Fast simulation of quantum algorithms using circuit optimization

Gian Giacomo Guerreschi

Intel Labs, Intel Corporation, 2200 Mission College Blvd, Santa Clara, CA 95054

Classical simulators play a major role in the development and benchmark of quantum algorithms and practically any software framework for quantum computation provides the option of running the algorithms on simulators. However, the development of quantum simulators was substantially separated from the rest of the software frameworks which, instead, focus on usability and compilation. Here, we demonstrate the advantage of co-developing and integrating simulators and compilers by proposing a specialized compiler pass to reduce the simulation time for arbitrary circuits. While the concept is broadly applicable, we present a concrete implementation based on the Intel Quantum Simulator, a high-performance distributed simulator. As part of this work, we extend its implementation with additional functionalities related to the representation of quantum states. The communication overhead is reduced by changing the order in which state amplitudes are stored in the distributed memory, a concept analogous to the distinction between local and global qubits for distributed Schrödinger-type simulators. We then implement a compiler pass to exploit the novel functionalities by introducing special instructions governing data movement as part of the quantum circuit. Those instructions target unique capabilities of simulators and have no analogue in actual quantum devices. To quantify the advantage, we compare the time required to simulate random circuits with and without our optimization. The simulation time is typically halved.

1 Introduction

The field of quantum computation has recently graduated from scientific research to technology development. One of the most visible changes has been the creation of increasingly sophisticated software frameworks to enable the execution of algorithms on actual quantum devices [1, 2, 3, 4, 5, 6, 7, 8, 9]. A major component of the frameworks is the compiler, sometimes also called transpiler, mapper or scheduler depending on its specific role [10, 11, 12, 13]. The compiler is required to translate a relatively abstract algorithm into instructions executable by the electronic controller of the quantum device. Every advanced framework also includes the option of running the algorithm with a simulator, i.e. classical software specialized in simulating quantum circuits.

Gian Giacomo Guerreschi: gian.giacomo.guerreschi@intel.com

Simulators have at least two important applications. On one hand they allow to benchmark quantum algorithms without the limitations of existing hardware, in particular finite coherence time [14, 15, 16, 17, 18, 19, 20, 21]. On the other hand, when equipped with realistic noise models, simulators help characterizing hardware noise and decoherence and, possibly, inform on its origin [22, 23]. In spite of their importance, no specialized optimization is integrated in the popular compiler frameworks that mostly deal with restrictions on the qubit connectivity and on the type of available gates. Noticeably, ProjectQ compiler [3] accepts multi-qubit operations in the algorithm's description without forcing their decomposition in 1- and 2-qubit gates, a feature suitable for simulation since multi-qubit gates are not native in most quantum hardware.

In this work we present a new compiler pass dedicated to reducing the simulation time for arbitrary quantum circuits. It takes advantage of the specific implementation of the simulator and is, therefore, a backend-aware compilation pass in contrast to hardware-agnostic passes earlier in the compiler chain [24, 11]. We adopt the Intel Quantum Simulator (IQS) [14, 25], a high-performance simulator exhibiting both multi-threading and distributed parallelization, and extend it by introducing a flexible way to represent the quantum state as a distributed vector. The functionality is available to all users of IQS via its open-source distribution at `https://github.com/iqusoft/intel-qs`. Then we create a compiler pass minimizing the communication overhead incurred when multiple computing nodes are involved in the simulation. The optimization is based on the distinction between gates that require internode communication and those that do not, minimizing the number of the former ones by means of a greedy search.

While inspired by the implementation of distributed state-vector simulators, similar optimization strategies may be helpful in other scenarios when

qubits are divided in two or more categories. For example, one can think of a network of quantum computing chips all involved in the same algorithm. It is easy to envision that the qubits used to communicate between chips differ from in-chip qubits, and therefore the distinction of local/global qubits used in the optimization may be adapted to correspond to qubits for computing/communicating. The scenario of a network of quantum chips is both part of companies road-maps [26] and the starting point of recent research [27, 28] and open-source software [29, 30].

Previous works on distributed simulators have proposed similar methods to reduce the communication overhead [31, 17, 20]. Other works suggested several optimizations based on circuit manipulation, like the notion of fusing several consecutive gates acting on a common subset of qubits [32, 4] and then re-express them with fewer operations. Notice that an efficient optimizing pass for gate fusion needs a way to express multi-qubit gates as part of the intermediate representation of the circuit [17]. In addition, ad-hoc circuit manipulation has been used to simulate the kind of random circuits proposed by Google for quantum supremacy [33] with a hybrid Schrödinger-Feynman simulator developed by NASA [23], a tensor network-based simulator by Alibaba [34], or by leveraging secondary storage as analyzed by IBM [35]. However, a central part of our view is that the optimization methods should not be part of a stand-alone simulator, but should occur inside the framework that compiles quantum circuits for different backends. When we describe co-development between compilers and simulators, we have this shift of perspective in mind [36, 32].

To evaluate the efficacy of the specific optimization discussed in this work, we compare the time to simulate random circuits with and without the proposed compiler pass. We vary the ratio of 1- and 2-qubit gates in the circuits and report a reduction by one order of magnitude in the number of gates requiring inter-node communication. This translates in the overall simulation time being typically halved.

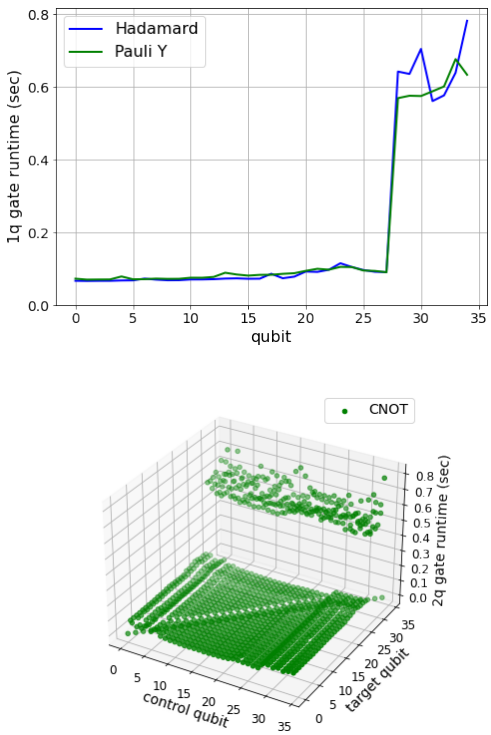

Figure 1: Simulations of $n = 35$ qubits using 128 MPI processes, each running on one node of Frontera supercomputer. Each process uses 24 OpenMP threads. **(Top)** Time to execute a 1-qubit gate as a function of the qubit involved. When the gate is executed on qubit with index $m = n - \lfloor \log_2 k \rfloor$ (with k being the number of MPI processes), the communication between the MPI tasks is happening between sockets of the same node. For qubits with index larger than m, communication is between distinct nodes. **(Bottom)** Time to execute a 2-qubit gate as a function of the involved qubits. CNOT is the controlled Pauli X gate.

2 Extending Intel Quantum Simulator

IQS is a state-of-the-art Schrödinger simulator which represents the full state of n qubits by storing all its 2^n complex amplitudes. All operations required by quantum circuits, namely state preparation and several 1- and 2-qubit gates, are implemented as linear manipulation of the amplitudes. Since the state vector grows exponentially in size with the number of qubits, it soon exceeds the memory available in a single computing node. Hence the necessity of distributing the state across multiple nodes, each with its own local memory.

The implementation is presented in references [14, 25], together with the communication pattern of the Message Passing Interface (MPI) protocol. Here, we provide a high-level summary of the properties exploited by the compiler pass. State preparation and measurement are done with little or no communication and efficiently parallelized over computing nodes and processor's cores (via OpenMP). Gate operations are limited to those gates that update amplitudes in pairs: arbitrary 1-qubit gates and 2-qubit gates in the form of controlled 1-qubit gates. Notice that the 2-qubit gate known as CNOT is part of this set and, thus, IQS can simulate all circuits (arbitrary 1-qubit gates and CNOT allow universal quantum computation).

We divide the qubits into two distinct groups of local, respectively global, qubits [17]. The number of local *vs* global qubits depends on the number k of MPI processes, specifically there are $\lfloor \log_2 k \rfloor$ global qubits and $m = n - \lfloor \log_2 k \rfloor$ local qubits. Inter-process communication is required by all gates acting on any of the global qubits (apart from the special case of a controlled 2-qubit gate with global control and local target), and the communication overhead is about an order of magnitude. Apart from the communication

overhead, the cost of 1- and 2-qubit gates is not significantly different. Figure 1 shows the time required to simulate 1- and 2-qubit gates as a function of the involved qubits. In the numerical experiments, the first $m = 28$ qubits are local and is evident that the communication overhead starts from gates on qubits with index 28 or greater. The simulations were run on Frontera supercomputer at the Texas Advanced Computing Center in which each computing node is equipped with 2 sockets of Intel® Xeon® Processor 8280 CPU (24 cores per socket).

In the original IQS implementation it is not possible to choose which qubits are local and which are global, by default the first m qubits are the local ones. Consider an arbitrary n-qubit state, it can be written in the computational basis as:

$$|\psi\rangle = \sum_{i\in\{0,1\}^n} \alpha_i \, |i_0\rangle \, |i_1\rangle \ldots |i_{n-1}\rangle$$

where $i = [i_{n-1} \ldots i_1 i_0]$ can be seen as the binary representation of integer $i = \sum_{q=0}^{n-1} i_q 2^q$. IQS stores the quantum state as a distributed vector whose i-th component corresponds to amplitude α_i. To provide opportunities for fast simulation, we want to increase the flexibility of this representation and, specifically, we allow to change the order in which the amplitudes are stored. Consider an arbitrary permutation σ of the qubit indices: under such permutation, qubit q will be associated with bit $\sigma(q)$ of the index of the vector's component. Therefore, amplitude α_i is stored as component $j(i,\sigma) = \sum_{q=0}^{n-1} i_q 2^{\sigma(q)}$ of the state vector.

We introduce the new function `PermuteQubits`(σ') to update the representation of quantum states in IQS. Consider that the current qubit permutation is σ and that we want to update it to σ'. The qubits are reordered in three steps: 1) local qubits (according to σ) are reordered among themselves, 2) global qubits (according to σ) are reordered among themselves, and 3) pairs formed by one local and one global qubits (according to either σ or, with opposite roles, σ') are exchanged. The cost of the three steps is analogous to that of a single local gate, a single global gate, and one global gate per exchanged pair respectively. We discuss the implementation of `PermuteQubits` in Appendix A, together with the pseudo-code to divide every transformation from qubit permutation σ to σ' in the three steps. As part of this contribution, we added the distributed implementation of SWAP gates to IQS.

The property of local *vs.* global qubits is still valid, but the distinction now depends on permutation σ. The local qubits are those with index q such that $\sigma(q) < m$. There still are $m = n - \lfloor \log_2 k \rfloor$ local qubits and $\lfloor \log_2 k \rfloor$ global qubits.

3 Compiler pass to optimize circuit simulation

From the benchmarks reported in Figure 1, it is clear that gates acting on global qubits take much longer to simulate than gates on local qubits alone. We propose a compiler pass that goes through an arbitrary quantum circuit and eliminates the communication overhead from most gates by adding qubit-reordering instructions. Communication is required to execute such instructions in IQS but, when they are chosen carefully, the overhead is reduced. As we quantify in the next section, even for unstructured circuits the simulation time is halved.

The compiler pass is described as Algorithm 1. It

Algorithm1 Compiler pass specialized for IQS

Require: The quantum circuit is provided as a DAG describing the logical dependency of the gates. The initial qubit order corresponds to the identity permutation: $\forall q,\ \sigma(q) = q$.

Ensure: The compiled circuit C is represented by a sequential list of instructions (either gates or calls to `PermuteQubits`).

```
G ← DAG of circuit
σ ←identity permutation
C ← {}
while G.vertices≠ ∅ do
    for v in G.vertices do
        if (v has no incoming edges) ∧ (v acts on local qubits according to σ) then
            add ApplyGate(v) to C
            remove v from G
        end if
    end for
    σ̃ ←identity permutation
    ã ← 0
    for l in local qubits according to σ do
        for g in global qubits according to σ do
            σ' ← σ ∘ (l g)
            a' ← count G.vertices on local qubits according to σ'
            if a' > ã then
                σ̃ ← σ'
                ã ← a'
            end if
        end for
    end for
    if ã > 0 then
        σ ← σ̃
        add PermuteQubits(σ) to C
    else
        v ← vertex of G without incoming edges
        add v to C
        remove v from G
    end if
end while
```

assumes that the quantum circuit is provided in terms of a directed acyclic graph (DAG) in which each vertex corresponds to a quantum gate and each edge to a logical dependency: the target vertex (i.e. gate) of the edge must be performed after the source vertex. The DAG may or may not be created considering that quantum gates may commute, but in our case we include commutativity to eliminate unnecessary edges. While common [37, 38, 39], this is not the only way to describe circuits as DAGs [40]. In describing the compiler pass, we will use the terms gate and vertex interchangeably.

The algorithm starts by scheduling all gates that act on local qubits only and do not have unresolved logical dependencies (lines 5:10). The latter condition corresponds to vertices without incoming edges. When it is not possible to schedule any further gate, the algorithm considers all possible qubit permutations obtained by exchanging one local and one global qubit according to the current qubit permutation σ (lines 11:22). There are $m(n-m) \leq n^2/4$ permutations to consider. Each permutation is evaluated depending on the number of gates that it allows to simulate on local qubits, and the permutation with the highest count is recorded as $\tilde{\sigma}$ in line 18. The compiler add instruction `PermuteQubit`$(\tilde{\sigma})$ if it removes communication from at least one gate, otherwise schedule the first gate without incoming edges even if it requires communication (lines 23:30). The algorithm iterates until all gates are scheduled.

We observe that the compiler pass uses a simplified interpretation of the simulation times reported in Figure 1. In particular, it considers only whether a qubit is local or global and neglects the fine-grained pattern of the actual simulation time. In addition, it does not consider the actual value of the communication overhead, but is only aware that gates on global qubits takes much longer than gates on local qubits. For example, the plot in Figure 1(top) would be abstracted as a step function: low simulation time until qubit $m-1$ and high simulation time from qubit m onward. By updating the qubit permutation σ, the compiler is able to remove certain communication overhead and, therefore, minimize the overall simulation time. More advanced compiler passes may use the full information from simulator's benchmarks. In Appendix B, we discuss how to implement the proposed pass in existing compiler pipelines.

4 Numerical results

To quantify the advantage provided by the compiler pass, we estimate the time to simulate random circuits before and after their optimization. We consider random circuits of the following form: fix the number of gates, every gate has a probability p to be a 2-qubit gate otherwise with probability $1-p$ it is a 1-qubit gate. Due to stochasticity, the number

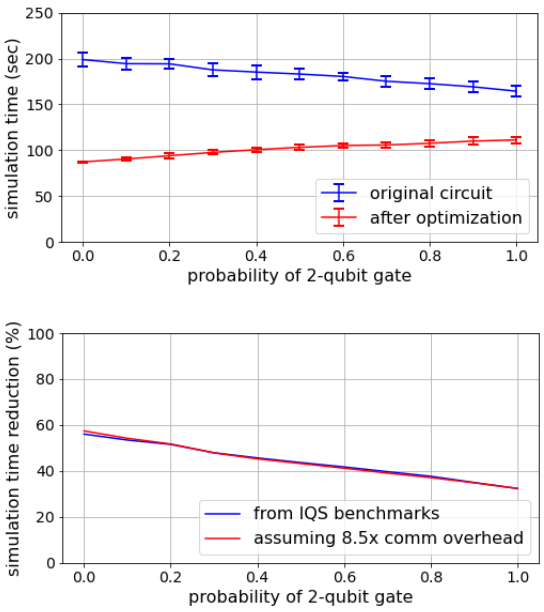

Figure 2: Effect of the compiler pass to optimize quantum circuits for simulation with IQS. **(Top)** Simulation time of random circuits of 1050 gates on $n = 35$ qubits. The timings have been estimated for the same configuration used in Figure 1: 128 computing nodes of Frontera supercomputer at TACC. There are $m = 28$ local qubits. Datapoints represent the average over 20 random circuits and the vertical bars indicate one standard deviation. **(Bottom)** Reduction of the simulation time between original and optimized circuit. The blue line is based on direct benchmarks of IQS, the red line is obtained by assuming that operations requiring communication have an overhead of $8.5\times$ with respect to operations implementable locally. This overhead is in line with the benchmarks in Figure 1.

of two-qubit gates is not strictly determined. The specific type of gate is selected uniformly at random in the sets $\{H, Y\}$ and $\{CNOT\}$ for 1- and 2-qubit gates respectively. Finally the qubits involved in each gate are chosen uniformly at random. Recall that the simulation time is largely independent on the specific type of gate and the choice is therefore arbitrary and inconsequential.

The random circuits we consider are simply a sequence of gates chosen at random, and do not aim at reproducing a Haar-random unitary on n qubits. They have been chosen to reflect the common situation in which programmers of quantum algorithms are not familiar with the design of the simulator and, thus, do not tailor their code (via gate selection or qubit indexing) for a specific simulator. In addition, while random circuits are not associated with a specific application, they are representative of algorithms like the Quantum Approximate Optimization Algorithm to solve combinatorial problems on class of random instances [41] or the dynamical evolution of spin systems with random interactions [42]. In Appendix C, we present results for circuits having a clear pattern of 2-qubit gates and show evidence supporting

an even greater reduction in simulation time.

In the first study, we fix the number of qubits to $n = 35$ and vary the probability p. Despite only a fraction $7/35 = 1/5$ of the qubits being global, the simulation time is reduced by between 32.3% (for $p = 1$) and 56.0% (for $p = 0$), according to the formula $1 - t_{\text{opt}}/t_{\text{orig}}$ in which t_{orig} is the simulation time for the original circuit and t_{opt} that for the optimized circuit. The results are shown in Figure 2 and have been obtained by averaging over 20 random circuits of $(30\,n)$ gates on $n = 35$ qubits. The diminished advantage when p increases is not related to the different probability that a 2-qubit gate requires communication compared to a 1-qubit gate, in fact for CNOT gates communication is required if and only if the target is global irrespective of the control qubit. We attribute the diminished reduction to the fact that a 2-qubit gate (requiring communication) excludes a larger part of the remaining circuit from execution since it has, on average, more outgoing edges in the DAG description of the circuit (see condition in line 6 of Algorithm 1).

The simulation time has been estimated by summing up the time required to simulate every gate separately. We start by benchmarking the execution of each type of gate acting on every qubit or pair of qubits. We also benchmark the time to reorganize the classical data representing the quantum state following a call to `PermuteQubits`. This can be achieved by benchmarking the time to execute a single SWAP gate on every pair of qubits since each new permutation is related to the previous one by the composition with a single 2-cycle or transposition (see line 15 in Algorithm 1). Notice that, unlike from the circuit optimization that is based on the dicotomy local *vs.* global qubits, to estimate the simulation time of both the original and optimized circuits we consider the full information content provided by benchmarks like those in Figure. 1.

In the second study, we fix the probability of 2-qubit gates to $p = 0.3$ and vary the number of local and global qubits. This choice corresponds to about two 1-qubit gates every 2-qubit operation. In Figure 3 we report the number of gates causing communication overhead (here including also the qubit reordering operations) as a fraction of the total number of original operations. In the top panel, the horizontal axis represents the percentage of qubits that are global when the total number of qubits is $n = 50$. This simulation size is at the threshold of simulability, even considering using all computing nodes of current supercomputers. However, we show in the bottom panel that, apart from small-size effects for $n \leq 30$, the behavior is independent on n. This allows us to estimate the advantages of the optimization at extremely large scale (even beyond the simulability threshold expected around 50 qubits). Specifically, when 1/5 of the qubits are global, the fraction of gates requiring communication is reduced from $\sim 20\%$ to $\sim 3.5\%$ independently of the number of qubits. Assuming an overhead of about $8.5\times$ for the communication (see bottom panel of Figure 2), this would correspond to a reduction of simulation time by $1 - t_{\text{opt}}/t_{\text{orig}} = 1 - (0.965 + 0.035 \times 8.5)/(0.8 + 0.2 \times 8.5) \simeq 49.5\%$. Notice that our simple optimization may be improved, but with limited gain since even the impossible case of completely eliminating communication would lead to a reduction of $1 - t_{\text{no-comm}}/t_{\text{orig}} = 1 - 1/(0.8 + 0.2 \times 8.5) \simeq 60\%$ in this scenario.

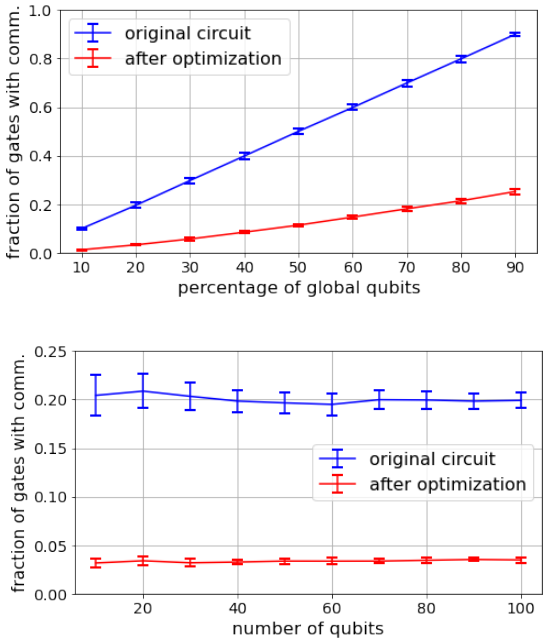

Figure 3: Effect of the compiler pass to optimize quantum circuits for simulation with IQS. Fraction of gates requiring communication for the original (blue) and optimized (red) circuit. **(Top)** Fixing $n = 50$, we vary the percentage of global qubits from 10% to 90%. **(Bottom)** Fixing the ratio global *vs.* total qubits to 1:5, we vary n. In both panels, datapoints are averaged over 20 random circuits and the vertical bars indicate one standard deviation. The random circuits have $(30\,n)$ gates.

The latter observation raises the question whether a $\sim 2\times$ speedup in simulation is an important benchmark. We believe so, especially considering that the typical use case of IQS, and of increasingly more simulators due to the progress of quantum hardware technology, are large-scale simulations running on hundreds or thousands of computing nodes for an extended period. Halving the simulation costs, not only the temporal cost but also the monetary and energy cost [21], with no additional user's overhead is certainly remarkable.

5 Conclusion and outlook

Quantum circuits are compiled for execution on actual quantum hardware, but an important feature of quantum computing frameworks is the possibility of simulating algorithms on classical computers. We propose a new compiler pass specialized for circuit simulation. Our results demonstrate that the co-development and integration between circuit compiler and simulator is largely beneficial and reduces the simulation time in half. The integration requires dedicated functions in both software programs: on one hand, we extend Intel Quantum Simulator, a high-performance, distributed and open-source simulator, with flexible data structures and qubit re-ordering functionalities; on the other hand, we propose a novel compiler pass based on the minimization of the number of operation requiring communication between computing nodes. We believe that compiler passes specialized for circuit simulation will become an essential component of every quantum computation framework.

Acknowledgements

The author acknowledges the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing HPC resources that have contributed to the research results reported in this work. http://www.tacc.utexas.edu The author thanks Fabio Baruffa, Justin Hogaboam, and Nicolas P.D. Sawaya for helpful discussions, and Salvatore Mandrà for comments on a previous version of the manuscript.

References

[1] Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoit Valiron. "Quipper: A scalable quantum programming language". Proceedings PLDI 13 **48**, 333–342 (2013).

[2] David Wecker and Krysta M. Svore. "LIQUi|⟩: A software design architecture and domain-specific language for quantum computing" (2014). arXiv:1402.4467.

[3] Damian S. Steiger, Thomas Häner, and Matthias Troyer. "ProjectQ: An Open Source Software Framework for Quantum Computing". Quantum **2**, 49 (2018).

[4] The Qiskit Contributors. "Qiskit: An open-source framework for quantum computing" (2019). https://github.com/Qiskit/qiskit.

[5] The Cirq Contributors. "Cirq, a python framework for creating, editing, and invoking noisy intermediate scale quantum (NISQ) circuits". https://github.com/quantumlib/Cirq.

[6] Robert S. Smith, Michael J. Curtis, and William J. Zeng. "A practical quantum instruction set architecture" (2016).

[7] Nathan Killoran, Josh Izaac, Nicolás Quesada, Ville Bergholm, Matthew Amy, and Christian Weedbrook. "Strawberry Fields: A Software Platform for Photonic Quantum Computing". Quantum **3**, 129 (2019).

[8] The Orquestra Contributors. "Orquestra". https://www.zapatacomputing.com/orquestra/.

[9] The Braket Contributors. "Braket". https://aws.amazon.com/braket/.

[10] Ali Javadiabhari, Shruti Patil, Daniel Kudrow, Jeff Heckey, Alexey Lvov, Frederic T. Chong, and Margaret Martonosi. "ScaffCC: A framework for compilation and analysis of quantum computing programs". CF '14 Proceedings of the 11th ACM Conference on Computing Frontiers (2014).

[11] Alexander Cowtan, Silas Dilkes, Ross Duncan, Alexandre Krajenbrink, Will Simmons, and Seyon Sivarajah. "On the qubit routing problem". Leibniz International Proceedings in Informatics, LIPIcs **135**, 5:1—-5:32 (2019).

[12] Andrew M. Childs, Eddie Schoute, and Cem M. Unsal. "Circuit transformations for quantum architectures". Leibniz International Proceedings in Informatics, LIPIcs **135**, 1–24 (2019).

[13] Carmen G. Almudever, Lingling Lao, Robert Wille, and Gian G. Guerreschi. "Realizing Quantum Algorithms on Real Quantum Computing Devices". Proceedings of the 2020 Design, Automation and Test in Europe Conference and Exhibition, DATE 2020Pages 864–872 (2020).

[14] Mikhail Smelyanskiy, Nicolas P. D. Sawaya, and Alán Aspuru-Guzik. "qHiPSTER: The quantum high performance software testing environment" (2016). arXiv:1601.07195.

[15] Edwin Pednault, John A. Gunnels, Giacomo Nannicini, Lior Horesh, Thomas Magerlein, Edgar Solomonik, Erik W. Draeger, Eric T. Holland, and Robert Wisnieff. "Breaking the 49-qubit barrier in the simulation of quantum circuits" (2017). arXiv:1710.05867.

[16] Hans De Raedt, Fengping Jin, Dennis Willsch, Madita Willsch, Naoki Yoshioka, Nobuyasu Ito, Shengjun Yuan, and Kristel Michielsen. "Massively parallel quantum computer simulator, eleven years later". Computer Physics Communications **237**, 47–61 (2019).

[17] Thomas Häner and Damian S. Steiger. "0.5 petabyte simulation of a 45-qubit quantum circuit". Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC 17 (2017).

[18] N. Khammassi, I. Ashraf, X. Fu, C.G. Almudever, and K. Bertels. "QX: A high-performance quantum computer simulation platform". Design,

Automation & Test in Europe Conference & Exhibition (DATE), 2017 (2017).

[19] Tyson Jones, Anna Brown, Ian Bush, and Simon C. Benjamin. "QuEST and high performance simulation of quantum computers". Scientific Reports**9** (2019).

[20] The Huawei HiQ Team. "Huawei hiq: A high-performance quantum computing simulator and programming framework". http://hiq.huaweicloud.com.

[21] Benjamin Villalonga, Dmitry Lyakh, Sergio Boixo, Hartmut Neven, Travis S. Humble, Rupak Biswas, Eleanor G. Rieffel, Alan Ho, and Salvatore Mandrà. "Establishing the quantum supremacy frontier with a 281 Pflop/s simulation". Quantum Science and Technology **5**, 034003 (2020).

[22] Thomas E. O'Brien, B. Tarasinski, and Leo DiCarlo. "Density-matrix simulation of small surface codes under current and projected experimental noise". npj Quantum Information **3**, 39 (2017).

[23] Benjamin Villalonga, Sergio Boixo, Bron Nelson, Christopher Henze, Eleanor Rieffel, Rupak Biswas, and Salvatore Mandrà. "A flexible high-performance simulator for verifying and benchmarking quantum circuits implemented on real hardware". npj Quantum Information **5**, 1–16 (2019).

[24] Thomas Häner, Damian S. Steiger, Krysta M. Svore, and Matthias Troyer. "A Software Methodology for Compiling Quantum Programs". Quantum Science and Technology **3**, 020501 (2018).

[25] Gian Giacomo Guerreschi, Justin Hogaboam, Fabio Baruffa, and Nicolas P. D. Sawaya. "Intel Quantum Simulator: A cloud-ready high-performance simulator of quantum circuits". Quantum Science and Technology **5**, 034007 (2020).

[26] Chapman, Peter. "Scaling IonQ's Quantum Computers: The Roadmap". https://ionq.com/posts/december-09-2020-scaling-quantum-computer-roadmap/.

[27] Santiago Rodrigo, Medina Bandic, Sergi Abadal, Hans van Someren, Eduard Alarcón, and Carmen G. Almudéver. "Scaling of multi-core quantum architectures". CF '21: Proceedings of the 18th ACM International Conference on Computing FrontiersPages 144–151 (2021).

[28] Thomas Häner, Damian S. Steiger, Torsten Hoefler, and Matthias Troyer. "Distributed Quantum Computing with QMPI". SC '21: Proceedings of the International Conference for High Performance Computing, Networking, Storage and AnalysisPage 16 (2021).

[29] Stephen Diadamo, Janis Nötzel, Benjamin Zanger, and Mehmet Mert Beşe. "QuNetSim: A Software Framework for Quantum Networks". IEEE Transactions on Quantum Engineering **2**, 2502512 (2021).

[30] Axel Dahlberg and Stephanie Wehner. "SimulaQron - A simulator for developing quantum internet software". Quantum Science and Technology **4**, 015001 (2019).

[31] K. De Raedt, K. Michielsen, H. De Raedt, B. Trieu, G. Arnold, M. Richter, Th. Lippert, H. Watanabe, and N. Ito. "Massively parallel quantum computer simulator". Computer Physics Communications **176**, 121–136 (2007).

[32] Yasunari Suzuki, Yoshiaki Kawase, Yuya Masumura, Yuria Hiraga, Masahiro Nakadai, Jiabao Chen, Ken M. Nakanishi, Kosuke Mitarai, Ryosuke Imai, Shiro Tamiya, Takahiro Yamamoto, Tennin Yan, Toru Kawakubo, Yuya O. Nakagawa, Yohei Ibe, Youyuan Zhang, Hirotsugu Yamashita, Hikaru Yoshimura, Akihiro Hayashi, and Keisuke Fujii. "Qulacs: a fast and versatile quantum circuit simulator for research purpose". Quantum **5**, 559 (2021).

[33] Frank et al. Arute. "Quantum supremacy using a programmable superconducting processor". Nature **574**, 505–510 (2019).

[34] Cupjin Huang, Fang Zhang, Michael Newman, Junjie Cai, Xun Gao, Zhengxiong Tian, Junyin Wu, Haihong Xu, Huanjun Yu, Bo Yuan, Mario Szegedy, Yaoyun Shi, and Jianxin Chen. "Classical Simulation of Quantum Supremacy Circuits" (2020). arXiv:2005.06787.

[35] Edwin Pednault, John A. Gunnels, Giacomo Nannicini, Lior Horesh, and Robert Wisnieff. "Leveraging Secondary Storage to Simulate Deep 54-qubit Sycamore Circuits" (2019).

[36] Xiu-Zhe Luo, Jin-Guo Liu, Pan Zhang, and Lei Wang. "Yao.jl: Extensible, Efficient Framework for Quantum Algorithm Design". Quantum **4**, 341 (2020).

[37] Gian Giacomo Guerreschi and Jongsoo Park. "Two-step approach to scheduling quantum circuits". Quantum Science and Technology **3**, 045003 (2018).

[38] Lingling Lao, Hans Van Someren, Imran Ashraf, and Carmen G. Almudever. "Timing and Resource-Aware Mapping of Quantum Circuits to Superconducting Processors". IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **41**, 359–371 (2022).

[39] Toshinari Itoko, Rudy Raymond, Takashi Imamichi, and Atsushi Matsuo. "Optimization of quantum circuit mapping using gate transformation and commutation". Integration **70**, 43–50 (2020).

[40] Seyon Sivarajah, Silas Dilkes, Alexander Cowtan, Will Simmons, Alec Edgington, and Ross Duncan. "t|ket⟩: A Retargetable Compiler for NISQ

Devices". Quantum Science and Technology **6**, 014003 (2021).

[41] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. "A quantum approximate optimization algorithm" (2014). arXiv:1411.4028.

[42] Y. Salathé, M. Mondal, M. Oppliger, J. Heinsoo, P. Kurpiers, A. Potočnik, A. Mezzacapo, U. Las Heras, Lucas Lamata, Enrique Solano, S. Filipp, and Andreas Wallraff. "Digital quantum simulation of spin models with circuit quantum electrodynamics". Physical Review X **5**, 021027 (2015).

A Appendix A: Transformation between two qubit permutations

Consider the current qubit permutation σ such that amplitude α_i of the computational basis state $|i_{n-1}\rangle \dots |i_1\rangle |i_0\rangle$ corresponds to the component $j = \sum_{q=0}^{n-1} i_q 2^{\sigma(q)}$ of the state vector. We want to update the qubit permutation to σ' taking into account that m of the n qubits are local and the remaining $n-m$ are global. The IQS method implementing the update proceeds in three steps: The first one reorders local qubits only, the second one reorders global qubits only, and the third step exchanges pairs formed by one local and one global qubits.

The three steps have separate implementations: reordering local qubits is obtained by copying and rearranging data in each local memory independently, reordering global qubits is obtained by swapping the local memory of two distinct processes, and exchanging a local and a global qubit corresponds to the execution of a SWAP gate from the point of view of data movement. A SWAP-like gate is implemented via the same communicate-compute-communicate approach of controlled 1-qubit gates described in [14], but with a different pattern of inter-process communication.

Below, we provide pseudocode to divide the transformation $\sigma \mapsto \sigma'$ in three transformations that, composed sequentially, correspond to the desired one. For clarity of terminology, we will say that qubit q is mapped to position $\sigma(q)$ and, viceversa, that position p is mapped to qubit $\sigma^{-1}(p)$. Inverting the permutation is efficient and takes $\mathcal{O}(n)$ operations. Algorithm 2 constructs the intermediate permutations such that $\sigma \mapsto \sigma_1 \mapsto \sigma_2 \mapsto \sigma'$ satisfies the criteria expressed above.

Algorithm2 Decompose transformation $\sigma \mapsto \sigma'$

Three steps: $\sigma \mapsto \sigma_1 \mapsto \sigma_2 \mapsto \sigma'$
1: $\sigma \leftarrow$ current qubit permutation
2: $\sigma' \leftarrow$ desired qubit permutation
3: $\sigma_1 \leftarrow$ identity permutation
4: $\sigma_2 \leftarrow$ identity permutation
5: **for** $p = 0, 1, \dots, m-1$ **do**
6: $\quad p' \leftarrow \sigma'(\sigma^{-1}(p))$
7: $\quad$ **while** $p' \geq m$ **do**
8: $\quad\quad p' \leftarrow \sigma'(\sigma^{-1}(p'))$
9: $\quad$ **end while**
10: $\quad \sigma_1^{-1}(p') \leftarrow \sigma^{-1}(p)$
11: **end for**
12: **for** $p = m, m+1, \dots, n-1$ **do**
13: $\quad p' \leftarrow \sigma'(\sigma_1^{-1}(p))$
14: $\quad$ **while** $p' < m$ **do**
15: $\quad\quad p' \leftarrow \sigma'(\sigma_1^{-1}(p'))$
16: $\quad$ **end while**
17: $\quad \sigma_2^{-1}(p') \leftarrow \sigma_1^{-1}(p)$
18: **end for**

B Appendix B: Implement optimization pass in existing compiler pipelines

In Algorithm 1 of the main text, we provide the pseudocode for the optimization pass suggested in this work. While we do not provide an explicit implementation, we would like to comment on how it could be integrated in existing compiler pipelines with minimal modifications.

From the pseudocode it should be clear that the effect of "**add** `PermuteQubits`(σ) to C" in line 25 is to change the qubit order by swapping only two qubits, one local and the other global. Therefore, one can introduce two separate instructions in the compiler pipeline: SWAP and Emulated-SWAP. While SWAP corresponds to the usual 2-qubit gate (this instruction is typically present in any quantum compiler), Emulated-SWAP can be seen as the same operation executed in two different ways: For hardware backends it corresponds to a usual SWAP, for Intel Quantum Simulator backend it updates the qubit order without data movement.

At this point, "**add** `PermuteQubits`(σ) to C" corresponds to adding a SWAP instruction followed by an Emulated-SWAP on the same pair of qubits. The intermediate representation of the quantum circuit is easy to interpret since Emulated-SWAP can be treated as any other 2-qubit gate and, if needed, the code can also run for hardware backends or other simulators.

C Appendix C: Reduction of the simulation time for QFT-like circuits

In the main text we applied the proposed compiler pass to random circuits. Being part of the compiler pipeline, the pass can be applied to any quantum circuit without additional effort by the user writing the quantum algorithm. However, its effectiveness depends on the pattern of 2-qubit gates. We expect a large reduction in simulation time for circuits which can be divided in a small number of subcircuits, each involving only a fraction of the total qubits. This would allow to permute the qubits not involved in the subcircuit at most once during the subcircuit, making them global qubits if they were local and avoiding any further permutation involving them during the subcircuit.

For our study in Section **Numerical results**, we chose circuits with 2-qubit gates involving qubits chosen uniformly at random. In this case, it is difficult to identify the subcircuit structure discussed above and therefore we believe that the random circuit benchmarks reflect unfavorable situations.

To confirm this intuition, we consider the opposite scenario of circuits presenting a clear pattern of 2-qubit gates. As an example, we analyze QFT-like circuits (QFT stays for Quantum Fourier Transform) in which the controlled-Rz gates are substituted by CNOT gates. The reason behind the substitution is that controlled-Rz gates are diagonal in the computational basis and do not require communication, while CNOT gates do require communication as benchmarked in Figure 1. The circuits are of the form illustrated in Figure 4 below.

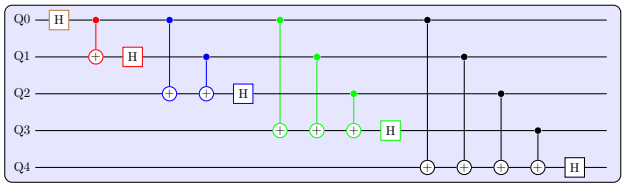

Figure 4: Quantum circuit with the same structure as the Quantum Fourier Transform circuit, but with CNOT gates substituting controlled-Rz rotations.

With the trivial qubit ordering in which low-index qubits are local and high-index qubits are global, one can compute the number of gates requiring communication as a function of the number of qubits n and the fraction of local qubits m/n. Additionally, one can observe that at most $2(n-m)$ qubit permutations are needed to eliminate all communication overhead (two permutations for each of the same-color gates with Hadamard on a global qubit). Figure 5 clarifies that the pass we propose is particularly effective for QFT-like circuits.

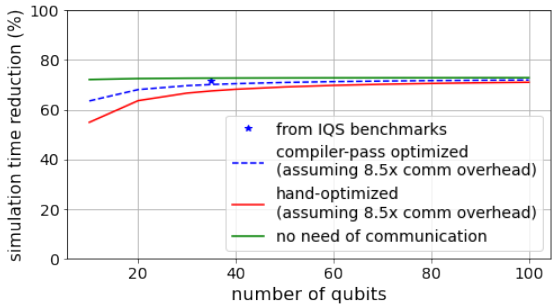

Figure 5: Effect of the compiler pass to optimize QFT-like quantum circuits for simulation with IQS. The blue star uses the benchmark results from Figure 1, while the dashed blue line uses a simplified model in which operations requiring communication have an overhead of 8.5x with respect to operations implementable locally (see caption of Figure 2). The green line represents the limiting case in which no operation would require communication. For all circuit sizes, it has been assumed that 20% of the qubits are global.