

Polynomial speedup in exact Torontonian calculation by a scalable recursive algorithm

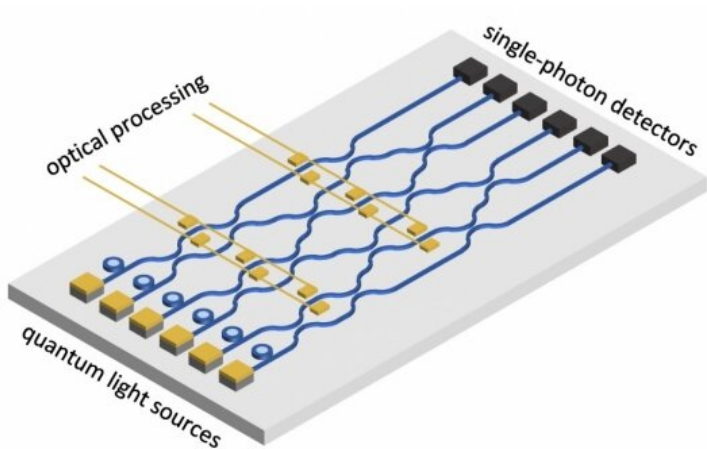
Ágoston Kaposi

Eötvös Loránd University, Faculty of Informatics

21. June 2022

Photonic Quantum Computing

A photonic quantum computer stores information in independent optical modes, called **qumodes**. Between the modes there can be several parametrized connections which can be described by an interferometer matrix.



Gaussian states

The fully general case is hard to simulate and hard to build physically. One add conditions for the light sources, the detectors and the gates between them.

Gaussian states are the states which have at most quadratic Hamiltonian. (E.g. Beamsplitter, Squeezing, Displacement)

A Gaussian state can be fully characterized by their mean vector $\mu \in \mathbb{C}^{2N}$ and their covariance matrix $\sigma \in \mathbb{C}^{2N \times 2N}$.

Reduction

We take a vector of 0, 1 numbers $S = \{s_1, \dots, s_N\}$ and a matrix $A \in \mathbb{C}^{N \times N}$. We can note the reduction of A into vector S , where we take the reduction on all columns and rows corresponding to S : if it is 0, we eliminate and if it is 1, we leave it in matrix. We denote it by A_S .

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}_{(0,1,0,1)} = \begin{pmatrix} a_{22} & a_{24} \\ a_{42} & a_{44} \end{pmatrix}$$

In our case we have block-reduction which means at a vector S the reduction of S, S .

Gaussian boson sampling with threshold detection

We consider those Gaussian states which have a mean vector 0 (without Displacement gates). We consider the measurement with threshold detectors which means that we do not differentiate between the number of photons, just the fact whether any photons were detected or not.

From the complex covariance matrix σ , we create the Husimi covariance matrix $\Sigma = \sigma + \mathbb{1}/2$.

Probability of a possible outcome vector S (0 means no particle detection, 1 means particle detection) can be exactly calculated with the help of the Torontonian matrix function:

$$p(S) = \frac{\text{Tor}(O_S)}{\sqrt{\det(\Sigma)}}, \quad (1)$$

where $O = \mathbb{1} - \Sigma^{-1}$ and O_S means the block reduction of O to S .

Torontonian function

The Torontonian is a matrix function which can be defined in the following way in case of a rectangular matrix A :

$$\text{Tor}(A) = \sum_{Z \in P_N} \frac{(-1)^{N/2-|Z|}}{\sqrt{|\det(\mathbb{1} - A_Z)|}}, \quad (2)$$

where P_N means all powerset of $\{1, \dots, N\}$ and A_Z means the row and column reduction of A corresponding to Z .

Complexity of the determinant is $\mathcal{O}(N^\omega)$ where N denotes the matrix size and $\omega > 2$ even in Hermitian cases. This results as a complexity of the Torontonian function in $\mathcal{O}(N^\omega \cdot 2^N)$.

Torontonian function of Hermitian matrices

Our observation was that the computational costs of evaluating the Torontonian function can be significantly reduced by a wise reuse strategy of the intermediate results during the evaluation of the determinants in Equality (2). In our algorithm we calculate determinants via Cholesky decomposition [1] bringing an $N \times N$ positive definite Hermitian matrix A into a product form $A = LL^\dagger$, where L is a lower triangular matrix. Then the determinant of A can be calculated from the diagonal elements of L by

$$\det(A) = \prod_{i=1}^N |L_{ii}|^2.$$

We proposed an algorithm to calculate the Torontonian where the determinant calculation routines are called recursively providing a possibility to partially reuse the Cholesky decomposition of each submatrix A_Z in the next step of the recursive chain. Our algorithm was designed to take advantage from reusable computational data, while keeping up an efficient parallelization technique and polynomial memory needs.

Cholesky-decomposition

Cholesky decomposition for a positive definite Hermitian matrix A:

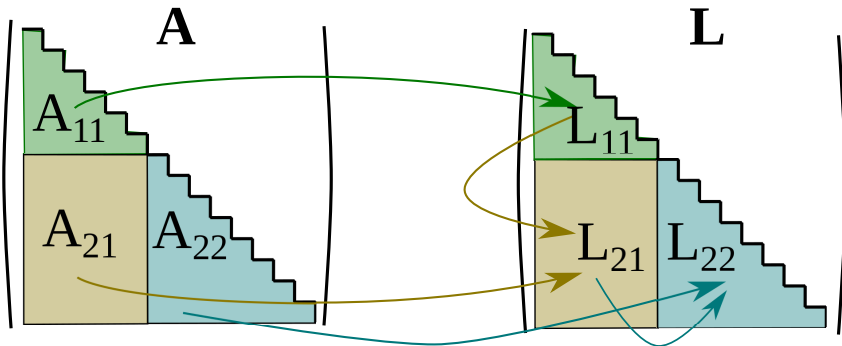
$$L_{i,i} = \sqrt{A_{i,i} - \sum_{k=1}^{i-1} |L_{i,k}|^2}, \quad i = 1, \dots, N$$

$$L_{i,j} = \frac{1}{L_{j,j}} \left(A_{i,j} - \sum_{k=1}^{j-1} L_{i,k} \overline{L_{j,k}} \right), \quad i = 1, \dots, N, j = 1, \dots, i-1$$

Since A is Hermitian and L is lower triangular, it is sufficient to consider only the lower triangular part of both matrices.

Data dependency in Cholesky decomposition

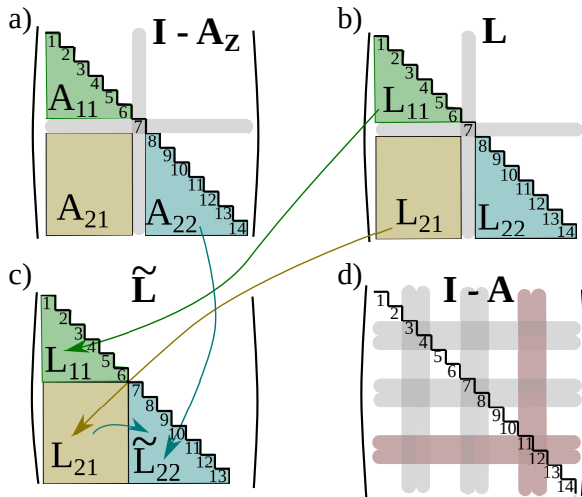
The idea behind our algorithm is based on the fact that during the Cholesky-decomposition the sub-block L_{11} depends only on A_{11} , L_{21} can be calculated using A_{21} and L_{11} , while L_{22} is determined from elements of A_{22} and L_{21} .



This dependency enables us to reuse already calculated data in further calculations.

Matrix reduction with Cholesky-decomposition

During the Torontonian calculation there are many places, where between two vectors S_1 and S_2 the difference is at only one place which means one row reduction.



Recursive algorithm

```
L = cholesky(I-A)
determinant = product_diagonal_element_abs(L)
tor = 1 / sqrt(determinant)

recursive_torontonian(L, [])

def recursive_torontonian(L, modes):
    if modes.size == 0:
        start = 0
    else:
        start = modes[-1] + 1
    for i in range(start, N/2):
        next_modes = modes ++ [i]
        A_Z = create_AZ(L, next_modes)
        next_L = cholesky(I - A_Z)
        determinant = product_diagonal_element_abs(next_L)
        torontonian += -1 ** (len(next_modes)) / sqrt(determinant)
        recursive_torontonian(next_L, next_modes)
```

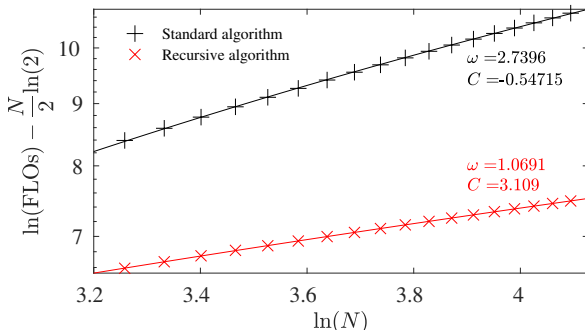
Complexity of the algorithm

Due to high data dependency in the recursive algorithm, we sum up all floating-point operations (FLOs) for both standard and recursive algorithms. We expect the complexity of the algorithms to be proportional to $N^\omega 2^{N/2}$.

The value of ω can be obtained from the logarithm of the overall FLOs as follows:

$$\omega \ln(N) = \ln(\text{FLOs}) - \frac{N}{2} \ln(2) - C, \quad (3)$$

where C is a constant number. After calculating ω and a linear fitting.



Complexity speedup

According to the numerical results, the recursive reuse of the Cholesky decomposition in subsequent determinant calculations reduces the computational complexity

$$\mathcal{C}(\text{standard}) \sim N^{2.7355} 2^{N/2}$$

of the standard algorithm to

$$\mathcal{C}(\text{recursive}) \sim N^{1.0695} 2^{N/2}$$

leading to a polynomial speedup by a factor of

$$\sim N^{1.666}$$

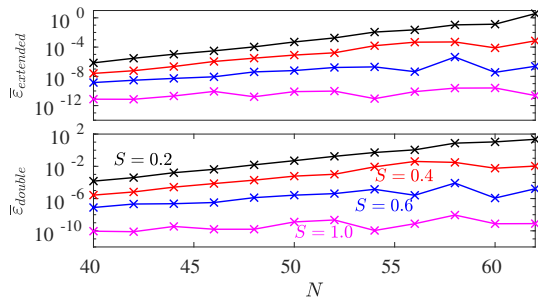
Fidelity of the calculation

Numerically the classical calculation and the recursive one have the same numerical fidelity since the addends in the most outer sum are the same.

We compared our implementation to a well known software tool developed by Xanadu (TheWalrus [2]). We plotted the relative difference

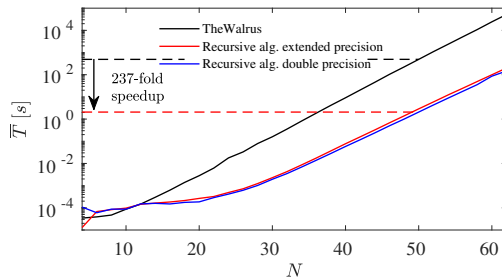
$$\varepsilon_X = \frac{|\text{Tor}_{TheWalrus} - \text{Tor}_X|}{\text{Tor}_X} \quad (4)$$

between the results of TheWalrus package and our implementation at different precisions ($X = double$ or $X = extended$). Here we note that TheWalrus package calculates the Torontonian only in extended precision.






Performance benchmark

We ran performance benchmark of our recursive algorithm to the TheWalrus package with averaged Torontonian calculation time \bar{T} of random sampling matrices of size N . Our implementation of the recursive algorithm shows much better performance than TheWalrus package version 0.15 which means 237-fold speedup.



The benchmark was done on an *Intel Xeon Gold 6130* platform using 24 threads in shared memory model. \bar{T} was determined by the average time of 100, 10 and 2 independent Torontonian computing cycles for matrices of size $N = 4 \dots 52$, $N = 54 \dots 58$ and $N = 60 \dots 62$ respectively.

Thank you for your attention![3]

-  Nicholas J. Higham.
Cholesky factorization.
Wiley Interdisciplinary Reviews: Computational Statistics, 1(2):251–254, September 2009.
-  Brajesh Gupt, Josh Izaac, and Nicolás Quesada.
The walrus: a library for the calculation of hafnians, hermite polynomials and gaussian boson sampling.
Journal of Open Source Software, 4(44):1705, 2019.
-  Kolarovszki Z. Kozsik T. Zimborás Z. Rakyta P. Kaposi, Á.
Polynomial speedup in torontonian calculation by a scalable recursive algorithm.
2021.