PAPER • OPEN ACCESS

High performance Boson sampling simulation via data-flow engines

To cite this article: Gregory Morse *et al* 2024 *New J. Phys.* **26** 033033

IOP Publishing New J. Phys. 26 (2024) 033033 https://doi.org/10.1088/1367-2630/ad313b

New Journal of Physics
The open access journal at the forefront of physics

Deutsche Physikalische Gesellschaft DPG
IOP Institute of Physics

Published in partnership with: Deutsche Physikalische Gesellschaft and the Institute of Physics

OPEN ACCESS

RECEIVED
17 December 2023

REVISED
27 February 2024

ACCEPTED FOR PUBLICATION
7 March 2024

PUBLISHED
19 March 2024

PAPER

High performance Boson sampling simulation via data-flow engines

Gregory Morse[1], Tomasz Rybotycki[2,3,4], Ágoston Kaposi[1,5], Zoltán Kolarovszki[1,5], Uroš Stojčić[6], Tamás Kozsik[1], Oskar Mencer[6], Michał Oszmaniec[7,8], Zoltán Zimborás[1,5,9] and Péter Rakyta[5,10,*]

[1] Department of Programming Languages and Compilers, Eötvös Loránd University, Pázmány Péter sétány 1/a, Budapest 1117, Hungary
[2] AstroCeNT—Particle Astrophysics Science and Technology Centre—International Research Agenda, Nicolaus Copernicus Astronomical Center, Polish Academy of Sciences, Warsaw, Poland
[3] ACC Cyfronet, University of Science and Technology, Cracow, Poland
[4] Systems Research Institute, Polish Academy of Sciences, Warsaw, Poland
[5] Wigner Research Center for Physics, 29-33 Konkoly-Thege Miklos Str, Budapest 1121, Hungary
[6] Maxeler Technologies, a Groq company, 16192 Coastal Hwy, Lewes 19958, DE, United States of America
[7] Center for Theoretical Physics, Polish Academy of Sciences, Warsaw, Poland
[8] NASK National Research Institute, Kolska 12, Warszawa 01-045, Poland
[9] Algorithmiq Ltd, Kanavakatu 3 C, Helsinki 00160, Finland
[10] Department of Physics of Complex Systems, Eötvös Loránd University, Pázmány Péter sétány 1/a, Budapest 1117, Hungary
[*] Author to whom any correspondence should be addressed.

E-mail: rakyta.peter@wigner.hu

Keywords: boson sampling simulation, FPGA, permanent, data-flow, lossy boson sampling

Supplementary material for this article is available online

Abstract

Boson sampling (BS) is viewed to be an accessible quantum computing paradigm to demonstrate computational advantage compared to classical computers. In this context, the evolution of permanent calculation algorithms attracts a significant attention as the simulation of BS experiments involves the evaluation of vast number of permanents. For this reason, we generalize the Balasubramanian–Bax–Franklin–Glynn permanent formula, aiming to efficiently integrate it into the BS strategy of Clifford and Clifford (2020 Faster classical boson sampling). A reduction in simulation complexity originating from multiplicities in photon occupation was achieved through the incorporation of a n-ary Gray code ordering of the addends during the permanent evaluation. Implementing the devised algorithm on FPGA-based data-flow engines, we leverage the resulting tool to accelerate boson sampling simulations for up to 40 photons. Drawing samples from a 60-mode interferometer, the achieved rate averages around 80 s per sample, employing 4 FPGA chips. The developed design facilitates the simulation of both ideal and lossy boson sampling experiments.

1. Introduction

The main idea behind quantum supremacy [1–3] is to use a quantum processor to solve a mathematical problem that is intractable for the classical computers, that is the solution of the problem on classical hardware would require resources (like the execution time) scaling exponentially with the problem size [4–11]. Undoubtedly, it is fundamental for the researchers to ascertain that the quantum device used in the experiment indeed solves its designed task in the way it is expected. Without such confirmation, one could not measure the strength of the quantum supremacy claims. For this reason, the development of powerful validation protocols is of high importance.

One of the proposed quantum algorithms to surpass classical computing devices is the so-called boson sampling (BS) [4, 5] in which bosons are drawn from a distribution of indistinguishable particles. In case the relation $m \gg n^2$ between the number of the output modes m and the number of input photons n holds on, the total execution time of the simulation scales exponentially with n, since there are no efficient classical

methods to simulate the quantum correlations originating from the peculiar nature of multi-body bosonic systems consisting of indistinguishable particles. For this reason, BS has been subjected to numerous theoretical and experimental investigations. Due to the intensive interest shown towards this topic, several flavors of BS have been formulated such as the Gaussian BS [12–14], scattershot BS [15–17], translating BS into time domain [18, 19], or implementing the BS with trapped ion technology [20, 21], ultra cold atoms [22] and with microwave cavities [23]. From an experimental point of view, after the proof-of-principle realizations of small-scaled photonic interferometers [24–27], researchers were experimenting to increase the number of modes on the photonic chip [28] and by multiplexing on-chip single-photon sources [29–31]. Currently, the largest conventional BS experiment was reported in the work of Wang *et al* [32] via a 60 mode interferometer with 20 input photons and 14 measured photons.

The period between 2020 and 2023 was especially fruitful for outstanding achievements in the experimental realization of BS from Gaussian states [33–35]. These experimental designs turned to be inaccessible to be simulated within a reasonable time frame even for the most efficient exact algorithms developed to simulate Gaussian BS [36–38]. Approximate samplers [37, 39, 40], while for smaller systems can generate faked samples being closer to the ideal distribution than the experiments, they also become computational too expensive at these measures.

Because of the dedicated race to experimentally demonstrate proven quantum advantage the validation protocols also increased in their importance [40]. Shortly after the BS proposal researchers tried to find a way to verify the results of the samplers. Initially, the validators used statistical tests to show that the samplers draw samples from distribution being different from classical counterparts [41] such as the mean-field approach or uniform distribution. During recent years, we could see the emergence of more sublime BS validation approaches [42, 43]. Some of those new ideas arise from the techniques used in a different branch of computer science, like pattern recognition [44] or computer vision [45]. These validators require permanent computation or access to the samples drawn from a bona fide boson sampler, which means they can benefit from a high-performance permanent computation technique.

From a mathematical point of view, the key element in the simulation of BS is a fast evaluation of the permanent function. The reason for that is the connection between the permanents of specific matrices and probabilities of BS outcomes. Thus, in BS simulation many permanent computations are required even with the most efficiently known Clifford and Clifford algorithm [46, 47]. The permanent of an $n \times n$ matrix A is defined by the formula

$$\mathrm{perm}\,(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} a_{i,\sigma(i)}, \tag{1}$$

where S_n is a set of non-repeating n element permutations of $S = \{1, \ldots, n\}$. The formula has a factorial time complexity of $\mathcal{O}(n!n) \leqslant \mathcal{O}(n^{n+1})$. Valiant [48] proved that the evaluation of the permanent of a matrix containing only zeros and ones belongs to complexity class #P (sharp P)-complete, a class which is at least as hard as NP-complete with many counting problems e.g. perfect matchings, graph colorings of size k, satisfiable assignments to general Boolean formula, etc. Unlike the determinant having quite a similar definition to equation (1), there is not any linear algebra simplification to calculate the permanent in polynomial time. Currently, the most efficient approaches to calculate the permanent have a computational complexity of $\mathcal{O}(n^2 \cdot 2^n)$ which can be further reduced to $\mathcal{O}(n \cdot 2^n)$ if data recycling of intermediate results is implemented via Gray code ordering. The $\mathcal{O}(n^2 \cdot 2^n)$ variants of the so-called Ryser [49] and the Balasubramanian–Bax–Franklin–Glynn (BB/FG) [50] formulas were benchmarked on the Tianhe 2 supercomputer in [51], implying better numerical accuracy of the BB/FG method.

In this work, we designed a novel scalable implementation to calculate the permanent via the BB/FG formula. In order to account for a collision of photons on the optical modes of the interferometer we introduce an extended version of the BB/FG formula based on the concept of the n-ary Gray code ordering [52] to take into account data repetition during the permanent evaluation and improve the performance of the BS simulation. We also implemented the designed computational model on FPGA-based data-flow engines (DFEs) further improving the computational performance of the BS simulation provided by the Piquasso package [53]. The so-called data-flow programming model [54] utilizes a fundamentally different approach to increase the computational concurrency than traditional parallel programming models on GPU and CPU hardware. The basic concept of data-flow programming can be explained by thinking of a stream of computational data flowing through hardware. Each of the utilized resources performs a fixed logical transformation on the elements of the stream, transforming a single data element in each clock cycle. By chaining up hardware resources we end up with a data-flow computing model: while one hardware element is transforming the ith element in the data stream, the next hardware element in the chain is already working on the previously transformed, $i-1$th element of the stream. By providing a long enough data stream to the

hardware one can realize an efficient parallel execution, even if the computational tasks have a high degree of data dependency.

The high-level DFE development framework of Maxeler Technologies provides an efficient programming background to design data-flow applications in terms of high-level building blocks (such as a support for fix-point arithmetic operations with complex numbers, automatic pipeline scheduling, stream-holds, memory controllers, etc) instead of the lingering work of programming low-level VHSIC Hardware Description Language (VHDL) components. By combining a novel permanent computational approach based on the BB/FG formula with the data-flow programming model we developed a permanent computing DFE capable of supporting exact BS simulations (with and without photon losses) up to 40 photons. The computational complexity of the BS simulation might be significantly reduced if photon occupation multiplicities on the optical modes are taken into consideration. Our DFE permanent calculator is adapted to this generalization by streaming the addends in n-ary Gray code ordering directly on the FPGA chip. With this accessory, it took ∼90 s per sample to draw 40 photon samples from a random 60 mode interferometer via four concurrently operating DFEs.

The manuscript is organized as follows: in section 2 we discuss the n-ary Gray code implementation to evaluate the permanent function with the BB/FG formula, accounting for photon occupation multiplicities on the photonic modes. Then, in section 3 we describe the DFE implementation of the permanent calculation engine developed for FPGA chips. Finally, in section 4 we provide the results of our numerical experiments on the classical simulation of BS incorporating DFE permanent calculation implementations.

2. Evaluation of the permanent function: a novel scalable approach

The two lowest complexity methods of computing the permanent involve the formulas of Ryser [49] and BB/FG [50] when using Gray code ordering of the addends. Ryser's approach computes the permanent as

$$\mathrm{perm}\,(A) = (-1)^n \sum_{S\supseteq\{1,\ldots,n\}} (-1)^{|S|} \prod_{i=1}^{n} \sum_{j\in S} a_{i,j}. \tag{2}$$

The formula has an exponential computational complexity of $\mathcal{O}(n^2\cdot 2^n)$, which is significantly reduced when the subsets $S\supseteq\{1,\ldots,n\}$ are ordered in a way that only a single element of S is changed in the subsequent S. In this case, the value of the inner sum of the matrix elements can be reused, i.e. only a single matrix element should be added or subtracted from the reused sum to obtain the new value corresponding to the next S. By reducing the complexity of the inner sum from $\mathcal{O}(n)$ to $\mathcal{O}(1)$ this way, the Ryser formula can be evaluated by an overall $\mathcal{O}(n\cdot 2^n)$ complexity. By a later technique reported in [55] the complexity can be further reduced by a factor of 2 in the outer sum using the expression:

$$\mathrm{perm}\,(A) = (-1)^{n-1}\cdot 2\cdot \sum_{S\supseteq\{1,\ldots,n-1\}} (-1)^{|S|} \prod_{i=1}^{n} \left(x_i + \sum_{j\in S} a_{i,j} \right), \tag{3}$$

where x_i is defined by $x_i = a_{i,n} - (a_{i,1} + a_{i,2} + \cdots + a_{i,n})/2$. Another highly efficient method to calculate the permanent is provided by the BB/FG formula:

$$\mathrm{perm}\,(A) = \frac{1}{2^{n-1}} \sum_{\delta} \left(\prod_{k=1}^{n} \delta_k \right) \prod_{j=1}^{n} \sum_{i=1}^{n} \delta_i a_{i,j}, \tag{4}$$

where $\delta = (\delta_1, \delta_2, \ldots, \delta_n)$ with $\delta_1 = 1$ and $\delta_i = \pm 1$ for $1 \leqslant i \leqslant n$. Notice, that in contrast with the traditional definition of the BB/FG formula, in the inner sum of equation (4) we rather calculate the column sums of the input matrix instead of row sums. This choice is motivated by a practical reason implied by the following reasoning: regarding the unitary matrix describing the scattering in the optical interferometer, the rows of the unitary are associated with the output states, while the columns are related to the input modes. Non-trivial photon multiplicities on the output modes (expected to occur more often than multiplicities in the input modes) are described by repeated rows in the unitary. As we will show in section 2.2, by accounting for these multiplicities one can significantly reduce the complexity of the permanent evaluation.

The computational complexity of the BB/FG formula is $\mathcal{O}(n^2\cdot 2^{n-1})$, while it can be reduced to $\mathcal{O}(n\cdot 2^{n-1})$ if Gray code ordering is applied in the evaluation of the outer sum. The Ryser and the BB/FG formulas follow quite different approaches to evaluate the permanent, also resulting in dissimilar numerical properties. In the benchmark comparison of the two methods reported in [51] the authors found that the BB/FG formula gives numerically more precise results than Ryser's formula in the context of bounded

Table 1. An example showing a 3-bit reflected Gray code sequence. In each row a single bit is changed in the Gray code compared to the previous row.

Decimal index	Gray code
0	000
1	001
2	011
3	010
4	110
5	111
6	101
7	100

bit-sized data types. This finding was further justified by our numerical analysis reported in section 3.2 comparing the numerical accuracy of the individual algorithms with the numerical results obtained by multiple-precision floating-point number arithmetics provided by the GNU MPFR library [56].

2.1. Parallel BB/FG implementation to calculate the permanent

In this section we discuss the technical foundations of our CPU implementation to calculate the permanent of a square matrix, that can be further improved to account for photon multiplicities discussed in section 2.2. Our algorithm implementing a Gray code ordered BB/FG formula (4) has a computational complexity of $\mathcal{O}(n \cdot 2^{n-1})$ while minimizing the overhead of processing the logic associated with the generation of auxiliary data needed for the Gray code ordering. Table 1 shows an example of a so-called binary reflected Gray code sequence encoded by 3 bits. Via Gray code ordering in each cycle of the outer sum of equation (4) only one element of the $\boldsymbol{\delta}$ vector is changed, making it possible to reuse the column sum (i.e. the inner sum in equation (4)) in the next cycle and subtract/add only elements of a single row of the input matrix (see [51] for details).

Here we note some important properties of reflected Gray code counting allowing us to efficiently implement the algorithm in a parallel environment. First of all, the Gray code corresponding to a decimal index $0 \leqslant i < 2^{n-1}$ is $g_i = i \oplus (i >> 1)$ where $\oplus$ is a bit-wise logical XOR operation and $>>$ is a bitshift operation. The ability to determine the Gray code corresponding to any decimal index i in a constant time implies an efficient way to evaluate the permanent in parallel execution. Simply, we divide the set of $0 \leqslant i < 2^{n-1}$ decimal indices into smaller contiguous subsets that can be processed concurrently on the available CPU threads. For the first element in the subset, the column sum is initialized with n arithmetic operations. Whether an $a_{i,j}$ element is subtracted or added to the column sum is determined by the elements δ_i which are mapped to the individual bits of the Gray code: δ_i is considered to be $+1$ (-1) if the corresponding Gray code bit is 0 (1). After initializing the initial column sums on each CPU thread, additional addends to the permanent are calculated via sequential iterations on the elements of the given subset of the decimal indices $\{i_{\min}, \ldots, i_{\max}\}$. To this end, we need to determine the changing bit in the Gray code and its position. In principle, the bit mask of the changed bit is given by a bit-wise comparison of the Gray code with its prior value $g_i \oplus (g_i >> 1) \oplus (g_i - 1) \oplus ((g_i - 1) >> 1)$. However, according to the generation rule of the Gray code g_i from the decimal index i, we can use a more efficient way to determine the changed bit. Namely, in each cycle, the position of the changed bit in the Gray code is determined by the position of the lowest bit in the decimal index which is 1. Also, since in each iteration only one element of the Gray code is changed we can keep track of the "parity" of the $\boldsymbol{\delta}$ vector (i.e. whether the sum of the elements δ_i is odd or even) from cycle to cycle in an efficient way without iterating over the elements of the vector. We provide a reference implementation of our recursive permanent algorithm in the Piquasso Boost library [57] providing high-performance C++ computing engines for the Python-based universal bosonic quantum computer simulator framework Piquasso. We also notice that the described method can be used to scale up the computation of the permanent over multiple computing nodes (using Message Processing Interface (MPI) protocol), so in contrast with the claim of [51] the Gray coded permanent evaluation can be efficiently turned into parallel execution.

2.2. Handling row and column multiplicities in the input matrix

In the general case, multiple photons might occupy the same optical mode at the output of the interferometer. Since the output modes are associated with the row indices of the unitary describing the scattering process of the photons, the multi-photon collision at the output modes is mathematically described by repeated rows in the unitary. In particular, the number for which a specific row is repeated in the unitary is identical to the number of photons occupying the corresponding optical mode. Following the

permanent evaluation strategy encoded by the BB/FG formula it becomes clear that by having row multiplicities in the input matrix some of the addends to the permanent would show up multiple times during the calculation. Such a complexity reduction was already reported in several previous works of [47, 58, 59] by generalizing the Ryser formula, or [60] by turning the permanent calculation into the evaluation of the Hafnian function accounting for multiplicities [36]. Here we argue that it is possible to make use of the addend multiplicities and reduce the overall complexity of the BB/FG permanent calculation as well. According to our best knowledge, this improvement to the BB/FG algorithm was not reported before.

For example, let us assume that the kth output mode ($k>1$) is occupied by M_k photons, resulting in M_k identical rows in the unitary at row indices $i_k, i_k+1, \ldots, i_k+M_k-1$. Consequently, the

$$\sum_{i=i_k}^{i_k+M_k-1} \delta_i\, a_{i,j} = \left(\sum_{i=i_k}^{i_k+M_k-1} \delta_i\right) a_{i_k,j} = (M_k - 2\Delta_k)\; a_{i_k,j} \tag{5}$$

column sum might result in M_k+1 different outcomes depending on the number Δ_k of -1 values and the number $M_k-\Delta_k$ of $+1$ values among the δ_i ($i_k \leqslant i \leqslant i_k+M_k-1$) elements of the $\boldsymbol{\delta}$ vector. The individual outcomes occur explicitly $\binom{M_k}{\Delta_k}$ times during the permanent evaluation. Taking the M_k multiplicities for each optical output mode labeled by k, in total

$$C = \prod_{k=1}^{\#modes} (M_k+1) \tag{6}$$

different outcomes of the inner column sum show up during the calculation, determining the overall complexity of the permanent evaluation. According to the BB/FG formula, the first row ($k=1$) of the matrix is always taken by a coefficient $\delta_1=1$, hence we always treat the first row with a multiplicity of 1, even if it is repeated in the matrix. Computationally the best practice is to move one of the non-repeating rows (if exists) to the top of the input matrix. The possible computational speedup achieved via the outlined combinatorial simplification depends on the specific use case. The exponential factor in the evaluation complexity is determined by the number of the involved rows of the input matrix. In general, the complexity of the calculations can be reduced to

$$\mathcal{O}\left(n\cdot 2^{M^*}\right), \qquad M^* = \log_2(C) = \sum_{k=1}^{\#modes} \log_2(M_k+1), \tag{7}$$

with $n=\sum M_k$ over the optical modes. In order to account for row multiplicities within the BB/FG formula we make use of reflected n-ary Gray code counting instead of the binary Gray code counting described in section 2.1:

$$\mathrm{perm}(\boldsymbol{A},\boldsymbol{M},\boldsymbol{N}) = \frac{1}{2^{n-1}}\cdot\sum_{\boldsymbol{\Delta}}\left[\left(\prod_{k=1}^{\#modes}(-1)^{\Delta_k}\binom{M_k}{\Delta_k}\right)\prod_{j=1}^{\#modes}\left(\sum_{k=1}^{\#modes}(M_k-2\Delta_k)\,a_{k,j}\right)^{N_j}\right], \tag{8}$$

where $\boldsymbol{A}=a_{ij}$ is a square matrix describing the interferometer, $\boldsymbol{M}$ and $\boldsymbol{N}$ are the row and column multiplicities respectively such that the photon count $n=\sum M_i=\sum N_j$ and $\boldsymbol{\Delta}$ is the n-ary Gray code, required for efficient computation. (For the physical meaning of Δ_k see equation (5)) The n-ary Gray code is also known as the non-Boolean Gray code or Guan code [61]. As the name implies, the individual "digits" of the n-ary Gray code are encoded by numbers from $0,1,\ldots,n-1$ instead of the binary values 0 and 1. While the limits of the individual Gray code digits can be different from each other, one can construct a specific n-ary Gray code counter in which the kth digit counts the number of $+1$ values of the δ_i elements corresponding to the repeated rows describing the kth output mode. Thus, according to our reasoning, the kth digit of the Gray code counts from 0 to M_k. In order to construct such a reflected n-ary Gray code counter, we followed the work of Kurt *et al* [52]. In each iteration, only a single digit changes its value, enabling one to reuse the calculated column sums in the next iterations, similarly to the case of binary-reflected Gray code ordering.

Due to the reflected nature of the counter, it is possible to determine the Gray code corresponding to a specific decimal index $0\leqslant i<C$ in a constant time (for details see [52]). Thus, the algorithm can be executed in parallel in a similar fashion then we described it for a binary-reflected Gray code counter. In the Piquasso Boost library, we provide our implementation of the BB/FG algorithm accounting for row multiplicities.

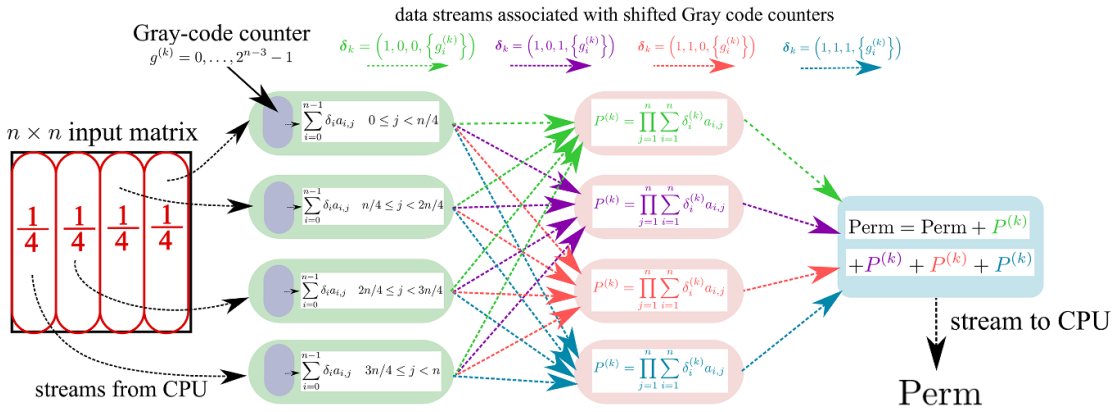

Figure 1. The structure of the computing kernels realized on the FPGA chip. The Xilinx Alveo U250 chip is organized into four super logic regions (SLRs) [62], with limited inter-SLR connectivity. The kernels are organized accounting for this specific hardware setup introducing 4 concurrent kernel blocks (mostly) bounded to the individual SLRs. The individual kernels are indicated with colored regions, also showing the mathematical operation they are evaluating. Further details of the data-flow scheme are discussed in section 3.

3. DFE design and implementation to evaluate the permanent

In order to further increase the computational speed of permanents, we developed a DFE implementation of the algorithm described in the previous section. Here in this section, we discuss the technical aspects of the developed FPGA (Field Programmable Gate Arrays) based permanent calculation engines realized via a static data-flow programming model. Since the configuration of the programmable hardware elements on the FPGA chip (or in other words the uploading of the program to the FGPA card) is time-consuming, the implementation needs to be general enough to avoid the need to re-configure the FPGA card during the BS runtime. In addition, by accounting for the particular needs implied by physics working behind the simulated BS architecture, we encountered further optimization requirements for the DFE implementation, such as the possibility to calculate the permanents of multiple matrices in one shot to amortize the initialization overhead even at small matrix sizes. In order to provide a thorough description of our implementation, while sparing the reader from too many low-level technical details, we will discuss these optimization details separately in the supplementary material. Here we rather focus on the description of the basic concept to evaluate permanents on DFEs.

Due to high resource requirements associated with floating point operations in the FPGA chip, we focused on designing a scheme where fixed point number representation was used to do arithmetic operations with gradually increasing the bitwidth of the number representation from the initial 64 bits (sufficient to represent the elements of the input unitary) up to 128 bits used to derive an accurate final result up to a unitary size of 40×40. (While the floating point units on modern CPUs are highly specialized and optimized, the FPGA with the aid of generic look-up tables (LUTs) and digital signal processing (DSP) units scales better with fixed point operations. For example, double precision floating point additions require a delay of 14 clock ticks, while 1 tick is sufficient for fixed point addition.) Starting with 64 bit double precision format of the input unitary on the CPU side, we perform a conversion of floating point numbers ($f = s \cdot 2^e$) into fixed point representation encoded by $b = 64$ bit variable a, such that a is the nearest integer to $f \cdot 2^{b_f}$, with $b_f = b - 2$ standing for the number of fractional bits. The remaining two bits are used to store the integer part (for the case if the matrix element is unity) and the sign of a in two's complement encoding. (Since the input matrix is expected to be unitary, the magnitude of the matrix elements would never be larger than unity.)

In order to ease the congestion on the FPGA chip and avoid timing issues associated with long routing paths, we split the input matrix into four blocks (see the left side of figure 1), and stream the column blocks into 4 distinct 'column summing' DFE kernels indicated by green colored boxes in figure 1). This way the computations could be partitioned over the Xilinx Alveo U250 FPGA chip used in our numerical experiments while increasing the computational concurrency at the same time. The purpose of these kernels is to (i) calculate the initial column sums $\sum \delta_i a_{i,j}$ for each column in the first n tick of the kernel (here n is the matrix size). Secondly, (ii) as the initial column sums are calculated, a gray code counter $g^{(k)} \in (0, 2^{n-3} - 1)$ is launched (in the nth clock tick) to stream the last $n - 3$ elements of the $\boldsymbol{\delta}^{(k)}$ vectors defined in the BB/FG permanent formula of equation (4). (The corresponding δ_i elements are given by the individual bits of $g^{(k)}$. Further details to generate the gray code counter logic via bit-wise operations are

discussed in section 2.1). The first element of each $\boldsymbol{\delta}^{(k)}$ vector is 1 by the definition, while the second and third elements become fixed by the following reasoning: in order to increase the computational concurrency on the DFE the gray code counter is multiplexed in order to create 4 concurrent streams of the $\boldsymbol{\delta}^{(k)}$ vectors by fixing the second and third elements to $(0,0)$, $(0,1)$, $(1,0)$ or $(1,1)$ as indicated by the colored $\boldsymbol{\delta}^{(k)}$ streams in figure 1. Consequently in each of the 4 'column sum' kernels 4 concurrent $\boldsymbol{\delta}^{(k)}$ vector streams are used to calculate the $\boldsymbol{\delta}^{(k)}$ weighted column sums. In each clock cycle 4 new column sums are concurrently calculated for each column index $0 \leqslant j < n$ (distributed between the 4 column sum kernels) corresponding to the 4 multiplexed $\boldsymbol{\delta}^{(k)}$ streams: from the value of the most recent column sum the new value is determined by adding or sub-tracking twice the matrix element taken from row i. The row index i corresponds to the element index where a change occurred in $\boldsymbol{\delta}^{(k)}$ compared to $\boldsymbol{\delta}^{(k-1)}$ (due to the design this index is the same for all of the multiplexed $\boldsymbol{\delta}^{(k)}$ streams).

The calculated column sum streams are gathered into groups according to the streams $\boldsymbol{\delta}^{(k)}$ to provide an input for the next layer of computing kernels. The red-colored kernels in figure 1 calculates the products of the incoming column sum streams (each of them corresponding to a specific column index $0 \leqslant j < n$), i.e. the stream data arriving at the same clock cycle are multiplied with each other over a binary tree reduction with a depth of $\lceil \log_2 n \rceil$.

The most expensive operations in the design are the multiplications of the complex numbers associated with the column sums. According to the most widely used approach, the complex multiplications are broken down as 4 fixed point multiplications and 2 additions as $(a+bi)(c+di) = (ac-bd) + (ad+bc)i$. The formula suggested by Knuth in [63] allows the computation of the product as $(c(a+b) - b(c+d))$ $+(c(a+b) + a(d-c))i$ which can be implemented with 3 multiplications and 5 additions. There is also the Karatsuba-style formula of $(ac-bd) + ((a+b)(c+d) - ac - bd)i$ which Knuth credits to Peter Ungar from 1963. This approach uses the same amount of operations as the prior one. From a pipelining perspective on the FPGA chip, the first formula is more balanced as the multiplications and additions to get the real and imaginary parts can be implemented concurrently. However, it was shown in work [64] that the numeric stability of the Ungar formula is better, as only the imaginary part would be affected by additional inaccuracy. Fortunately, in the context of fixed point numbers used in our design, this aspect is of less importance. It should be also noted that the addition resulting in the final real and imaginary components would fall within the range $[-2,2]$ requiring an extra leading bit in the intermediate computations. In addition, the fixed point multiplications need to be further broken down to deal with efficient tiling on the hardware units of the FPGA chip, such as digital signal processing (DSP) multipliers.

In our implementation, we followed the pioneering results of [65–67]. According to [65] one can use the Karatsuba multiplication formula by optimally splitting the input multiplicands into smaller bitwidth parts being the least common multiple of the width of the utilized input bit ports of the DSP units. (Thus, the most optimal selection of the tiling factors depends on the width of the input multiplicands.) A slightly different approach is to work out a DSP-number optimized recursive Karatsuba-like formula for the individual input bit size configurations by following the reasoning of [66, 67] being applicable for rectangular-sized tilings. (The Virtex 5 DSP units embedded inside our Xilinx Alveo U250 FPGA cards have 18×25 wide input ports to perform signed multiplications.) Since our DFE implementation is designed to handle at most 40×40 matrices, one needs to calculate the product reduction of 40 complex numbers. The tree-like reduction is performed over 6 levels, providing a balance between resource utilization and pipelining. On the first level 20 products are calculated from 64 bit wide input fixed point values (streamed from the columns sum kernels) resulting in 79 bit results (the remaining bits are discarded). On the second level, the 20 results are paired up to 10 pairs. The multiplications are performed again with 79 bit-wide results. The third level calculates the pair-wise product of 10 numbers resulting in 93 bit results. The remaining 3 levels to reduce the final 5 numbers are done with results of 110, 158 and 189 bit precision. All values are fixed points with 2 integer bits (including the sign bit) and the remaining as fractional bits. The final values are all accumulated at 192 (or 256) bits of precision, 6 of which are integer bits. During the development, numerous technical details were addressed to end up with an efficient strategy to multiply large bitwidth complex numbers implemented in our calculations. We solved many issues related to the correct management of the fan-outs of intermediate computations or to the careful choice between LUTs or DSP units to perform multiplications on small bitwidth tiles in order to reach the limits of our design being still decomposable on the FPGA chip. The most cumbersome limiting factor was the requirement to keep up with the numerical accuracy comparable to the CPU implementations using extended precision floating point number representation (see section 3.2 for the comparison of DFE implementation to the CPU ones). To this end, we needed to increase the bitwidth of the fixed point numbers in the multiplication binary tree as much as possible by utilizing more and more hardware resources, while keeping up with the timing constraints by designing a suitable pipeline structure on the chip.

Finally, the results of the four product kernels are streamed toward the final 'sum up' kernel indicated by the blue-colored region in figure 1. Due to the multiplexed four gray code streams four different column product reductions are entering the kernel in each tick. The incoming streams are summed up and added to the partial permanent labeled by *Perm* in figure 1. To sum up the permanent addends into a single scalar result, a single clock tick deep data-flow loop is designed (corresponding to the addition of two fixed point numbers) by removing any registers from the underlying logic. Finally, the result encoded by a $128 + 128$ bit complex number is streamed back to the CPU.

Beyond the discussed layout issues, many other optimizations were implemented to increase the potential usability of the developed permanent calculator DFE in quantum computing simulations. The most important issue in our work was to generalize the implementation for matrices of variable size. Since the initial uploading of the DFE program onto the FPGA card takes about ~ 1 minute, using the engine in realistic BS simulations without this generalization is infeasible. In section S1 of the supplementary material, we provide the details of our solution to generalize the DFE implementation to calculate the permanent of arbitrary-sized matrices. By preserving the computational accuracy (see section 3.2), however, the maximal size of the input matrix for which the layout could still fit onto the chip turned to be 48×48 at clock frequency 300 MHz (and 330 MHz at maximal matrix size of 40×40). For cases where the FPGA would not tick enough to get a result, i.e. the input matrix is smaller than 3×3, the permanent computations are performed solely on the CPU side. (While such cases could be computed in one tick on the FPGA with some additional logic, the communication delay with DFE would likely exceed the cost of the computation by the CPU side.) Also, we would like to highlight at this point another important optimization of the design. While feeding the permanent calculator DFE with more matrices sequentially (i.e. one after another) in a single execution, one can retrieve the calculated permanents for all of the matrices in one DFE execution. Since the execution of our program on the DFE takes up an overhead of about a millisecond (including I/O data transfer and other initialization overhead while the DFE program is already uploaded to the FPGA card), it is expected to provide a further advantage for the DFE, if one can calculate the permanents for more matrices in a single execution amortizing the overhead between the matrices. Following this ideology, in section S4 of the supplementary material we show that significant speedup can be realized in permanent evaluation even for small matrices, speeding up the simulation of BS if a large number of smaller matrices needs to be processed.

Depending on the specific needs of individual simulation tasks, it might be advantageous to introduce more concurrency into a single evaluation of permanents by splitting the calculations between multiple DFEs. This might be especially useful when dealing with larger matrices (up to 48×48 in our case) during the calculations. In section S2 of the Supplementary material we provide further details on the technical background related to scaling up permanent calculation over multiple DFEs. In order to utilize DFEs to support the simulation of photonic quantum computers (when multiple photons can collide onto the same output mode), one also needs to implement the necessary logic accounting for row multiplicities reducing the permanent calculation complexity. This optimization is discussed in section 2.2.

Finally, we can not finish this section without mentioning that in order to prevent integer overflow during the arithmetic operations on the DFE, we need to keep all of the intermediate results of calculations within certain bounds correspondingly to the limits of the used fixed point number representations. This can be achieved by a well-formulated normalization strategy of the input matrices, while the normalization factors can be used to re-scale the output result received from the DFE to end up with the correct result of permanent. We provide further details on the applied normalization strategy in the section S5 of the supplementary material.

3.1. Performance benchmark of the permanent calculator implementations

In this section, we provide the details regarding our performance benchmark of the DFE permanent calculator implementation compared to the fastest CPU implementations available today. The numerical experiments were conducted using Xilinx Alveo U250 FPGA cards implementing a bitstream generated from Very High Speed Integrated Circuit Hardware Description Language (VHDL) generated from Java source by Maxeler's MaxCompiler 2021.1. Thus the heart of the developed permanent calculator DFE is a static data-flow computation model translated into V4HDL by the MaxCompiler. In the performance benchmark, we compared the DFE implementation to the Piquasso Boost library's BB/FG parallel C++ implementation and TheWalrus v0.16.2 library's Ryser double and quad precision (implementing 128 bit floats when GNU or Intel compiler is used to build the source, and long double precision for other compilers). The measurements were computed by averaging the execution time of 5 computations for matrices of size smaller than 24×24, otherwise, just a single execution measurement was taken.

The CPU computations were performed on a two-socket CPU system consisting of AMD EPYC 7542 32-Core processors with two logical threads on each core and two non-uniform memory access (NUMA) nodes where we bound the computations to one of the NUMA nodes. (Thus the CPU benchmark was

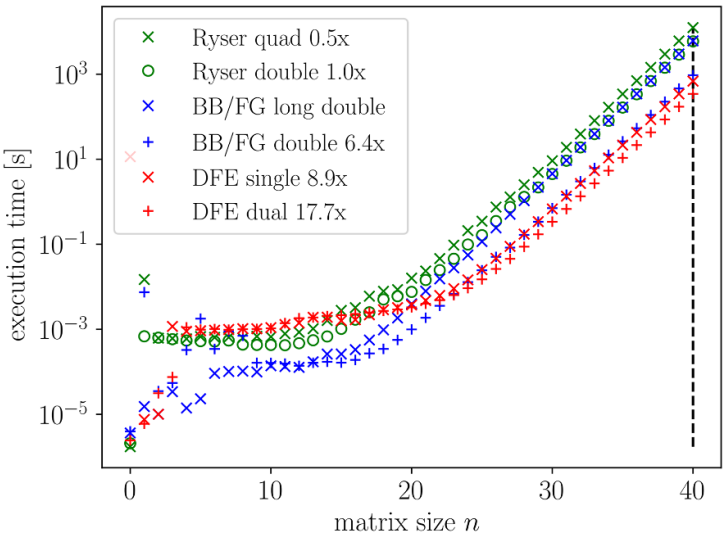

Figure 2. Performance comparison of the permanent calculator DFE to the CPU side BB/FG implementations. The input unitaries are constructed randomly occupying a single photon on each optical mode. As the photon count increases above 20, the DFE implementation becomes systematically faster than CPU implementations. The speedup factors compared to the long double precision BB/FG implementation are indicated in the legend.

measured by the performance on 64 computing threads.) The FPGA communication uses a high-speed PCI Express 3.0 channel over which the input matrix and series of other input parameters are uploaded, like scalar initialization parameters for each kernel, clock synchronization data, etc. The frequency of the DFE implementation was 320 MHz The initialization time of programming the DFE when uploading the compiled circuit from the generated VHDL program takes 56.2 s for a single DFE while it takes 112.9 s for a dual DFE mode (i.e. splitting the permanent evaluation over two DFEs). The uploaded bitstream program is preserved on the DFE until it is reprogrammed, hence the initial uploading time is needed to be spent only at the very start of the usage. The initialization of the TheWalrus library is around 6 milliseconds, while the BB/FG implementation of the Piquasso Boost library exhibits the most negligible loading delay with about 19.5 microseconds.

We compared the performance of our implementation provided in the Piquasso Boost library to the implementation of TheWalrus version 0.16.2 [60] package also having implemented parallelized C++ engines to evaluate the permanent function. (Newer versions on TheWalrus do not contain C++ engines, nor extended precision implementations.) The results are plotted in figure 2 showing the performance of the different libraries. The red-colored data points show the execution time of the single and dual DFE implementations. (For details related to the dual mode see section S2 of the supplementary material.) For matrices of size less than 20×20 the initialization overhead dominates the execution of the DFE resulting in a constant execution time. (The permanent evaluation of the smallest matrices is done solely on the CPU side explaining the first 3 data points in the set.) Above the crossover region, the advantage of the DFE execution becomes evident. Only the double precision BB/FG implementation comes close to the performance of the DFE. However, while the single DFE implementation is only about 1.4 times faster than the double precision BB/FG implementation, the accuracy of the DFE implementation is significantly better, having better or identical accuracy as the long double precision BB/FG implementation up to a matrix size 40×40. (For details see section 3.2.) The long double precision CPU implementations are significantly slower than the DFE (see the indicated speedup factors in the legend of figure 2), implying the advantage of DFE over CPU implementations when high numerical accuracy is required. The total run-time to evaluate the permanent of an $n \times n$ unitary on a single ($k = 2$) and dual ($k = 3$) DFE can be calculated as $t = t_0 + \frac{n-1+2^{n-1-k}}{f}$ where t_0 is a small PCI Express communication delay plus the pipeline delay of the kernels (approximately half a millisecond), and f is the frequency in Hertz. The $n-1$ clock cycles are spent to initialize the column sum, and in the remaining 2^{n-1-k} ticks the permanent in evaluated. Our design is compiled for 330 MHz frequency so the execution time for a 40×40 input matrix on the dual DFE build is in total 207 s. (equivalent to 337 GOPS (10^9 operations per seconds) For such 'long-run' execution, the t_0 initialization overhead is negligible. For comparison, [51] reported the required time to compute the permanent of a 40×40 matrix on 98304 CPU cores of the Tianhe 2 supercomputer in 24 s using Ryser's formula in double precision. Consequently, one CPU server with two DFE engines calculates a 40×40 permanent only 8.6 $\times$ slower than 4096 CPU nodes (each node containing of 24 cores) in the benchmark of [51].

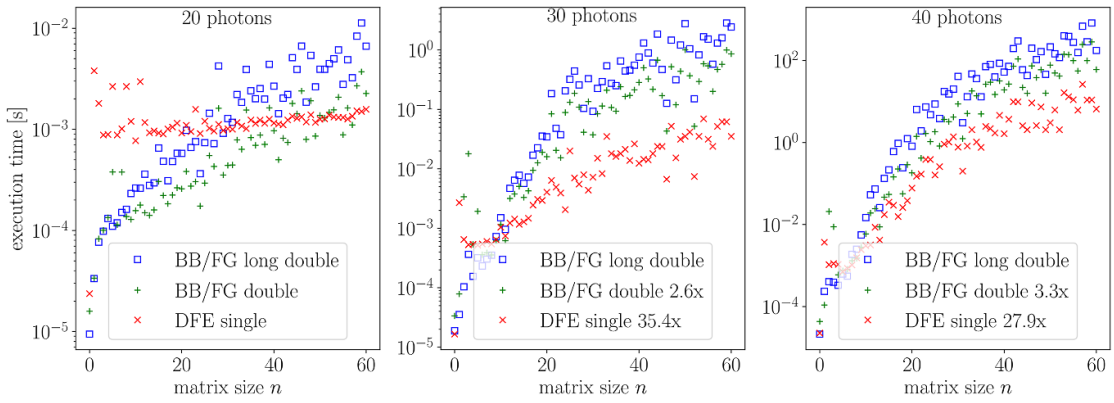

Figure 3. Performance comparison of the permanent calculator DFE to the CPU side BB/FG implementations accounting for row and column multiplicities in the input unitary. The $n \times n$ input unitaries, row, and column multiplicities are constructed randomly, such that they describe 20 (left) and 30 photons (right) in total. As the photon count increases above 20, the DFE implementation becomes systematically faster than the CPU implementations at matrix size larger than $n \approx 13$. The speedup factors compared to the long double precision BB/FG implementation are indicated in the legend.

From a practical usability point of view, we also examine the performance of the permanent calculation implementations when photon multiplicities on the photonic modes induce repeated rows and columns in the unitary describing the photonic scattering process. (See details in section 2.2.) In this case, the average photon multiplicity on the photonic modes is controlled by the unitary size n and the total photon number. In figure 3, we compare the performance of the DFE engine to the CPU implementations of the Piquasso Boost library for photon counts 20 and 30. The figure shows the execution time measured at permanent evaluation for random unitaries constructed for randomly distributed photons over the photonic modes. From our numerical results, we see that at a photon count larger than 20 the DFE turns out to be more efficient than the CPU implementations in the evaluation of permanents. While keeping up to the numerical accuracy of the long double precision BB/FG implementation, the DFE is about 3.6 times faster than the double precision BB/FG implementation at photon count 30 and about 7.4 times faster at photon count 40. (The speedup compared to long double precision implementations at the same photon counts are 35.4 $\times$ and 27.9 $\times$, respectively.) By using dual DFE implementation the speedup is further doubled. At lower photon count, however, the PCIe delay time of the DFE dominates the execution. (See the left side of figure 3) In the repeated row DFE variant of the permanent calculator, the columns sum initialization achieves the same performance by staggering computations the loop tiling/staggering technique. This requires the CPU to pre-compute and upload some additional data which scales on the order of the loop length. The loop length in practice is small or 9 cycles. The cycle delay is based upon multiplication in computing the binomial updates. To keep the parallelism of 4 or 8, we also force the first 3 or 4 row multiplicities to be 1 (the original formula already requires one such reduction and 2 or 3 extra for parallelism), which can slightly increase the operation count. However, by choosing the smallest multiplicities for row expansion, the consequence of maintaining power-of-2 parallelism is largely mitigated. The DFE implementation accounting for row/column multiplicities is also compiled for 330 MHz frequency.

Finally, in section S4 of the supplementary material we show that by streaming multiple input matrices to the DFE in a single execution, one needs to pay the PCIe delay time only once and divide it between the permanent calculation instances, thus lowering the crossover in the DFE-CPU performance down to a matrix size of $n \approx 13$.

3.2. Numerical accuracy of permanent calculator engines

As pointed out earlier by [51] the numerical accuracy becomes a notable issue with increasing the size of the input matrix. The final numerical precision of an implementation depends on the interplay of various factors. The number representation used in the individual arithmetic operations and the conversion between them, or the computational design of mathematical operations have both great effect on the accuracy of the final result. For example, [51] found that the BB/FG formula shows bigger numerical fidelity than the Ryser formula in the experiment of reproducing the analytical result evaluated for the identity matrix According to our reasoning, the different numerical properties of the two approaches can be explained by the difference in the number of addends in the inner sums of equations (2) and (4). While in equation (4) the sum involves always the same number of matrix elements before calculating the products of the column sums, in equation (2) the number of the summed matrix elements varies according to the actual partitioning S

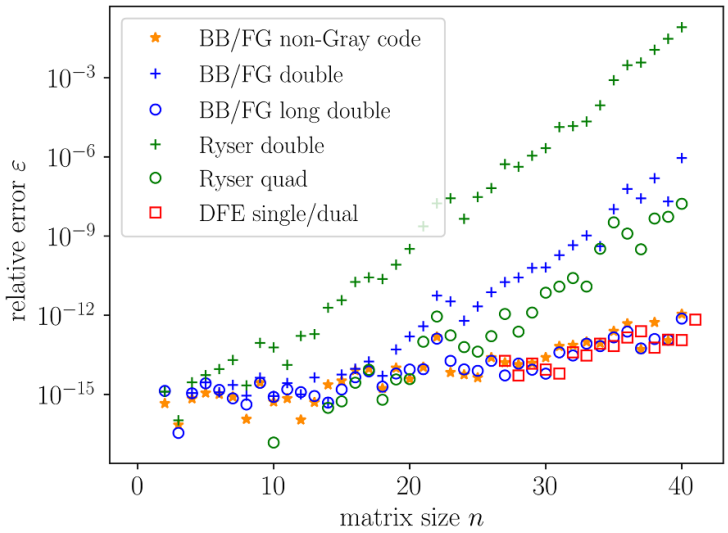

Figure 4. The numerical accuracy of different implementations to calculate the permanent function. The relative error defined by equation (9) was calculated for the permanent calculator engines implementing the Ryser formula implemented in TheWalrus package, the recursive BB/FG formula implemented in the Piquasso Boost library and the BB/FG formula implemented on the DFE. (For details of the DFE implementation see section 3) Among the CPU implementations, the highest accuracy was achieved by the recursive BB/FG implementation utilizing extended precision floating point number representation. The implementations designed for the DFE achieve almost identical precision, we experienced deviation from the ideal result only at larger matrices. However, even in this regime, the relative error is smaller than $\varepsilon \approx 10^{-8}$. The CPU benchmark was done on an *AMD EPYC 7542 32-Core Processor* platform, for DFE calculations Xilinx Alveo U250 FPGA cards were used.

resulting in a possible wider range of magnitude in the calculated products before summing them up. In order to increase the accuracy of the calculated permanent in the Piquasso Boost library, we evaluate the outer sum of the BB/FG formula by classifying the individual addends according to their magnitude, split them into 'pockets' and always calculate the sum of those partial results that are close to each other in magnitude. (We also applied this strategy in our previous work [68].)

Now we examine the numerical accuracy of the individual calculation methods. We implemented the designed scalable BB/FG algorithm with several levels of floating point number representations provided by the Piquasso Boost library and compared the result to the Ryser formula implemented in The Walrus package (version 0.16.2). Among them, the least accurate variant turned out to be the Ryser formula implemented by double precision (i.e. 64 bit) floating point arithmetic operations, followed by the double-precision BB/FG formula evaluated by the Gray code strategy. As default, the Piquasso Boost library implements extended precision floating point arithmetic operations to calculate the permanent providing high numerical accuracy for photonic quantum computer simulations up to photon numbers still accessible on traditional HPC nodes.

To establish a proper ground truth, we also incorporated the GNU Multiple Precision (GMP) arithmetic library's [56] extension of Multiple Precision Floating-Point Reliability (MPFR) to compute the permanent with 'infinite precision' using the designed recursive BB/FG algorithm. (The correctness of the 'infinite precision' implementation was tested on special input cases. namely, when evaluating the BB/FG formula on $m \times n$ rectangular shaped matrices with $m \geqslant n+2$, the final result should be inevitably 0 due to the exact cancellation of the addends. Such a computation, where approximation with normal floating or fixed point numbers would never return a proper 0 result, provides very convincing evidence for the correctness of the implementation.) Figure 4 shows the relative error

$$\varepsilon = \frac{\mathrm{abs}\left(\mathrm{perm}_{\mathrm{INF}}\left(\mathbf{A}\right) - \mathrm{perm}\left(\mathbf{A}\right)\right)}{\mathrm{abs}\left(\mathrm{perm}_{\mathrm{INF}}\left(\mathbf{A}\right)\right)} \tag{9}$$

of several benchmarked permanent calculation implementations with $\mathrm{perm}_{\mathrm{INF}}(\ldots)$ standing for the infinite precision implementation. Our numerical experiments carried out on random unitary matrices justify our expectations. (This choice is justified as our primary goal is to use the developed permanent calculation engines to support photonic quantum computer simulations, where the physical system is described by unitary matrices.) The 64 bit floating point representation is significantly less accurate than the extended precision counterparts. Secondly, our results revealed that the Ryser formula is less accurate than the BB/FG variant. (The accuracy of the double precision Gray coded BB/FG implementation is close to the Ryser formula evaluated with extended precision.) A reduction in accuracy associated with the extended precision

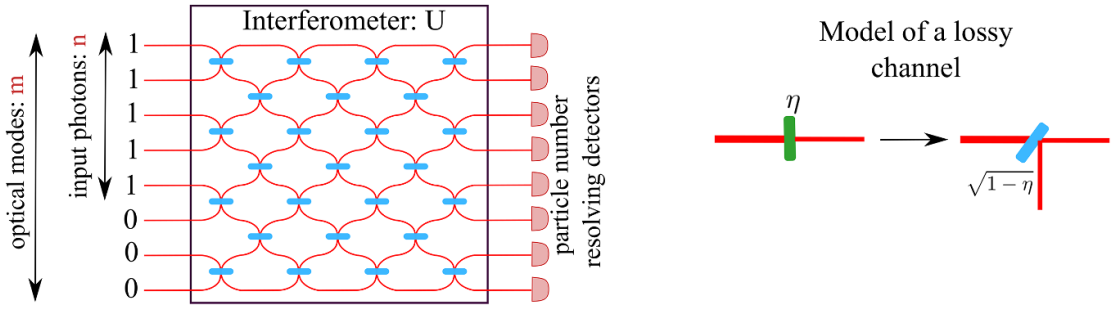

Figure 5. *Left:* a schematic of a standard BS experiment. The blue elements denote beam splitters inside the interferometer. A net of interconnected beam splitters is the usual physical realization of the interferometer. *Right:* a schematic representation of the loss model we use in this work. On the left side of the diagrammatic equation, we see a lossy channel on a single mode, with particle transmission probability η. On the right side of the equation, one can see the applied beam splitter that mimics such a channel. The transmitted beam labeled by $\sqrt{1-\eta}$ is not measured.

Ryser method first appears at a matrix size of $n \approx 20$ and the difference increases with the matrix size. (See the green circles in figure 4.) Though we leave the discussion of the implementation details of the DFE permanent calculator engine for section 3, here we just notice that the numerical accuracy of the DFE implementation stays close to the extended precision CPU implementation of the BB/FG formula, we experienced a deviance only at matrix size $n \geqslant 36$. However, even at such matrix size the accuracy of the DFE implementation still remains better than the extended precision Ryser formula.

According to our reasoning, the main source of the numerical error in the CPU implementations can be explained by the fact that in Gray code variants of the Ryser and BB/FG formula, some computational data are reused from cycle to cycle, in each turn modifying their value by addition/subtraction. Consequently, some addends to the permanent are derived via exponentially many arithmetic operations. This results in an accumulation of numerical error compared to the naive $\mathcal{O}(n^2 \cdot 2^n)$ implementations, where each addend added to the permanent is a result of only n^2 arithmetic operations. This reasoning leads us to the conclusion that the $\mathcal{O}(n^2 \cdot 2^n)$ implementations are expected to be more accurate than the Gray-coded $\mathcal{O}(n \cdot 2^n)$ variants. We justified our expectation by evaluating the numerical accuracy of the BB/FG formula without the Gray code strategy. The associated orange-colored data points in figure 4 revealed, that the double precision, non-Gray-coded BB/FG formula indeed gives as good accuracy as the Gray coded variant BB/FG formula evaluated with extended precision. The Gray-coded implementations, in turn, perform so much faster that executing them in extended precision (to obtain equivalent numerical precision) is still favorable.

4. Classical simulation of BS including photon losses

The classic experimental setup of BS is shown in figure 5. Given an m-mode n-particle Fock state $|\vec{S}\rangle = |1 \ldots 10 \ldots 0\rangle$, and the interferometer, described by a $m \times m$ matrix U, one performs the passive (particles-number preserving) linear-optical evolution of $|\vec{S}\rangle$ with the interferometer, then we measure the outcome using the particle-number resolving detectors.

In subsequent formulas, we use s_i and t_i to label the components of the occupation vector describing the input state $\vec{S}$ and the output state $\vec{T}$ as $\vec{S} = (s_1 = 1, s_2 = 1, \ldots, s_n = 1, s_{n+1} = 0, \ldots, s_m = 0)$ and $\vec{T} = (t_1, t_2, \ldots, t_m)$. In the particle-conserving case, $\sum_{i=1}^{m} s_i = \sum_{i=1}^{m} t_i$. The output probability p_U associated with a given process $\vec{S} \to \vec{T}$ is then given by (10)

$$p_U\left(\vec{S} \to \vec{T}\right) = \frac{|\mathrm{perm}\,(U_{ST})|^2}{\prod_{i=1}^{m} s_i! t_i!}, \tag{10}$$

where U_{ST} is an $n \times n$ matrix constructed from the unitary U describing the interferometer as follows. First, we construct an $m \times n$ matrix U_S, by taking ith column of U s_i times and then we take the jth row of U_S t_j times. The hardness of BS is a consequence of the fact that the probability amplitude in equation (10) is proportional to matrix permanent. The situation is especially clear in the regime $m \gg n$, where for typical Haar random U the probability distribution given in equation (10) is concentrated on collision-free sub-spaces spanned by states $|\vec{T}\rangle$ with $t_i \geqslant 1$. This is the regime in which arguments for hardness put forward in the original BS paper [4] hold (see also [69] for a more refined analysis of hardness of this problem). The regime when the number of photons is comparable to the number of modes was investigated by a separate analysis of [70] establishing an evidence on the hardness of BS simulation in this regime as well.

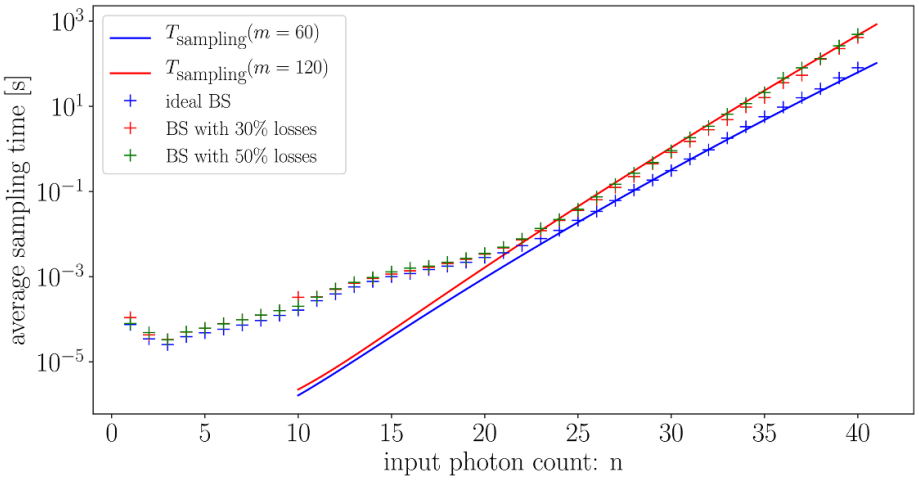

Figure 6. Performance of the developed BS simulation design executed on two *AMD EPYC 7542 32-Core Processor* servers, each server containing of 2 CPU nodes and two Xilinx Alveo U250 FPGA cards. The averaged sampling time T_{sampling} of a 60-mode BS simulations calculated over 1000, 300, 100 samples for input photon counts less or equal than 34, 38, 40, respectively.

From an experimental point of view, it should be noted that in real-world devices realizing BS photon losses occur on a regular basis. Thus,

$$n = \sum_{i=1}^{m} s_i \geqslant \sum_{i=1}^{m} t_i \tag{11}$$

where we previously defined all of the symbols. Lossy BS introduces many new challenges in simulations. In [71, 72] the authors discuss loss handling and a classical simulation algorithm for lossy BS simulation. In general, interferometers are a net of interconnected beam splitters. In the lossy scenario, the lossy beam splitter transfers the particle into an inaccessible (unmeasurable) mode. One can easily see the practicality of this approach as its implementation does not need any new tools besides beam splitters. Graphically, one usually presents it as in the right panel of figure 5. The authors of [72] also noticed that the uniform part of the losses (i.e. loss that applies uniformly to all of the modes) can be extracted from the interferometer and applied as a pure-loss channel at the beginning of the simulation leading to a new instance of the matrix U describing the interferometer. One may still incorporate further, non-uniform losses into the model, while the simulation will be computationally less demanding due to the lower number of particles at its input. It's worth pointing out that the matrix describing a lossy interferometer is no longer unitary as its eigenvalues can be lower than unity.

The most popular algorithm of BS simulation is the Clifford and Clifford algorithm [46, 47]. Although it was designed for the exact simulation of BS, the algorithm can be adopted for lossy BS simulations as well by virtually doubling the number of the optical modes according to the reasoning of [71], while taking the sample only from the first half of the outputs.

We implemented the Clifford and Clifford algorithm in the Piquasso Boost high-performance C++ library linked against traditional CPU and DFE permanent calculation engines. The BS experiment can be designed by the high-level programming interface of the Piquasso framework in Python language. The BS simulation can be scaled up over multiple servers via MPI communication protocol. In our numerical experiments, we used *AMD EPYC 7542 32-Core Processor* servers, each server containing 2 CPU nodes and two Xilinx Alveo U250 FPGA cards. We executed the BS simulation on 4 MPI processes, each process utilizing one CPU node with one FPGA card. Figure 6 shows the results of our BS simulation performance measurement carried out on host servers equipped with the DFE permanent evaluation engines. Following the reasoning in Clifford and Clifford [47], the theoretical prediction of the average sampling time (i.e. averaged over many samples) in the BS simulation (proportional to the average-case sampling complexity) can be given by the

$$T_{\text{sampling}} = T_0 \left[\frac{n(m+n)}{m} \binom{m+n}{n+1}^{-1} \binom{2m+n}{n+1} + n^2 m \right] \tag{12}$$

expression in the $n, m \to \infty$ limit. The expression contains a single parameter T_0 providing a portable measure to capture the performance of a BS simulator. We fitted equation (12) to three data sets obtained

from the averaged sampling time of a 60-mode interferometer. One data set corresponds to an ideal BS without any photon loss in the simulation. The second and third data sets were obtained by simulations assuming 30% and 50% losses. (The 60-mode interferometer and 30% loss correspond to the experimental setup of [32].) In the case of lossy BS, the simulation implies a doubling of the photonic modes, resulting in increased sampling time reported also in figure 6. We found that equation (12) describes the complexity of BS simulations very well at larger input photon counts. For the developed BS simulator design we obtained $T_0 = 7.5 \times 10^{-11}$ sec from the fitting. At lower input photon count, the measured sampling performance is different from the prediction of equation (12). At the corresponding 10^{-3}–10^{-6} s timescale the CPU-DFE performance crossover, the CPU parallelization overhead, and the initialization cost of the computational model (such as the Python-C++ binding, library loading, etc) dominates the execution time of the BS simulator.

Depending on the number of optical modes, when the number of optical modes (m) is significantly larger than the photon count ($m = 120$ case), the average execution time scales exponentially with the number of input photons. Conversely, when the interferometers feature fewer optical modes, the average time complexity tends towards sub-exponential behavior, as evidenced by the slight deviation of the blue data points in figure 6 from a line. This finding resembles prior theoretical results on the original hardness result for exact BS of Aaronson and Arkhipov requiring $m \geqslant 2n$ as a crucial condition in their hardness analysis.

Consequently, it is recommended that experimentalists design devices with a substantially higher number of optical modes than input photons. However, the ratio between the number of optical modes and the photon count does not necessarily need to be large. Based on this finding, achieving quantum advantage in boson sampling experiments appears feasible, as a moderate increase in system size relative to the photon count can suffice for this purpose.

Finally, we notice that the obtained T_0 fitting parameter is expected to be inversely proportional to the number of the incorporated DFEs, scaling ideally up to a large number of accelerators. Since the CPU servers hosting the DFEs are collecting the samples on their own, the gathering of the samples over the host servers will pose a single-time communication overhead that can be neglected compared to the overall sampling time. We have successfully demonstrated this design using two CPU servers hosting 4 FPGA cards in total.

5. Conclusions

In this work, we described a novel scalable approach to evaluate the permanent function based on the BB/FG formula. Utilizing the mathematical properties of reflected Gray code ordering one may concurrently evaluate partitions of the outer sum, paving the way for parallel computation of the permanent. We further generalized the algorithm for cases when columns or row multiplicities occur in the input matrix (corresponding to multiple occupations of photonic modes) and the complexity of the permanent calculation might be reduced. We achieved this by the utilization of generalized n-ary Gray code ordering of the outer sum in the BB/FG permanent formula, with digits varying between zero and the occupation number of the individual optical modes. This generalization makes the BB/FG formula applicable for high-performance BS simulation utilizing significant reduction in the computational complexity as it was previously demonstrated using the Ryser formula as well [47, 58, 59]. The main advantage of the BB/FG formula opposed to the Ryser variant lies in the numerical accuracy of the calculated permanent value. Our numerical experiments using the MPFR multi-precision library showed that Ryser's formula loses against the BB/FG method by several orders of magnitude in accuracy in both the double and extended precision calculations.

We also implemented the BB/FG method on FPGA-based DFEs. Our implementations are capable of handling matrices of arbitrary size (up to 40 columns) without the overhead of reprogramming the FPGA chips and accounting for row multiplicities in the input matrix via the N-ary Gray code ordering ported to the FPGA chips. The throughput of the DFEs was further increased by organizing the input matrices into batches and executing multiple permanent calculations on the FPGA chips in a single execution. Finally, the fix-point number representation implemented on the FPGA chips provides competitive accuracy to the extended precision BB/FG CPU implementation in the evaluation of the permanent of unitary matrices. The accuracy of the DFE implementation—equivalent to extended precision—holds up to matrices of size $n = 40$.

These features turned out to be essential to achieve an efficient BS simulator design supported by high-performance DFEs. We integrated our permanent evaluation DFEs into the computation flow of both the ideal and lossy BS simulation. Since the simulation of lossy BS involves twice as many photonic modes as the ideal variant of the same BS design, the simulation of lossy BS takes more time in general. On average, our setup of 4 Alveo U250 FPGA cards made it possible to take a single sample from a 60-mode interferometer with 40 input photons in ∼80 s without photon losses. Introducing photon losses into the design, our numerical experiments could draw a single sample in ∼360 s. The theoretical description of [47]

fits the measured performance data very well by fitting a single parameter (labeled by T_0 in equation (12) to all data points. The fitting parameter provides a portable measure to compare different BS simulator implementations. However, we did not find any competitive work in the literature on BS simulation providing similar performance measurements to ours. In turn, we can compare the performance of our simulator to a real BS experiment involving 60 photonic modes, 20 input photons, and 14 measured photons (i.e. loss of 30% on average). We could simulate the described design in ∼0.8 milliseconds per sample, while in [32] authors detected 150 valid samples of 14-photon coincidence measurements in 26 h.

We should mention that numerous possible sources can be named to cause discrepancies between the theory and experiments. Photon losses originating from the imperfection of the photon detectors or disappearing in the circuitry can be accounted for via different loss models. Our boson sampling simulator implements photon loss by means of unmeasured photonic modes over which the photon could virtually escape from the device without capturing them. A real experiment can be also affected by decoherency, imperfect sources and uncertainties of interferometer settings which are not captured by the simulation technique. These might be a sources for differences between the simulator and a real experiment. In addition, we should not forget about the high dimensionality of the photon distribution that makes it hard to make conclusive judgments about the equivalence or difference between numerical and experimental results. For this purpose several validation approaches have been developed [40–45] which have been mentioned in the introduction.

Finally, we notice that the BS simulation capabilities described in this work can be further improved by utilizing the concept of approximate BS described in [72], in which part of the optical modes are treated with MF approximation. In this approach, the number of the approximated modes is a hyper-parameter in the algorithm controlling both the speed and the fidelity of the BS simulation. We leave the study of approximate BS simulation with DFEs for future work.

Data availability statement

All data that support the findings of this study are included within the article (and any supplementary files).

Acknowledgments

This research was supported by the Ministry of Culture and Innovation and the National Research, Development and Innovation Office within the Quantum Information National Laboratory of Hungary (Grant No. 2022-2.1.1-NL-2022-00004), by the ÚNKP-22-5 New National Excellence Program of the Ministry for Culture and Innovation from the source of the National Research, Development and Innovation Fund, and by the Hungarian Scientific Research Fund (OTKA) Grants Nos. K134437 and FK135220. We also acknowledge funding by the QuantERA II project HQCC-101017733. R P acknowledges support from the Hungarian Academy of Sciences through the Bolyai János Stipendium (BO/00571/22/11) as well. We acknowledge the computational resources provided by the Wigner Scientific Computational Laboratory (WSCLAB) (the former Wigner GPU Laboratory). T R and M O acknowledge financial support by the Foundation for Polish Science through TEAM-NET project (Contract No. POIR.04.04.00-00-17C1/18-00). Supported by IRAP AstroCeNT (MAB/2018/7) funded by FNP from ERDF.

References

[1] Preskill J 2012 Quantum computing and the entanglement frontier (arXiv:1203.5813)
[2] Lund A P, Bremner M J and Ralph T C 2017 Quantum sampling problems, bosonsampling and quantum supremacy *npj Quantum Inf.* **3** 1–8
[3] Harrow A W and Montanaro A 2017 Quantum computational supremacy *Nature* **549** 203–9
[4] Aaronson S and Arkhipov A 2010 The computational complexity of linear optics (arXiv:1011.3245)
[5] Aaronson S and Arkhipov A 2014 The computational complexity of linear optics *Research in Optical Sciences* (Optical Society of America) p QTh1A.2
[6] Bremner M J, Montanaro A and Shepherd D J 2016 Average-case complexity versus approximate simulation of commuting quantum computations *Phys. Rev. Lett.* **117** 080501
[7] Boixo S, Isakov S V, Smelyanskiy V N, Babbush R, Ding N, Jiang Z, Bremner M J, Martinis J M and Neven H 2018 Characterizing quantum supremacy in near-term devices *Nat. Phys.* **14** 595–600
[8] Bouland A, Fefferman B, Nirkhe C and Vazirani U 2019 On the complexity and verification of quantum random circuit sampling *Nat. Phys.* **15** 159–63
[9] Haferkamp J, Hangleiter D, Bouland A, Fefferman B, Eisert J and Bermejo-Vega J 2020 Closing gaps of a quantum advantage with short-time hamiltonian dynamics *Phys. Rev. Lett.* **125** 250501
[10] Oszmaniec M, Dangniam N, Morales M E S and Zimborás Z 2020 Fermion sampling: a robust quantum computational advantage scheme using fermionic linear optics and magic input states (arXiv:2012.15825)

[11] Tangpanitanon J, Thanasilp S, Lemonde M-A, Dangniam N and Angelakis D G 2023 Signatures of a sampling quantum advantage in driven quantum many-body systems *Quantum Sci. Technol.* **8** 025019
[12] Hamilton C S, Kruse R, Sansoni L, Barkhofen S, Silberhorn C and Jex I 2017 Gaussian boson sampling *Phys. Rev. Lett.* **119** 170501
[13] Kruse R, Hamilton C S, Sansoni L, Barkhofen S, Silberhorn C and Jex I 2019 Detailed study of gaussian boson sampling *Phys. Rev.* A **100** 032326
[14] Quesada N, Miguel Arrazola J and Killoran N 2018 Gaussian boson sampling using threshold detectors *Phys. Rev.* A **98** 062322
[15] Lund A P, Laing A, Rahimi-Keshari S, Rudolph T, O'Brien J L and Ralph T C 2014 Boson sampling from a gaussian state *Phys. Rev. Lett.* **113** 100502
[16] Bentivegna M *et al* 2015 Experimental scattershot Boson sampling *Sci. Adv.* **1** e1400255
[17] Barkhofen S, Bartley T J, Sansoni L, Kruse R, Hamilton C S, Jex I and Silberhorn C 2017 Driven Boson sampling *Phys. Rev. Lett.* **118** 020502
[18] Motes K R, Gilchrist A, Dowling J P and Rohde P P 2014 Scalable Boson sampling with time-bin encoding using a loop-based architecture *Phys. Rev. Lett.* **113** 120501
[19] Pant M and Englund D 2016 High-dimensional unitary transformations and Boson sampling on temporal modes using dispersive optics *Phys. Rev.* A **93** 043803
[20] Shen C, Zhang Z and Duan L-M 2014 Scalable implementation of Boson sampling with trapped ions *Phys. Rev. Lett.* **112** 050504
[21] Toyoda K, Hiji R, Noguchi A and Urabe S 2015 Hong–ou–mandel interference of two phonons in trapped ions *Nature* **527** 74–77
[22] Robens C, Arrazola I nigo, Alt W, Meschede D, Lamata L, Solano E and Alberti A 2022 Boson sampling with ultracold atoms (arXiv:2208.12253)
[23] Peropadre B, Giacomo Guerreschi G, Huh J and Aspuru-Guzik A 2016 Proposal for microwave Boson sampling *Phys. Rev. Lett.* **117** 140505
[24] Broome M A, Fedrizzi A, Rahimi-Keshari S, Dove J, Aaronson S, Ralph T C and White A G 2013 Photonic Boson sampling in a tunable circuit *Science* **339** 794–8
[25] Spring J B *et al* 2013 Boson sampling on a photonic chip *Science* **339** 798–801
[26] Crespi A, Osellame R, Ramponi R, Brod D J, Galv ao E F, Spagnolo N'o, Vitelli C, Maiorino E, Mataloni P and Sciarrino F 2013 Integrated multimode interferometers with arbitrary designs for photonic Boson sampling *Nat. Photon.* **7** 545–9
[27] Tillmann M, Dakić B, Heilmann R, Nolte S, Szameit A and Walther P 2013 Experimental Boson sampling *Nat. Photon.* **7** 540–4
[28] Spagnolo N'o *et al* 2014 Experimental validation of photonic Boson sampling *Nat. Photon.* **8** 615–20
[29] Spring J B *et al* 2017 Chip-based array of near-identical, pure, heralded single-photon sources *Optica* **4** 90–96
[30] Faruque I I, Sinclair G F, Bonneau D, Rarity J G and Thompson M G 2018 On-chip quantum interference with heralded photons from two independent micro-ring resonator sources in silicon photonics *Opt. Express* **26** 20379–95
[31] Signorini S and Pavesi L 2020 On-chip heralded single photon sources *AVS Quantum Sci.* **2** 041701
[32] Wang H *et al* 2019 Boson sampling with 20 input photons and a 60-mode interferometer in a 10^{1}4-dimensional hilbert space *Phys. Rev. Lett.* **123** 250503
[33] Zhong H-S *et al* 2020 Quantum computational advantage using photons *Science* **370** 1460–3
[34] Madsen L S *et al* 2022 Quantum computational advantage with a programmable photonic processor *Nature* **606** 75–81
[35] Deng Y-H *et al* 2023 Gaussian Boson sampling with pseudo-photon-number-resolving detectors and quantum computational advantage *Phys. Rev. Lett.* **131** 150601
[36] Bulmer J F F *et al* 2022 The boundary for quantum advantage in gaussian Boson sampling *Sci. Adv.* **8** eabl9236
[37] Dellios A S, Reid M D and Drummond P D 2023 Simulating gaussian Boson sampling quantum computers *AAPPS Bull.* **33** 31
[38] Changhun O, Lim Y, Fefferman B and Jiang L 2022 Classical simulation of Boson sampling based on graph structure *Phys. Rev. Lett.* **128** 190501
[39] Villalonga B, Yuezhen Niu M, Li L, Neven H, Platt J C, Smelyanskiy V N and Boixo S 2022 Efficient approximation of experimental gaussian Boson sampling (arXiv:2109.11525)
[40] Changhun O, Jiang L and Fefferman B 2023 Spoofing cross-entropy measure in Boson sampling *Phys. Rev. Lett.* **131** 010401
[41] Neville A, Sparrow C, Clifford R, Johnston E, Birchall P M, Montanaro A and Laing A 2017 Classical Boson sampling algorithms with superior performance to near-term experiments *Nat. Phys.* **13** 1153–7
[42] Seron B, Novo L, Arkhipov A and Cerf N J 2022 Efficient validation of Boson sampling from binned photon-number distributions (arXiv:2212.09643)
[43] Chabaud U, Grosshans F'eric, Kashefi E and Markham D 2021 Efficient verification of Boson sampling *Quantum* **5** 578
[44] Agresti I, Viggianiello N, Flamini F, Spagnolo N'o, Crespi A, Osellame R, Wiebe N and Sciarrino F 2019 Pattern recognition techniques for Boson sampling validation *Phys. Rev.* X **9** 011013
[45] Flamini F, Spagnolo N'o and Sciarrino F 2019 Visual assessment of multi-photon interference *Quantum Sci. Technol.* **4** 024008
[46] Clifford P and Clifford R 2018 The classical complexity of Boson sampling *Proc. 29th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA'18)* (Society for Industrial and Applied Mathematics) pp 146–55
[47] Clifford P and Clifford R 2020 Faster classical Boson sampling (arXiv:2005.04214)
[48] Valiant L G 1979 The complexity of computing the permanent *Theor. Comput. Sci.* **8** 189–201
[49] Ryser H J 1963 *Combinatorial Mathematics (The Carus Mathematical Monographs)* (Mathematical Association of America)
[50] David G G 2013 Permanent formulae from the veronesean *Des. Codes Cryptogr.* **68** 39–47
[51] Junjie W, Liu Y, Zhang B, Jin X, Wang Y, Wang H and Yang X 2018 A benchmark test of Boson sampling on Tianhe-2 supercomputer *Natl Sci. Rev.* **5** 715–20
[52] Kurt M, Atilgan C and Berberler M E 2013 A dynamic programming approach for generating n-ary reflected gray code list *J. Faculty Sci.* **37** 31–37
[53] Kolarovszki Z *et al* 2024 Piquasso: A Photonic Quantum Computer Simulation Software Platform (arXiv:2403.04006)
[54] Johnston W M Paul Hanna J R and Richard J M 2004 Advances in dataflow programming languages *ACM Comput. Surv.* **36** 1–34
[55] Nijenhuis A and Herbert S W 1975 *Combinatorial Algorithms/Albert Nijenhuis and Herbert S. Wilf* (Academic)
[56] Fousse L, Hanrot G, Lefèvre V, Pélissier P and Zimmermann P 2007 Mpfr: a multiple-precision binary floating-point library with correct rounding *ACM Trans. Math. Softw.* **33** 13-es
[57] Rakyta P *et al* 2021 Piquasso boost libraries (available at: https://github.com/Budapest-Quantum-Computing-Group/piquassoboost)
[58] Chin S and Huh J 2018 Generalized concurrence in Boson sampling *Sci. Rep.* **8** 6101
[59] Lundow P H and Markström K 2022 Efficient computation of permanents, with applications to Boson sampling and random matrices *J. Comput. Phys.* **455** 110990

[60] Gupt B, Izaac J and Quesada N 2019 The walrus: a library for the calculation of hafnians, hermite polynomials and gaussian Boson sampling *J. Open Source Softw.* **4** 1705
[61] Guan D-J 1998 Generalized gray codes with applications *Proc. Natl. Sci. Counc. ROC(A)* **22** 6
[62] Xilinx2021 Alveo U200 and U250 Data Center Accelerator Cards Data Sheet (DS962) (available at: https://docs.xilinx.com/r/en-US/ds962-u200-u250/Summary)
[63] Donald E K 1997 *The Art of Computer Programming, Volume 2: Seminumerical Algorithms* 3rd edn (Addison-Wesley)
[64] Nicholas J H 1992 Stability of a method for multiplying complex matrices with three real matrix multiplications *SIAM J. Matrix Anal. Appl.* **13** 681–7
[65] Kumm M, Gustafsson O, de Dinechin F, Kappauf J and Zipf P 2018 Karatsuba with rectangular multipliers for fpgas *2018 IEEE 25th Symp. on Computer Arithmetic (ARITH)* pp 13–20
[66] de Dinechin F and Pasca B 2009 Large multipliers with fewer DSP blocks *2009 Int. Conf. on Field Programmable Logic and Applications* pp 250–5
[67] Montgomery P L 2005 Five, six and seven-term karatsuba-like formulae *IEEE Trans. Comput.* **54** 362–9
[68] Kaposi Agoston, Kolarovszki Z, Kozsik T, Zimborás Z and Rakyta P 2022 Polynomial speedup in torontonian calculation by a scalable recursive algorithm (arXiv:2109.04528)
[69] Bouland A, Fefferman B, Landau Z and Liu Y 2022 Noise and the frontier of quantum supremacy *2021 IEEE 62nd Annual Symp. on Foundations of Computer Science (FOCS)* pp 1308–17
[70] Bouland A, Brod D, Datta I, Fefferman B, Grier D, Hernandez F and Oszmaniec M 2023 Complexity-theoretic foundations of bosonsampling with a linear number of modes (arXiv:2312.00286)
[71] García-Patrón R, Renema J J and Shchesnovich V 2019 Simulating Boson sampling in lossy architectures *Quantum* **3** 169
[72] Jost Brod D and Oszmaniec M 2020 Classical simulation of linear optics subject to nonuniform losses *Quantum* **4** 267