different algorithms for computing the parity.

Table 3 shows a complete list of all QMPI collectives and the required resources in terms of entangled copy, move, reduce, and scan from Table 1.

[1]Future versions may support automatic compilation from a non-reversible implementation.

Table 1: Classical and quantum resources required for entangled copy, move, reduce, scan, and their respective inverse operations (or uncomputation) in brackets. Stated are the resources required per qubit in the message and for N nodes (reduce/scan).

	copy [uncopy]	move [unmove]	reduce [unreduce]	scan [unscan]
Quantum comm. (EPR pairs)	1 [0]	1 [1]	$N-1$ [0]	$N-1$ [0]
Classical comm. (bits)	1 [1]	2 [2]	$N-1$ [$N-1$]	$N-1$ [$N-1$]

Table 2: Point to point communication primitives in QMPI. In addition to blocking, also non-blocking variants are available, as in classical MPI. Resource requirements are given in terms of entangled copy and move from Table 1. (a): Same as Sendrecv with move semantics, (b): Resources may already have been used.

Operation	Reverse operation	Resources
QMPI_Send, QMPI_Bsend, QMPI_Ssend, QMPI_Rsend	QMPI_Unsend, QMPI_Bunsend QMPI_Sunsend, QMPI_Runsend	copy
QMPI_Recv, QMPI_Mrecv	QMPI_Unrecv, QMPI_Munrecv	copy
QMPI_Sendrecv	QMPI_Unsendrecv	copy
QMPI_Sendrecv_replace$^{(a)}$	QMPI_Unsendrecv_replace	move
QMPI_Cancel$^{(b)}$	—	
QMPI_Send_move, QMPI_Bsend_move, QMPI_Ssend_move, QMPI_Rsend_move	QMPI_Unsend_move, QMPI_Bunsend_move, QMPI_Sunsend_move, QMPI_Runsend_move	move
QMPI_Recv_move, QMPI_Mrecv_move	QMPI_Unrecv_move, QMPI_Munrecv_move	move

Table 3: Collective communication in QMPI. In addition to the blocking calls, also non-blocking variants [23] are available, as in classical MPI. Resource requirements are given in terms of entangled copy, move, reduce, and scan from Table 1. (a): For in-place: Move resources, (b): Operation must be reversible.

Operation	Reverse operation	Resources
QMPI_Bcast	QMPI_Unbcast	copy
QMPI_Gather, QMPI_Gatherv	QMPI_Ungather, QMPI_Ungatherv	copy
QMPI_Scatter, QMPI_Scatterv	QMPI_Unscatter, QMPI_Unscatterv	copy
QMPI_Allgather, QMPI_Allgatherv	QMPI_Unallgather, QMPI_Unallgatherv	copy
QMPI_Alltoall, QMPI_Alltoallv, QMPI_Alltoallw	QMPI_Unalltoall, QMPI_Unalltoallv, QMPI_Unalltoallw	copy/move$^{(a)}$
QMPI_Reduce	QMPI_Unreduce	reduce$^{(b)}$
QMPI_Allreduce	QMPI_Unallreduce	reduce$^{(b)}$ + copy
QMPI_Reduce_scatter, QMPI_Reduce_scatter_block	QMPI_Unreduce_scatter, QMPI_Unreduce_scatter_block	reduce$^{(b)}$
QMPI_Scan, QMPI_Exscan	QMPI_Unscan, QMPI_Unexscan	scan$^{(b)}$
QMPI_Gather_move, QMPI_Gatherv_move	QMPI_Ungather_move, QMPI_Ungatherv_move	move
QMPI_Scatter_move, QMPI_Scatterv_move	QMPI_Unscatter_move, QMPI_Unscatterv_move	move
QMPI_Alltoall_move, QMPI_Alltoallv_move, QMPI_Alltoallw_move	QMPI_Unalltoall_move, QMPI_Unalltoallv_move, QMPI_Unalltoallw_move	move

4.6 Communication Resources

The tables with all point-to-point and collective operations give the resource requirements in terms of four basic primitives (and their inverses for uncomputing communicated data): entangled copy, move, reduce, and scan. Table 1 can be used to translate from these basic primitives to the number of EPR pairs to be established, and the number of classical bits to be communicated. We note that the stated numbers for reduce and scan are representative of one particular implementation, and there are a host of different tradeoffs to consider in practice.

In particular, the stated numbers for reduce and scan are valid if sufficient logical qubits are available to store intermediate results. Using a linear communication schedule, both reduce and scan can be performed using a single output register per node and a total of $N-1$ EPR pairs per qubit to send, and uncomputation only requires classical communication. In contrast, a binary-tree reduction either requires more local storage, or intermediate results must be uncomputed, and later recomputed during `QMPI_Unreduce`, which also increases EPR pair usage. Similar considerations apply for the scan primitive.

For a more detailed discussion of the tradeoffs involved in optimizing collectives such as `QMPI_Bcast` and `QMPI_Reduce`, we refer the reader to Section 7.1.

4.7 Future Extension: Persistent Requests

Persistent communication requests allow further optimization beyond what is possible classically. All required EPR pairs can be

prepared before starting communication and, in particular, before the data to be sent is available. Point-to-point or collective quantum communication can then be performed with purely classical communication. This allows for overlaying quantum communication with computation performed *prior* to the communication start, which once more is impossible classically. Of course, this optimization is possible only if sufficient qubits are available to store the established EPR pairs and if there is sufficient time to establish all EPR pairs before the communication is started.

5 THE SENDQ MODEL

Analogously to classical performance models such as the LogP model [11], whose parameters characterize the performance of the network interconnecting classical nodes, our SENDQ model captures the features of a distributed quantum computer that are most essential to performance. We envision an architecture where multiple nodes are interconnected with both a classical and a quantum-coherent network, the latter of which may be used to send and receive quantum information. In particular, the quantum-coherent network is used to establish EPR pairs between two nodes.

We anticipate a relatively low logical clock speed for quantum computers due to the overhead introduced by the quantum error correction (QEC) protocol (cf. Section 3). As a consequence, we do not expect that classical communication will have a significant effect on performance and we thus choose to ignore classical communication in our model.

To account for optimizations that overlay communication with local computation, it is crucial to model the performance of both local and nonlocal operations. Our proposed model thus consists of two sets of parameters – one to model (coherent) communication, and the other to model local computation.

In order to model the communication performance, we choose the following parameters.

- S: The number of qubits used to store EPR pairs (per node).
- E: (Upper bound on) the time it takes for a node to establish an EPR pair with any other node. Any node can be involved in at most one remote EPR pair creation at any point. We assume latencies are negligible.
- N: The number of nodes.

The local computation can be modeled using an abstract quantum circuit model that only considers width and depth of the circuit. The parameters are thus

- D: The delay incurred due to local computation
- Q: The number of logical qubits available for computation (per node)

5.1 Discussion of parameters

We now briefly discuss the parameters that make up our performance model for distributed quantum computing.

Parameter *S*. Our model of quantum communication includes a parameter for local storage, namely the number of logical qubits S dedicated to buffering of EPR pairs. This is different from classical performance models such as the LogP model [11], which does not contain such parameters. This parameter is important because performance models with unlimited local storage allow for a simple and unrealistic exploit. Namely, all required EPR pairs may be shared and stored locally ahead of time. As a consequence, all quantum communication could then be implemented in constant time (ignoring the delay of classical communication), see Section 7.1 for an in-depth explanation for the example of `QMPI_Bcast`.

Parameter *E*. E specifies the upper bound on the time it takes to establish a logical EPR pair with any other node, assuming exclusive communication. A logical EPR pair may be established by sharing many physical EPR pairs, followed by a distillation protocol. As we ignore latency, E^{-1} can be seen as the EPR pair injection bandwidth per node into the quantum network.

Parameter *N*. The number of quantum nodes in the distributed quantum computer is denoted by N.

Parameters *D* and *Q*. Our model also includes parameters to model local compute as an integral part because logical qubits for computation can also be used for storing EPR pairs when unused. In general, only the total number of qubits $Q + S$ is constant on each node. Depending on the algorithm, one may choose Q and S to be fixed to a constant value in order to simplify the model even further. The delay D can be specified in more detail if desired. For example, a common choice for a fault tolerant quantum computer is to ignore the delays of all gates and measurements except for the most costly rotation gates (arbitrary rotations and T gates), as discussed in Section 3. Note that we consider the number of logical qubits per node Q equivalent to the number of compute elements, i.e., the number of qubits onto which operations can be applied in parallel. This is due to the fact that current schemes for fault tolerance require full parallelism on all qubits in order to just store information and this parallelism can be used to apply gates.

For the applications presented in this paper, we assume that all parameters are constant throughout the execution of a given quantum algorithm.

5.2 Possible Extensions

The objective of SENDQ is to be simple enough to allow reasoning about a distributed quantum algorithm, while still capturing the most relevant performance characteristics. This tradeoff naturally introduces some limitations that may be addressed with extensions to SENDQ.

First, making the parameters constant throughout the entire execution of an algorithm disallows subroutine-specific tradeoffs such as using some of the data qubits from the previous subroutine as EPR pair storage. While this makes reasoning about the performance of an algorithm more involved, the SENDQ parameters may be assumed to be constant only while executing a given subroutine, or they may be assumed to be completely variable, in order to account for such optimizations.

Second, while the quantum communication parameter E takes into account the limited injection bandwidth per node, it is a single parameter that aims to capture the performance characteristics of the quantum-coherent network. This may be insufficient in certain cases, especially when connectivity among nodes is limited. For example, in the extreme case of a linear chain of nodes with nearest-neighbor connectivity, all nodes are involved in communication between the nodes at the two ends of the chain. To address such scenarios, E_i may be defined for each node i or E_{ij} may be introduced to represent EPR pair generation time between two

nodes i and j. While such an extension makes reasoning about the performance more difficult, this is a viable option for use in a performance simulator.

6 PROTOTYPE IMPLEMENTATION OF QMPI

We have implemented a QMPI prototype in C++ using MPI and multi-threading leveraging the C++ standard library. While some quantum programming frameworks such as qcor [40] are written in C++, other quantum programming frameworks are embedded into different programming languages such as Python [1, 51], or they feature a stand-alone programing language [53], and hence the QMPI interface will need to be ported to these languages.

Our prototype supports a variety of standard quantum gates and the point-to-point as well as collective functions described in the previous section. The current implementation only supports qubit types, and no higher-level datatypes that may be constructed from qubits.

At the core of the library is a full state simulator that allows users to test and debug their distributed quantum algorithms. To ensure that the state vector faithfully represents the quantum state of the distributed quantum computer at any point throughout the computation, all ranks forward quantum operations to rank 0, which then applies the operation to the state vector. Qubit allocations, deallocations, and measurements are handled similarly. Rank 0 runs a separate thread that waits to receive gate operations to execute. Consequently, all ranks (including rank 0) may be used in a quantum computation.

The following example shows how to establish an EPR pair between two QMPI ranks. The simulation output is as expected: Both ranks observe the same value when measuring their share of the EPR pair.

```
#include "qmpi.hpp"
#include <iostream>

using namespace QMPI;

int main() {
  QMPI_Init(0, 0);
  auto qubit = QMPI_Alloc_qmem(1); // allocate 1 qubit
  int rank;
  QMPI_Comm_rank(QMPI_COMM_WORLD, &rank);
  int dest = rank == 0 ? 1 : 0;
  // prepare EPR pair between rank and dest
  int ret = QMPI_Prepare_EPR(qubit, dest, 0,
     QMPI_COMM_WORLD);
  if (ret != MPI_SUCCESS)
    MPI_Abort(QMPI_COMM_WORLD, 0);
  // measure the local qubit
  bool res = Measure(qubit);
  std::cout << rank << ": " << res << std::endl;
  QMPI_Free_qmem(qubit, 1); // free 1 qubit
  QMPI_Finalize();
  return 0;
}
```

In the next section, we describe more examples and we present the corresponding implementation based on our QMPI prototype.

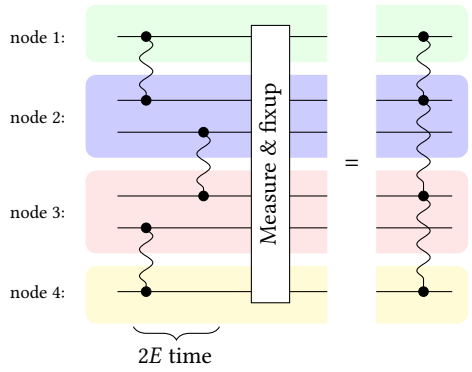


Figure 4: Quantum circuit for establishing a cat state on $n = 4$ nodes in constant quantum depth and classical $O(\log n)$ depth. A reduction of measurement outcomes is required for computing the fixup operation for each node (see text).

7 APPLICATIONS

In this section, we show how quantum algorithms for applications from physics and chemistry may be implemented in QMPI and how the SENDQ model can be used to inform algorithmic decisions.

7.1 Optimizing Collectives

Collective operations allow hardware vendors to optimize these operations by taking into account their hardware parameters. In this section, we describe how to optimize `QMPI_Bcast`. In Section 7.3, we show an example of how to optimize `QMPI_Reduce` for systems with either one qubit per node dedicated to storing EPR paris ($S = 1$) or systems with $S \geq 2$.

Optimizing QMPI_Bcast. This first example presents a simple implementation of `QMPI_Bcast` in terms of `QMPI_Send / Recv` and shows how it can be analyzed and optimized using SENDQ. For simplicity, we assume that only one qubit is sent.

A log-depth implementation of broadcast can be achieved by constructing a binary tree of calls to `QMPI_Send / Recv`: In the k-th step (starting with $k = 0$), 2^k nodes send the broadcast message to 1 other node, thus doubling the number of nodes that have received the message at every step. Since each node communicates with (at most) one node at every step, only one EPR pair must be established between each pair of nodes that communicates. As a result, $S = 1$ is sufficient and the runtime of a broadcast is $E\lceil \log_2 N \rceil$.

This implementation may be optimized by realizing that a cat state, which is an n-qubit generalization of an EPR pair, that is,

$$|\mathrm{cat}(n)\rangle := \frac{1}{\sqrt{2}}(|\underbrace{0\cdots0}_{n}\rangle + |\underbrace{1\cdots1}_{n}\rangle),$$

can be prepared in constant depth [25]. In QMPI, $|\mathrm{cat}(n)\rangle$ can be prepared by first connecting all n nodes with EPR pairs along the edges of a spanning tree, and then combining the individual EPR pairs using a parity-measurement of the different EPR pair qubits on each node (no parity-measurement is performed on leaf nodes), see Fig. 4 for a simplified diagram of this process. The measurement outcomes are used to compute whether or not a given node must apply a Pauli X correction. Specifically, each node k applies

$X^{r_1 \oplus \cdots \oplus r_{k-1}}$ to the qubit that will be part of the cat state, where r_i denotes the outcome of the (in-place) parity measurement on node i, and $\bigoplus_{i=1}^{k-1} r_i$ can be computed with a classical `MPI_Exscan`.

This procedure can be extended to an implementation of `QMPI_Bcast` by also performing a parity measurement between the qubit to broadcast and the one EPR pair qubit on the root node. This implementation runs in quantum time

$$2E + D_M + D_F,$$

where D_M and D_F denotes the time it takes to perform a local two-qubit parity measurement and to apply an X gate (the fixup operation), respectively. The classical `QMPI_Exscan`, which is used to determine whether or not a local X gate correction is needed, can be performed in $O(\log N)$ classical communication steps [46].

7.2 Transverse-field Ising model

In this second example, we show how to simulate the time evolution of a transverse-field Ising model (TFIM) with n spins, whose Hamiltonian is given by

$$H_{\text{TFIM}} = \sum_{\langle i,j \rangle} J_{ij} \sigma_z^{(i)} \sigma_z^{(j)} - \sum_{i=0}^{n-1} \Gamma_i \sigma_x^{(i)},$$

where $\sigma_x^{(i)}, \sigma_z^{(i)}$ denotes a Pauli X and Z, respectively, acting on spin index $i < n$, J_{ij} denotes the coupling constant, and Γ_i is the (local) strength of the transverse field. The first sum runs over all connected spins i, j, which we denote by $\langle i, j \rangle$.

Time evolution under this Hamiltonian can be used as a building block to solve optimization problems leveraging the adiabatic theorem [7]: One first maps the optimization problem to a classical Ising model (thus defining the connectivity and the parameters J_{ij}). Then, starting with $J_{ij} = 0, \Gamma_i = 1$ and in the ground state of the corresponding Hamiltonian (which is $|+\rangle^{\otimes n}$), one slowly changes the parameters to $\Gamma_i = 0$ and J_{ij} equal to the computed values, aiming to remain in the groundstate of all intermediate Hamiltonians. Upon success, a final measurement of all qubits yields the solution of the optimization problem.

In the following, we assume linear nearest-neighbor connectivity for the spins and $J_{ij} = J, \Gamma_i = \Gamma$ for simplicity. The time evolution operator for the Hamiltonian H is given by $U(t) = e^{-itH}$, where t is the time to evolve and $i^2 = -1$. One possible approach to implement a TFIM simulation on a quantum computer is to first map each spin to a qubit. $U(t)$ may then be implemented by first decomposing it using a Trotter-Suzuki expansion. For example, a first-order approximation is

$$U(t) \approx \left(e^{i\delta t H_1} e^{i\delta t H_2} \right)^{\frac{t}{\delta t}},$$

for small δt and $H_1 := -J \sum_{\langle i,j \rangle} \sigma_z^{(i)} \sigma_z^{(j)}$, $H_2 := \Gamma \sum_i \sigma_x^{(i)}$. The individual terms in H_1 commute, and so do the terms within H_2. Therefore,

$$e^{-i\delta t H_1} = \prod_{\langle i,j \rangle} e^{-i\delta t J \sigma_z^{(i)} \sigma_z^{(j)}}, \text{ and } e^{-i\delta t H_2} = \prod_i e^{i\delta t \Gamma \sigma_x^{(i)}}.$$

Each term in the first product can be implemented by computing the parity between spin i and j using a CNOT gate, followed by a rotation gate $R_z(\theta) = e^{-0.5i\theta\sigma_z}$ and another CNOT gate to uncompute the parity. The terms in the second product are just rotation gates $R_x(\theta) = e^{-0.5i\theta\sigma_x}$ acting on qubit i.

The complete code for the simulation can be found in the appendix, see Listing 1. While the prototype implementation uses blocking send/receive calls, we note that one would use an asynchronous version in practice: The EPR pairs could be established while applying the local operations.

Analysis with SENDQ. Each Trotter step requires N EPR pairs, where N denotes the number of nodes, and each node prepares an EPR pair with the two nodes that contain adjacent spins. We assume that rotation gates cannot be executed in parallel due to the cost (in space) of T-state factories. Since the delay of each rotation gate D_R is much larger than the logical gate time, we ignore the cost of CNOTs. As a result, the delay of one Trotter step is approximately

$$D_{\text{Trotter}} = 2\frac{n}{N} D_R = 2QD_R,$$

assuming that n is divisible by N.

To ensure that communication is not a bottleneck (assuming asynchronous send/receive implementations), the time spent on local gates should be at least as large as the time it takes to establish two EPR pairs. For $S \geq 2$, this means that

$$D_{\text{Trotter}} \geq 2E.$$

In turn, this allows us to inform our choice of the number of nodes N if sufficient space is available per node to temporarily store the two EPR pairs: N should be chosen such that

$$E^{-1} n D_R \geq N.$$

If, on the other hand, space per node is a limiting factor and $S = 1$ while $Q \geq 2$, then one may return to the $S \geq 2$ case by increasing the number of nodes to $N \geq \lceil \frac{n}{Q-1} \rceil$.

We now address the case where increasing the number of nodes is not an option. Specifically, we show that our model correctly predicts an overhead for $S = 1$ compared to $S \geq 2$ even with an optimized communication schedule that allows for halting local computations at any point, e.g., during execution of a local rotation gate. With $S = 1$, a request for EPR pair creation can only be initiated once the buffer qubit has been cleared. As a result, there is an additional delay D_R between EPR pair creation requests because the rotation must be applied before the remote qubit can be unreceived. The delay per Trotter step with an optimized schedule for initiating EPR pair creation requests is thus

$$\max\left(D_{\text{Trotter}}, 2E + 2D_R\right),$$

in contrast to the $S \geq 2$ case, where the delay per Trotter step is $\max\left(D_{\text{Trotter}}, 2E\right)$.

This TFIM example shows that SENDQ can be used to model various tradeoffs in the implementation of a distributed quantum algorithm. Crucially, smaller S results in longer runtimes, even if the communication schedule is optimized.

7.3 Chemistry

Simulation of molecules is currently one of the most promising applications to first show a quantum advantage for a practical problem compared to classical supercomputers. The goal is to determine the energy eigenstates of molecules described by a Hamiltonian H.

This then allows, for example, to investigate and optimize chemical catalysis [58].

For large molecules the best quantum algorithms to find ground state energies are based on phase estimation of a unitary operator which depends only on the Hamiltonian of the molecule H. For a given molecule, the full quantum circuit is known at circuit compilation time, i.e., there are no branches in the program depending to measurements during runtime which influence performance. Hence, one can use expensive quantum circuit optimization techniques to reduce the quantum resources and increase performance ahead of time. We will highlight a few optimization possibilities for a distributed quantum computer.

We consider the algorithm of expressing the Hamiltonian H of a molecule of interest in second quantization, expressed in a basis of n spin-orbitals. This algorithm requires at least n data qubits which might be distributed onto different nodes. We perform phase estimation on the time evolution operator of the system, e^{-itH}, which we implement using a Trotter-Suzuki expansion from Section 7.2. The majority of the algorithm is only one primitive operation, namely, a time evolution operator of the form:

$$e^{-itZ_{i_1}Z_{i_2}\cdots Z_{i_k}}, \qquad \{i_1,\ldots,i_k\} \subseteq \{0,\ldots,n-1\}, t \in \mathbb{R}. \tag{1}$$

The qubit indices and parameter t involved in each of these operators depend on the molecule and on the choices of how to represent its Hamiltonian.

Analysis with SENDQ. For a given molecule to be simulated, there are several choices to be made when mapping the problem to a quantum computer. In particular, different choices lead to different Hamiltonians, even if they all describe the same molecule. For example, the fermionic Hamiltonian needs to be transformed into a Hamiltonian that acts on qubits. This can be achieved using the Jordan-Wigner transformation [27, 43, 50], the Bravyi-Kitaev encoding [9], or by using more than n data qubits [62]. For example, a Jordan-Wigner transformation will result in operations as in Eq. (1), which may act on all data qubits. In contrast, the operators resulting from a Bravyi-Kitaev encoding only act on at most $O(\log n)$ qubits, which may lead to savings in a distributed setting, at least without considering further optimizations to the Jordan-Wigner approach. The mapping differences are illustrated in Fig. 5 for the example of a hydrogen ring (data was generated using Refs. [34, 51, 52]).

Once the encoding has been fixed, the individual operators must be implemented in terms of quantum gates. Here, we discuss the tradeoffs of three different approaches to implementing operators of the form given by Eq. (1). For simplicity, we assume that each of the qubits involved is on a different node and that rotation gates take much longer to execute than measurements and other (local) quantum gates, allowing us to ignore the latter. Fig. 6 illustrates the three approaches for an operator acting on $k = 4$ qubits. Each of these circuits consists of the same three subroutines: a parity computation of all involved qubits into a target qubit, a single qubit rotation $R_z(2t)$ on that target qubit, and a final uncomputation of the parity.

The circuit in Fig. 6(a) computes the parity in place using a binary tree of distributed CNOT gates. Consequently, the full circuit requires $2(k-1)$ EPR pairs and has a runtime of

$$2E\lceil \log_2 k \rceil + D_R,$$

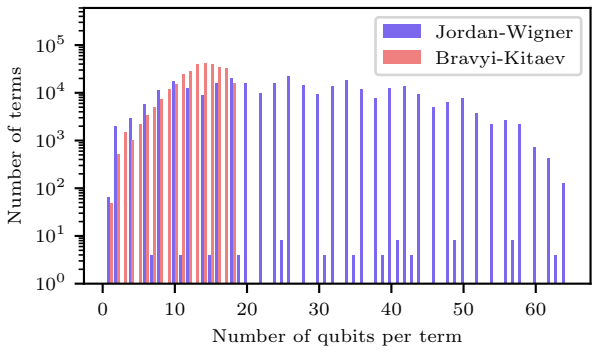
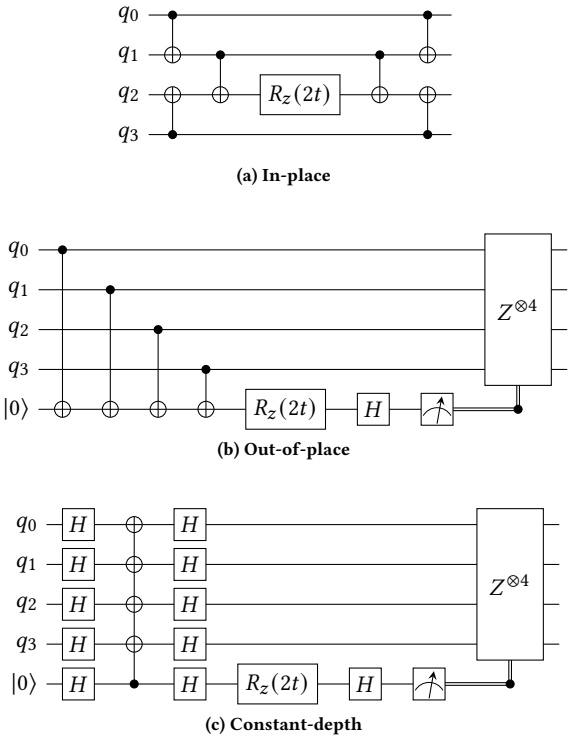


Figure 5: Mapping of the Hamiltonian representing a hydrogen ring with 32 atoms in the STO-3G basis set to a Hamiltonian acting on 64 qubits. The number of qubits involved in each term of the form defined by Eq. (1) is plotted as a histrogram for two different encoding methods.



(a) In-place

(b) Out-of-place

(c) Constant-depth

Figure 6: Three different methods to implement $e^{-itZ_0Z_1\cdots Z_{k-1}}$ for $k = 4$.

where D_R is the delay to execute one rotation gate.

The circuit in Fig. 6(b) computes the parity into an auxiliary qubit. The downside is that the distributed CNOT gates now must be performed serially (unless more auxiliary qubits are available),

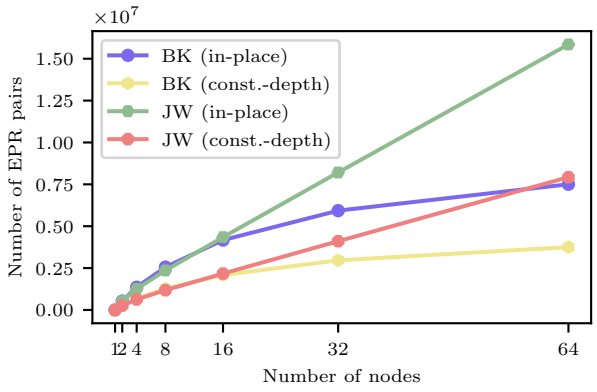


Figure 7: Number of EPR pairs required for communication to simulate one first-order Trotter step for a hydrogen ring of 32 atoms in the STO-3G basis set as a function of the number of nodes. We used either the Bravyi-Kitaev (BK) or the Jordan-Wigner (JW) encoding, see also Fig. 5. One implementation uses the in-place circuit of Fig. 6(a) which we compare to the circuit in Fig. 6(c). The constant-depth circuit requires more local resources such as $S \geq 2$ and we additionally assumed that the rotation can be performed on an auxiliary qubit on one of the nodes already storing one of the involved orbitals. We did not consider advanced optimisations and the spin-orbitals are fixed in our example to a specific node for the full duration.

but the uncomputation can be performed using only classical communication (see Fig. 1). As a result, only k EPR pairs are required, but the circuit delay is

$$Ek + D_R.$$

The parity computation of both Fig. 6(a) and (b) can be expressed as a reduction in QMPI, i.e., with a call to `QMPI_Reduce`.

In contrast to the first two circuits, Fig. 6(c) illustrates that a constant-depth implementation is possible in quantum computing using a parallel implementation of the multi-target CNOT. Specifically, this involves fanning out the control qubit using `QMPI_Bcast` to each node, which requires k EPR pairs to establish a cat state, see Section 7.1, and, thus, $S \geq 2$ is needed [25, 28]. The delay of this constant-depth implementation is

$$2E + D_R.$$

As the full quantum circuit is known at circuit generation time, a compiler may choose the optimal method for each term, given the available resources at that point in the program. See Fig. 7 for an example of a straight forward implementation without any advanced optimization applied to it.

8 CONCLUSIONS AND OUTLOOK

We introduce QMPI, an extension of MPI to distibuted quantum computing. This enables the development of portable high-performance distributed quantum programs. Complementary, we introduce the machine-independent SENDQ performance model for distributed quantum computing. The model is motivated by technological trends in building large-scale fault-tolerant quantum machines. These considerations allowed us to simplify the model by, e.g., not modeling the overhead due to classical communication as the clock cycle rate of logical quantum operations is expected to be significantly lower. As a consequence, we end up with a deliberately simple model with only a small set of general parameters.

The SENDQ model thus allows us to expose different tradeoffs of distributed quantum algorithms in a machine-agnostic fashion and without having to deal with unnecessary details. We illustrate use cases from quantum chemistry and from condensed matter physics. A common performance model makes distributed algorithms comparable and thus encourages algorithm designers to start thinking about qubit placement in a distributed setting, and overlaying communication with local computation.

The high-level modeling of the quantum network without specifying details allows hardware developers to explore implementation choices such as different quantum network topologies, and to quantify their impact in terms of the effect on the runtime of large-scale quantum computing applications. As quantum hardware matures and distributed quantum computers become available, our performance model may be further refined to capture the performance of the system more accurately.

ACKNOWLEDGMENTS

We thank Vadym Kliuchnikov for helpful discussions. This project received support from the Microsoft Swiss Joint Research Center.

REFERENCES

[1] Gadi Aleksandrowicz, Thomas Alexander, Panagiotis Barkoutsos, Luciano Bello, Yael Ben-Haim, D Bucher, FJ Cabrera-Hernández, J Carballo-Franquis, A Chen, CF Chen, et al. 2019. Qiskit: An open-source framework for quantum computing. *Accessed on: Mar* 16 (2019).

[2] David Awschalom, Karl K. Berggren, Hannes Bernien, Sunil Bhave, Lincoln D. Carr, Paul Davids, Sophia E. Economou, Dirk Englund, Andrei Faraon, Martin Fejer, Saikat Guha, Martin V. Gustafsson, Evelyn Hu, Liang Jiang, Jungsang Kim, Boris Korzh, Prem Kumar, Paul G. Kwiat, Marko Lončar, Mikhail D. Lukin, David A.B. Miller, Christopher Monroe, Sae Woo Nam, Prineha Narang, Jason S. Orcutt, Michael G. Raymer, Amir H. Safavi-Naeini, Maria Spiropulu, Kartik Srinivasan, Shuo Sun, Jelena Vučković, Edo Waks, Ronald Walsworth, Andrew M. Weiner, and Zheshen Zhang. 2021. Development of Quantum Interconnects (QuICs) for Next-Generation Information Technologies. *PRX Quantum* 2 (Feb 2021), 017002. Issue 1. https://doi.org/10.1103/PRXQuantum.2.017002

[3] Robert Beals, Stephen Brierley, Oliver Gray, Aram W Harrow, Samuel Kutin, Noah Linden, Dan Shepherd, and Mark Stather. 2013. Efficient distributed quantum computing. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 469, 2153 (2013), 20120686.

[4] Charles H Bennett. 1973. Logical reversibility of computation. *IBM journal of Research and Development* 17, 6 (1973), 525–532.

[5] Charles H. Bennett, Gilles Brassard, Sandu Popescu, Benjamin Schumacher, John A. Smolin, and William K. Wootters. 1996. Purification of Noisy Entanglement and Faithful Teleportation via Noisy Channels. *Phys. Rev. Lett.* 76 (Jan 1996), 722–725. Issue 5. https://doi.org/10.1103/PhysRevLett.76.722

[6] Benjamin Bichsel, Maximilian Baader, Timon Gehr, and Martin Vechev. 2020. Silq: A high-level quantum language with safe uncomputation and intuitive semantics. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*. 286–300.

[7] Max Born and Vladimir Fock. 1928. Beweis des adiabatensatzes. *Zeitschrift für Physik* 51, 3-4 (1928), 165–180.

[8] Sergey Bravyi and Alexei Kitaev. 2005. Universal quantum computation with ideal Clifford gates and noisy ancillas. *Physical Review A* 71, 2 (2005), 022316.

[9] Sergey B. Bravyi and Alexei Yu. Kitaev. 2002. Fermionic Quantum Computation. *Annals of Physics* 298, 1 (2002), 210–226. https://doi.org/10.1006/aphy.2002.6254

[10] Daniel Collins, Noah Linden, and Sandu Popescu. 2001. Nonlocal content of quantum operations. *Phys. Rev. A* 64 (Aug 2001), 032302. Issue 3. https://doi.org/10.1103/PhysRevA.64.032302

[11] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten Von Eicken. 1993. LogP: Towards a realistic model of parallel computation. In *Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming*. 1–12.

[12] Axel Dahlberg, Matthew Skrzypczyk, Tim Coopmans, Leon Wubben, Filip Rozpędek, Matteo Pompili, Arian Stolk, Przemysław Pawełczak, Robert Knegjens, Julio de Oliveira Filho, et al. 2019. A link layer protocol for quantum networks. In *Proceedings of the ACM Special Interest Group on Data Communication*. 159–173.

[13] Axel Dahlberg and Stephanie Wehner. 2018. SimulaQron—a simulator for developing quantum internet software. *Quantum Science and Technology* 4, 1 (2018), 015001.

[14] Sebastian Debone, Runsheng Ouyang, Kenneth Goodenough, and David Elkouss. 2020. Protocols for creating and distilling multipartite GHZ states with Bell pairs. *IEEE Transactions on Quantum Engineering* (2020).

[15] Stephen DiAdamo, Marco Ghibaudi, and James Cruise. 2021. Distributed Quantum Computing and Network Control for Accelerated VQE. arXiv:2101.02504 [quant-ph]

[16] A. Einstein, B. Podolsky, and N. Rosen. 1935. Can Quantum-Mechanical Description of Physical Reality Be Considered Complete? *Phys. Rev.* 47 (May 1935), 777–780. Issue 10. https://doi.org/10.1103/PhysRev.47.777

[17] J. Eisert, K. Jacobs, P. Papadopoulos, and M. B. Plenio. 2000. Optimal local implementation of nonlocal quantum gates. *Phys. Rev. A* 62 (Oct 2000), 052317. Issue 5. https://doi.org/10.1103/PhysRevA.62.052317

[18] Moritz Forsch, Robert Stockill, Andreas Wallucks, Igor Marinković, Claus Gärtner, Richard A Norte, Frank van Otten, Andrea Fiore, Kartik Srinivasan, and Simon Gröblacher. 2020. Microwave-to-optics conversion using a mechanical oscillator in its quantum ground state. *Nature Physics* 16, 1 (2020), 69–74.

[19] Jay Gambetta. 2020. IBM's Roadmap For Scaling Quantum Technology. https://www.ibm.com/blogs/research/2020/09/ibm-quantum-roadmap/. Accessed: 16.03.2021.

[20] Craig Gidney and Martin Ekerå. 2019. How to factor 2048 bit rsa integers in 8 hours using 20 million noisy qubits. *arXiv preprint arXiv:1905.09749* (2019).

[21] Alexander S Green, Peter LeFanu Lumsdaine, Neil J Ross, Peter Selinger, and Benoît Valiron. 2013. Quipper: a scalable quantum programming language. In *Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation*. 333–342.

[22] Thomas Häner, Samuel Jaques, Michael Naehrig, Martin Roetteler, and Mathias Soeken. 2020. Improved quantum circuits for elliptic curve discrete logarithms. In *International Conference on Post-Quantum Cryptography*. Springer, 425–444.

[23] Torsten Hoefler, Prabhanjan Kambadur, Richard L Graham, Galen Shipman, and Andrew Lumsdaine. 2007. A case for standard non-blocking collective operations. In *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*. Springer, 125–134.

[24] Julian Hofmann, Michael Krug, Norbert Ortegel, Lea Gérard, Markus Weber, Wenjamin Rosenfeld, and Harald Weinfurter. 2012. Heralded Entanglement Between Widely Separated Atoms. *Science* 337, 6090 (2012), 72–75. https://doi.org/10.1126/science.1221856 arXiv:https://science.sciencemag.org/content/337/6090/72.full.pdf

[25] Peter Høyer and Robert Špalek. 2005. Quantum fan-out is powerful. *Theory of computing* 1, 1 (2005), 81–103.

[26] Ali JavadiAbhari, Shruti Patil, Daniel Kudrow, Jeff Heckey, Alexey Lvov, Frederic T Chong, and Margaret Martonosi. 2015. ScaffCC: Scalable compilation and analysis of quantum programs. *Parallel Comput.* 45 (2015), 2–17.

[27] P. Jordan and E. Wigner. 1928. Über das Paulische Äquivalenzverbot. *Zeitschrift für Physik* 47, 9 (1928), 631–651. https://doi.org/10.1007/BF01331938

[28] Vadym Kliuchnikov and Alexander Vaschillo. 2021. Layout based on cat states. *In preparation* (2021).

[29] Joonho Lee, Dominic Berry, Craig Gidney, William J Huggins, Jarrod R McClean, Nathan Wiebe, and Ryan Babbush. 2020. Even more efficient quantum computations of chemistry through tensor hypercontraction. *arXiv preprint arXiv:2011.03494* (2020).

[30] Bjoern Lekitsch, Sebastian Weidt, Austin G Fowler, Klaus Mølmer, Simon J Devitt, Christof Wunderlich, and Winfried K Hensinger. 2017. Blueprint for a microwave trapped ion quantum computer. *Science Advances* 3, 2 (2017), e1601540.

[31] Daniel Litinski. 2019. Magic state distillation: Not as costly as you think. *Quantum* 3 (2019), 205.

[32] Paul Magnard, Simon Storz, Philipp Kurpiers, Josua Schär, Fabian Marxer, Janis Lütolf, T Walter, J-C Besse, M Gabureac, K Reuer, et al. 2020. Microwave quantum link between superconducting circuits housed in spatially separated cryogenic systems. *Physical Review Letters* 125, 26 (2020), 260502.

[33] Sam McArdle, Suguru Endo, Alán Aspuru-Guzik, Simon C. Benjamin, and Xiao Yuan. 2020. Quantum computational chemistry. *Rev. Mod. Phys.* 92 (Mar 2020), 015003. Issue 1. https://doi.org/10.1103/RevModPhys.92.015003

[34] Jarrod R McClean, Nicholas C Rubin, Kevin J Sung, Ian D Kivlichan, Xavier Bonet-Monroig, Yudong Cao, Chengyu Dai, E Schuyler Fried, Craig Gidney, Brendan Gimby, et al. 2020. OpenFermion: the electronic structure package for quantum computers. *Quantum Science and Technology* 5, 3 (2020), 034014.

[35] Rodney Van Meter, WJ Munro, Kae Nemoto, and Kohei M Itoh. 2008. Arithmetic on a distributed-memory quantum multicomputer. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 3, 4 (2008), 1–23.

[36] Rodney Doyle Van Meter III. 2006. Architecture of a quantum multicomputer optimized for shor's factoring algorithm. *arXiv preprint quant-ph/0607065* (2006).

[37] David L Moehring, Peter Maunz, Steve Olmschenk, Kelly C Younge, Dzmitry N Matsukevich, L-M Duan, and Christopher Monroe. 2007. Entanglement of single-atom quantum bits at a distance. *Nature* 449, 7158 (2007), 68–71.

[38] C. Monroe, R. Raussendorf, A. Ruthven, K. R. Brown, P. Maunz, L.-M. Duan, and J. Kim. 2014. Large-scale modular quantum-computer architecture with atomic memory and photonic interconnects. *Phys. Rev. A* 89 (Feb 2014), 022317. Issue 2. https://doi.org/10.1103/PhysRevA.89.022317

[39] Hartmut Neven. 2020. Google Quantum AI updates at Quantum Summer Symposium 2020. https://www.youtube.com/watch?v=TJ6vBNEQReU Online; posted 3-September-2020, accessed 25-March-2021.

[40] Thien Nguyen, Anthony Santana, Tyler Kharazi, Daniel Claudino, Hal Finkel, and Alexander McCaskey. 2020. Extending C++ for Heterogeneous Quantum-Classical Computing. *arXiv preprint arXiv:2010.03935* (2020).

[41] Naomi H. Nickerson, Joseph F. Fitzsimons, and Simon C. Benjamin. 2014. Freely Scalable Quantum Technologies Using Cells of 5-to-50 Qubits with Very Lossy and Noisy Photonic Links. *Phys. Rev. X* 4 (Dec 2014), 041041. Issue 4. https://doi.org/10.1103/PhysRevX.4.041041

[42] Michael A Nielsen and Isaac Chuang. 2002. Quantum computation and quantum information.

[43] Gerardo Ortiz, James E Gubernatis, Emanuel Knill, and Raymond Laflamme. 2001. Quantum algorithms for fermionic simulations. *Physical Review A* 64, 2 (2001), 022319.

[44] Stefano Pirandola, Ulrik L Andersen, Leonardo Banchi, Mario Berta, Darius Bunandar, Roger Colbeck, Dirk Englund, Tobias Gehring, Cosmo Lupo, Carlo Ottaviani, et al. 2020. Advances in quantum cryptography. *Advances in Optics and Photonics* 12, 4 (2020), 1012–1236.

[45] Markus Reiher, Nathan Wiebe, Krysta M Svore, Dave Wecker, and Matthias Troyer. 2017. Elucidating reaction mechanisms on quantum computers. *Proceedings of the National Academy of Sciences* 114, 29 (2017), 7555–7560.

[46] Peter Sanders and Jesper Larsson Träff. 2006. Parallel prefix (scan) algorithms for MPI. In *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*. Springer, 49–57.

[47] VM Schäfer, CJ Ballance, K Thirumalai, LJ Stephenson, TG Ballance, AM Steane, and DM Lucas. 2018. Fast quantum logic gates with trapped-ion qubits. *Nature* 555, 7694 (2018), 75–78.

[48] Artur Scherer, Benoît Valiron, Siun-Chuon Mau, Scott Alexander, Eric Van den Berg, and Thomas E Chapuran. 2017. Concrete resource analysis of the quantum linear-system algorithm used to compute the electromagnetic scattering cross section of a 2D target. *Quantum Information Processing* 16, 3 (2017), 1–65.

[49] Peter W Shor. 1994. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*. Ieee, 124–134.

[50] Rolando Somma, Gerardo Ortiz, James E Gubernatis, Emanuel Knill, and Raymond Laflamme. 2002. Simulating physical phenomena by quantum networks. *Physical Review A* 65, 4 (2002), 042323.

[51] Damian S Steiger, Thomas Häner, and Matthias Troyer. 2018. ProjectQ: an open source software framework for quantum computing. *Quantum* 2 (2018), 49.

[52] Qiming Sun, Timothy C. Berkelbach, Nick S. Blunt, George H. Booth, Sheng Guo, Zhendong Li, Junzi Liu, James D. McClain, Elvira R. Sayfutyarova, Sandeep Sharma, Sebastian Wouters, and Garnet Kin-Lic Chan. 2017. PySCF: the Python-based simulations of chemistry framework. , e1340 pages. https://doi.org/10.1002/wcms.1340 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/wcms.1340

[53] Krysta Svore, Alan Geller, Matthias Troyer, John Azariah, Christopher Granade, Bettina Heim, Vadym Kliuchnikov, Mariia Mykhailova, Andres Paz, and Martin Roetteler. 2018. Q# enabling scalable quantum computing and development with a high-level dsl. In *Proceedings of the Real World Domain Specific Languages Workshop 2018*. 1–10.

[54] Yuta Tsuchimoto, Patrick Knüppel, Aymeric Delteil, Zhe Sun, Martin Kroner, and Ata ç Imamoğlu. 2017. Proposal for a quantum interface between photonic and superconducting qubits. *Phys. Rev. B* 96 (Oct 2017), 165312. Issue 16. https://doi.org/10.1103/PhysRevB.96.165312

[55] S. J. van Enk, J. I. Cirac, and P. Zoller. 1997. Ideal Quantum Communication over Noisy Channels: A Quantum Optical Implementation. *Phys. Rev. Lett.* 78 (Jun 1997), 4293–4296. Issue 22. https://doi.org/10.1103/PhysRevLett.78.4293

[56] Rodney Van Meter and Simon J Devitt. 2016. The path to scalable distributed quantum computing. *Computer* 49, 9 (2016), 31–42.

[57] Rod Van Meter, Kae Nemoto, and W Munro. 2007. Communication links for distributed quantum computation. *IEEE Trans. Comput.* 56, 12 (2007), 1643–1653.

[58] Vera von Burg, Guang Hao Low, Thomas Häner, Damian S Steiger, Markus Reiher, Martin Roetteler, and Matthias Troyer. 2020. Quantum computing enhanced computational catalysis. *arXiv preprint arXiv:2007.14460* (2020).

[59] David W Walker and Jack J Dongarra. 1996. MPI: a standard message passing interface. *Supercomputer* 12 (1996), 56–68.
[60] Adam Bene Watts, Robin Kothari, Luke Schaeffer, and Avishay Tal. 2019. Exponential Separation between Shallow Quantum Circuits and Unbounded Fan-in Shallow Classical Circuits. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing* (Phoenix, AZ, USA) *(STOC 2019)*. Association for Computing Machinery, New York, NY, USA, 515–526. https://doi.org/10.1145/3313276.3316404
[61] Stephanie Wehner, David Elkouss, and Ronald Hanson. 2018. Quantum internet: A vision for the road ahead. *Science* 362, 6412 (2018).
[62] James D. Whitfield, Vojt ěch Havlíček, and Matthias Troyer. 2016. Local spin operators for fermion simulations. *Phys. Rev. A* 94 (Sep 2016), 030301. Issue 3. https://doi.org/10.1103/PhysRevA.94.030301
[63] Ze-Liang Xiang, Mengzhen Zhang, Liang Jiang, and Peter Rabl. 2017. Intracity quantum communication via thermal microwave networks. *Physical Review X* 7, 1 (2017), 011035.
[64] Anocha Yimsiriwattana and Samuel J Lomonaco Jr. 2004. Distributed quantum computing: A distributed Shor algorithm. In *Quantum Information and Computation II*, Vol. 5436. International Society for Optics and Photonics, 360–372.
[65] Anocha Yimsiriwattana and Samuel J Lomonaco Jr. 2004. Generalized GHZ states and distributed quantum computing. *arXiv preprint quant-ph/0402148* (2004).
[66] Youpeng Zhong, Hung-Shen Chang, Audrey Bienfait, Étienne Dumur, Ming-Han Chou, Christopher R Conner, Joel Grebel, Rhys G Povey, Haoxiong Yan, David I Schuster, et al. 2021. Deterministic multi-qubit entanglement in a quantum network. *Nature* 590, 7847 (2021), 571–575.

A EXAMPLE IMPLEMENTATIONS

A.1 Moving Qubits

Here, we give an example implementation of `QMPI_Send_move` and `QMPI_Recv_move` in our QMPI prototype. The sender executes `QMPI_Send_move`, which sends a qubit (with move semantics), and the receiver calls the corresponding `QMPI_Recv_move`. Both of these functions can be implemented using EPR-pair preparation and local quantum operations as follows:

```
void QMPI_Send_move(QMPI_QUBIT_PTR qubit, int dest, int
     tag, MPI_Comm comm) {
  auto epr_qubit = QMPI_Alloc_qmem(1);
  QMPI_Prepare_EPR(epr_qubit, dest, tag, comm);
  CNOT(qubit, epr_qubit);
  int r=0;
  r = Measure(epr_qubit);
  H(qubit);
  r |= 2 * Measure(qubit);
  QMPI_Free_qmem(epr_qubit, 1);
  MPI_Send(&r, 1, MPI_INT, dest, tag, comm);
}
void QMPI_Recv_move(QMPI_QUBIT_PTR qubit, int src, int
     tag, MPI_Comm comm) {
  QMPI_Prepare_EPR(qubit, src, tag, comm);
  int r;
  MPI_Recv(&r, 1, MPI_INT, src, tag, comm,
     MPI_STATUS_IGNORE);
  if (r&1)
    X(qubit);
  if (r&2)
    Z(qubit);
}
```

We note that these functions may also be implemented by relying on `QMPI_Send / Recv` and their inverses: Once the value of a qubit is shared between two nodes, it is no longer possible to distinguish sender from receiver. Therefore, the two involved nodes may exchange roles when calling the inverses of `QMPI_Send / Recv`, resulting in a slightly less efficient implementation of teleportation (since measurement results are communicated using two one-bit messages instead of one two-bit message).

A.2 Transverse-field Ising Model (TFIM).

As a second code example, we provide an implementation of time evolution under a TFIM Hamiltonian below. Note that the code also includes annealing from a fully transverse-field model to a fully classical Ising model. We note that the code can be significantly optimized by using asynchronous communication primitives. However, we use blocking calls only to simplify the presentation.

```
#include "qmpi.hpp"
#include <iostream>

using namespace QMPI;

void tfim_time_evolution(double const& J, double const&
      g, double const& time, QMPI_QUBIT_PTR qubits,
     unsigned num_spins, unsigned num_trotter) {
  int rank, size;
  QMPI_Comm_size(QMPI_COMM_WORLD, &size);
  QMPI_Comm_rank(QMPI_COMM_WORLD, &rank);
  auto dt = time/num_trotter;
  for (unsigned step=0; step < num_trotter; ++step) {
    for (unsigned site = 0; site < num_spins-1; ++site)
      {
      CNOT(qubits+site, qubits+site+1);
      Rz(qubits+site+1, 2.0 * J * dt);
      CNOT(qubits+site, qubits+site+1);
    }
    if (size == 1) { // single rank: no communication
     required
      CNOT(qubits+num_spins-1, qubits);
      Rz(qubits, 2.0 * J * dt);
      CNOT(qubits+num_spins-1, qubits);
    }
    else {
      for (unsigned odd = 0; odd < 2; ++odd) {
        if ((rank&1) == odd) {
          QMPI_Send(qubits, (rank-1+size)%size, 0,
     QMPI_COMM_WORLD);
          QMPI_Unsend(qubits, (rank-1+size)%size, 0,
     QMPI_COMM_WORLD);
        }
        else {
          auto tmpqubit = QMPI_Alloc_qmem(1);
          QMPI_Recv(tmpqubit, (rank+1)%size, 0,
     QMPI_COMM_WORLD);
          CNOT(qubits+num_spins-1, tmpqubit);
          Rz(tmpqubit, 2.0 * J * dt);
          CNOT(qubits+num_spins-1, tmpqubit);
          QMPI_Unrecv(tmpqubit, (rank+1)%size, 0,
     QMPI_COMM_WORLD);
          QMPI_Free_qmem(tmpqubit, 1);
        }
      }
    }
    for (unsigned site = 0; site < num_spins; ++site)
      Rx(qubits+site, -2.0*g*dt);
  }
}
int main() {
  QMPI_Init(0, 0);
  int rank, size;
  QMPI_Comm_size(QMPI_COMM_WORLD, &size);
  QMPI_Comm_rank(QMPI_COMM_WORLD, &rank);
  // Number of spins per node:
  unsigned num_local_spins = 2;
  // Number of annealing steps:
  double num_annealing_steps = 100;
  // Trotter number
```

```
  unsigned num_trotter = 1;
  double time = 1; // time to evolve per annealing step
  // Parameters of transverse-field Ising model
  double J = 0.; // coupling strength
  double g = 1.; // transverse field
  // allocate spins:
  auto qubits = QMPI_Alloc_qmem(num_local_spins);
  // init to ground state
  for (unsigned i = 0; i < num_local_spins; ++i)
    H(qubits+i);
  // run annealing schedule
  for (unsigned step = 0; step < num_annealing_steps;
     ++step) {
    J = step * 1.0/num_annealing_steps;
    g = 1.0-J;
    tfim_time_evolution(J, g, time, qubits,
     num_local_spins, num_trotter);
  }
  // Measure
  std::vector<int> res(num_local_spins);
  for (unsigned i = 0; i < num_local_spins; ++i)
    res[i] = Measure(qubits+i);
  QMPI_Free_qmem(qubits, num_local_spins);
  // Gather all (classical) results and output
  std::vector<int> allres(num_local_spins*size);
  MPI_Gather(&res[0], num_local_spins, MPI_INT, &allres
     [0], num_local_spins, MPI_INT, 0, QMPI_COMM_WORLD)
     ;
  if (rank == 0) {
    std::cout << "Measurements: ";
    for (auto r : allres)
      std::cout << r << " ";
    std::cout << std::endl;
  }
  QMPI_Finalize();
  return 0;
}
```

Listing 1: QMPI code for TFIM time evolution and annealing.

Appendix: Artifact Description/Artifact Evaluation

SUMMARY OF THE EXPERIMENTS REPORTED

Our paper presents a performance model for distributed quantum computing called SENDQ and a specification of quantum MPI, an extension of MPI that enables communication of quantum data. In order to ensure that our proposed QMPI specification is consistent, we implemented a prototype of the proposed point-to-point and collective functions based on MPI that forwards all quantum-specific calls (i.e., generation of EPR pairs and quantum gates / measurements) to a quantum circuit simulator (similar to [1]) running as a separate thread of the root MPI process. We are unable to provide the code for the QMPI prototype at this time because we are still awaiting company-internal approval for this software disclosure. However, the manuscript does not contain any data that were obtained from this prototype (it was only used to check the QMPI specification).

All data in our manuscript were obtained directly from the presented SENDQ model with the exception of the chemistry example, for which we used OpenFermion and its PySCF plugin, as described in the manuscript. Specifically, the steps are as follows:

1) use OpenFermion [2] and its PySCF plugin [3] with PySCF [4] to generate Hamiltonian

from openfermion.chem import MolecularData, make_atomic_ring

from openfermionpyscf import run_pyscf

hydrogen_ring32 = make_atomic_ring(n_atoms=32, spacing=1.2, basis='sto-3g', atom_type='H', charge=0, filename='hydrogen_ring_32')

run_pyscf(hydrogen_ring32, run_scf=1)

2) load the Hamiltonian into OpenFermion and transform it using a Bravyi-Kitaev or Jordan-Wigner encoding

from openfermion.chem import MolecularData

from openfermion.transforms import get_fermion_operator, bravyi_kitaev, jordan_wigner

molecule = MolecularData(filename="hydrogen_ring_32.hdf5")

fermion_hamiltonian = get_fermion_operator(molecule.get_molecular_hamiltonian())

jordan_wigner_hamiltonian = jordan_wigner(fermion_hamiltonian)

bravyi_kitaev_hamiltonian = bravyi_kitaev(fermion_hamiltonian)

3) jordan_wigner_hamiltonian and bravyi_kitaev_hamiltonian are now Hamiltonians expressed as sums of multi-qubit Pauli operators. A histogram of the number of qubits in each term is in Fig. 5

4) Fig. 6 shows the cost to implement one first order Trotter step of the Hamiltonian. The resources per term are given in the text and in Fig. 6.

References

[1] https://github.com/ProjectQ-Framework/ProjectQ/blob/develop/projectq/backends/_sim/_cppkernels/simulator.hpp

[2] https://pypi.org/project/openfermion/1.0.1/

[3] https://github.com/quantumlib/OpenFermion-PySCF/tree/v0.5

[4] https://pypi.org/project/pyscf/1.7.5.2/