Article

Large-Scale Simulation of Shor's Quantum Factoring Algorithm

Dennis Willsch [1,*], Madita Willsch [1,2], Fengping Jin [1], Hans De Raedt [1,3] and Kristel Michielsen [1,2,4]

1 Jülich Supercomputing Centre, Institute for Advanced Simulation, Forschungszentrum Jülich, 52425 Jülich, Germany; m.willsch@fz-juelich.de (M.W.); f.jin@fz-juelich.de (F.J.); deraedthans@gmail.com (H.D.R.); k.michielsen@fz-juelich.de (K.M.)
2 AIDAS, 52425 Jülich, Germany
3 Zernike Institute for Advanced Materials, University of Groningen, Nijenborgh 4, 9747 AG Groningen, The Netherlands
4 Department of Physics, RWTH Aachen University, 52056 Aachen, Germany
* Correspondence: d.willsch@fz-juelich.de

Abstract: Shor's factoring algorithm is one of the most anticipated applications of quantum computing. However, the limited capabilities of today's quantum computers only permit a study of Shor's algorithm for very small numbers. Here, we show how large GPU-based supercomputers can be used to assess the performance of Shor's algorithm for numbers that are out of reach for current and near-term quantum hardware. First, we study Shor's original factoring algorithm. While theoretical bounds suggest success probabilities of only 3–4%, we find average success probabilities above 50%, due to a high frequency of "lucky" cases, defined as successful factorizations despite unmet sufficient conditions. Second, we investigate a powerful post-processing procedure, by which the success probability can be brought arbitrarily close to one, with only a single run of Shor's quantum algorithm. Finally, we study the effectiveness of this post-processing procedure in the presence of typical errors in quantum processing hardware. We find that the quantum factoring algorithm exhibits a particular form of universality and resilience against the different types of errors. The largest semiprime that we have factored by executing Shor's algorithm on a GPU-based supercomputer, without exploiting prior knowledge of the solution, is 549,755,813,701 = 712,321 × 771,781. We put forward the challenge of factoring, without oversimplification, a non-trivial semiprime larger than this number on any quantum computing device.

Keywords: quantum computing; quantum algorithms; Shor's factoring algorithm; high performance computing; computer simulation; parallelization

MSC: 81P68; 68Q12; 11A51

check for updates

Citation: Willsch, D.; Willsch, M.; Jin, F.; De Raedt, H.; Michielsen, K. Large-Scale Simulation of Shor's Quantum Factoring Algorithm. *Mathematics* **2023**, *11*, 4222. https://doi.org/10.3390/math11194222

Academic Editor: Artur Czerwinski

Received: 24 August 2023
Revised: 29 September 2023
Accepted: 7 October 2023
Published: 9 October 2023

1. Introduction

The challenge of factoring integers is one of the oldest problems in mathematics [1,2]. Famous mathematicians such as Fermat, Euler, and Gauss have made substantial contributions to the problem, and even algorithms discovered by the ancient Greeks—the Euclidean algorithm and the sieve of Eratosthenes—are still in use today. The state-of-the-art algorithms are based on the general number field sieve [3] and have recently achieved the factorization of RSA-250 from the famous RSA factoring challenge [4]. Still, all known algorithms exhibit at best subexponential time and space complexity [4,5]. The difficulty of solving this type of problem using classical computers is an integral aspect of modern data and communication security [6,7].

In 1994, Peter Shor proposed an algorithm to factor integers on quantum computers with an exponential speedup [8–10] over the best known classical algorithms. Factoring an L-bit integer N with the conventional Shor algorithm [10] requires at least $3L$ qubits: $L = \lfloor \log_2 N \rfloor + 1$ qubits to represent N, and $t = \lceil 2 \log_2 N \rceil \approx 2L$ qubits for the Quantum

Fourier Transform (QFT), plus $O(L)$ qubits for the modular exponentiation [6,11]. Kitaev, Griffiths, and Niu realized that by replacing the QFT with a semiclassical Fourier transform, only a single qubit can be reused t times to obtain the same result [12,13] (also known as *qubit recycling* [14,15] or *dynamic quantum computing* [16]). It is thus possible to run Shor's algorithm with only $L+1$ qubits to factor L-bit integers (which is less than required by the best adiabatic algorithm [17,18]). We refer to this variant as the *iterative Shor algorithm*.

The iterative Shor algorithm has been executed on real quantum computing devices to factor 15, 21, and 35 [15,19,20], without relying on oversimplification [21]. Implementing the algorithm for integers beyond 35 continues to pose substantial experimental challenges [6,22].

To study the performance of Shor's algorithm for much larger integers than those feasible for testing on real quantum devices, we have developed a massively parallel simulator called `shorgpu` [23], specifically designed to execute the iterative Shor algorithm on multiple GPUs. Using `shorgpu`, we have examined over 60,000 factoring scenarios for integers up to $N_{\max}$ = 549,755,813,701, significantly surpassing previous achievements using statevector simulators [24–26], matrix product states [27,28] (in [28], the authors simulated 60 qubits to factor N = 961,307), and tensor networks [29,30]. Note that $N_{\max}$ is still "small" for cryptographic purposes. In order to handle integers of the size of $N_{\max}$, `shorgpu` uses a new technique (see Section 2) to perform the distributed memory communications.

To factor $N_{\max}$, the conventional Shor algorithm would need 117 qubits. The iterative Shor algorithm, however, needs only 40 qubits. It is important to note that the resulting quantum algorithm is still an honest implementation of Shor's algorithm: it produces the same results, does not require exponentially large classical resources (given a large enough quantum computer) and, most importantly, does not exploit a priori knowledge of the factors [21].

We emphasize that for all our simulations, we do not require the solution of the factoring problem to be known. If one presumes knowledge of the solution, and one is not interested in simulating the effect of quantum errors, it is possible to study even larger, cryptographically relevant cases using Qunundrum [31].

The procedure used to factor integers is shown in Figure 1: first, a factoring problem is selected, consisting of a semiprime $N = p \times q$ to factor and a random integer $1 < a < N$ comprime to N (i.e., $\gcd(a, N) = 1$). Using this as input, `shorgpu` executes the iterative Shor algorithm with $n = L + 1$ qubits to produce several bitstrings. Each bitstring j is processed using either Shor's [8–10,32] or Ekerå's [33,34] post-processing method, which may or may not produce a factor of N (see the yellow section in Figure 1). An important step on the way is to extract a candidate r for the *order* $\hat{r} = \mathrm{ord}_N(a)$. Here, $\mathrm{ord}_N(a)$ denotes the order of a modulo N, defined as the smallest exponent $\hat{r} > 0$ such that $a^{\hat{r}} \bmod N = 1$.

Note that "success" is not guaranteed by Shor's algorithm; in particular, the sampled bitstring might produce an $r \neq \hat{r}$ that is not the order, or r might be odd, in which case Shor's post-processing method is not guaranteed to work. However, if the blind application of Shor's factoring procedure still yields at least one factor, we count this execution as a "lucky" case. As shown below, a "lucky" factor is found much more often than expected.

In principle, the green section in Figure 1 representing `shorgpu` can be completely replaced by a sufficiently large, error-corrected quantum computing device. With this in mind, we put forward the challenge of *indirect quantum supremacy* [35] (a.k.a. *limited quantum speedup* [36]) for a future quantum computer. Here, "indirect" means that the simulator (running on a conventional computer) is required to simulate an ideal quantum computational model that executes the same quantum algorithm as the quantum computer, without using any prior knowledge of the solution. More specifically, the challenge for a gate-based quantum computer would be to factor, using Shor's algorithm without oversimplification [21], an "interesting" semiprime—where "interesting" means that the two distinct prime factors shall have the same number of digits—that is larger than the largest semiprime that can be factored by the quantum computer simulator.

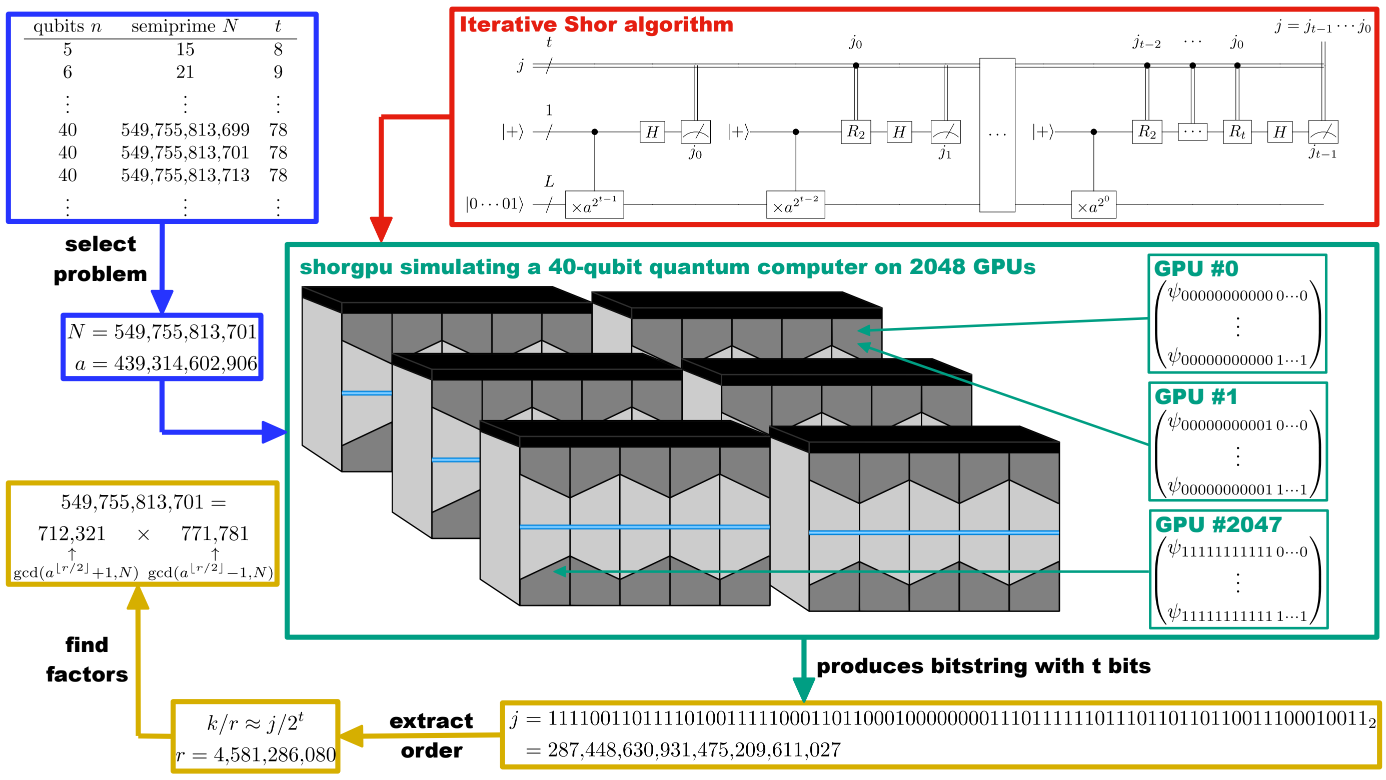
qubits n | semiprime N | t
5 | 15 | 8
6 | 21 | 9
40 | 549,755,813,699 | 78
40 | 549,755,813,701 | 78
40 | 549,755,813,713 | 78
select problem
$N = 549{,}755{,}813{,}701$
$a = 439{,}314{,}602{,}906$
Iterative Shor algorithm
$j = j_{t-1} \cdots j_0$
$|+\rangle$
$|0\cdots01\rangle$
$\times a^{2^{t-1}}$
$\times a^{2^{t-2}}$
$\times a^{2^0}$
H
R_2
R_t
shorgpu simulating a 40-qubit quantum computer on 2048 GPUs
GPU #0
GPU #1
GPU #2047
produces bitstring with t bits
$j = 1111001101111101001111100011011000100000000111011111110111011011011001110001001 1_2$
$= 287{,}448{,}630{,}931{,}475{,}209{,}611{,}027$
extract order
$k/r \approx j/2^t$
$r = 4{,}581{,}286{,}080$
find factors
$549{,}755{,}813{,}701 = 712{,}321 \times 771{,}781$
$\gcd(a^{\lfloor r/2 \rfloor}+1, N)$ $\gcd(a^{\lfloor r/2 \rfloor}-1, N)$

Figure 1. Scheme to test Shor's algorithm. After selecting an L-bit semiprime $N = p \times q$ to factor and a random integer $1 < a < N$ coprime to N (blue), a quantum computer or quantum computer simulator with $n = L + 1$ qubits runs the iterative Shor algorithm (red) and produces several bitstrings j with t bits (green). Here, "iterative" means that one qubit is measured and reused t times to produce the t classical bits of each j. Every bitstring j is analyzed using either Shor's [8–10,32] or Ekerå's [33,34] post-processing method (yellow), independent of whether certain algorithmic requirements on j are satisfied or not. Here, k (r) denotes the numerator (denominator) obtained from a continued fraction expansion of $j/2^t$. Note that the expression for the factors in the yellow section is specific to Shor's post-processing; for Ekerå's post-processing, $\gcd(a^{\lfloor r/2 \rfloor} \pm 1, N)$ has to be replaced by $\gcd(x_i^{r_i} - 1, N)$, where $x_i \neq a$ is a random element of $\mathbb{Z}_N^*$ and r_i is usually a multiple of the largest odd divisor of r (see below). We remark that conceptually, it does not matter whether the green section is performed by a quantum computer simulator or a real quantum computing device.

1.1. Related Work

There is a large body of literature on Shor's quantum factoring algorithm; the related work can be roughly classified into five main categories. In this section, we give a survey of their main goals and discuss several individual results.

1. Theory: the first class of articles focuses on theoretical perspectives such as algorithmic modifications and improved lower bounds on the success probability [11,14,37–60], many of which consider the case that some parameters of Shor's algorithm are modified and certain trade-offs are made. This class contains work that estimates the number of resources required when using different levels of quantum computer technology [6,22]. This line of work culminates in Ekerå's post-processing algorithms [34], by which the success probability for a single run of the quantum part can be brought arbitrarily close to one (see below).
2. Simulation: second, Shor's algorithm has been studied by using simulators running on conventional computers. Some use universal quantum computer simulators [24–26,61], sometimes also called Schrödinger simulators, since they propagate the full quantum statevector. Another approach is to use so-called Feynman simulators, which can only access certain amplitudes from the full statevector, but may require fewer computational

resources; they are often based on tensor networks or matrix product states [27–30]. Finally, there is software designed to directly sample from the probability distributions generated by Shor's algorithm (cf. Equation (26) below) and various extensions thereof. This class contains the suite of programs called Qunundrum [31,62], which can simulate distributions for large, cryptographically relevant cases. Note, however, that the solution to the factoring problem (i.e., the order or the discrete logarithm) must be known in advance, and the effect of errors in the quantum part cannot be simulated.

3. Alternative Algorithms: a third line of work studies alternative ways to use gate-based quantum computers to solve the factoring problem. Some of them use Shor's discrete logarithm quantum algorithm [62–64], which is also an instance of the hidden subgroup problem [65]. In the Ekerå-Håstad scheme [63], the idea to factor a semiprime $N = pq$ is to pick a random $g \in \mathbb{Z}_N^*$, compute $y = g^{N+1} \bmod N$ with unknown order r, and then obtain $d \equiv \log_g y \equiv pq + 1 \equiv pq + 1 - \phi(N) \equiv p + q \pmod{r}$ (using that $r \mid \phi(N) = (p-1)(q-1)$ [66]). If $r > p + q$ (which is the case for many g), we have $d = p + q$, and additionally knowing $N = pq$ allows one to compute p and q. Another alternative way to solve the factoring problem is given in [67] and is based on the classical number field sieve [3]. In particular, Bernstein et al. propose to use Grover's quantum search algorithm [68] (and/or Shor's algorithm for a much smaller subproblem) to accelerate the number field sieve. This proposal is asymptotically worse in time complexity than using Shor's algorithm directly, but it requires fewer qubits and is therefore possibly easier to realize in near-term physical devices. Finally, Li et. al. [69] presented an algorithm with an exponential speedup (beyond the framework of the hidden subgroup problem [65]) that solves the square-free decomposition problem—a problem related to factoring in which the task is to find, for any integer $N > 0$, the unique integers N_r and N_s^2 of the square-free decomposition $N = N_r N_s^2$.
4. Gate-Based Experiments: fourth, there have been several experimental efforts to implement Shor's factoring algorithm on existing gate-based quantum computer devices [15,19,20,70–76]. However, many of these have made use of prior knowledge about the factors to simplify the experimental setup [21]. In the extreme case (namely when a base $a \in \mathbb{Z}_N^*$ with order $\mathrm{ord}_N(a) = 2$ is used), this reduces the computational problem to the equivalent of flipping coins. Experiments that have not used such an oversimplified method can be found in [15,19,20].
5. Other Experiments: finally, quantum annealers and adiabatic quantum computers have been used to study alternative factoring algorithms [17,77–82]. The quantum annealing approach requires at most $O(L^2)$ qubits to factor an L-bit number. Quantum annealing and adiabatic quantum computation are technologically significantly ahead of gate-based quantum computing, in that larger quantum processing units with more than 5000 qubits exist and that they can solve much larger problems [83,84]. In particular, numbers up to and above 200,000 have been factored on the D-Wave 2000Q [78,79] and 1,005,973 has been factored using D-Wave hybrid [80]. Although significantly larger than the numbers factored on gate-based quantum computers (without oversimplification), these numbers are still much smaller than $N_{\max}$ = 549,755,813,701 factored in this work using `shorgpu`.

1.2. Outline

This paper is structured as follows. In Section 2, we describe the algorithmic details of `shorgpu`. In particular, we explain how to implement the modular multiplication as a systematic communication scheme between the compute nodes. In Section 3, we present our results from over 60,000 quantum computer simulations using up to 2048 GPUs. Section 4 contains our conclusions.

2. Materials and Methods

For almost all results reported in this work, we use `shorgpu` to simulate the iterative Shor algorithm (the source code is available online [23]). It propagates an $n = L + 1$-

qubit statevector $|\psi\rangle$ consisting of 2^{L+1} complex numbers through the quantum circuit for factoring an L-bit number shown in Figure 2. Each step in the quantum circuit corresponds to an operation on $|\psi\rangle$. In this section, we describe each of these operations in a linear algebraic context. The probability distribution generated by Shor's algorithm is derived and visualized in Appendix A.

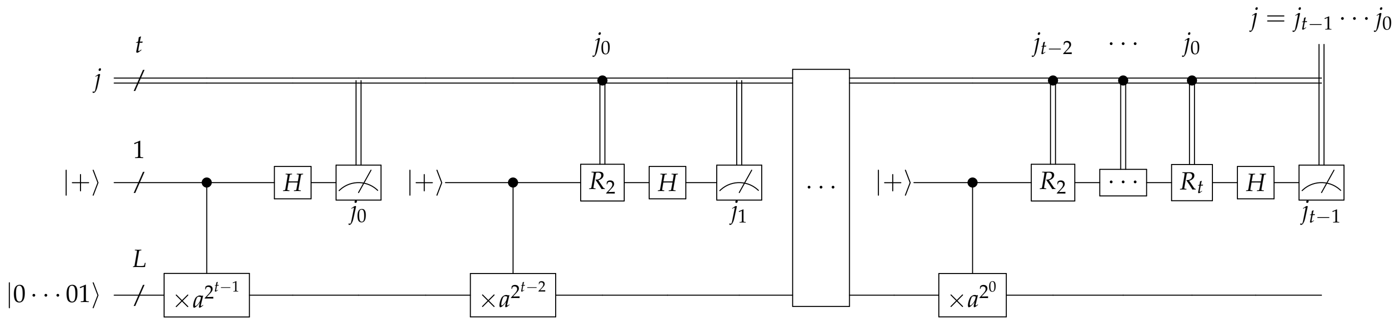

Figure 2. Quantum circuit of the iterative Shor algorithm. The circuit consists of $L+1$ qubits that undergo t separate stages $\texttt{cbit} = 0, \ldots, t-1$, in which the classical bit $j_{\texttt{cbit}}$ is measured. Each stage starts with the first qubit in the initial state $|+\rangle$ and ends with this qubit being measured (middle row). Between initialization and measurement, each stage consists of a controlled modular multiplication (bottom row) with some power of a (see Equation (6)), then a rotation gate controlled by all previously measured classical bits (see Equation (8)), and finally a Hadamard gate. The resulting bit $j_{\texttt{cbit}}$ is used to assemble the classical bitstring $j = j_{t-1} \cdots j_0$ (top row).

As the total memory is the bottleneck of such statevector simulations, the 2^{L+1} complex numbers $|\psi\rangle$ are distributed over the memory of up to 2048 GPUs (cf. Figure 1). Communication between the GPUs is managed using the Message Passing Interface (MPI) [85].

We use the Jülich Universal Quantum Computer Simulator (JUQCS) [24,25,86] for verification. JUQCS was previously used to simulate the conventional Shor algorithm for $N \leq 65{,}531$ with up to $n = 48$ qubits on the Sunway TaihuLight and the K computer [25]. A few features had to be added to JUQCS to be able to also simulate the iterative Shor algorithm. The latter made it possible to simulate one bitstring for N = 4,194,293 with $n = 23$ qubits in about 720 s (using four A100 GPUs). However, our JUQCS implementation of the oracle which performs the modular exponentiation becomes highly inefficient as the number n of qubits increases because it does not distribute well over many cores or GPUs. In contrast, the new, dedicated algorithm described below generates a bitstring in about 0.4 s for the same problem and the same number of GPUs. For the largest problem simulated (N = 549,755,813,701 with $n = 40$ qubits), `shorgpu` generates a bitstring in about 200 s using 2048 GPUs. We verified that the iterative Shor algorithm simulated with `shorgpu` produces the same results as JUQCS for problems of the size that can be simulated with JUQCS.

2.1. Initialization

To simulate the iterative Shor algorithm for factoring an L-bit semiprime N, `shorgpu` simulates the full quantum circuit with $n = L+1$ qubits shown in Figure 2. This is conducted by computing all complex coefficients of the statevector

$$|\psi\rangle = \sum_{k_L \cdots k_0 = 0,1} \psi_{k_L \cdots k_0} |k_L \cdots k_0\rangle = \begin{pmatrix} \psi_{0\cdots 00} \\ \psi_{0\cdots 01} \\ \vdots \\ \psi_{1\cdots 11} \end{pmatrix}. \tag{1}$$

These 2^{L+1} complex double-precision numbers $\psi_{k_L \cdots k_0}$ are distributed over $N_{\mathrm{GPU}} \in \{2, 4, 8, \ldots, 2048\}$ GPUs. The distributed memory communication between the GPUs uses CUDA-aware MPI. In our approach, each GPU is identified by its MPI rank, i.e., an n_{global}-bit integer called $\texttt{mpi\_rank} = 0, \ldots, N_{\mathrm{GPU}} - 1$. Here, n_{global} denotes the number of so-called *global qubits* (see [24,25,86]). We have $N_{\mathrm{GPU}} = 2^{n_{\mathrm{global}}}$ GPUs. The other $n_{\mathrm{local}} = n - n_{\mathrm{global}}$ qubits are called *local qubits*, since each GPU holds in its local memory all $2^{n_{\mathrm{local}}}$ complex coefficients

$$\begin{pmatrix} \psi_{\mathrm{bin}(\texttt{mpi\_rank})0\cdots00} \\ \psi_{\mathrm{bin}(\texttt{mpi\_rank})0\cdots01} \\ \vdots \\ \psi_{\mathrm{bin}(\texttt{mpi\_rank})1\cdots11} \end{pmatrix}. \tag{2}$$

The GPUs (i.e., the MPI processes) are further divided into two separate groups, identified by the most significant bit of the MPI rank,

$$\mathrm{bin}(\texttt{mpi\_rank}) = \texttt{mpi\_x}\,\mathrm{bin}(\texttt{mpi\_xrank}). \tag{3}$$

Here, $\texttt{mpi\_x} = 0, 1$ identifies the group and $0 \leq \texttt{mpi\_xrank} < N_{\mathrm{GPU}}/2$ identifies the GPU within each group. Thus, `shorgpu` requires at least two GPUs to work (unless a single GPU is used with 2 MPI processes in overscheduling mode). The reason for the separation into two groups is that the implementation of the controlled modular multiplication gate (see below) requires an all-to-all communication between all GPUs with $\texttt{mpi\_x} = 1$.

At the start of the simulation, the statevector $|\psi\rangle$ is initialized in the state $|+\rangle\,|0\cdots01\rangle$, where $|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$. This means that we set

$$\psi_{00\cdots01} = \frac{1}{\sqrt{2}}, \tag{4}$$

$$\psi_{10\cdots01} = \frac{1}{\sqrt{2}}, \tag{5}$$

and all other coefficients to zero. This type of initialization is always used unless `shorgpu` is used to assess the effect of quantum initialization errors (for information on this mode, see Section 2.6 below).

2.2. Controlled Modular Multiplication Gate

The first gate in each of the t stages in Figure 2 is the controlled modular multiplication gate (also called *oracle gate*), controlled by the first qubit. Mathematically, its operation is defined by

$$CU_{\texttt{a}}\,|x\rangle\,|y\rangle = \begin{cases} |0\rangle\,|y\rangle & (x = 0) \\ |1\rangle\,|\texttt{a}y \bmod N\rangle & (x = 1 \text{ and } 0 \leq y < N), \\ |1\rangle\,|y\rangle & (x = 1 \text{ and } N \leq y) \end{cases} \tag{6}$$

where x denotes the first qubit, $y = 0, \ldots, N_{\mathrm{GPU}}/2$ denotes the other qubits, and $\texttt{a} \in \{a^{2^{t-1}} \bmod N, a^{2^{t-2}} \bmod N, \ldots, a\}$ stands for one of the powers of a in Figure 2. Note that each individual modular exponentiation is always precomputed for *any* realization of this circuit (we use the shift-and-multiply algorithm). This is independent of whether the circuit is executed by a quantum computer simulator or a real quantum computer (see also [6,19,20]).

Looking at Equation (6), we see that the oracle gate performs a permutation of all complex coefficients among the GPUs in the $\texttt{mpi\_x} = 1$ group. `shorgpu` implements this unitary operation by computing, on each GPU, all indices of the coefficients that are sent to other GPUs (stored in a GPU buffer `oracle_idxsend`) and those that are received from other GPUs (stored in a GPU buffer `oracle_idxrecv`) using the precomputed modular inverse $\texttt{ainv} = \texttt{a}^{-1} \bmod N$, which is efficiently computable using the extended Euclidean algorithm. The MPI communication scheme for an example with 16 GPUs is shown in Figure 3.

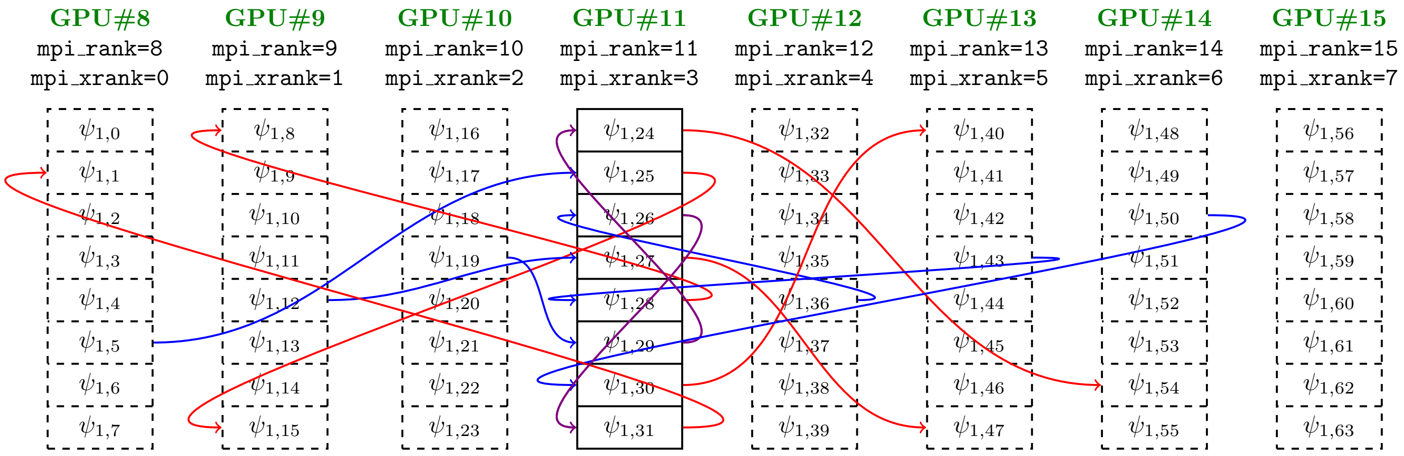

Figure 3. Illustration of the MPI communication scheme for the implementation of the controlled modular multiplication gate given by Equation (6) for the quantum circuit to factor $N = 55 = 5 \times 11$ with $a = 16$. This circuit needs $n = 7$ qubits, i.e., the first qubit for the measurement (middle line in Figure 2) and $L = 6$ qubits to represent N. There are $N_{\text{GPU}} = 16$ GPUs in this example, so we have $n_{\text{global}} = 4$ global and $n_{\text{local}} = 3$ local qubits (note that this is only for illustration purposes; in practice one would use much fewer GPUs for 7 qubits). Shown is the implementation of the last oracle gate in Figure 2 (the controlled modular multiplication with $\mathtt{a} = a^{2^0} = 16$) from the perspective of GPU#11. $\psi_{x,y}$ denotes, in the notation of Equation (1), the statevector coefficient $\psi_{x\,\text{bin}(y)}$. Red arrows represent coefficients that are sent from GPU#11 to another GPU (whose index is computed from $\mathtt{a}y \bmod N$). Blue arrows represent coefficients that are sent to GPU#11 from another GPU (whose index is computed from $\mathtt{a}^{-1}y \bmod N$). Purple arrows represent coefficients that stay on GPU#11. Note that GPU#15 is not involved in the communication because $N \leq y$ for all $\psi_{x,y}$ of GPU#15, so the oracle gate does not permute these coefficients (the last case in Equation (6)).

The complexity of the permutation depends on the value of a. For instance, in the special case that the order of a is a small power of 2, we have $\mathtt{a} = 1$ in many of the early stages, so the oracle gate would not require MPI communication between different GPUs. In general the communication scheme can be very complicated. Figure 3 shows a typical instance where each GPU sends (red arrows) and receives (blue arrows) some coefficients from other GPUs.

To implement this communication scheme between the GPUs, `shorgpu` uses non-blocking point-to-point communication in a circular fashion using `MPI_Isend` and `MPI_Irecv`. Additionally, before the send operations, each GPU first arranges all coefficients that are sent to a particular other GPU in a contiguous block of memory, schematically denoted by $\psi^{(\text{cont})}$. This is imperative since for large N, this part of the simulation takes a significant fraction of the total run time. Alternative implementations using one-sided communication such as `MPI_Put`, collective communication using `MPI_Alltoallv`, or communication based on custom MPI data types (see [85] for more information) performed significantly worse in our experiments.

2.3. Rotation Gate

After the oracle gate, each stage (except the first stage) of the quantum circuit in Figure 2 contains a sequence of rotation gates defined by

$$R_l = \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^l} \end{pmatrix}. \tag{7}$$

These are controlled by bits resulting from previous measurements. Specifically, at the stage $\texttt{cbit} = 0, \ldots, t-1$, in which the classical bit $j_{\texttt{cbit}}$ is being measured, the sequence of these controlled rotation gates reads

$$\prod_{l=2}^{1+\texttt{cbit}} CR_l = \prod_{l=2}^{1+\texttt{cbit}} \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i j_{1+\texttt{cbit}-l}/2^l} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\varphi_{\texttt{cbit}}} \end{pmatrix}, \tag{8}$$

where the phase $\varphi_{\texttt{cbit}}$ at stage $\texttt{cbit}$ amounts to

$$\varphi_{\texttt{cbit}} = 2\pi \sum_{l=2}^{1+\texttt{cbit}} \frac{j_{1+\texttt{cbit}-l}}{2^l} = \frac{\pi j^{(\texttt{cbit})}}{2^{\texttt{cbit}}}, \tag{9}$$

and $j^{(\texttt{cbit})} = j_{\texttt{cbit}-1} j_{\texttt{cbit}-2} \cdots j_1 j_0$ is the integer assembled from all classical bits measured up to this point.

As the phase gate given by Equation (8) only affects coefficients $\psi_{k_L \cdots k_0}$ where the first qubit index $k_L = 1$, this operation only needs to be implemented by the GPUs in the $\texttt{mpi\_x} = 1$ group. This is performed directly after the implementation of the oracle gate, when moving all coefficients out of the contiguous memory blocks $\psi^{(\text{cont})}$, according to

$$\begin{aligned} &\text{Re}(\psi_{1\,\text{bin}(\texttt{mpi\_xrank})\,*\cdots*}) \\ &\qquad \leftarrow \cos(\varphi_{\texttt{cbit}})\text{Re}(\psi^{(\text{cont})}_{1\,\text{bin}(\texttt{mpi\_xrank})\,*\cdots*}) \\ &\qquad - \sin(\varphi_{\texttt{cbit}})\text{Im}(\psi^{(\text{cont})}_{1\,\text{bin}(\texttt{mpi\_xrank})\,*\cdots*}), \end{aligned} \tag{10}$$

$$\begin{aligned} &\text{Im}(\psi_{1\,\text{bin}(\texttt{mpi\_xrank})\,*\cdots*}) \\ &\qquad \leftarrow \cos(\varphi_{\texttt{cbit}})\text{Im}(\psi^{(\text{cont})}_{1\,\text{bin}(\texttt{mpi\_xrank})\,*\cdots*}) \\ &\qquad + \sin(\varphi_{\texttt{cbit}})\text{Re}(\psi^{(\text{cont})}_{1\,\text{bin}(\texttt{mpi\_xrank})\,*\cdots*}). \end{aligned} \tag{11}$$

2.4. Hadamard Gate

The implementation of the Hadamard gate on the first qubit transforms the statevector coefficients as

$$\begin{aligned} &\psi_{0\,\text{bin}(\texttt{mpi\_xrank})\,*\cdots*} \\ &\quad \leftarrow \frac{\psi_{0\,\text{bin}(\texttt{mpi\_xrank})\,*\cdots*} + \psi_{1\,\text{bin}(\texttt{mpi\_xrank})\,*\cdots*}}{\sqrt{2}}, \end{aligned} \tag{12}$$

$$\begin{aligned} &\psi_{1\,\text{bin}(\texttt{mpi\_xrank})\,*\cdots*} \\ &\quad \leftarrow \frac{\psi_{0\,\text{bin}(\texttt{mpi\_xrank})\,*\cdots*} - \psi_{1\,\text{bin}(\texttt{mpi\_xrank})\,*\cdots*}}{\sqrt{2}}. \end{aligned} \tag{13}$$

For every GPU in the $\texttt{mpi\_x} = 0$ group, this requires two-sided MPI communication with exactly one GPU in the $\texttt{mpi\_x} = 1$ group.

2.5. Measurement Operation

At the end of each stage in Figure 2, the classical bit $j_{\texttt{cbit}}$ is measured, where $\texttt{cbit} = 0, \ldots, t-1$ enumerates the stage. This amounts to adding up the probabilities

$$p_1 = \sum_{k_{L-1} \cdots k_0 = 0,1} |\psi_{1\,k_{L-1} \cdots k_0}|^2, \tag{14}$$

which is an MPI reduction over all GPUs belonging to the $\texttt{mpi\_x} = 1$ group. The probability to measure 1 (0) is then given by p_1 ($p_0 = 1 - p_1$). This probability can be used to sample $j_{\texttt{cbit}}$, which is performed by drawing a uniform random number $R \in [0, 1)$, and assigning $j_{\texttt{cbit}} = 1$ if this $R < p_1$ and $j_{\texttt{cbit}} = 0$ otherwise.

2.6. Reset Operation

The reset operation performs both the von-Neumann projection of the statevector to the result of the measurement and the reinitialization of the first qubit in $|+\rangle$ at the same time. If the result of the measurement is given by $j_{\texttt{cbit}} = 0, 1$, this operation is performed by transforming all coefficients according to

$$\begin{pmatrix} \psi_{00\cdots 0} \\ \vdots \\ \psi_{01\cdots 1} \\ \psi_{10\cdots 0} \\ \vdots \\ \psi_{11\cdots 1} \end{pmatrix} \leftarrow \begin{pmatrix} \psi_{j_{\texttt{cbit}}0\cdots 0} / \sqrt{2p_{j_{\texttt{cbit}}}} \\ \vdots \\ \psi_{j_{\texttt{cbit}}1\cdots 1} / \sqrt{2p_{j_{\texttt{cbit}}}} \\ \psi_{j_{\texttt{cbit}}0\cdots 0} / \sqrt{2p_{j_{\texttt{cbit}}}} \\ \vdots \\ \psi_{j_{\texttt{cbit}}1\cdots 1} / \sqrt{2p_{j_{\texttt{cbit}}}} \end{pmatrix}. \tag{15}$$

Of course, in the case of quantum errors, the coefficients have to be replaced accordingly (cf. Equations (25a) and (25b)).

This operation requires an MPI transfer of all coefficients from the GPUs in the group $\texttt{mpi\_x} = j_{\texttt{cbit}}$ to the GPUs in the group $\texttt{mpi\_x} = 1 - j_{\texttt{cbit}}$.

2.7. Initialization Errors

There are two different types of initialization errors that `shorgpu` can simulate, namely an amplitude initialization error and a phase initialization error. In both cases, a slightly different initial state $|+'\rangle$ is used instead of $|+\rangle$ for the first qubit in all stages $\texttt{cbit} = 0, \ldots, t-1$ of the circuit in Figure 2. The slightly erroneous state $|+'\rangle$ is parameterized in terms of an error parameter $\delta \in [0,1]$. Our motivation to prioritize the recycled qubit for a study of initialization errors instead of the other "internal" qubits is that this qubit is measured and reinitialized successively in every stage of the iterative Shor algorithm.

2.7.1. Amplitude Initialization Error

We define an amplitude initialization error as the case in which, at the beginning of each stage in Figure 2, the quantum state is not initialized in the equal superposition $|+\rangle$ but the slightly unequal superposition

$$|+'_{\text{ampl}}(\delta)\rangle = \sqrt{\frac{1+\delta}{2}}\,|0\rangle + \sqrt{\frac{1-\delta}{2}}\,|1\rangle\,. \tag{16}$$

This expression is motivated by the observation that quantum computer prototypes from the NISQ era sometimes tend to prefer $|0\rangle$ over $|1\rangle$ when brought to a uniform superposition by multiple quantum gates [87]. Furthermore, one of the most prominent decoherence and noise processes in qubit systems is a decay from $|1\rangle$ to $|0\rangle$, a so-called T_1 relaxation process [88–90].

2.7.2. Phase Initialization Error

As a second type of initialization error, we consider a phase initialization error defined as

$$|+'_{\text{phase}}(\delta)\rangle = \frac{1}{\sqrt{2}}\,|0\rangle + \frac{e^{i\pi\delta}}{\sqrt{2}}\,|1\rangle\,. \tag{17}$$

This expression is motivated by the fact that in addition to the T_1 relaxation process, dephasing processes are other prominent consequences of decoherence and noise in quantum systems [88,89,91].

2.7.3. Effective Single-Qubit Error Probability

For both initialization errors, the error parameter $\delta \in [0,1]$ can be related to an effective, single-qubit error probability, defined as the probability that the erroneous state would correctly be observed as a $|+\rangle$ state when measured along the x axis:

$$p^{\text{error}}_{\text{ampl}}(\delta) = 1 - |\langle + | +'_{\text{ampl}}(\delta)\rangle|^2 = \frac{1-\sqrt{1-\delta^2}}{2}, \tag{18}$$

$$p^{\text{error}}_{\text{phase}}(\delta) = 1 - |\langle + | +'_{\text{phase}}(\delta)\rangle|^2 = \frac{1-\cos(\pi\delta)}{2}. \tag{19}$$

Note, however, that this interpretation is not unique; depending on the particular realization of the quantum circuit, there may be more reasonable, alternative interpretations of δ in relation to an effective error probability.

2.8. Measurement Errors

For quantum processors, a measurement is often a slow and susceptible process by which destructive influences from the environment can enter the quantum system [92–96]. Moreover, it is particularly challenging to implement quantum non-demolition readout required for midcircuit measurements [16,97]. We distinguish between two different types of measurement errors, namely a classical error corresponding to a misclassification of the quantum measurement result, and a quantum error that may occur during or before each measurement.

2.8.1. Classical Measurement Error

We define a classical measurement error as a misclassification that occurs right after the quantum measurement process with a given, constant error probability δ. It is defined by flipping only the resulting bit $j_{\texttt{cbit}}$, while leaving the internal quantum state unchanged.

Simulating a classical measurement error requires a second sampling step, by drawing another uniform random number $R_2 \in [0,1)$ and flipping the bit if $R_2 < \delta$. In case of a misclassification error, we simply use $j^{(\text{observed})}_{\texttt{cbit}} = 1 - j_{\texttt{cbit}}$ for the classical bitstring in Figure 2. The quantum state, however, is left in its original state with the first qubit projected on $|j_{\texttt{cbit}}\rangle$.

Note that even such a single misclassification error can have non-trivial consequences, since this error affects the angles of all subsequent rotation gates (see Figure 2). This has an influence on the measurements of the following bits $\texttt{cbit} + 1, \ldots, t-1$. Therefore, a single bit flip error can induce a change in more than one classical bit of the output bitstring j.

2.8.2. Quantum Measurement Error

Quantum errors are conventionally modeled as operations $\rho \mapsto \mathcal{E}(\rho)$ on the system's density matrix $\rho = |\psi\rangle\langle\psi|$. If such an operation is a completely positive, trace-preserving map, it is called a *quantum channel* or *error channel* (see [32,98,99] for more information).

We model a quantum measurement error by applying a depolarizing error channel in every measurement process (which, on quantum computer hardware, is a time evolution that can take a significant amount of time [96]). The depolarizing error channel is defined by the quantum operation

$$\begin{aligned}\mathcal{E}_{\text{dep}}(\tilde{\rho}) &= (1-p_x-p_y-p_z)\rho \\ &\quad + p_x\sigma^x\tilde{\rho}\sigma^x + p_y\sigma^y\tilde{\rho}\sigma^y + p_z\sigma^z\tilde{\rho}\sigma^z,\end{aligned} \tag{20}$$

where $\tilde{\rho}$ is a single-qubit density matrix, $(\sigma^x, \sigma^y, \sigma^z)$ are the Pauli matrices, and (p_x, p_y, p_z) represent the error probabilities for the respective Pauli errors. After the application of $\mathcal{E}_{\text{dep}}$ to the first qubit, the density matrix that describes the state of the full quantum computer reads

$$\begin{aligned}\rho &= (\mathcal{E}_{\text{dep}} \otimes I)(|\psi\rangle\langle\psi|) \\ &= \sum_{\bar{k}\bar{k}'} \mathcal{E}_{\text{dep}}\Big(\psi_{0\bar{k}}\psi^*_{0\bar{k}'}|0\rangle\langle 0| + \psi_{0\bar{k}}\psi^*_{1\bar{k}'}|0\rangle\langle 1| \\ &\quad +\psi_{1\bar{k}}\psi^*_{0\bar{k}'}|1\rangle\langle 0| + \psi_{1\bar{k}}\psi^*_{1\bar{k}'}|1\rangle\langle 1|\Big) \otimes |\bar{k}\rangle\langle\bar{k}'|,\end{aligned} \tag{21}$$

where I is an identity operation on the remaining L qubits, and $\bar{k}, \bar{k}' = k_{L-1}\cdots k_0$ enumerate their 2^L different indices.

A measurement of the first qubit is quantum-mechanically described by the measurement operators $\mathcal{M}_0 = |0\rangle\langle 0| \otimes I$ and $\mathcal{M}_1 = |1\rangle\langle 1| \otimes I$. Using Equation (20), we find the probability to measure $j_{\texttt{cbit}} = 0, 1$ as

$$\begin{aligned}p'_{j_{\texttt{cbit}}} &= \operatorname{Tr} \mathcal{M}_{j_{\texttt{cbit}}}\rho\mathcal{M}^\dagger_{j_{\texttt{cbit}}} \\ &= (1 - p_x - p_y)p_{j_{\texttt{cbit}}} + (p_x + p_y)p_{1-j_{\texttt{cbit}}},\end{aligned} \tag{22}$$

where $p_{j_{\texttt{cbit}}} = \sum_{\bar{k}} |\psi_{j_{\texttt{cbit}}\bar{k}}|^2$ is computed according to Equation (14). A calculation of the post-measurement state $\rho'_{j_{\texttt{cbit}}}$ yields

$$\begin{aligned}\rho'_{j_{\texttt{cbit}}} &= \frac{M_{j_{\texttt{cbit}}}\rho\mathcal{M}^\dagger_{j_{\texttt{cbit}}}}{\operatorname{Tr} M_{j_{\texttt{cbit}}}\rho\mathcal{M}^\dagger_{j_{\texttt{cbit}}}} \\ &= p^{(\text{correct})}_{j_{\texttt{cbit}}} |\psi^{(\text{correct})}_{j_{\texttt{cbit}}}\rangle\langle\psi^{(\text{correct})}_{j_{\texttt{cbit}}}| \\ &\quad + p^{(\text{error})}_{j_{\texttt{cbit}}} |\psi^{(\text{error})}_{j_{\texttt{cbit}}}\rangle\langle\psi^{(\text{error})}_{j_{\texttt{cbit}}}|,\end{aligned} \tag{23}$$

where

$$p^{(\text{correct})}_{j_{\texttt{cbit}}} = \frac{(1 - p_x - p_y)p_{j_{\texttt{cbit}}}}{(1 - p_x - p_y)p_{j_{\texttt{cbit}}} + (p_x + p_y)p_{1-j_{\texttt{cbit}}}}, \tag{24a}$$

$$p^{(\text{error})}_{j_{\texttt{cbit}}} = \frac{(p_x + p_y)p_{1-j_{\texttt{cbit}}}}{(1 - p_x - p_y)p_{j_{\texttt{cbit}}} + (p_x + p_y)p_{1-j_{\texttt{cbit}}}}, \tag{24b}$$

and

$$|\psi^{(\text{correct})}_{j_{\texttt{cbit}}}\rangle = \sum_{\bar{k}} \frac{\psi_{j_{\texttt{cbit}}\bar{k}}}{\sqrt{p_{j_{\texttt{cbit}}}}} |j_{\texttt{cbit}}\bar{k}\rangle, \tag{25a}$$

$$|\psi^{(\text{error})}_{j_{\texttt{cbit}}}\rangle = \sum_{\bar{k}} \frac{\psi_{(1-j_{\texttt{cbit}})\bar{k}}}{\sqrt{p_{1-j_{\texttt{cbit}}}}} |j_{\texttt{cbit}}\bar{k}\rangle. \tag{25b}$$

Here, the superscript "correct" ("error") refers to the probability and the state in the case that no error (an error) has occurred. Furthermore, the expressions show that both Pauli x and y errors only occur in combination, so we define the joint quantum error probability $\delta = p_x + p_y$, by analogy with the classical case.

As in the classical case, a simulation of the quantum error process requires two sampling operations: first, a random number $R \in [0, 1)$ is sampled to assign the measurement result with probability $p'_{j_{\texttt{cbit}}}$ given by Equation (22), i.e., we assign $j_{\texttt{cbit}} = 1$ if this $R < p'_1$ and $j_{\texttt{cbit}} = 0$ otherwise.

Second, a random number $R_2 \in [0, 1)$ is sampled to determine whether an error has happened or not. If $R_2 < p^{(\text{error})}_{j_{\texttt{cbit}}}$, an error has happened while measuring $j_{\texttt{cbit}}$, and the simulation continues with the state $|\psi^{(\text{error})}_{j_{\texttt{cbit}}}\rangle$ given by Equation (25b). Otherwise, the simulation continues with the state $|\psi^{(\text{correct})}_{j_{\texttt{cbit}}}\rangle$.

Note that the quantum error has a more direct influence on the quantum state than the classical error, since the projection to either $|\psi^{(\text{correct})}_{j_{\texttt{cbit}}}\rangle$ or $|\psi^{(\text{error})}_{j_{\texttt{cbit}}}\rangle$ directly affects the quantum state, not only implicitly through the angles of subsequent rotation gates.

2.9. Details on Memory and Computing Time

The largest part of the memory needed by `shorgpu` is taken by the coefficients of the statevector $|\psi\rangle$ (see Equation (1)). For a 40-qubit iterative Shor circuit, which can be used to factor 39-bit integers, the statevector needs 2^{40} complex double-precision floating point numbers, so $16 \times 2^{40}\,\mathrm{B} = 16\,\mathrm{TiB}$. For performance reasons, two statevector buffers are used in the implementation of the oracle gate and the following single-qubit gates. In addition to the two statevector buffers, `shorgpu` requires two 32-bit integer buffers for the implementation of the oracle gate, called `oracle_idxrecv` and `oracle_idxsend` (see above). Each of these takes another $4 \times 2^{40}\,\mathrm{B} = 4\,\mathrm{TiB}$. The total GPU memory required is thus slightly larger than 40 TiB. When using $N_{\mathrm{GPU}} = 2048$ GPUs, the required memory per GPU is slightly larger than 20 GiB.

We performed all simulations on JUWELS Booster [100,101], a GPU cluster with 3744 NVIDIA A100 Tensor Core GPUs [102], each of which has 40 GiB of GPU memory. Note that the implementation of the algorithm requires the number of GPUs to be a power of two (cf. Section 2.1), so the maximum number of NVIDIA A100 GPUs that we can use on JUWELS Booster is 2048. The total computing time used to perform the simulations amounts to 594 core years (corresponding to 49.5 GPU years since each node contains four A100 GPUs and 48 physical CPU cores). We note that the total computing time is 22% of the 2700 core years used for the recent factoring record of RSA-250—a number with 829 binary digits from the famous RSA factoring challenge [4].

3. Results

In this section, we describe and interpret the results obtained from simulating Shor's algorithm according to Figure 1. For our analysis, we generated 61,362 factoring problems (N, a), 52,077 of which were chosen to have uniformly distributed prime factors to ensure unbiased results, and the rest comprise individual factoring problems for large semiprimes (see Appendix B). We consider Shor's original post-processing in Section 3.1 and Ekerå's post-processing in Section 3.2.

3.1. Using Shor's Post-Processing

A given factoring problem for Shor's algorithm consists of a semiprime N and a random integer $1 < a < N$ coprime to N. For each such factoring problem (N, a), Shor's algorithm produces a sample of M bitstrings (we typically consider $M = 1024$ samples). Each bitstring j is analyzed using the so-called standard procedure (see Appendix C). If all checks on j from the standard procedure pass, the algorithm was successful and we count the bitstring j as "success". However, if certain checks on j fail, we *still* evaluate j and test if it yields a factor of N. If it does, we count this factoring attempt as "lucky". Figure 4a shows a scatter plot of all "success" and "success+lucky" probabilities for all uniformly distributed problems ($n < 30$ qubits) and the individual large problems ($30 \leq n \leq 40$ qubits).

Surprisingly, "lucky" occurs much more often than expected. In Figure 4b, we see that on average only 25% of all bitstrings yield "success". Including the "lucky" cases, however, a factor of the semiprime N can be extracted from over 50% of all bitstrings on average. Additionally, all average success probabilities are significantly larger than the theoretical bound of 3–4% (see Appendix A.2). We hypothesize that asymptotically, the average success probability for "success + lucky" approaches 50% from above (further evidence is given in Section 3.1.1 below, where we give a classification of the different "lucky" scenarios and show that the main contribution saturates around 25%). This observation is remarkable, as it shows that factoring a semiprime with Shor's algorithm is often successful, even though the order-finding procedure actually fails.

Simulating Shor's algorithm for semiprimes N between 536,870,861 and 549,755,813,701 requires substantial computational resources. Therefore, only individual cases are shown in Figure 4a. These cases correspond to the largest "interesting" semiprimes for a given number of qubits $n = 30, \ldots, 40$. A noteworthy case is the factoring problem for (N, a) = (8,589,933,181, 3,974,323,683) ($n = 34$ qubits). Here, the "lucky" cases raise the success probability from 56.25% to 100% (yellow square) among all M bitstrings. Furthermore, the factoring problem for (N, a) = (274,877,906,893, 226,009,433,972) ($n = 39$ qubits, second from the right) has a success probability of 0% when the sufficient conditions for Shor's algorithm are presupposed (green circle). However, when ignoring the violations of these conditions, we find that Shor's algorithm can indeed factor N with a "lucky" success probability of 12.5% (yellow squares).

The unexpectedly large success probabilities when the lucky cases are included prompt the question "how many bitstrings do we need to sample until a factor is found?" This is a relevant question, since for large problems, computing time on both classical and quantum computers is an essential resource. Figure 5a demonstrates that, for more than half of all factoring problems examined, the first sampled bitstring already yields a factor of N. Furthermore, in only 7.7% of all factoring problems (N, a), none of the 1024 bitstrings produced a factor. In this case, the reason is usually that the choice of a was bad, which can be estimated to happen with probability 50% (see Proposition A3 in Appendix A.2). Clearly, the failure probability of 7.7% is much smaller than the theoretical estimate would suggest.

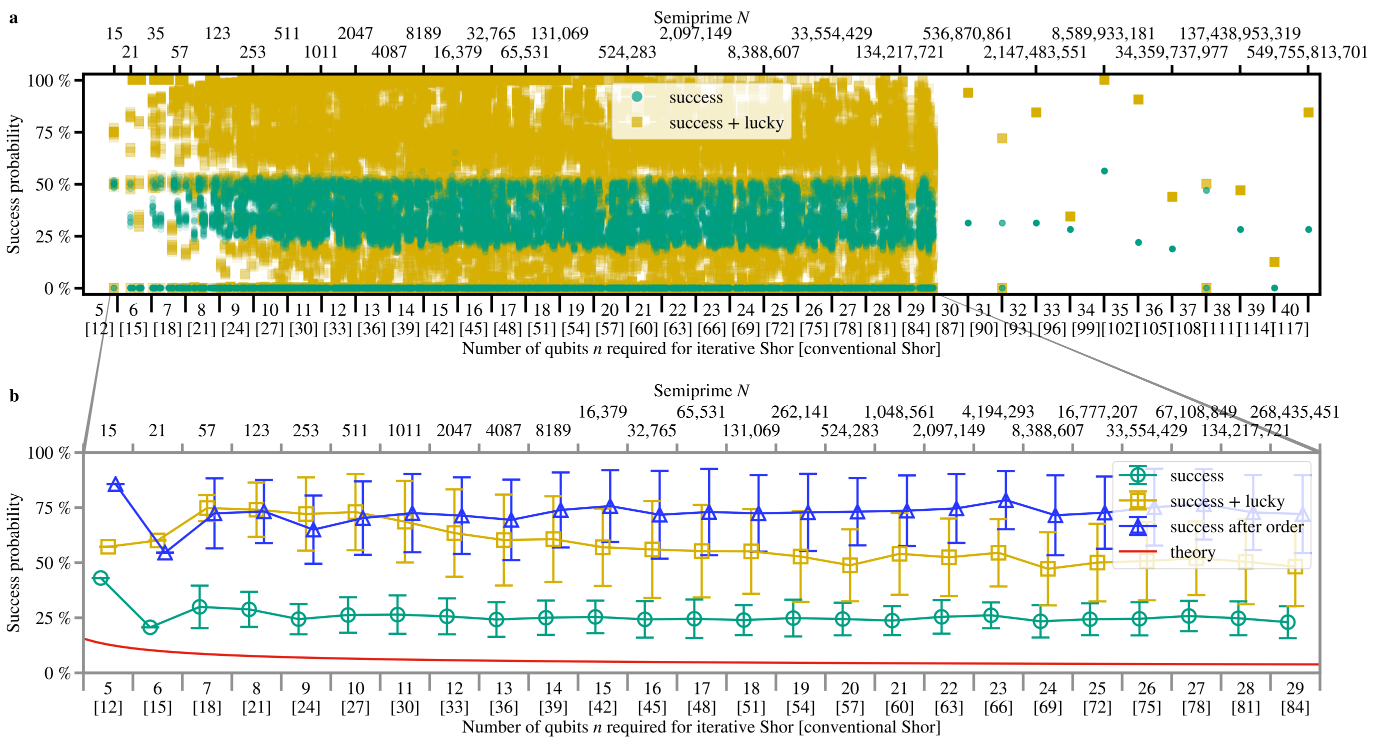

Figure 4. Success probabilities for Shor's factoring algorithm. For each factoring problem (N, a), the success probabilities are given by the ratio of all M bitstrings (sampled from Shor's algorithm) that yield a factor. If a bitstring satisfies all conditions of Shor's (original) algorithm, it is counted as "success" (green circles). If the bitstring yields a factor, even if these conditions are *not* met, it is counted as "success + lucky" (yellow squares). (**a**) Individual success probabilities for each of the 61,362 factoring problems (N, a). The markers are placed at the positions of the factored semiprime according to the top axis. The number of qubits required using both the iterative and the conventional Shor algorithm is indicated on the bottom axis. (**b**) Average success probabilities for the 52,077 uniform factoring problems, averaged over all problems (N, a) for a given number of qubits n (see text). Error bars indicate the root-mean-square deviations of the averages of a across different semiprimes N for the same n. Blue triangles represent the success probabilities for all factoring problems (N, a) that can be solved after a bitstring has yielded the order of a modulo N (which always happened within the first 33 bitstrings, see Figure 5b). The red line represents the theoretical bound for Shor's post-processing, given by $2e^{-\gamma}/\pi^2 \log\log N$ (see Appendix A.2). Lines are guides to the eye.

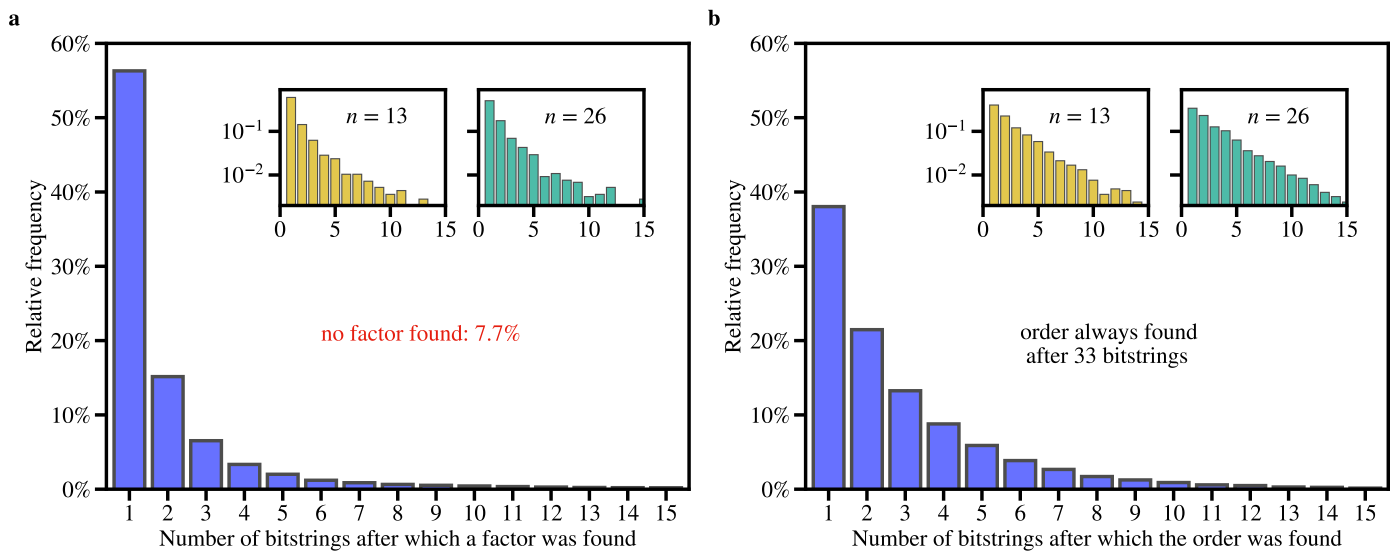

Figure 5. Statistical analysis of how often Shor's algorithm has to be executed to solve a factoring problem. Shown is the number of bitstrings that had to be generated until (**a**), a factor of N, or (**b**), the order of a modulo N could be found. The main plots show the statistics extracted from all 52,077 uniformly distributed factoring problems (N, a), for which the total number of sampled bitstrings is $M = 1024$. Insets show the same information (on a logarithmic scale) for subsets of 2500 problems that all require the same number of qubits $n = 13$ and $n = 26$ for the iterative Shor algorithm (corresponding to $n = 36$ and $n = 75$ for the conventional Shor algorithm, respectively).

Figure 5b further reveals that, even when the order-finding procedure in Shor's algorithm fails, the first bitstring often still produces a factor. In 38% of all cases, the first bitstring yields the order of a modulo N (leftmost bar). From Figure 4b, we know that on average, 75% of all factoring problems can be solved after the order is known (blue triangles). Thus we expect approximately 38% $\times$ 75% $\approx$ 29% of all factoring problems to be solved by the correct order after the first bitstring. However, in Figure 5a, we see that 56% of all problems are solved by processing the first bitstring. This percentage obviously is much larger than 29%, implying that it is easier to find a factor with Shor's algorithm than to solve the underlying order-finding problem.

Another interesting result is observed when reducing the number of bits t in the sampled bitstring below the recommended $\lceil 2 \log_2 N \rceil$ (cf. Appendix C). This saves resources in both versions of Shor's algorithm. For the conventional Shor algorithm, it reduces the required number of qubits and gates required for the QFT. For the iterative Shor algorithm, it linearly reduces the number of quantum gates and thus the execution time.

Surprisingly, in almost all cases, reducing the number of bits t still allows for a successful factorization. Three representative cases are shown in Figure 6. First, Figure 6a shows that reducing t may even increase the frequency of "lucky" factorizations to over 99%, as it does for $10 \leq t \leq 13$ in this case. Second, in Figure 6b, we see that even though the success probabilities decrease with t, at half of the recommended number of classical bits, that is at $t = 14$, there are still "lucky" cases, allowing for successful factorization. Finally, in the case shown in Figure 6c, the success and lucky probabilities are essentially constant for $4 \leq t \leq 28$.

Although it is known that reducing t may still allow for non-zero success probabilities [39,44,46,61], the surprising robustness (or even increase) of the "lucky" success probabilities has not been appreciated. In conclusion, Shor's algorithm can still be successful (sometimes even more successful) if much less classical bits t are sampled than the recommended $t = \lceil 2 \log_2 N \rceil$.

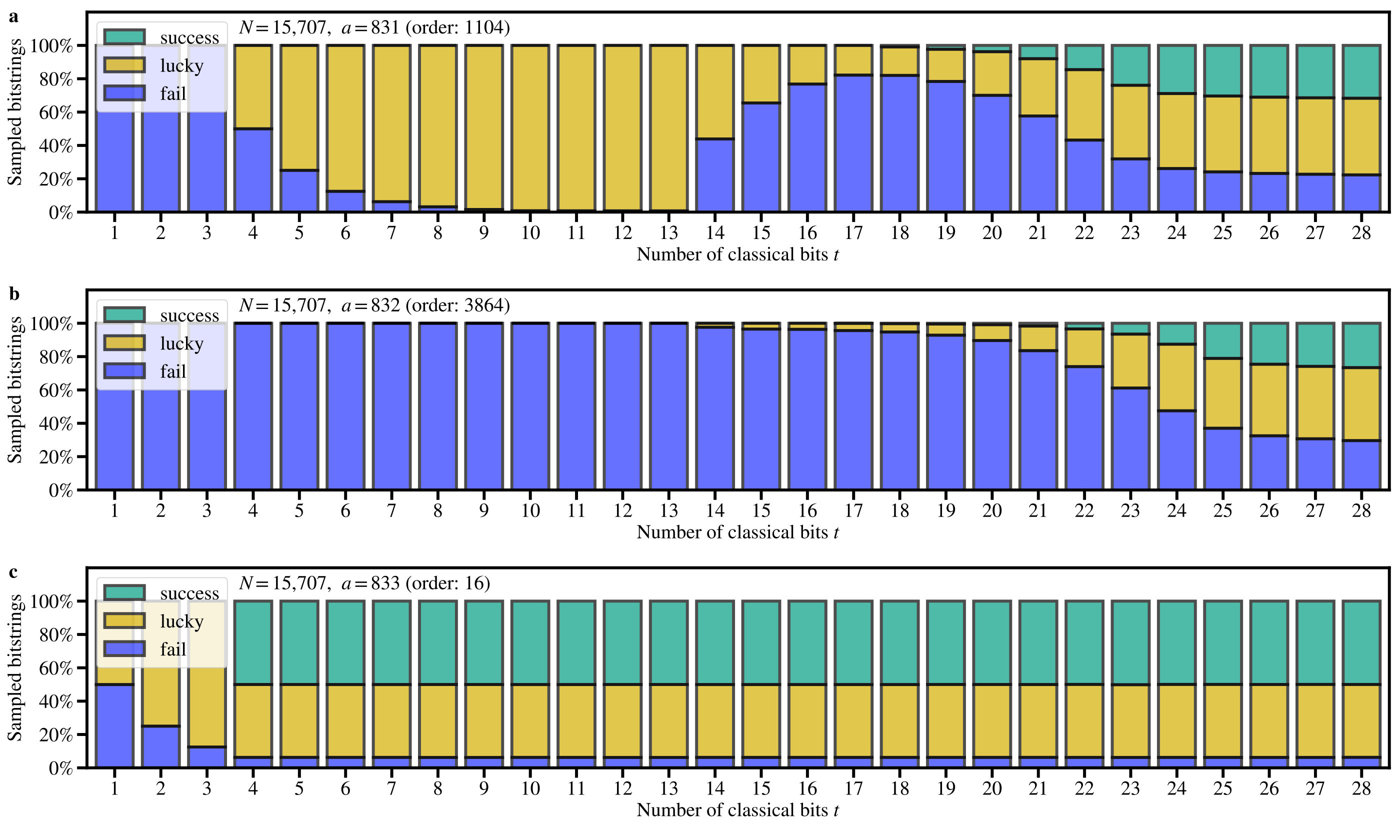

Figure 6. Success probabilities of Shor's algorithm when less than the recommended $t = \lceil 2 \log_2 N \rceil$ classical bits are extracted from the QFT. Every bar represents the fraction of 1 million sampled bitstrings classified as "success" (green), "lucky" (yellow), and "fail" (blue). The factored semiprime is N = 15,707 (such that $t = 28$) with (**a**) $a = 831$, (**b**) $a = 832$, and (**c**) $a = 833$. The corresponding orders of a modulo N are indicated at the top. Increasing t beyond 28 does not further improve the success probabilities.

3.1.1. Classification of the "Lucky" Scenarios

If a sampled bitstring j does not pass the standard tests required by Shor's algorithm, but still produces a factor with the procedure shown in Figure 1, we call this a "lucky" case. As shown above, this happens much more often than expected. In this section, we explain and classify the different scenarios that can happen.

For a given factoring problem with an L-bit semiprime N and an integer a coprime to N, let $\hat{r}$ be the multiplicative order of a modulo N. Furthermore, let r be the denominator and k be the numerator extracted from the convergent k/r to $j/2^t$ using the continued fractions algorithm from the standard procedure (i.e., the largest $r < N$ such that k/r is a convergent to $j/2^t$ with $|k/r - j/2^t| \leq 1/2r^2$ [10]). We distinguish between three scenarios in which the standard checks for Shor's algorithm fail:

(**n,e**) $r \neq \hat{r}$ is **not** the order but r is **even**,
(**n,o**) $r \neq \hat{r}$ is **not** the order and r is **odd**,
(**o,o**) $r = \hat{r}$ is the **order** but the order is **odd**.

In Figure 7, we show a breakdown of the average success probability for these scenarios. We see that the (**n,e**) scenario makes up a large fraction of the successful factorizations and its relevance grows for larger integers. In contrast, the (**n,o**) and (**o,o**) scenarios, where the extracted r is odd, only matter for smaller integers. We explain the reasons for this in the discussions of each individual scenario below.

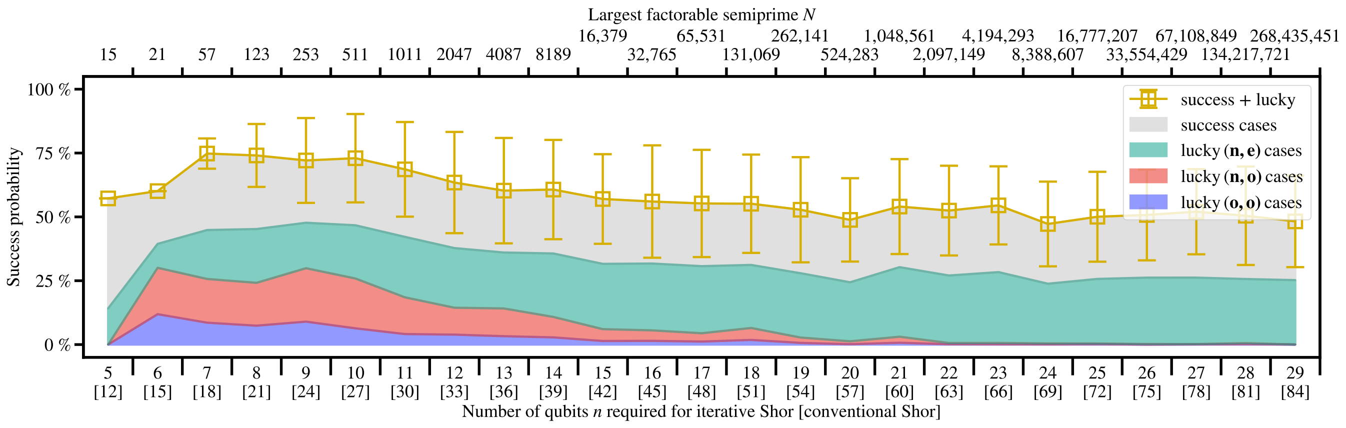

Figure 7. Breakdown of the average success probabilities for successful factoring scenarios. The contributions to the average "success + lucky" probability (yellow squares) from Figure 4b are divided into the "success" cases (gray), the lucky (**n,e**) cases (green), the lucky (**n,o**) cases (red), and the lucky (**o,o**) cases (blue). Note that an L-bit semiprime requires $n = L + 1$ qubits for the iterative Shor algorithm and roughly $3L$ qubits using the conventional Shor algorithm, as indicated on the bottom axis. The largest odd semiprime that can be factored with a given number of qubits is shown on the top axis. Lines are guides to the eye.

Figure 8 shows the number of cases for each scenario among the 52,077 uniformly drawn factoring problems. We see that the cases where bitstrings yield a factor in the (**n,e**) scenario are responsible for a significant fraction of all successful factorizations. Indeed, as Figure 7 suggests, the relevance of this scenario also grows on average and tends to saturate above 25% for larger L. We expect that this contribution persists for even larger semiprimes.

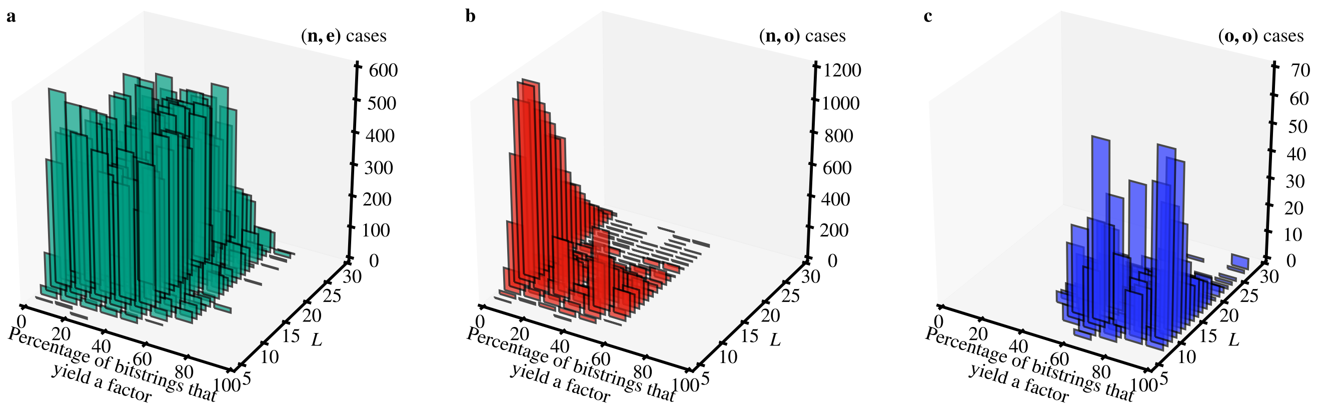

Figure 8. Classification of the different "lucky" scenarios. Shown is the absolute number of (**a**) (**n,e**) cases, (**b**) (**n,o**) cases, and (**c**) (**o,o**) cases, which yield a factor even though the sufficient conditions for Shor's algorithm are not met. For each bit length $L = 9, \ldots, 29$, the total number of cases is given by 2500 uniformly distributed factoring problems (for smaller L, the total number is smaller than 2500 because all possibilities for factoring problems (N, a) are exhausted, see Appendix B). Every bar represents a 10%-wide half-open interval $(P - 10\%, P]$ with $P = 10\%, 20\%, \ldots, 100\%$ for the percentage of all $M = 1024$ bitstrings that yield a factor in this case. For instance, the large leftmost red bar at $L = 11$ and $P = 10\%$ in panel (**b**) means that in 1197 out of 2500 cases, up to 10% of all sampled bitstrings yield a factor even though they represent an (**n,o**) case, i.e., they produce an odd number r that is not the order. Similarly, the large rightmost blue bar at $L = 9$ in panel (**c**) means that in 68 out of 2500 cases, more than 90% of all sampled bitstrings yield a factor even though the correctly extracted order $r = \hat{r}$ is odd.

Contributions from the (**n,o**) and (**o,o**) scenarios seem to be responsible only for a small number of successful factorizations, and mostly only for small semiprimes up to $L = 10$. It is remarkable, however, that for many factoring problems that can be factored in the (**o,o**) scenario, 50–100% of all bitstrings yield a factor (see Figure 8c).

To understand these effects, we discuss the different scenarios individually. The goal is to obtain an understanding for why the different scenarios occur. The number-theoretic ideas are similar to the algorithm used in [33], which can be traced back to Miller's algorithm ([103], Lemma 5).

3.1.2. The (**n,e**) Scenario

From the quantum circuit of Shor's algorithm, one can compute the probability distribution for the bitstrings j that are sampled at the measurement [104] (see Appendix A for the derivation),

$$p_{\hat{r},t}(j) = \frac{\hat{r}}{2^{2t}} \left(\frac{\sin(\pi \hat{r} j \lfloor \frac{2^t}{\hat{r}} \rfloor / 2^t)}{\sin(\pi \hat{r} j / 2^t)} \right)^2 + \frac{2^t - \hat{r} \lfloor \frac{2^t}{\hat{r}} \rfloor}{2^{2t}} \frac{\sin(\pi \hat{r} j [2 \lfloor \frac{2^t}{\hat{r}} \rfloor + 1] / 2^t)}{\sin(\pi \hat{r} j / 2^t)}, \tag{26}$$

where $\lfloor 2^t / \hat{r} \rfloor$ denotes the integral number of times that $\hat{r}$ fits into 2^t. Note that for a given factoring problem (N, a) with t classical bits per bitstring, this distribution only depends on the order $\hat{r}$ of a modulo N. This is a consequence of the fact that the QFT in Shor's algorithm is used to determine the period of the function $f(k) = a^k \bmod N$, which is exactly $\hat{r}$.

The distribution in Equation (26) is shown for a few representative cases in Figure A1 in Appendix A. It is strongly peaked at $\hat{r}$ bitstrings (see also [51])

$$j \in \{\mathrm{round}(\hat{k} \times 2^t / \hat{r}) \, : \, \hat{k} = 0, \ldots, \hat{r} - 1\}, \tag{27}$$

where $\hat{k}$ enumerates the $\hat{r}$ peaks. Given j, the continued fractions algorithm yields a convergent $k/r = \hat{k}/\hat{r}$ to $j/2^t$. However, if $\hat{k}$ and $\hat{r}$ have a common factor, the denominator r from the extracted convergent will not be equal to the order $\hat{r}$. For instance, this is the reason that the "success" cases in Figure 4a are typically below 50%, since every second peak corresponds to an even $\hat{k}$, and an even order $\hat{r}$ is a sufficient condition for success; hence, at least a factor of two is lost in $\hat{k}/\hat{r}$. We note that with a very small probability, this procedure may also yield an $r > \hat{r}$ if the sampled bitstring j is not at one of the $\hat{r}$ peaks of $p_{\hat{r},t}(j)$.

To understand why $r \neq \hat{r}$ may still yield a factor of N, we consider the case that the order $\hat{r}$ yields a factor of N (as Figure 4b shows, this case occurs with approximately 75% frequency). In this case, we have

$$\gcd(a^{\hat{r}/2} - 1, N) = p, \tag{28}$$
$$\gcd(a^{\hat{r}/2} + 1, N) = q, \tag{29}$$

where p and q are the two prime factors of N. Let 2^d be the largest power of 2 in $\hat{r}$ such that $2 \nmid \hat{r}/2^d$ (meaning 2 does no divide $\hat{r}/2^d$). Note that often, $2^d \geq 4$ since the multiplicative order of the whole group $\mathbb{Z}_N^*$ is $\phi(N) = (p-1)(q-1)$, which is at least divisible by 4 (here, $\phi(N) = |Z_N^*|$ is Euler's totient function). In this case, Equation (28) can be written as

$$\gcd((a^{\hat{r}/4} - 1)(a^{\hat{r}/4} + 1), N) = p, \tag{30}$$

so either $(a^{\hat{r}/4} - 1)$ or $(a^{\hat{r}/4} + 1)$ contain the prime factor p. Since every second $\hat{k}$ in Equation (27) is even, it is likely that $r \in \{\hat{r}/2, \hat{r}/4, \ldots, \hat{r}/2^{d-1}\}$ (each case decreasing in likelihood). If $r = \hat{r}/2$, Equation (30) shows that by testing both $\gcd(a^{r/2} \pm 1, N)$, a factor will be found. If $r = \hat{r}/4$, knowing that r is even, we can further write Equation (30) as

$$\gcd((a^{\hat{r}/8} - 1)(a^{\hat{r}/8} + 1)(a^{\hat{r}/4} + 1), N) = p, \tag{31}$$

so if p does not happen to be in $(a^{\hat{r}/4} + 1)$, also a factor will be found. This reasoning can be iterated up to the unlikely case that $r = \hat{r}/2^{d-1}$, where Equation (28) becomes

$$\begin{aligned} \gcd(\quad & (a^{\hat{r}/2^d} - 1)(a^{\hat{r}/2^d} + 1) \\ & \times (a^{\hat{r}/2^{d-1}} + 1) \cdots (a^{\hat{r}/4} + 1), N) = p. \end{aligned} \tag{32}$$

Similarly, if $r = \hat{r}/3$, we can write Equation (28) as

$$\gcd((a^{r/2} - 1)(a^r + a^{r/2} + 1), N) = p, \tag{33}$$

in which case we also find a factor if p happens to be in the first part of the product. Finally, if $r = \hat{r}/l$ with $l > 4$, we can write Equation (28) as

$$\begin{aligned} \gcd(\quad & (a^{r/2} - 1) \\ & \times (a^{r/2 \times (l-1)} + a^{r/2 \times (l-2)} + \cdots + a^{r/2} + 1), N) = p. \end{aligned} \tag{34}$$

Note that as l grows, this case becomes increasingly unlikely since l would need to be a factor of $\hat{k}$ already. However, also in this case, there is a small chance that when evaluating $\gcd(a^{r/2} - 1, N)$, a factor can be found. We remark that when $r/2 = \hat{r}/2l$ is prime, the decomposition in Equation (34) is irreducible [66], such that no further polynomial in $a^{r/2}$ including p can be factored out.

The distribution of the fractions $r/\hat{r}$ (extracted from the data generated from the uniformly distributed factoring problems) is shown in Figure 9a. Indeed, we see that very often a small multiple of r is equal to the order $\hat{r}$. In particular, the significance of fractions up to $r/\hat{r} = 1/11$ seems to increase with L, i.e., with increasingly large semiprimes N. This observation agrees well with the argument given in [33].

Interesting examples for the lucky (**n,e**) scenario are the individual problems for $n = 34$ and $n = 39$ discussed in Section 3.1. In particular, the $n = 39$ case with N = 274,877,906,893 = 364,303 $\times$ 754,531, a = 226,009,433,972 and order $\hat{r}$ = 45,812,798,010 violates the condition $a^{\hat{r}/2} \not\equiv -1 \pmod N$. As this is one of the sufficient conditions for Shor's algorithm to guarantee successful factorization, the corresponding "success" probability is zero (green circle). However, 12.5% of the sampled bitstrings yield even integers $r \in \{\hat{r}/3, \hat{r}/5, \hat{r}/111\}$, which still allow for a successful "lucky" factorization of N (the corresponding quadratic residues a^r do not have a trivial square root, i.e., $a^{r/2} \not\equiv -1 \pmod N$).

For large N, the (**n,e**) scenario makes up the majority of all "lucky" factorizations (see Figure 7). We hypothesize that on average, the probability of "success+lucky" factorizations asymptotically approaches 50% due to the (**n,e**) scenario.

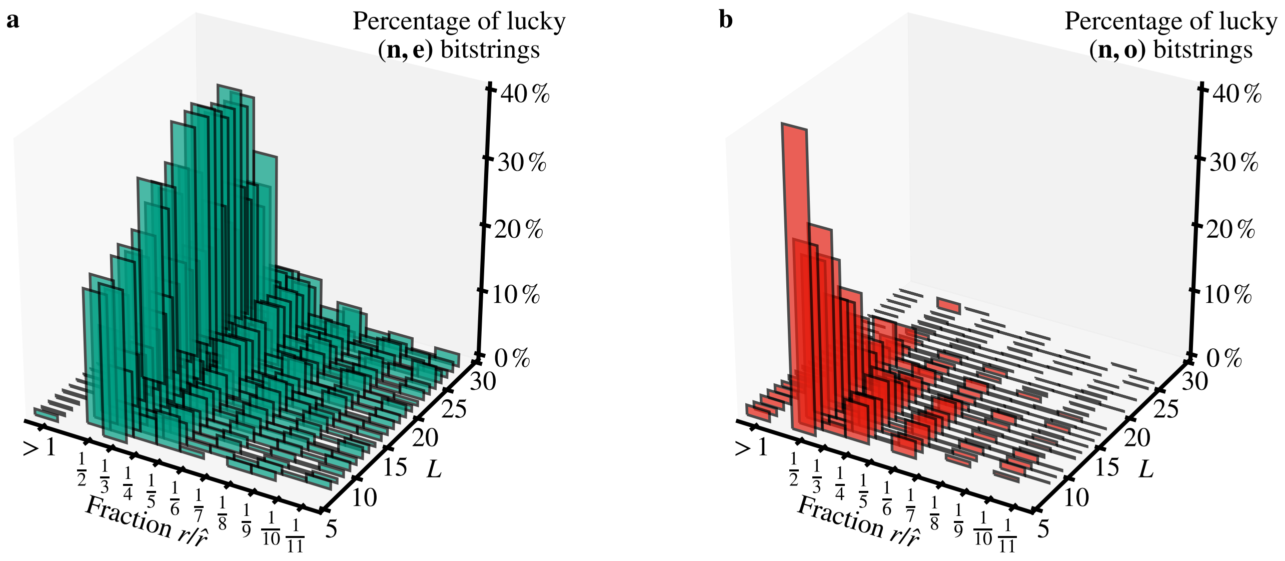

Figure 9. Histograms of the fractions $r/\hat{r}$ between the order $\hat{r}$ and the denominator r extracted from the continued fractions algorithm. Shown is the percentage of bitstrings for $L \geq 5$ (normalized by all "lucky" bitstrings) that yield a factor in the (**a**) (**n,e**) and (**b**) (**n,o**) scenario, respectively. Here, r is not the order but often a large divisor of it. Additionally, the bars at the very left for $r/\hat{r} > 1$ represent the number of cases where the extracted r is actually *larger* than the order $\hat{r}$, which may occasionally happen when a bitstring j is sampled at a non-zero probability $p_{\hat{r},t}(j)$ that is not a peak (cf. Figure A1b,c).

3.1.3. The (**n,o**) Scenario

In the (**n,o**) scenario, the bitstring j yields an integer r that is neither the order $\hat{r}$ nor even. Using the reasoning from the (**n,e**) scenario, this happens only if $r \mid \hat{r}/2^d$, so all d powers of 2 must have been in $\hat{k}$ from Equation (27) already (or $d = 0$, in which case $\hat{r}$ is odd). This is an unlikely scenario given that all $\hat{k} \in \{0, \ldots, \hat{r}-1\}$ occur with roughly the same probability, see Figure A1 in Appendix A. Moreover, as Figure 8 shows, the frequency of this scenario tends to zero for larger semiprimes N. However, it does occur for smaller semiprimes with up to 25% frequency on average (see the red area in Figure 7), so it is instructive to understand how a factor can be found in this case. In what follows, we exclude the irrelevant case $r = 1$ as it will never yield a factor.

Let $r = \hat{r}/l$ where $2^d \mid l$ and $2 \nmid r$ (note that $d = 0$ is possible if the order $\hat{r}$ is odd, but we always have $l \geq 2$ since $r \neq \hat{r}$). In this case, using the procedure depicted in Figure 1, we test

$$\gcd(a^{\lfloor r/2 \rfloor} \pm 1, N) = \gcd(a^{(\hat{r}/l-1)/2} \pm 1, N) \tag{35}$$

to find a factor of N.

Since Equation (35) seems somewhat arbitrary from the perspective of the original theory behind Shor's algorithm, one might think that a factor may only be found by coincidence. For instance, when $N = p \times q$ has a very small prime factor (say 3, 5, or 7), then whatever number is computed by Equation (35) might have a chance of including the small prime factor.

That this reasoning does not always hold is shown in Figure 10a, where we list the percentage of bitstrings that yield a factor in the (**n,o**) scenario as a function of the smallest prime factor of N. Indeed, we see that also larger prime factors can be found in certain cases. The most important of these are the cases in which a has a small order with respect to either p or q (purple squares, black crosses, and red pluses in Figure 10a). Indeed, one can prove

Theorem 1. *If*

$$\min\{\mathrm{ord}_p(a), \mathrm{ord}_q(a)\} \in \{1, 2\}, \tag{36}$$

then $\gcd(a^{\lfloor r/2 \rfloor} \pm 1, N)$ *with* $r \neq \hat{r}$ *odd yields a factor of* N.

Proof. Without loss of generality, we assume $\mathrm{ord}_p(a) \in \{1, 2\}$. This implies that $a \equiv \pm 1 \pmod{p}$ (because if $\mathrm{ord}_p(a) = 1$, we have $a \equiv 1 \pmod{p}$, and if $\mathrm{ord}_p(a) = 2$, we have $a \equiv -1 \pmod{p}$). Hence,

$$x_p := a^{\lfloor r/2 \rfloor} \bmod p = (\pm 1)^{\lfloor r/2 \rfloor} \bmod p = \pm 1. \tag{37}$$

Moreover, since $\hat{r} = \mathrm{lcm}(\mathrm{ord}_p(a), \mathrm{ord}_q(a))$, where lcm denotes the least common multiple, $\mathrm{ord}_p(a) \in \{1, 2\}$ implies that $\mathrm{ord}_q(a) \in \{\hat{r}, \hat{r}/2\}$.

Thus we have $\mathrm{ord}_q(a) \geq \hat{r}/2 \geq \hat{r}/l > \hat{r}/l - 1$ (where $l \geq 2$ is defined above Equation (35)), and therefore

$$x_q := a^{\lfloor r/2 \rfloor} \bmod q = a^{(\hat{r}/l - 1)/2} \bmod q \neq \pm 1 \tag{38}$$

(since otherwise $\hat{r}/l - 1$ would be a multiple of the order of a modulo q). Applying the Chinese remainder theorem [66] to Equations (37) and (38), we obtain

$$a^{\lfloor r/2 \rfloor} \equiv x_q p p_q + x_p q q_p \pmod{N}, \tag{39}$$

where p_q is the inverse of $p \pmod{q}$ and q_p is the inverse of $q \pmod{p}$. Using that $pp_q + qq_p \equiv 1 \pmod{N}$, we finally have

$$\gcd(a^{\lfloor r/2 \rfloor} \pm 1, N) = \gcd((x_q \pm 1) p p_q + (x_p \pm 1) q q_p, N). \tag{40}$$

Thus, since $x_q \neq \pm 1$, the case where $x_p \pm 1 = 0$ will yield the factor p. □

One can estimate how often the situation given by Equation (36) happens. Choosing a uniformly at random from $\{2, \ldots, N-1\}$ with $\gcd(a, N) = 1$ is equivalent to choosing $a \bmod p$ ($a \bmod q$) uniformly at random from $\{1, \ldots, p-1\}$ ($\{1, \ldots, q-1\}$) with the exception of $a \bmod p = a \bmod q = 1$. Thus there are $(p-2)(q-2) - 1 \sim pq$ choices for a, $2p + 2q - 13 \sim p + q$ of these satisfying either $a \bmod p = \pm 1$ or $a \bmod q = \pm 1$. Therefore, the contribution of cases with $\min\{\mathrm{ord}_p(a), \mathrm{ord}_q(a)\} = 1, 2$ becomes negligible for large p and q. We remark that individual cases with $\min\{\mathrm{ord}_p(a), \mathrm{ord}_q(a)\} = 3, 4, \ldots,$ may further contribute to lucky (**n,o**) factorizations, as Figure 10a shows.

3.1.4. The (**o,o**) Scenario

In the (**o,o**) scenario, the bitstring j yields the order $r = \hat{r}$ and the order is odd (see also [15,54–56]). Interestingly, as Figure 10b suggests, this case can be fully classified:

Theorem 2.

$$\gcd(a^{\lfloor r/2 \rfloor} - 1, N) \textit{ yields a factor} \Leftrightarrow 1 \in \{a \bmod p, a \bmod q\}. \tag{41}$$

Note that $a > 1$ *by construction, so one of* $\{a \bmod p, a \bmod q\}$ *is larger than 1, and in particular* $r > 1$.

Proof. The "⇐" case is a special case of Theorem 1 from the (**n,o**) scenario. If $a \bmod p = 1$, we have $x_p = 1$ in Equation (37) and $x_q \neq \pm 1$ in Equation (38), so $\gcd(a^{\lfloor r/2 \rfloor} - 1, N) = p$ by Equation (40).

We therefore only need to show the "⇒" case. Without loss of generality, let $\gcd(a^{\lfloor r/2 \rfloor} - 1, N) = p$, so $p \mid (a^{\lfloor r/2 \rfloor} - 1)$. From this follows that $a^{\lfloor r/2 \rfloor} \equiv 1 \pmod{p}$, so $\mathrm{ord}_p(a) \mid \lfloor r/2 \rfloor = (r-1)/2$. However, we also have $\mathrm{ord}_p(a) \mid r$ because $r = \mathrm{lcm}(\mathrm{ord}_p(a),$

$\mathrm{ord}_q(a))$. Since $\gcd((r-1)/2, r) = 1$ (using the Euclidean algorithm), this is only possible if $\mathrm{ord}_p(a) = 1$, which means $a \bmod p = 1$. □

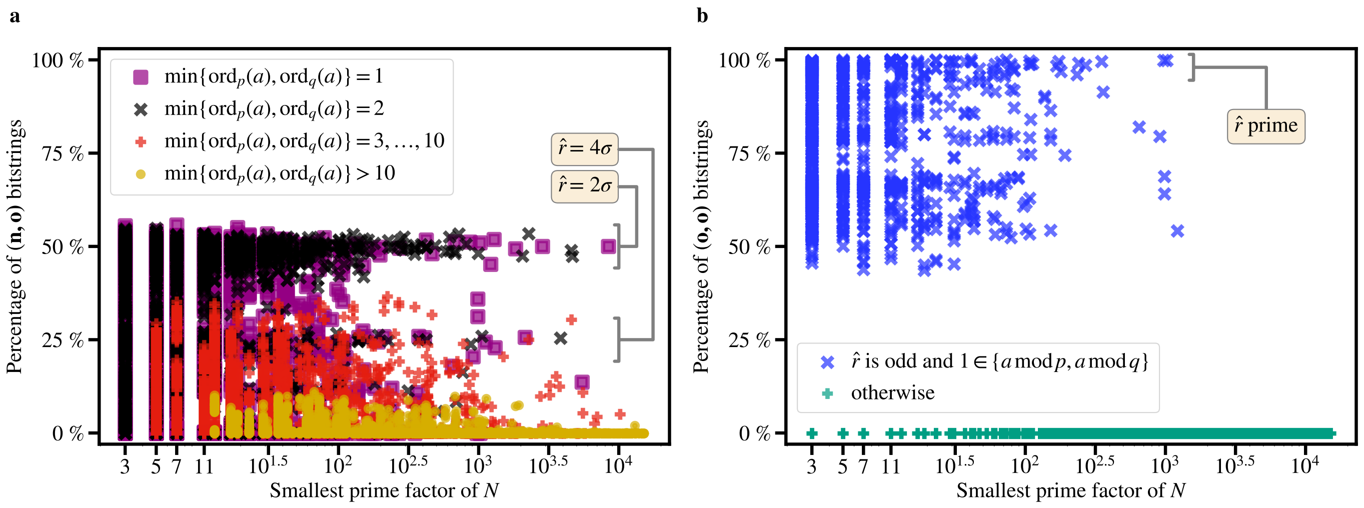

Figure 10. Classification of "lucky" factorizations for the case that the extracted integer r is odd, using $\lfloor r/2 \rfloor = (r-1)/2$ instead of $r/2$. Shown is the percentage of bitstrings that yield a factor as a function of the smallest prime factor of the semiprime $N = p \times q$. (**a**) The lucky (**n,o**) probabilities, classified in terms of the minimum order of a modulo p and q (see legend; prioritized from top to bottom). (**b**) The lucky (**o,o**) probabilities, fully classified in terms of the cases where $1 \in \{a \bmod p, a \bmod q\}$ and $\hat{r}$ is odd (blue crosses) and the rest (green pluses). In both (**a**) and (**b**), annotations indicate an additional condition that is satisfied by many (but not all) cases at this percentage level (σ denotes an odd integer). We use the notation $\mathrm{ord}_p(a)$ to denote the order of a modulo p, i.e., the smallest integer m such that $a^m \equiv 1 \pmod p$.

Theorem 3. *The "+" case, i.e.,* $\gcd(a^{\lfloor r/2 \rfloor} + 1, N)$*, never gives a factor in the case* $r = \mathrm{ord}_N(a)$ *odd.*

Proof. Assume $\gcd(a^{(r-1)/2} + 1, N) = p$. This means that $a^{(r-1)/2} \equiv -1 \pmod p$, and thus $a^{r-1} \equiv 1 \pmod p$. The former implies that $\mathrm{ord}_p(a) \nmid (r-1)/2$ and the latter implies that $\mathrm{ord}_p(a) \mid r-1$. Therefore, $2 \mid \mathrm{ord}_p(a)$. From $r = \mathrm{lcm}(\mathrm{ord}_p(a), \mathrm{ord}_q(a))$ then follows that $2 \mid r$, but this is a contradiction because r was assumed to be odd. □

Finally, we can show that it does not matter whether we round $r/2$ up or down, i.e., whether we take $\lfloor r/2 \rfloor = (r-1)/2$ or $\lceil r/2 \rceil = (r+1)/2$ in Figure 1, since one of them yields a factor whenever the other one also yields a factor:

$$\begin{aligned} \gcd(a^{\lfloor r/2 \rfloor} - 1, N) &= \gcd(a^{\lfloor r/2 \rfloor}(a^{\lceil r/2 \rceil} - 1), N) \\ &= \gcd(a^{\lceil r/2 \rceil} - 1, N), \end{aligned} \tag{42}$$

where we used that $a^{\lfloor r/2 \rfloor} a^{\lceil r/2 \rceil} \equiv a^{(r-1)/2} a^{(r+1)/2} \equiv a^r \equiv 1 \pmod N$ and furthermore that $a^{\lfloor r/2 \rfloor}$ cannot have a common factor with N (since a was chosen coprime to N),

A special, additional condition that yields a success probability of almost 100% with the (**o,o**) scenario is when $\hat{r}$ is prime, as indicated in Figure 10b. In this case, almost all bitstrings j are sampled at the peaks of Shor's bitstring distribution $p_{\hat{r},t}(j)$ given by Equation (27) and directly yield the order $r = \hat{r}$, because $\hat{k}$ and $\hat{r}$ are always coprime. The only exceptions are either when j belongs to the first peak corresponding to $\hat{k} = 0$, or when j lies in the neighborhood of one of the peaks of Equation (26) where the probability is small.

3.2. Using Ekerå's Post-Processing

In 2022, Ekerå has proven a lower bound for the success probability that takes into account additional, efficient classical post-processing procedures [34] (his implementation of the procedures can be found in [105]). While with Shor's post-processing, a factor is found with more than 50% probability on average after a single run (see Figure 5a), with Ekerå's post-processing, it is possible to increase this probability arbitrarily close to unity. The bound reads

$$p(\text{success} \mid N) \geq \underbrace{\left(1 - \frac{1}{\pi^2}\left(\frac{2}{B} + \frac{1}{B^2} + \frac{1}{3B^2}\right) - \frac{\pi^2(2B+1)}{\sqrt{2^{m+\ell}}}\right)}_{j \text{ sampled } \pm B \text{ bitstrings around peak}} \underbrace{\left(1 - \frac{1}{c \log cm}\right)}_{\text{peak yields order } \hat{r}} \underbrace{\left(1 - 2^{-k}\binom{n_F}{2} - \frac{1}{2\varsigma^2 \log^2 \varsigma L}\right)}_{\text{order } \hat{r} \text{ yields factors of } N}, \tag{43}$$

where L is the bit length of N, $m + \ell = t$ is the number of classical bits obtained from Shor's algorithm, n_F is the number of distinct prime factors of N (i.e., $n_F = 2$ for semiprimes), and $B, c, k, \varsigma \geq 1$ are constants of the post-processing algorithms that can be freely selected. We choose $m = L$ and $\ell = t - L$ so that the results are in line with the analysis presented above. Note that the only technical requirement is $2^m > \hat{r}$ and $2^{m+\ell} > \hat{r}^2$, so $m = L - 1$ is possible [34]; this does not make a difference for the results that follow. We remark that the three factors in Equation (43) are directly related to the three Propositions A1–A3 discussed in Appendix A.2. We discuss each factor in turn.

The first factor in Equation (43) comes from the idea that whenever the bitstring j is not sampled at one of the $\hat{r}$ peaks (see Figure A1 in Appendix A), it is often sampled very close to a peak. Thus, one can try out all bitstrings in the range $\{j - B, \ldots, j + B\}$ for some small B. The probability to find the peak among these bitstrings can be estimated from the distribution $p_{\hat{r},t}(j)$ in Equation (26). Instead of $4/\pi^2$ (see Equation (A20); this would correspond to $B = 0$), we then obtain a larger probability, given by the first factor. Here, we choose $B = L$, i.e., the number of bits in N.

The second factor in Equation (43) stems from the idea that, when the continued fraction method does not yield the order $\hat{r}$, it will often yield a large divisor $r = \hat{r}/D$ for some small D (see Figure 9). Starting from r, Ekerå gives several classical algorithms in [34] to efficiently recover the real order $\hat{r} = r \times D$. The corresponding success probability is given by $1 - 1/c \log cm$, where $c \geq 1$ is a parameter that is free to choose. Its derivation is based on the probability that D is cm-smooth, meaning that $D > 0$ is not divisible by any prime power larger than cm. For our numerical work we choose $c = 1$.

Finally, the third factor in Equation (43) follows from the algorithm presented in [33] (see also [54]). This algorithm describes the factoring of an arbitrary composite integer N (with $n_F \geq 2$ distinct prime factors) given the order $\hat{r}$ of a single element $a \in \mathbb{Z}_N^*$ selected uniformly at random. The corresponding success probability depends on two parameters $k, \varsigma \geq 1$ (ς is called c in [33]) that can be freely selected. For our numerical work we choose $k = 100$ and $\varsigma = 1$.

Probably the most important consequence of Ekerå's result given by Equation (43) is that, as the size of the factoring problem becomes very large, i.e., $N, \hat{r} \to \infty$, the success probability approaches one. This trend can already be seen in Figure 11a (gray line), which shows that the bound is increasing—even though it is already quite large for our modest choice of $(B, c, k, \varsigma) = (L, 1, 100, 1)$. Furthermore, the gray diamonds in Figure 11a represent the actual success probabilities, obtained from applying Ekerå's post-processing to the largest scenarios studied above. They are all larger than 93% and thus even closer to unity than expected (this potential underestimation was noted in [34]).

3.3. Errors during the Execution of Shor's Algorithm

With Ekerå's post-processing [33,34], the expected success probability using only a single run of the quantum part of the algorithm can be brought arbitrarily close to 100% by properly selecting the constants (B, c, k, ς) (cf. Equation (43)). However, these probabilistic estimates still require a successful execution of the quantum part of Shor's algorithm. Since fully error-corrected, fault-tolerant quantum computers will probably not become available for several years to come [106–108], it is an interesting, relevant question to study how the performance of the post-processing algorithms is affected by errors during the execution of the quantum algorithm.

In this section, we consider five different models for errors arising in the quantum part of Shor's algorithm. Each of these is shown in the inset of Figure 11b, which schematically marks the places in the iterative Shor algorithm (cf. Figure 2) at which the respective errors may occur.

1. Classical measurement errors (blue squares) are defined as misclassifications occurring directly after each quantum measurement process with a constant error probability $p_{\text{error}}(\delta) = \delta$ (see Section 2.8.1).
2. Quantum measurement errors (yellow circles) are modeled as depolarizing quantum noise during the measurement process with effective error probability $p_{\text{error}}(\delta) = \delta = p_x + p_y$ (see Section 2.8.2).
3. Amplitude initialization errors (green upward-pointing triangles) are modeled by initializing the recycled qubit not in the uniform superposition $|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$, but by increasing the amplitude of $|0\rangle$ as a function of δ. The effective error probability is $p_{\text{error}}(\delta) = (1 - \sqrt{1-\delta^2})/2$ (see Section 2.7.1).
4. Phase initialization errors (red down-pointing triangles) are defined by introducing a relative phase $e^{i\pi\delta}$ between the states $|0\rangle$ and $|1\rangle$ in the initialization. The effective error probability is $p_{\text{error}}(\delta) = (1 - \cos(\pi\delta))/2$ (see Section 2.7.2).
5. Bit flip errors (purple stars) are defined by flipping each bit in the final bitstring j with probability $p_{\text{error}}(\delta) = \delta$. This error model, in contrast to the others, does not affect the execution of the quantum part of the iterative Shor algorithm. While such an error (e.g., a fault in the classical computer memory) may be considered unlikely, it is still interesting to compare its consequences to the errors in the quantum part.

We consider the case that for each of these errors, Ekerå's post-processing algorithm is applied to the resulting bitstrings, without the user being aware that one or more errors may have occurred.

Figure 11a shows the success probabilities for the different errors and problem sizes. We see that errors with $\delta = 0.01$ (which corresponds to 1% error probability for the blue squares, yellow circles, and purple stars) can decrease the success probability below the bound Equation (43) indicated by the solid gray line. Furthermore, the success probabilities show a decrease as a function of problem size that rivals the increasing success probability from the bound Equation (43). Nevertheless, Ekerå's post-processing algorithm still produces correct factors even in the presence of errors.

Figure 11b shows the scaling of the success probability as a function of the effective single-qubit error probability for the 30-qubit case N = 536,870,903. For all errors, we see that the performance of the factoring algorithm including Ekerå's post-processing scales similarly, despite the fundamental differences in the error models. This type of universal behavior is an interesting and unexpected observation.

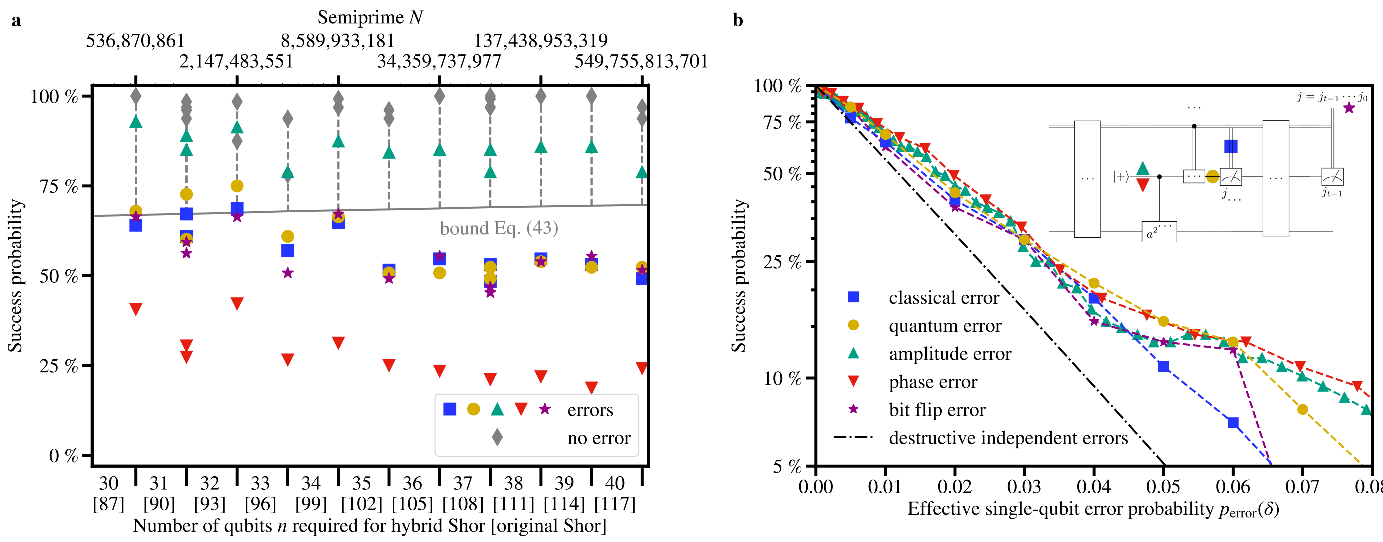

Figure 11. Performance of Shor's algorithm using Ekerå's post-processing in the presence of errors. (**a**) Success probability using Ekerå's post-processing with $(B, c, k, \varsigma) = (L, 1, 100, 1)$ for the largest scenarios from Figure 4a in the case of classical measurement errors with $\delta = 0.01$ (blue squares), quantum measurement errors with $\delta = 0.01$ (yellow circles), amplitude initialization errors with $\delta = 0.1$ (green upward-pointing triangles), phase initialization errors with $\delta = 0.1$ (red downward-pointing triangles), bit flip errors (purple stars) with $\delta = 0.01$, or no errors (gray diamonds). Additionally, the solid gray line shows the bound Equation (43) for semiprimes with $n_F = 2$ prime factors, and the vertical dashed gray lines show by how much the bound underestimates the actual performance (cf. [34]). (**b**) Success probability as a function of the effective single-qubit error probability $p_{\text{error}}(\delta)$ for the 30-qubit case N = 536,870,903 from panel (**a**). The black dash-dotted line represents the success probability in the case of independent destructive errors, $(1 - p_{\text{error}})^t$. The inset shows schematically in which parts of the quantum circuit in Figure 2 each of the different errors happen.

It is also instructive to compare the simulation results to $(1 - p_{\text{error}})^t$ (black dash-dotted line in Figure 11b), which represents the probability to obtain a bitstring for which no error occurred (under the assumption that errors for individual bits are independent). Since we know from Figure 11a that the considered 30-qubit case is solved with 100% success, $(1 - p_{\text{error}})^t$ represents the assumption that errors are destructive, i.e., an error in one of the bits prevents a successful factorization. Hence, the systematic gap between the dash-dotted line and the other dashed lines in Figure 11b shows that the quantum factoring problem can still be solved with Ekerå's post-processing in the presence of errors. The fact that the success probability is systematically larger than the one for independent errors is encouraging, because an error at one stage in the iterative Shor algorithm affects the operation of all subsequent gates that depend on previous measurement results (see inset of Figure 11b). Such an error can thus propagate through the quantum algorithm and induce further, correlated errors. Our simulation results reveal a certain resilience when using Ekerå's post-processing in combination with the iterative Shor algorithm for factoring integers.

3.4. Discussion of Limitations and Future Directions

Some of our design choices, made to achieve a large-scale simulation of Shor's algorithm for many factoring scenarios, result in certain practical limitations to what we can simulate. In this section, we list these design choices, state the accompanying limitations, and discuss interesting future research directions that alternative choices could offer.

1. In a practical realization of Shor's quantum circuit shown in Figure 2, most of the work is expected to be in the implementation of the exponentiation in terms of the controlled modular multiplications (see Section 2.2). Our choice to simulate

the multiplications using direct permutations with no extra qubits, while allowing the large-scale MPI scheme sketched in Figure 3, prevents the direct simulation of quantum errors during the multiplications (which is why, in Figure 11, essentially only initialization errors before and measurement errors after the multiplications are shown). The alternative would be to implement a general multiplication circuit using standard quantum gates and additional workspace qubits. To pursue this research direction to allow the study of errors during the multiplications, an informative exposition to start from is the construction by Gidney and Ekerå [6], which combines and optimizes many techniques discovered over the past decades to implement the modular multiplications.

2. Although Shor's order-finding algorithm is the most prominent quantum algorithm for factoring, a practical solution of the factoring problem on gate-based quantum computers might rather use the Ekerå-Håstad factoring scheme [63] based on the discrete logarithm quantum algorithm (see point 3 in Section 1.1). Instead of $t \approx 2L$ stages in the iterative quantum circuit (cf. Figure 2) using the semiclassical Fourier transform, this algorithm requires at most $t \approx 3L/2$ stages, with a systematic option to reduce t further at the cost of reducing the success probability below 99% [64]. In the context of quantum circuit simulation, the Ekerå-Håstad scheme would save valuable execution time (cf. Section 2.9), allowing for more statistics to be gathered for larger factoring scenarios.
3. The `shorgpu` implementation used for this work maintains two full statevector buffers `psi` and `psibuf`, which reduce the simulation time by enabling contiguous memory transfer through the MPI network (see [23]). However, the total amount of memory fixes the maximum number of qubits that can be simulated, which puts a limit on the size of simulatable factoring problems. An alternative choice would be to use only a single statevector buffer, thereby having to replace the contiguous memory transfer with interleaved communication and computation. This choice (potentially combined with reducing computing time by switching to the Ekerå-Håstad scheme, see previous item) would allow the simulation of yet another qubit, to push the boundary of simulatable factoring problems and the threshold of the proposed challenge one step further.

4. Conclusions

In this paper, we have introduced a method to simulate the iterative Shor algorithm on supercomputers with thousands of GPUs. The simulation software [23] allowed us to push the size of factoring problems far beyond what has been achieved previously. We have used the simulation software to perform an in-depth analysis of the iterative Shor algorithm.

Using Shor's original post-processing [8–10,32], we have shown that a significant amount of "lucky" factorizations raises the expected success probability from 3–4% to above 50%. We have given number-theoretic arguments for the existence of the lucky cases, and we hypothesize that they continue to contribute with approximately 25% beyond the size of integer factoring problems investigated in this paper.

Using Ekerå's post-processing [33,34], the success probability for a factoring scenario can be brought close to unity using only a single bitstring obtained by executing the iterative Shor algorithm. However, Ekerå's post-processing method assumes that the quantum part has been executed without errors, an assumption which is unlikely to hold for quantum processors in the near future. Therefore, we have studied how additional classical and quantum errors, as present in today's quantum information processing hardware [106], influence the performance of the post-processing procedure. Remarkably, we find that Ekerå's post-processing procedure exhibits a particular form of universality and resilience. Here, "universality" means that the decrease in success probability is roughly independent of the particular type of error and "resilience" means that the success probability is systematically larger than the success probability expected from independent bit flip errors.

Although these results might inspire confidence in the quantum factoring procedure, the first successful factorization of a cryptographically relevant number—say RSA-2048 from the famous RSA factoring challenge—is still out of reach [6,22]. Therefore, a more modest challenge towards true quantum supremacy might be to demonstrate that a real quantum computing device can factorize an interesting semiprime which is larger than $N_{\max}$ = 549,755,813,701. In fact, since gate-based quantum computers might already require full error correction for this purpose, it is conceivable that this challenge is first met by a quantum annealer [17,77–82].

Author Contributions: Conceptualization, D.W., M.W., F.J., H.D.R. and K.M.; software, D.W. and H.D.R.; validation, D.W., M.W. and H.D.R.; investigation, D.W., M.W., F.J. and H.D.R.; writing—original draft preparation, D.W.; writing—review and editing, D.W., M.W., F.J., H.D.R. and K.M.; visualization, D.W.; project administration, H.D.R. and K.M.; funding acquisition, K.M. All authors have read and agreed to the published version of the manuscript.

Funding: The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu, accessed on 18 September 2023) for funding this project by providing computing time on the GCS Supercomputer JUWELS [101] at Jülich Supercomputing Centre (JSC). D.W. and M.W. acknowledge support from the project Jülich UNified Infrastructure for Quantum computing (JUNIQ) that has received funding from the German Federal Ministry of Education and Research (BMBF) and the Ministry of Culture and Science of the State of North Rhine-Westphalia.

Data Availability Statement: The source code of the `shorgpu` simulator used to generate the data for this study is available at [23]. The data that supports the findings in this study including the generated bitstrings for the individual factoring scenarios as well as further statistics are available from the corresponding author upon reasonable request.

Acknowledgments: D.W. thanks Martin Ekerå and Viv Kendon for helpful and stimulating discussions. D.W. and H.D.R. thank Andreas Herten, Markus Hrywniak, and Jiri Kraus for help in optimizing the GPU-based simulation, in particular the MPI communication scheme.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
GPU	Graphics Processing Unit
JSC	Jülich Supercomputing Centre
JUNIQ	Jülich UNified Infrastructure for Quantum computing
JUQCS	Jülich Universal Quantum Computer Simulator
JUWELS	Jülich Wizard for European Leadership Science
MPI	Message Passing Interface
NISQ	Noisy Intermediate-Scale Quantum
QFT	Quantum Fourier Transform
RSA	Rivest Shamir Adleman
gcd	Greatest Common Divisor
lcm	Least Common Multiple

Appendix A. The Probability Distribution Generated by Shor's Algorithm

The construction of Shor's algorithm starts by assuming that there are two quantum registers of size t and L, respectively, in the initial state $|\psi_0\rangle = |0\rangle\,|0\rangle$. The first step is to bring the first register in a uniform superposition using Hadamard gates such that the state becomes

$$|\psi_1\rangle = \frac{1}{2^{t/2}} \sum_{k=0}^{2^t-1} |k\rangle\,|0\rangle\,. \tag{A1}$$

Then, application of the oracle corresponding to the function $f(k) = a^k \bmod N$ brings the state to

$$|\psi_2\rangle = \frac{1}{2^{t/2}} \sum_{k=0}^{2^t-1} |k\rangle\,|f(k)\rangle\,. \tag{A2}$$

The next step is to apply the quantum Fourier transform to the first register, which yields

$$|\psi_3\rangle = \frac{1}{2^t} \sum_{k=0}^{2^t-1} \sum_{j=0}^{2^t-1} e^{-2\pi i kj/2^t}\,|j\rangle\,|f(k)\rangle\,. \tag{A3}$$

Since $f(k)$ is a periodic function with period $\hat{r}$ (i.e., the multiplicative order of a modulo N), the second register can only take $\hat{r}$ different values. Combining all amplitudes with equal second register gives

$$|\psi_3\rangle = \frac{1}{2^t} \sum_{k=0}^{\hat{r}-1} \sum_{j=0}^{2^t-1} e^{-2\pi i kj/2^t} \left(1 + e^{-2\pi i \hat{r} j/2^t} + e^{-2\pi i 2\hat{r} j/2^t} + \cdots + e^{-2\pi i (s-1)\hat{r} j/2^t} + e^{-2\pi i s\hat{r} j/2^t}\,\delta_{[k+s\hat{r}<2^t]}\right) |j\rangle\,|f(k)\rangle\,, \tag{A4}$$

where $s = \lfloor 2^t/\hat{r} \rfloor$, and $\delta_{[k+s\hat{r}<2^t]}$ indicates that the last term only contributes if $k + s\hat{r} < 2^t$. Identifying a geometric sequence for the first s terms, we have

$$|\psi_3\rangle = \frac{1}{2^t} \sum_{k=0}^{\hat{r}-1} \sum_{j=0}^{2^t-1} e^{-2\pi i kj/2^t} \left(\frac{1 - e^{-2\pi i s\hat{r} j/2^t}}{1 - e^{-2\pi i \hat{r} j/2^t}} + e^{-2\pi i s\hat{r} j/2^t}\,\delta_{[k+s\hat{r}<2^t]}\right) |j\rangle\,|f(k)\rangle\,. \tag{A5}$$

Finally, to obtain the probability $p_{\hat{r},t}(j)$ to measure the bitstring j in the first register, we trace out the second register,

$$\begin{aligned} p_{\hat{r},t}(j) &= \tfrac{1}{2^{2t}} \textstyle\sum_{k=0}^{\hat{r}-1} \left| \frac{1-e^{-2\pi i s\hat{r} j/2^t}}{1-e^{-2\pi i \hat{r} j/2^t}} + e^{-2\pi i s\hat{r} j/2^t}\,\delta_{[k+s\hat{r}<2^t]} \right|^2 \\ &= \tfrac{\hat{r}}{2^{2t}} \left(\frac{\sin(\pi s\hat{r} j/2^t)}{\sin(\pi \hat{r} j/2^t)}\right)^2 + \frac{2^t - s\hat{r}}{2^{2t}}\,\frac{\sin(\pi[2s+1]\hat{r} j/2^t)}{\sin(\pi \hat{r} j/2^t)}\,. \end{aligned} \tag{A6}$$

This result is the same as in [104], correcting some misprints in [24,109]. Note that the singularities at $\sin(\pi\hat{r}j/2^t) = 0$ are removable singularities.

The resulting bitstring distribution is shown for a few representative cases in Figure A1a–c. It is strongly peaked at $\hat{r}$ bitstrings given by

$$j \in \{0, \mathrm{round}(2^t/\hat{r}), \mathrm{round}(2 \times 2^t/\hat{r}), \ldots, \mathrm{round}((\hat{r}-1) \times 2^t/\hat{r})\}\,. \tag{A7}$$

Note that when $\hat{r} \geq 2^t$, the distribution becomes a uniform distribution that is "peaked" everywhere. Furthermore, when $\hat{r}$ divides 2^t (such that $s = \lfloor 2^t/\hat{r} \rfloor = 2^t/\hat{r}$), only the first term in Equation (26) contributes, with the same value of $1/\hat{r}$ at all $\hat{r}$ peaks; all other bitstrings then have probability zero (see Figure A1a).

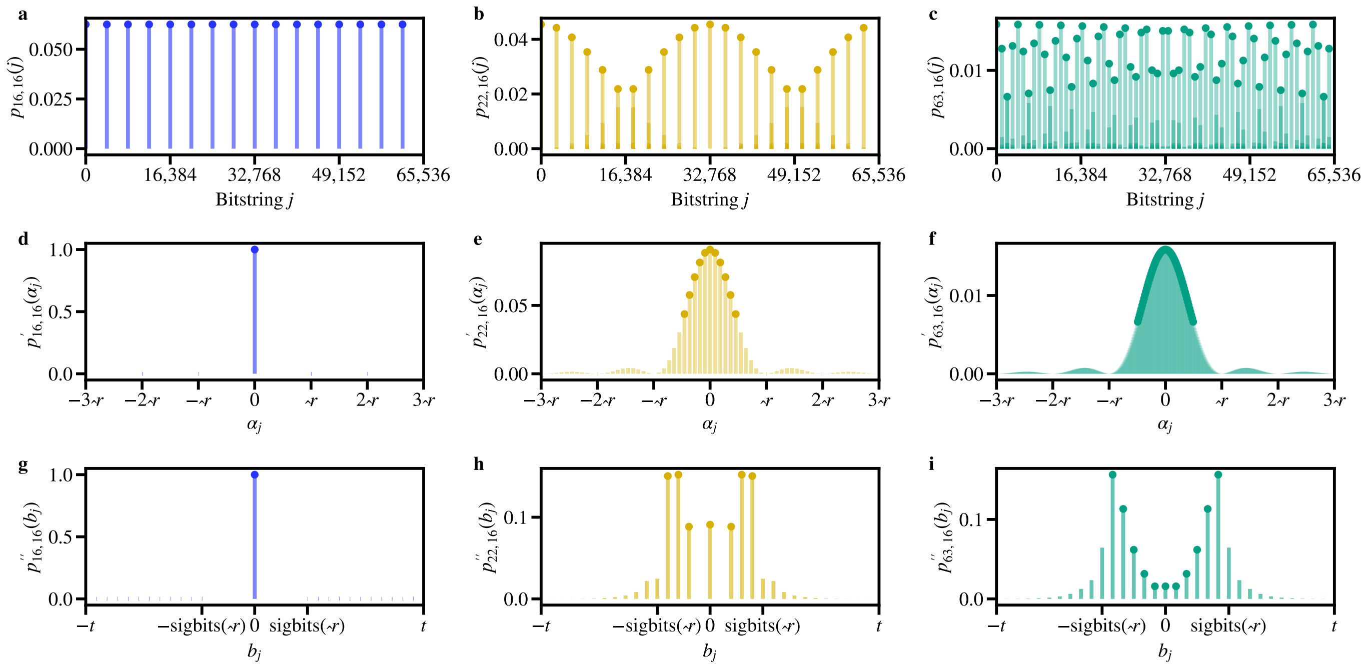

Figure A1. Representative bitstring distributions produced by Shor's algorithm. Shown is the probability distribution $p_{\hat{r},t}(j)$ given by Equation (A6) for $t = 16$ (such that the integer representation of the bitstrings j ranges from 0 to $2^t - 1 = 65{,}535$) and multiplicative orders (**a**) $\hat{r} = 16$, (**b**) $\hat{r} = 22$, and (**c**) $\hat{r} = 63$. Each distribution has exactly $\hat{r}$ peaks (solid circles) given by Equation (A7). The peaks are approximately $2^t/\hat{r}$ bitstrings apart. Note that in (**a**), the peaks are equidistant, equally large with probability $1/\hat{r}$, and all other probabilities are exactly zero. These properties are lost if $\hat{r}$ does not divide 2^t evenly, as can be seen in the presence of small but non-zero bars next to the peaks in (**b**,**c**). (**d**–**f**) The corresponding distributions $p'_{\hat{r},t}(\alpha_j)$ expressed in terms of $\alpha_j = \{\hat{r}j\}_{2^t} \in \{-2^t/2, \ldots, 2^t/2 - 1\}$ (see Equation (A10)). (**g**–**i**) The corresponding distributions $p''_{\hat{r},t}(b_j)$ expressed in terms of $b_j = \text{sigbits}(\alpha_j)$ (see Equation (A12)).

Appendix A.1. Alternative Representations of the Probability Distribution

A useful, alternative representation of the probability distribution $p_{\hat{r},t}(j)$ in Equation (A6) can be obtained by identifying all bitstrings j that yield equivalent arguments of the sine functions. Due to the periodicity of the sine function, these arguments can be represented by

$$\alpha_j = \{\hat{r}j\}_{2^t} = (\hat{r}j + 2^t/2) \bmod 2^t - 2^t/2, \tag{A8}$$

where the notation $\{x\}_y$ denotes $x \bmod y$ constrained to $\{-y/2, \ldots, y/2 - 1\}$.

All bitstrings j that yield the same α_j can be enumerated by solving the equation $\alpha_j \equiv \hat{r}j \pmod{2^t}$ for $j \in \{0, \ldots, 2^t - 1\}$. To do that, let 2^d denote the largest power of two dividing $\hat{r}$. Then $\hat{r}/2^d$ is coprime to 2^t, so it has an inverse modulo 2^t which we denote by $(\hat{r}/2^d)^{-1}$. Thus, we find $\alpha_j(\hat{r}/2^d)^{-1} \equiv 2^d j \pmod{2^t}$. This means that there is an integer $l \in \mathbb{Z}$ such that $2^d j = \alpha_j(\hat{r}/2^d)^{-1} + 2^t l$. From Equation (A8), we furthermore see that $2^d \mid \alpha_j$. Dividing by 2^d (note that we require $\hat{r} < 2^t$; the other case has been discussed above) and using that $j \in \{0, \ldots, 2^t - 1\}$, we thus obtain

$$j = \left(\frac{\alpha_j}{2^d}\left(\frac{\hat{r}}{2^d}\right)^{-1} + 2^{t-d} l\right) \bmod 2^t. \tag{A9}$$

Here, $l = 0, \ldots, 2^d - 1$ enumerates all 2^d different bitstrings j.

As each α_j has multiplicity 2^d according to Equation (A9), and each admissible α_j must be a multiple of 2^d according to Equation (A8), we can write the probability distribution for $\alpha_j \in \{-2^t/2, \ldots, 2^t/2-1\} \cap 2^d\mathbb{Z}$ as

$$p'_{\hat{r},t}(\alpha_j) = 2^d \left(\frac{\hat{r}}{2^{2t}} \left(\frac{\sin(\pi s \alpha_j/2^t)}{\sin(\pi \alpha_j/2^t)} \right)^2 + \frac{2^t - s\hat{r}}{2^{2t}} \frac{\sin(\pi[2s+1]\alpha_j/2^t)}{\sin(\pi\alpha_j/2^t)} \right). \tag{A10}$$

This distribution is shown in Figure A1d–f. The first term has the typical structure of a Fraunhofer diffraction pattern. Note in particular that all peaks given by Equation (A7) correspond to the values of α_j with $-\hat{r}/2 \leq \alpha_j \leq \hat{r}/2$ (see also [8–10]).

The advantage of using this representation is that it is the basis of a viable method to sample from the distribution, even for cryptographically large bitstrings [31,33,34,62,64]. The key is that the distribution as a function of α_j is quite regular and smooth, so it can be numerically integrated to obtain a cumulative distribution function.

More precisely, one groups α_j into logarithmically spaced regions identified by

$$b_j = \mathrm{sigbits}(\alpha_j) = \begin{cases} \mathrm{sign}(\alpha_j)(\lfloor \log_2 |\alpha_j| \rfloor + 1) & (\alpha_j \neq 0) \\ 0 & (\alpha_j = 0) \end{cases}, \tag{A11}$$

which denotes the signed number of bits needed to represent the integer α_j. This means that $2^{|b_j|-1} \leq |\alpha_j| < 2^{|b_j|}$ (note that for the numerical integration, one can use subregions of the form $2^{|b_j|-1+\xi/2^\nu} \leq |\alpha_j| < 2^{|b_j|-1+(\xi+1)/2^\nu}$ [62], along with Simpson's rule and Richardson extrapolation [110]).

The corresponding distribution,

$$p''_{\hat{r},t}(b_j) = \sum_{\mathrm{sigbits}(\alpha_j)=b_j} p'_{\hat{r},t}(\alpha_j), \tag{A12}$$

is shown in Figure A1g–i. The characteristic property for large t is that most of the probability mass is located around $b_j \approx \pm\mathrm{sigbits}(\hat{r})$. In other words, most of the sampled bitstrings j have approximately as many bits as the order $\hat{r}$. This is independent of the particular value of $\hat{r}$ (unless $\hat{r}$ contains an artificially large power of 2). This trend is already observable for $t = 16$ in Figure A1i.

In the terminology of information theory, this means that a sampled bitstring j provides approximately $t - |\mathrm{sigbits}(\hat{r})|$ bits of information on the order $\hat{r}$. This interpretation provides another intuition for the success of Shor's algorithm. For a typical factoring problem for an L-bit semiprime $N = p \times q$, bitstrings with $t \approx 2L$ classical bits in the recommended setting (see main text) are sampled. The multiplicative order $\hat{r}$ always requires *less than L* bits (the argument for this is that the largest possible order $\hat{r}$ is the least common multiple $\mathrm{lcm}(p-1, q-1)$, which is at least divisible by two, so it requires less bits than $N = p \times q$). Note that in [62], the "worst" case that $|\mathrm{sigbits}(\hat{r})| \approx L$ is considered, and even then two runs of the order-finding algorithm are sufficient.

The distribution $p_{\hat{r},t}(j)$ over bitstrings j with $t = 16$ bits is shown in Figure A1. We used `shorgpu` to generate samples from the distributions $p_{\hat{r},t}(j)$ with up to $t = 78$ bits, without knowing the solution to the specified factoring problem. If, however, the solution to the factoring problem is known, one can use the trick explained above to generate samples of $p_{\hat{r},t}(j)$ with up to t = 16,384 bits and beyond (see [62]).

Appendix A.2. Probability Theory for Shor's Factoring Procedure

In this section, we relate the results extracted from the large data sets to relations and theorems about Shor's algorithm found in the literature. We first reformulate the theoretical success probability for Shor's original factoring procedure in terms of probabilities for the different conditions. Then we relate each contribution to known theorems from the literature. This framework can be seen as the basis to interpret the results of Ekerå's post-processing stated in Section 3.2.

Given an integer N to factor, Shor's algorithm states that one should first pick a random $a \in \mathbb{Z}_N^*$ and then run the quantum algorithm. Formally, the success probability for one run of the quantum algorithm (i.e., one sampled bitstring j) therefore reads

$$p(\text{success} \mid N) = \sum_{a \in \mathbb{Z}_N^*} p(\text{success} \mid a, N)\, p(a \mid N). \tag{A13}$$

We pick a uniformly, so $p(a \mid N) = 1/|\mathbb{Z}_N^*| = 1/\phi(N)$, where $\phi(N)$ is Euler's totient function. Furthermore, the conditions for "success" stated in the literature [8–10,32] are that the sampled bitstring j yields the order $\hat{r} = \text{ord}_N(a)$, $\hat{r}$ is even, and $a^{\hat{r}/2} \not\equiv \pm 1 \pmod{N}$. Thus,

$$p(\text{success} \mid N) = \frac{1}{\phi(N)} \sum_{a \in \mathbb{Z}_N^*} p(j \text{ yields } \hat{r} \wedge \hat{r} \text{ even} \wedge a^{\hat{r}/2} \not\equiv_N \pm 1 \mid a, N). \tag{A14}$$

We know that the bitstring j yields the order $\hat{r}$ if j is sampled at one of the $\hat{k} = 0, \dots, \hat{r} - 1$ peaks of $p_{\hat{r},t}(j)$ given by Equations (A6) and (A7), and the peak enumerator $\hat{k}$ is coprime to $\hat{r}$ (so that the continued fraction method yields the convergent $k/r = \hat{k}/\hat{r}$ with $r = \hat{r}$). Hence,

$$p(\text{success} \mid N) = \frac{1}{\phi(N)} \sum_{a \in \mathbb{Z}_N^*} p(\underbrace{j \text{ sampled at a peak}}_{\mathbf{A}} \wedge \underbrace{\hat{k} \text{ coprime to } \hat{r}}_{\mathbf{B}} \wedge \underbrace{\hat{r} \text{ even} \wedge a^{\hat{r}/2} \not\equiv_N \pm 1}_{\mathbf{C}} \mid a, N), \tag{A15}$$

where we defined the Propositions A1–A3, the probabilities of each of which have known estimates (see below). Using the product rule [111], we have

$$\begin{aligned} p(\text{success} \mid N) = \tfrac{1}{\phi(N)} \textstyle\sum_{a \in \mathbb{Z}_N^*} \quad & p(j \text{ sampled at a peak} \mid a, N) \\ \times \quad & p(\hat{k} \text{ coprime to } \hat{r} \mid A1, a, N) \\ \times \quad & p(\hat{r} \text{ even} \wedge a^{\hat{r}/2} \not\equiv_N \pm 1 \mid A2, A1, a, N). \end{aligned} \tag{A16}$$

Substituting Equations (A20), (A24) and (A26) derived below, we arrive at the theoretical bound for the success probability,

$$p(\text{success} \mid N) \gtrsim \frac{4}{\pi^2} \times \frac{e^{-\gamma}}{\log \log N} \times \frac{1}{2}. \tag{A17}$$

Figure A2 shows the combined bounds from Propositions A1 and A2 in comparison with the corresponding data extracted from the simulations. We see that when the bound of $4/\pi^2$ for Proposition A1 in Equation (A20) is included, the estimate becomes very weak. If it is not included (red crosses), the values lie only slightly above the data points (at least for all uniform factoring problems with enough samples). In other words, the probability of sampling j at one of the peaks is much larger than $4/\pi^2$.

The bound Equation (A17) takes values between 3 and 4% for semiprimes N between 2^{20} and 2^{40}. Since the actual performance of Shor's algorithm shown in Figure 4b is clearly much better, it would be interesting to obtain better estimates and, in particular, to find statements about the averages instead of lower bounds.

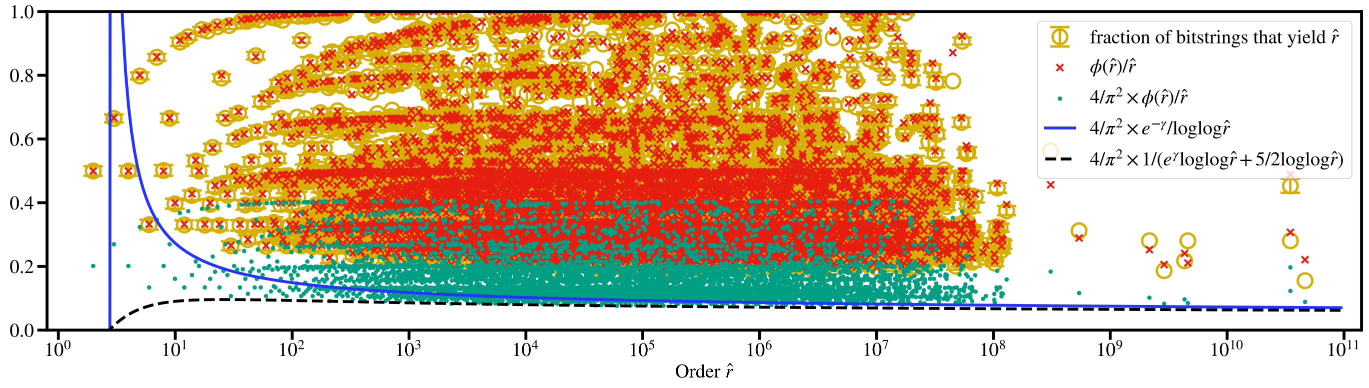

Figure A2. Comparison of the bounds for Propositions A1 and A2 proven in the literature and the corresponding frequencies extracted from the simulations. The 52,077 uniform factoring problems plus the 13 individual large cases from Figure 4a are grouped as a function of increasing $\hat{r} = \mathrm{ord}_N(a)$. Yellow circles represent the average fraction of sampled bitstrings (normalized by a total of 1024 for the uniform cases and 32 for the large cases) that yield the correct order $\hat{r}$ (meaning that they satisfy Propositions A1 and A2 in Equation (A15)); error bars show the corresponding standard deviation for problems with the same $\hat{r}$. Red crosses indicate the corresponding values of $\phi(\hat{r})/\hat{r}$. Green points, the solid blue line, and the dashed black line indicate the bounds in Equations (A21)–(A23), respectively, combined with the lower bound of $4/\pi^2$ for Proposition A1 in Equation (A20).

Proposition A1. *j sampled at a peak.*

A known lower bound for the probability $p_{\hat{r},t}(j)$ at one of the $\hat{r}$ peaks is $4/\pi^2$ [8–10]. This result can be obtained from the distribution $p_{\hat{r},t}(j)$ given by Equation (A6): at a peak, we have by Equation (A7) that the bitstring j satisfies $j = \mathrm{round}(2^t\hat{k}/\hat{r}) = 2^t\hat{k}/\hat{r} + \delta$ with $|\delta| \leq 1/2$. Using $|\sin x| \geq |x|/(\pi/2)$ when $|x| \leq \pi/2$, $|\sin x| \leq |x|$ for all x, and the periodicity of $|\sin|$, we have for the numerator and the denominator of the first term,

$$\left|\sin(\pi j\hat{r}\lfloor 2^t/\hat{r}\rfloor/2^t)\right| \geq 2|\delta|\frac{\lfloor 2^t/\hat{r}\rfloor}{2^t/\hat{r}} \approx 2|\delta|, \tag{A18}$$

$$\left|\sin(\pi j\hat{r}/2^t)\right| \leq \pi|\delta|\frac{\hat{r}}{2^t}. \tag{A19}$$

We note that the second term of $p_{\hat{r},t}(j)$ in Equation (A6) is usually neglected in the literature or simply assumed to be positive. Indeed, the signs of both numerator and denominator are often dominated by the sign of δ. However, it may become negative for certain values such as $\hat{r} = 15$, $t = 5$, and $j = \mathrm{round}(2^t \times 4/15)$. In any case, its contribution is negligible with respect to the first term. Hence, we have $p_{\hat{r},t}(j) \gtrsim 4/\pi^2\hat{r}$. Since there are exactly $\hat{r}$ peaks, we obtain

$$p(j \text{ sampled at a peak} \mid a, N) \gtrsim \frac{4}{\pi^2} \approx 40.5\%. \tag{A20}$$

We remark that considering bitstrings j that are only a few steps away from a peak may also work (see [48] and also Section 3.2).

Proposition A2. *$\hat{k}$ coprime to $\hat{r}$.*

The probability that an integer $\hat{k} = 0, \ldots, \hat{r}-1$ is coprime to $\hat{r}$ is given by

$$p(\hat{k} \text{ coprime to } \hat{r} \mid A, a, N) = \frac{\phi(\hat{r})}{\hat{r}}, \tag{A21}$$

since there are exactly $\phi(\hat{r})$ elements in $\mathbb{Z}_{\hat{r}}$ that are coprime to $\hat{r}$. There are several estimates for this quantity in the literature. Shor [8,10] uses an estimate of the form

$$\frac{\phi(\hat{r})}{\hat{r}} \gtrsim \frac{e^{-\gamma}}{\log\log\hat{r}}, \tag{A22}$$

where γ is Euler's constant such that $e^{-\gamma} \approx 0.561$. This estimate is based on the fact that $\underline{\lim}(\phi(\hat{r})\log\log\hat{r}/\hat{r}) = e^{-\gamma}$ [66], Theorem 328. However, this is only an infimum limit, and one can in fact show that there are infinitely many $\hat{r}$ violating this bound [112]. Ekert and Jozsa [9] mostly argue with $\phi(\hat{r})/\hat{r} > 1/\log\hat{r}$ (using the prime number theorem), but this bound is only valid for $\hat{r} \gtrsim 10^6$ and then becomes a rather weak bound. A better, strict lower bound to $\phi(\hat{r})/\hat{r}$ has been proven by Rosser and Schoenfeld in [113],

$$\frac{\phi(\hat{r})}{\hat{r}} > \frac{1}{e^{\gamma}\log\log\hat{r} + \frac{5}{2\log\log\hat{r}}}, \tag{A23}$$

which is valid for all $\hat{r} \geq 2$ except 223092870 (in which case $5/2$ must be replaced with 2.50637).

Due to the presence of $\log\log\hat{r}$, both bounds in Equations (A22) and (A23) show an extremely weak dependence on $\hat{r}$ (e.g., for $\hat{r} \in \{10^2, 10^{11}, 2^{4096}\}$, $\log\log\hat{r}$ varies only between 1 and 8). Therefore, either bound is suitable for the present estimate. For the same reason, we may safely approximate $\log\log\hat{r} \approx \log\log\phi(N) \approx \log\log N$ such that the bound becomes independent of $\hat{r}$. Thus, we obtain

$$p(\hat{k} \text{ coprime to } \hat{r} \mid A, a, N) \gtrsim \frac{e^{-\gamma}}{\log\log N}. \tag{A24}$$

Proposition A3. $\hat{r}$ even $\wedge\, a^{\hat{r}/2} \not\equiv \pm 1 \pmod N$.

Combining the results for Propositions A1 and A2 (which are now independent of the particular $a \in \mathbb{Z}_N^*$), the remaining part of Equation (A16) is

$$\frac{1}{\phi(N)} \sum_{a\in\mathbb{Z}_N^*} p(\hat{r} \text{ even} \wedge a^{\hat{r}/2} \not\equiv_N \pm 1 \mid B, A, a, N). \tag{A25}$$

We note that an *erroneous bound* of $1 - 1/2^{n_F}$ was given for this probability in both Shor's original paper [8] and in the book by Nielsen and Chuang [32]. The correct versions were given in Shor's later paper [10] and in an errata list by Nielsen [114]. An extensive proof can be found in the review by Ekert and Jozsa [9], which states the result as follows.

Theorem A1. *Let N be odd with prime factorization $N = p_1^{e_1} p_2^{e_2} \cdots p_{n_F}^{e_{n_F}}$. Suppose a is chosen at random, satisfying* $\gcd(a, N) = 1$*. Let r be the order of $a \bmod N$. Then*

$$\text{prob}(r \textit{ is even and } a^{r/2} \not\equiv \pm 1 \pmod N) \geq 1 - \frac{1}{2^{n_F - 1}}, \tag{A26}$$

*where "*prob*" means the frequency when enumerating all $a \in \mathbb{Z}_N^*$, which directly corresponds to the sum present in Equation (A25). We remark that the condition $a^{r/2} \not\equiv +1$ is actually superfluous since this case does not occur if r is the order (otherwise $r/2$ would already be the order).*

Proof. The idea of the proof is to study the converse, namely that r is odd or $a^{r/2} \equiv -1$. This only happens if all multiplicative orders r_j of $a_j = a \bmod p_j^{e_j}$ contain exactly the same power of 2 as r. In other words, $r/2^d$ and $r_j/2^{d_j}$ are odd integers with $d_j = d$. Summing over all possible d (which may be different for different a) yields

$$\text{prob}(r \text{ is odd or } a^{r/2} \equiv -1 \pmod N) = \sum_d \text{prob}(d_1 = d) \cdots \text{prob}(d_{n_F} = d). \tag{A27}$$

When enumerating all a_j, the case $d_j = d$ occurs with frequency $\leq 1/2$. Approximating the last $n_F - 1$ factors $\leq 1/2$ and using the first factor $\mathrm{prob}(d_1 = d)$ to remove the sum yields the bound $1/2^{n_F-1}$. $\square$

As the blue triangles in Figure 4 show, this statement is in agreement with the data, since the average of 75% is above the bound of 50% for $n_F = 2$ (some error bars might extend to below 50% which is due to the fact that we do not simulate the full set of all $a \in \mathbb{Z}_N^*$). Furthermore, the theoretical bound in Equation (A26) is also tight: for $N = 21$, we have exactly 50% of all $a \in \mathbb{Z}_{21}^*$ that have either an odd order r or $a^{r/2} \equiv -1$. We do not know whether one can prove the observed average frequency of 75% in Figure 4b, using that N is generated by uniformly drawing the prime factors p and q from the integers.

Appendix B. Generation of the Factoring Problems

We have generated 61,362 factoring problems (N, a). Of these, 52,077 are referred to as "uniform" factoring problems because they have been generated by a procedure, to be described next, to ensure a uniform distribution of prime factors that is not biased towards small primes. For a given number of bits L, we sample the first prime factor from a uniformly distributed set of integers $p \in \{3, \ldots, \lfloor\sqrt{2^L}\rfloor\}$ until a primality test asserts that p is prime. The second prime is similarly sampled from $q \in \{\lceil 2^{L-1}/p \rceil, \ldots, \lfloor 2^L/p \rfloor\}$ until $q > p$ and $N = p \times q$ is an L-bit semiprime.

We remark that the reason to consider semiprimes is that they yield the hardest factoring problems when factoring is reduced to order finding. This is because many elements in $\mathbb{Z}_N^*$ have large orders, but the largest order $\lambda(N)$ (i.e., the Carmichael function [115]) is always less than $N/2^{n_F-1}$, where $n_F \geq 2$ is the number of distinct prime factors in N ([34], Claim 7). Thus, if N has more than $n_F = 2$ factors, the orders become smaller on average and thus easier to find.

For each $L = 4, \ldots, 28$, this procedure is conducted for 50 different N. For each N, we subsequently draw 50 different $a \in \{2, \ldots, N-1\}$ coprime to N. This procedure exhausts all N for $4 \leq N \leq 8$ and generates 2500 unique problems (N, a) for each $L > 8$. For each problem, `shorgpu` generated $M = 1024$ bitstrings.

In addition to the uniform factoring problems, we generated 9285 individual problems relevant for Figures 4a, 6, and 11. In particular, these problems include the individual "large" cases with $30 \leq n \leq 40$ qubits, for which we always choose the largest interesting semiprimes N (see Table A1 in Appendix D). The number of sampled bitstrings is $M = 32$ ($M = 128$) for the results presented in Figure 4a (Figure 11a). In case none of these bitstrings yields a factor, we continue with a second random a. This is the reason that for $n = 31$ and $n = 37$, one pair of "success" and "success+lucky" markers is at 0% and only the second pair is above 0%.

Appendix C. Standard Procedure: Shor's Post-Processing

Executing Shor's algorithm for a given factoring problem (N, a) yields a bitstring j with t bits (the recommended number of bits is $t = \lceil 2\log_2 N \rceil$, which comes from the requirement that $N^2 \leq 2^t < 2N^2$ [10]; so we always have $t \in \{2L, 2L-1\}$ since N is no power of two). From the continued fraction expansion of $j/2^t$ (using the integer representation of the bitstring j), one takes the convergent k/r with the largest denominator $r < N$ [8–10] (we remark that in principle, it is better to stop at the largest denominator $r < 2^{t/2}$, otherwise one can construct pathological examples for smaller t for which going up to N skips the order and yields an unrelated, larger integer; see also ([34], Lemma 6)). The resulting r is often (cf. Figure 5b) equal to the order $\hat{r}$ of a modulo N, i.e., the smallest exponent such that $a^{\hat{r}} \bmod N = 1$. The standard procedure dictates that if r is even, $a^r \bmod N = 1$, and $a^{r/2} \bmod N \neq -1$, then computing the greatest common divisors $\gcd(a^{r/2} \pm 1, N)$ has a high probability of yielding a factor of N. Recall that in this work, if one of these checks on r fails but $\gcd(a^{\lfloor r/2 \rfloor} \pm 1, N)$ still produces a factor, the bitstring j is counted as "lucky".

Appendix D. List of Semiprimes

In Table A1, we give a list of the largest interesting semiprimes with $L < 50$ bits, for which a factorization using the iterative Shor algorithm shown in Figure 2 would need up to $n = 50$ qubits.

Table A1. List of the largest interesting semiprimes (where "interesting" means that the two prime factors are distinct and have the same number of decimal digits) that can be factored using the iterative Shor algorithm for a given number of qubits $n = L + 1$, where L is the number of bits required to represent the semiprime. For each semiprime $N = p \times q$, $t = \lceil \log_2 N^2 \rceil$ is the recommended minimum number of classical bits to read out (cf. Figure 2).

Qubits n	Semiprime N	Factor p	Factor q	t
5	15	3	5	8
6	21	3	7	9
7	35	5	7	11
8	35	5	7	11
9	253	11	23	16
10	493	17	29	18
11	1007	19	53	20
12	2047	23	89	22
13	4087	61	67	24
14	8051	83	97	26
15	16,241	109	149	28
16	32,743	137	239	30
17	65,509	109	601	32
18	131,029	283	463	34
19	262,099	349	751	36
20	524,137	557	941	38
21	1,048,351	1009	1039	40
22	2,097,101	1399	1499	42
23	4,194,163	1307	3209	44
24	8,388,563	2357	3559	46
25	16,777,207	4093	4099	48
26	33,554,089	3797	8837	50
27	67,108,147	8011	8377	52
28	134,217,449	11,119	12,071	54
29	268,435,247	12,589	21,323	56
30	536,870,861	22,717	23,633	58
31	1,073,741,687	27,779	38,653	60
32	2,147,483,551	32,063	66,977	62
33	4,294,967,213	57,139	75,167	64
34	8,589,933,181	89,597	95,873	66
35	17,179,869,131	125,627	136,753	68
36	34,359,737,977	117,517	292,381	70
37	68,719,476,733	242,819	283,007	72
38	137,438,953,319	189,853	723,923	74
39	274,877,906,893	364,303	754,531	76
40	549,755,813,701	712,321	771,781	78
41	1,099,511,623,591	1,002,817	1,096,423	80
42	2,199,023,255,179	1,286,533	1,709,263	82
43	4,398,046,510,399	2,014,013	2,183,723	84
44	8,796,093,021,439	2,217,443	3,966,773	86
45	17,592,186,044,353	2,005,519	8,771,887	88
46	35,184,372,088,787	3,769,453	9,334,079	90
47	70,368,744,177,439	8,388,593	8,388,623	92
48	140,737,488,355,141	11,150,957	12,621,113	94
49	281,474,976,708,763	15,847,327	17,761,669	96
50	562,949,953,421,083	16,619,039	33,873,797	98

References

1. Bressoud, D.M. *Factorization and Primality Testing*; Springer: New York, NY, USA, 1989. [CrossRef]
2. Lehman, R.S. Factoring large integers. *Math. Comput.* **1974**, *28*, 637–646. [CrossRef]
3. Lenstra, A.K.; Lenstra, H.W. *The Development of the Number Field Sieve*; Lecture Notes in Mathematics; Springer: Berlin/Heidelberg, Germany, 1993. [CrossRef]
4. Boudot, F.; Gaudry, P.; Guillevic, A.; Heninger, N.; Thomé, E.; Zimmermann, P. Comparing the Difficulty of Factorization and Discrete Logarithm: A 240-Digit Experiment. In Proceedings of the Advances in Cryptology—CRYPTO 2020, Virtual, 17–21 August 2020; Micciancio, D., Ristenpart, T., Eds.; Springer: Cham, Switzerland, 2020; pp. 62–91. [CrossRef]
5. Kleinjung, T.; Aoki, K.; Franke, J.; Lenstra, A.K.; Thomé, E.; Bos, J.W.; Gaudry, P.; Kruppa, A.; Montgomery, P.L.; Osvik, D.A.; et al. Factorization of a 768-Bit RSA Modulus. In Proceedings of the Advances in Cryptology—CRYPTO 2010, Santa Barbara, CA, USA, 15–19 August 2010; Rabin, T., Ed.; Springer: Berlin/Heidelberg, Germany, 2010; pp. 333–350. [CrossRef]
6. Gidney, C.; Ekerå, M. How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. *Quantum* **2021**, *5*, 433. [CrossRef]
7. Biasse, J.F.; Bonnetain, X.; Kirshanova, E.; Schrottenloher, A.; Song, F. Quantum algorithms for attacking hardness assumptions in classical and post-quantum cryptography. *IET Inf. Secur.* **2023**, *17*, 171–209. [CrossRef]
8. Shor, P.W. Algorithms for quantum computation: Discrete logarithms and factoring. In Proceedings of the 35th Annual Symposium on Foundations of Computer Science, Santa Fe, NM, USA, 20–22 November 1994; pp. 124–134. [CrossRef]
9. Ekert, A.; Jozsa, R. Quantum computation and Shor's factoring algorithm. *Rev. Mod. Phys.* **1996**, *68*, 733–753. [CrossRef]
10. Shor, P.W. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM J. Comput.* **1997**, *26*, 1484–1509. [CrossRef]
11. Van Meter, R.; Itoh, K.M. Fast quantum modular exponentiation. *Phys. Rev. A* **2005**, *71*, 052320. [CrossRef]
12. Kitaev, A.Y. Quantum measurements and the Abelian Stabilizer Problem. *arXiv* **1995**, arXiv:quant-ph/9511026.
13. Griffiths, R.B.; Niu, C.S. Semiclassical Fourier Transform for Quantum Computation. *Phys. Rev. Lett.* **1996**, *76*, 3228–3231. [CrossRef]
14. Parker, S.; Plenio, M.B. Efficient Factorization with a Single Pure Qubit and log$N$ Mixed Qubits. *Phys. Rev. Lett.* **2000**, *85*, 3049–3052. [CrossRef]
15. Martín-López, E.; Laing, A.; Lawson, T.; Alvarez, R.; Zhou, X.Q.; O'Brien, J.L. Experimental realization of Shor's quantum factoring algorithm using qubit recycling. *Nat. Photonics* **2012**, *6*, 773–776. [CrossRef]
16. Córcoles, A.D.; Takita, M.; Inoue, K.; Lekuch, S.; Minev, Z.K.; Chow, J.M.; Gambetta, J.M. Exploiting Dynamic Quantum Circuits in a Quantum Algorithm with Superconducting Qubits. *Phys. Rev. Lett.* **2021**, *127*, 100501. [CrossRef] [PubMed]
17. Peng, X.; Liao, Z.; Xu, N.; Qin, G.; Zhou, X.; Suter, D.; Du, J. Quantum Adiabatic Algorithm for Factorization and Its Experimental Implementation. *Phys. Rev. Lett.* **2008**, *101*, 220405. [CrossRef] [PubMed]
18. Hegade, N.N.; Paul, K.; Albarrán-Arriagada, F.; Chen, X.; Solano, E. Digitized adiabatic quantum factorization. *Phys. Rev. A* **2021**, *104*, L050403. [CrossRef]
19. Monz, T.; Nigg, D.; Martinez, E.A.; Brandl, M.F.; Schindler, P.; Rines, R.; Wang, S.X.; Chuang, I.L.; Blatt, R. Realization of a scalable Shor algorithm. *Science* **2016**, *351*, 1068–1070. [CrossRef] [PubMed]
20. Amico, M.; Saleem, Z.H.; Kumph, M. Experimental study of Shor's factoring algorithm using the IBM Q Experience. *Phys. Rev. A* **2019**, *100*, 012305. [CrossRef]
21. Smolin, J.A.; Smith, G.; Vargo, A. Oversimplifying quantum factoring. *Nature* **2013**, *499*, 163–165. [CrossRef]
22. Gouzien, E.; Sangouard, N. Factoring 2048-bit RSA Integers in 177 Days with 13 436 Qubits and a Multimode Memory. *Phys. Rev. Lett.* **2021**, *127*, 140503. [CrossRef]
23. `shorgpu`: Simulation of Shor's Algorithm with the Semiclassical Fourier Transform Using Multiple GPUs and MPI. Available online: https://jugit.fz-juelich.de/qip/shorgpu.git (accessed on 18 September 2023).
24. De Raedt, K.; Michielsen, K.; De Raedt, H.; Trieu, B.; Arnold, G.; Richter, M.; Lippert, T.; Watanabe, H.; Ito, N. Massively parallel quantum computer simulator. *Comput. Phys. Commun.* **2007**, *176*, 121. [CrossRef]
25. De Raedt, H.; Jin, F.; Willsch, D.; Willsch, M.; Yoshioka, N.; Ito, N.; Yuan, S.; Michielsen, K. Massively parallel quantum computer simulator, eleven years later. *Comput. Phys. Commun.* **2019**, *237*, 47 – 61. [CrossRef]
26. Tankasala, A.; Ilatikhameneh, H. Quantum-Kit: Simulating Shor's Factorization of 24-Bit Number on Desktop. *arXiv* **2020**, arXiv:1908.07187.
27. Wang, D.S.; Hill, C.D.; Hollenberg, L.C.L. Simulations of Shor's algorithm using matrix product states. *Quantum Inf. Process.* **2017**, *16*, 176. [CrossRef]
28. Dang, A.; Hill, C.D.; Hollenberg, L.C.L. Optimising Matrix Product State Simulations of Shor's Algorithm. *Quantum* **2019**, *3*, 116. [CrossRef]
29. Dumitrescu, E. Tree tensor network approach to simulating Shor's algorithm. *Phys. Rev. A* **2017**, *96*, 062322. [CrossRef]
30. Zhao, Y.Q.; Li, R.G.; Jiang, J.Z.; Li, C.; Li, H.Z.; Wang, E.D.; Gong, W.F.; Zhang, X.; Wei, Z.Q. Simulation of quantum computing on classical supercomputers with tensor-network edge cutting. *Phys. Rev. A* **2021**, *104*, 032603. [CrossRef]
31. Ekerå, M. Qunundrum. 2020. Available online: https://github.com/ekera/qunundrum.git (accessed on 18 September 2023).
32. Nielsen, M.A.; Chuang, I.L. *Quantum Computation and Quantum Information: 10th Anniversary Edition*; Cambridge University Press: New York, NY, USA, 2010. [CrossRef]

33. Ekerå, M. On completely factoring any integer efficiently in a single run of an order-finding algorithm. *Quantum Inf. Process.* **2021**, *20*, 205. [CrossRef]
34. Ekerå, M. On the success probability of quantum order finding. *arXiv* **2022**, arXiv:2201.07791v2.
35. Arute, F.; Arya, K.; Babbush, R.; Bacon, D.; Bardin, J.C.; Barends, R.; Biswas, R.; Boixo, S.; Brandao, F.G.S.L.; Buell, D.A.; et al. Quantum supremacy using a programmable superconducting processor. *Nature* **2019**, *574*, 505–510. [CrossRef] [PubMed]
36. Rønnow, T.F.; Wang, Z.; Job, J.; Boixo, S.; Isakov, S.V.; Wecker, D.; Martinis, J.M.; Lidar, D.A.; Troyer, M. Defining and detecting quantum speedup. *Science* **2014**, *345*, 420–424. [CrossRef]
37. Knill, E. *On Shor's Quantum Factor Finding Algorithm: Increasing the Probability of Success and Tradeoffs Involving the Fourier Transform Modulus*; Tech. Rep. LAUR-95-3350; Los Alamos National Laboratory: Santa Fe, NM, USA, 1995.
38. DiVincenzo, D.P. Quantum Computation. *Science* **1995**, *270*, 255–261. [CrossRef]
39. Barenco, A.; Ekert, A.; Suominen, K.A.; Törmä, P. Approximate quantum Fourier transform and decoherence. *Phys. Rev. A* **1996**, *54*, 139–146. [CrossRef]
40. Vedral, V.; Barenco, A.; Ekert, A. Quantum networks for elementary arithmetic operations. *Phys. Rev. A* **1996**, *54*, 147–153. [CrossRef]
41. Seifert, J.P. Using Fewer Qubits in Shor's Factorization Algorithm via Simultaneous Diophantine Approximation. In Proceedings of the Topics in Cryptology—CT-RSA 2001, San Francisco, CA, USA, 8–12 April 2001; Naccache, D., Ed.; Springer: Berlin/Heidelberg, Germany, 2001; pp. 319–327. [CrossRef]
42. McAnally, D. A Refinement of Shor's Algorithm. *arXiv* **2001**, arXiv:quant-ph/0112055.
43. Leander, G. Improving the Success Probability for Shor's Factoring Algorithm. *arXiv* **2002**, arXiv:quant-ph/0208183.
44. Coppersmith, D. An approximate Fourier transform useful in quantum factoring. *arXiv* **2002**, arXiv:quant-ph/0201067.
45. Beauregard, S. Circuit for Shor's algorithm using 2n+3 qubits. *Quantum Inf. Comput.* **2003**, *3*, 175. [CrossRef]
46. Fowler, A.G.; Hollenberg, L.C.L. Scalability of Shor's algorithm with a limited set of rotation gates. *Phys. Rev. A* **2004**, *70*, 032329. [CrossRef]
47. Kendon, V.M.; Munro, W.J. Entanglement and its Role in Shor's Algorithm. *Quantum Inf. Comput.* **2006**, *6*, 630. [CrossRef]
48. Gerjuoy, E. Shor's factoring algorithm and modern cryptography. An illustration of the capabilities inherent in quantum computers. *Am. J. Phys* **2005**, *73*, 521–540. [CrossRef]
49. Devitt, S.J.; Fowler, A.G.; Hollenberg, L.C.L. Robustness of Shor's algorithm. *Quantum Inf. Comput.* **2006**, *6*, 616–629. [CrossRef]
50. Zalka, C. Shor's algorithm with fewer (pure) qubits. *arXiv* **2016**, arXiv:quant-ph/0601097.
51. Bourdon, P.S.; Williams, H.T. Sharp Probability Estimates for Shor's Order-Finding Algorithm. *Quantum Inf. Comput.* **2007**, *7*, 522–550.
52. Markov, I.L.; Saeedi, M. Constant-optimized quantum circuits for modular multiplication and exponentiation. *Quantum Inf. Comput.* **2012**, *12*, 0361. [CrossRef]
53. Markov, I.L.; Saeedi, M. Faster quantum number factoring via circuit synthesis. *Phys. Rev. A* **2013**, *87*, 012310. [CrossRef]
54. Grosshans, F.; Lawson, T.; Morain, F.; Smith, B. Factoring Safe Semiprimes with a Single Quantum Query. *arXiv* **2015**, arXiv:1511.04385.
55. Lawson, T. Odd orders in Shor's factoring algorithm. *Quantum Inf. Process.* **2015**, *14*, 831–838. [CrossRef]
56. Johnston, A. Shor's Algorithm and Factoring: Don't Throw Away the Odd Orders. Cryptology ePrint Archive, Report 2017/083. 2017. Available online: https://ia.cr/2017/083 (accessed on 18 September 2023).
57. Häner, T.; Roetteler, M.; Svore, K.M. Factoring using 2n+2 qubits with Toffoli based modular multiplication. *Quantum Inf. Comput.* **2017**, *17*, 0673. [CrossRef]
58. Davis, E.D. Benchmarks for quantum computers from Shor's algorithm. *arXiv* **2021**, arXiv:2111.13856.
59. Bastos, D.C.; Brasil Kowada, L.A. How to detect whether Shor's algorithm succeeds against large integers without a quantum computer. *Procedia Comput. Sci.* **2021**, *195*, 145–151. [CrossRef]
60. Antipov, A.V.; Kiktenko, E.O.; Fedorov, A.K. Efficient realization of quantum primitives for Shor's algorithm using PennyLane library. *PLoS ONE* **2022**, *17*, e0271462. [CrossRef] [PubMed]
61. Nam, Y.S.; Blümel, R. Performance scaling of Shor's algorithm with a banded quantum Fourier transform. *Phys. Rev. A* **2012**, *86*, 044303. [CrossRef]
62. Ekerå, M. Quantum algorithms for computing general discrete logarithms and orders with tradeoffs. *J. Math. Cryptol.* **2021**, *15*, 359–407. [CrossRef]
63. Ekerå, M.; Håstad, J. Quantum Algorithms for Computing Short Discrete Logarithms and Factoring RSA Integers. In Proceedings of the Post-Quantum Cryptography, Utrecht, The Netherlands, 26–28 June 2017; Lange, T., Takagi, T., Eds.; Springer: Cham, Switzerland, 2017; pp. 347–363. [CrossRef]
64. Ekerå, M. On post-processing in the quantum algorithm for computing short discrete logarithms. *Des. Codes Cryptogr.* **2020**, *88*, 2313–2335. [CrossRef]
65. Jozsa, R. Quantum factoring, discrete logarithms, and the hidden subgroup problem. *Comput. Sci. Eng.* **2001**, *3*, 34–43. [CrossRef]
66. Hardy, G.H.; Wright, E.M. *An Introduction to the Theory of Numbers*, 6th ed.; Oxford University Press: Oxford, UK, 2008.
67. Bernstein, D.J.; Biasse, J.F.; Mosca, M. A Low-Resource Quantum Factoring Algorithm. In Proceedings of the Post-Quantum Cryptography, Utrecht, The Netherlands, 26–28 June 2017; Lange, T., Takagi, T., Eds.; Springer: Cham, Switzerland, 2017; pp. 330–346. [CrossRef]

68. Grover, L.K. A Fast Quantum Mechanical Algorithm for Database Search. In Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, Philadelphia, PA, USA, 22–24 May 1996; pp. 212–219. [CrossRef]
69. Li, J.; Peng, X.; Du, J.; Suter, D. An Efficient Exact Quantum Algorithm for the Integer Square-free Decomposition Problem. *Sci. Rep.* **2012**, *2*, 1–5. [CrossRef]
70. Vandersypen, L.M.K.; Steffen, M.; Breyta, G.; Yannoni, C.S.; Sherwood, M.H.; Chuang, I.L. Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance. *Nature* **2001**, *414*, 883–887. [CrossRef] [PubMed]
71. Lu, C.Y.; Browne, D.E.; Yang, T.; Pan, J.W. Demonstration of a Compiled Version of Shor's Quantum Factoring Algorithm Using Photonic Qubits. *Phys. Rev. Lett.* **2007**, *99*, 250504. [CrossRef] [PubMed]
72. Lanyon, B.P.; Weinhold, T.J.; Langford, N.K.; Barbieri, M.; James, D.F.V.; Gilchrist, A.; White, A.G. Experimental Demonstration of a Compiled Version of Shor's Algorithm with Quantum Entanglement. *Phys. Rev. Lett.* **2007**, *99*, 250505. [CrossRef]
73. Politi, A.; Matthews, J.C.F.; O'Brien, J.L. Shor's Quantum Factoring Algorithm on a Photonic Chip. *Science* **2009**, *325*, 1221–1221. [CrossRef] [PubMed]
74. Lucero, E.; Barends, R.; Chen, Y.; Kelly, J.; Mariantoni, M.; Megrant, A.; O'Malley, P.; Sank, D.; Vainsencher, A.; Wenner, J.; et al. Computing prime factors with a Josephson phase qubit quantum processor. *Nat. Phys.* **2012**, *8*, 719–723. [CrossRef]
75. Skosana, U.; Tame, M. Demonstration of Shor's factoring algorithm for N = 21 on IBM quantum processors. *Sci. Rep.* **2021**, *11*, 16599. [CrossRef]
76. Abhijith, J.; Adedoyin, A.; Ambrosiano, J.; Anisimov, P.; Casper, W.; Chennupati, G.; Coffrin, C.; Djidjev, H.; Gunter, D.; Karra, S.; et al. Quantum Algorithm & Implementations for Beginners. *ACM Trans. Quantum Comput.* **2022**, *3*, 18. [CrossRef]
77. Andriyash, E.; Bian, Z.; Chudak, F.; Drew-Brook, M.; King, A.D.; Macready, W.G.; Roy, A. *Boosting Integer Factoring Performance via Quantum Annealing Offsets*; Technical Report; D-Wave Technical Report Series 14-1002A-B; D-Wave Systems Inc.: Burnaby, BC, Canada, 2016.
78. Dridi, R.; Alghassi, H. Prime factorization using quantum annealing and computational algebraic geometry. *Sci. Rep.* **2017**, *7*, 43048. [CrossRef]
79. Jiang, S.; Britt, K.A.; McCaskey, A.J.; Humble, T.S.; Kais, S. Quantum Annealing for Prime Factorization. *Sci. Rep.* **2018**, *8*, 17667. [CrossRef] [PubMed]
80. Peng, W.; Wang, B.; Hu, F.; Wang, Y.; Fang, X.; Chen, X.; Wang, C. Factoring larger integers with fewer qubits via quantum annealing with optimized parameters. *Sci. China Phys. Mech. Astron.* **2019**, *62*, 60311. [CrossRef]
81. Mengoni, R.; Ottaviani, D.; Iorio, P. Breaking RSA Security With A Low Noise D-Wave 2000Q Quantum Annealer: Computational Times, Limitations And Prospects. *arXiv* **2020**, arXiv:2005.02268.
82. Wang, B.; Hu, F.; Yao, H.; Wang, C. Prime factorization algorithm based on parameter optimization of Ising model. *Sci. Rep.* **2020**, *10*, 7106. [CrossRef]
83. King, A.D.; Suzuki, S.; Raymond, J.; Zucca, A.; Lanting, T.; Altomare, F.; Berkley, A.J.; Ejtemaee, S.; Hoskinson, E.; Huang, S.; et al. Coherent quantum annealing in a programmable 2,000 qubit Ising chain. *Nat. Phys.* **2022**, *18*, 1324–1328. [CrossRef]
84. King, A.D.; Raymond, J.; Lanting, T.; Harris, R.; Zucca, A.; Altomare, F.; Berkley, A.J.; Boothby, K.; Ejtemaee, S.; Enderud, C.; et al. Quantum critical dynamics in a 5,000-qubit programmable spin glass. *Nature* **2023**, *617*, 61–66. [CrossRef] [PubMed]
85. Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard Version 4.0*; 2021.
86. Willsch, D.; Willsch, M.; Jin, F.; Michielsen, K.; De Raedt, H. GPU-accelerated simulations of quantum annealing and the quantum approximate optimization algorithm. *Comput. Phys. Commun.* **2022**, *278*, 108411. [CrossRef]
87. Michielsen, K.; Nocon, M.; Willsch, D.; Jin, F.; Lippert, T.; De Raedt, H. Benchmarking gate-based quantum computers. *Comput. Phys. Commun.* **2017**, *220*, 44 – 55. [CrossRef]
88. Weiss, U. *Quantum Dissipative Systems*, 4th ed.; World Scientific: Singapore, 2012. [CrossRef]
89. Paladino, E.; Galperin, Y.M.; Falci, G.; Altshuler, B.L. 1/f noise: Implications for solid-state quantum information. *Rev. Mod. Phys.* **2014**, *86*, 361–418. [CrossRef]
90. Carroll, M.; Rosenblatt, S.; Jurcevic, P.; Lauer, I.; Kandala, A. Dynamics of superconducting qubit relaxation times. *NPJ Quantum Inf.* **2022**, *8*, 132. [CrossRef]
91. Fox, M. *Quantum Optics: An Introduction*; Oxford Master Series in Physics; Oxford University Press: Oxford, UK, 2006.
92. Wallraff, A.; Schuster, D.I.; Blais, A.; Frunzio, L.; Majer, J.; Devoret, M.H.; Girvin, S.M.; Schoelkopf, R.J. Approaching Unit Visibility for Control of a Superconducting Qubit with Dispersive Readout. *Phys. Rev. Lett.* **2005**, *95*, 060501. [CrossRef] [PubMed]
93. Gambetta, J.; Blais, A.; Schuster, D.I.; Wallraff, A.; Frunzio, L.; Majer, J.; Devoret, M.H.; Girvin, S.M.; Schoelkopf, R.J. Qubit-photon interactions in a cavity: Measurement-induced dephasing and number splitting. *Phys. Rev. A* **2006**, *74*, 042318. [CrossRef]
94. Reed, M.D.; DiCarlo, L.; Johnson, B.R.; Sun, L.; Schuster, D.I.; Frunzio, L.; Schoelkopf, R.J. High-Fidelity Readout in Circuit Quantum Electrodynamics Using the Jaynes-Cummings Nonlinearity. *Phys. Rev. Lett.* **2010**, *105*, 173601. [CrossRef]
95. Jacobs, K. *Quantum Measurement Theory and Its Applications*; Cambridge University Press: Cambridge, MA, USA, 2014. [CrossRef]
96. Naghiloo, M. Introduction to Experimental Quantum Measurement with Superconducting Qubits. *arXiv* **2019**, arXiv:1904.09291.
97. Boissonneault, M.; Gambetta, J.M.; Blais, A. Improved Superconducting Qubit Readout by Qubit-Induced Nonlinearities. *Phys. Rev. Lett.* **2010**, *105*, 100504. [CrossRef]
98. Holevo, A.S. *Quantum Systems, Channels, Information: A Mathematical Introduction*; De Gruyter: Berlin, Germany, 2019. [CrossRef]
99. Wilde, M.M. *Quantum Information Theory*; Cambridge University Press: Cambridge, UK, 2017. [CrossRef]

100. Jülich Supercomputing Centre. JUWELS: Modular Tier-0/1 Supercomputer at the Jülich Supercomputing Centre. *J. Large-Scale Res. Facil.* **2019**, *5*, A135. [CrossRef]
101. Jülich Supercomputing Centre. JUWELS Cluster and Booster: Exascale Pathfinder with Modular Supercomputing Architecture at Juelich Supercomputing Centre. *J. Large-Scale Res. Facil.* **2021**, *7*, A138. [CrossRef]
102. NVIDIA A100 Tensor Core GPU. Data Sheet. 2022. Available online: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-nvidia-us-2188504-web.pdf (accessed on 18 September 2023).
103. Miller, G.L. Riemann's hypothesis and tests for primality. *J. Comput. Syst. Sci* **1976**, *13*, 300–317. [CrossRef]
104. Michielsen, K.; De Raedt, K.; De Raedt, H. Simulation of Quantum Computation: A Deterministic Event-Based Approach. *J. Comput. Theor. Nanosci.* **2005**, *2*, 227–239. [CrossRef]
105. Ekerå, M. Quppy. 2023. To Appear. Available online: https://github.com/ekera/quppy.git (accessed on 6 October 2023).
106. Google Quantum AI. Exponential suppression of bit or phase errors with cyclic error correction. *Nature* **2021**, *595*, 383–387. [CrossRef]
107. Krinner, S.; Lacroix, N.; Remm, A.; Di Paolo, A.; Genois, E.; Leroux, C.; Hellings, C.; Lazar, S.; Swiadek, F.; Herrmann, J.; et al. Realizing repeated quantum error correction in a distance-three surface code. *Nature* **2022**, *605*, 669–674. [CrossRef] [PubMed]
108. Sivak, V.V.; Eickbusch, A.; Royer, B.; Singh, S.; Tsioutsios, I.; Ganjam, S.; Miano, A.; Brock, B.L.; Ding, A.Z.; Frunzio, L.; et al. Real-time quantum error correction beyond break-even. *Nature* **2023**, *616*, 50–55. [CrossRef] [PubMed]
109. Einarsson, G. Probability Analysis of a Quantum Computer. *arXiv* **2003**, arXiv:quant-ph/0303074.
110. Press, W.H.; Teukolsky, S.A.; Vetterling, W.T.; Flannery, B.P. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*; Cambridge University Press: New York, NY, USA, 2007.
111. Jaynes, E.T.; Bretthorst, G.L. *Probability Theory: The Logic of Science*; Cambridge University Press: Cambridge, MA, USA, 2003. [CrossRef]
112. Nicolas, J.L. Petites valeurs de la fonction d'Euler. *J. Number Theory* **1983**, *17*, 375–388. [CrossRef]
113. Rosser, J.B.; Schoenfeld, L. Approximate formulas for some functions of prime numbers. *Illinois J. Math.* **1962**, *6*, 64. [CrossRef]
114. Nielsen, M.A. Errata List for "Quantum Computation and Quantum Information". 2014. Available online: https://michaelnielsen.org/qcqi/errata/errata/errata.html (accessed on 18 September 2023).
115. Carmichael, R.D. Note on a new number theory function. *Bull. Am. Math. Soc.* **1910**, *16*, 232–238. [CrossRef]