Benchmarking 50-Photon Gaussian Boson Sampling on the Sunway TaihuLight

Yuxuan Li, *Student Member, IEEE*, Lin Gan, *Member, IEEE*, Mingcheng Chen, Yaojian Chen, Haitian Lu, Chaoyang Lu, Jianwei Pan, Haohuan Fu, *Member, IEEE*, and Guangwen Yang, *Member, IEEE*

Abstract—Boson sampling is expected to be an important milestone that will demonstrate quantum computational advantage (or quantum supremacy). This work establishes the benchmarking of Gaussian boson sampling (GBS) with threshold detection based on the Sunway TaihuLight supercomputer. To achieve the best performance and provide a competitive scenario for future quantum computing studies, the selected simulation algorithm is fully optimized based on a set of innovative approaches, including a parallel framework with almost perfect load balance and an instruction-level optimizing scheme based on a shortest-path-based instruction scheduling. In addition, data precision is carefully processed by an integer-instruction-based and multiple-precision fixed-point implementation, including 128- and 256-bit precison mode, which can be appropriately selected based on an adaptive precision optimizing scheme. Based on these methods, a highly efficient parallel quantum sampling algorithm is designed. The largest run enables us to obtain one Torontonian function of a 100×100 submatrix from 50-photon GBS within 20 hours in 128-bit precision and 2 days in 256-bit precision. To our knowledge, this was the largest quantum computing simulation based on Boson Sampling by using modern supercomputers.

Index Terms—Boson sampling simulation, quantum computation, parallel computing, sunway TaihuLight supercomputer

1 Introduction

The *extended Church-Turing* (ECT) thesis states that a classical computer can efficiently simulate any physical process with only polynomial overheads [1]. In the 1980s, Feynman observed that many-body quantum problems cannot be efficiently solved by classical computers due to the exponential size of quantum state Hilbert space [2]. This observation inspired Feynman to envision a quantum computer to solve quantum problems. Efficient quantum algorithms were proposed for classically hard problems, such as integer factoring [3], [4]. Today, achieving *quantum computational advantage* (or *quantum supremacy*) based on quantum computers is anticipated to be an important milestone in the post-Moore era.

Recently, it was found that quantum computers can simulate quantum sampling problems in polynomial time, while classical computers require exponential time unless the polynomial hierarchy (PH) collapses [5]. Therefore, the quantum sampling problem has become a feasible way to demonstrate the quantum computational advantage on noisy intermediate-scale quantum (NISQ) devices [6]. Based on numerical estimation, quantum sampling with $50 \sim 100$ quantum particles is beyond the computational capabilities of state-of-the-art supercomputers [7].

Candidates for quantum sampling problems include the instantaneous quantum polynomial time circuit [8], random circuit sampling (RCS) [9], [10], boson sampling [5], and Gaussian boson sampling (GBS) [11], [12]. To implement quantum computing, an instantaneous quantum polynomial time circuit and RCS are based on quantum bits, while boson sampling and GBS are based on bosons, such as single photons.

Because the proposal of GBS shows its capacity to solve several NP-problems with polynomial time and markedly simplifies its quantum implementation, quantum sampling based on bosons becomes an important branch of demonstrating quantum computational advantage [1], [13]. Based on the method of GBS with threshold detection, H. Zhong announced a quantum machine prototype that can manage up to 76 output photon-clicks. The work was published in SCIENCE in Oct. 2020 [14], and was claimed to be a significant milestone in achieving quantum computational advantage. Our work in this paper is an important part for the designing of the prototype, and works as convincing classical benchmarking for GBS with threshold detection. With little modifications, corresponding optimizing techniques are also applicable to other computational problems with similar algorithms.

More specifically, in this work, we establish the quantum computational advantage frontier of classical simulation for

- *Yuxuan Li, Lin Gan, Yaojian Chen, Haohuan Fu, and Guangwen Yang are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China. E-mail: yxuanl1995@gmail.com, {lingan, haohuan}@tsinghua.edu.cn, chen-yj17@mails.tsinghua.edu.cn, ygw@mail.tsinghua.edu.cn.*
- *Mingcheng Chen, Chaoyang Lu, and Jianwei Pan are with the Hefei National Laboratory for Physical Sciences, Microscale Nanoscience Laboratory and Department of Modern Physics, University of Science and Technology of China, Hefei 230026, China, and also with Shanghai Branch, CAS Center for Excellence in Quantum Information and Quantum Physics, University of Science and Technology of China, Shanghai 201315, China. E-mail: {cmc, cylu, pan}@ustc.edu.cn.*
- *Haitian Lu is with the National Supercomputing Center, Wuxi 214072, China. E-mail: luhaitian1997@sina.com.*

Manuscript received 17 Feb. 2021; revised 4 Sept. 2021; accepted 4 Sept. 2021. Date of publication 9 Sept. 2021; date of current version 25 Oct. 2021.
This work was supported by the National Key R&D Program of China under Grant 2020YFB0204700.
(Corresponding author: Yuxuan Li.)
Recommended for acceptance by S. Chandrasekaran.
Digital Object Identifier no. 10.1109/TPDS.2021.3111185

1045-9219 © 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: ULAKBIM UASL - DOKUZ EYLUL UNIVERSITESI. Downloaded on October 28,2022 at 12:39:49 UTC from IEEE Xplore. Restrictions apply.

the boson-based quantum sampling problem. To achieve the best performance and provide a competitive and convincing benchmarking for quantum computers, the selected algorithm is fully optimized based on the Sunway TaihuLight supercomputer, which is one of the most powerful classical computers in the world. This work served as the classical counterpart of H. Zhong's study and can serve for future boson-based studies. More than that, since the GBS method links to potential applications in several fields, including certifiable random numbers, graph optimization [15], [16], [17], graph similarity [18], point process [19], molecular docking [20], quantum chemistry [21], [22], and quantum machine learning [23], this work has the potential to simulate practical applications.

The major contributions include:

- An integer-instruction-based and fixed-point precision scheme with careful precision analysis is designed to enable customizable precision modes for different precision requirements of different matrice.
- An adaptive framework based on upper- and lower-bound estimation is proposed and applied to automatically determine the best precision configurations.
- An effective parallel framework is proposed to reduce the requirements of cache-level storage and obtain a nearly ideal load balance of the selected boson sampling algorithm.
- To further accelerate kernels at the instruction level, a shortest-path-based method that uses DAG to describe dependences is used to develop the optimal instruction scheduling.

All contributions are aimed at resolving the precision and performance issues, which are critical for scientific applications. In terms of the precision issue, Sections 4 and 5 provide analysis and solutions to the precision problems, Section 4 analyzes the precision requirements of the worst case, and Section 5 provides a customized precision implementation and an adaptive framework. In terms of the performance optimizations, Section 6 introduces the proposed parallel framework, and Section 7 describes our sophisticated instruction scheduling strategy that provides the optimal performance based on the parallel framework.

Combining all the above mentioned innovations, we design a highly efficient parallel quantum sampling algorithm. Sustained performance of 2.78 PFLOPS for 128-bit precision and 1.27 PFLOPS for 256-bit precision can be achieved with a proper N. The largest run enables us to obtain one Torontonian function of a 100×100 submatrix from 50-photon GBS within 20 hours in 128-bit precision and 2 days in 256-bit precision.

2 Related Works

In 2019, Google first claimed the quantum computational advantage by using 53 superconducting quantum bits in an RCS experiment [24]. The quantum device took 200 seconds to sample one instance of a quantum circuit one million times, while classical benchmarking was expected to need 10,000 years on the Summit supercomputer. To confirm and establish this quantum computational advantage, there have been many developments in the classical benchmarking algorithm [25], [26], [27], [28], [29], [30] before and after the experiments. The latest result found by IBM indicates that by leveraging secondary storage, the calculation can be run on the same supercomputer within two and a half days rather than 10,000 years [31], [32]. Even though the classical supercomputer can solve the problem in a few days, the increase in speed of approximately one thousand times demonstrates the overwhelming power of quantum computing.

Additionally, many other studies have investigated the RCS problem. In general, the classical simulators of RCS problems can be categorized into three classes. The first and second classes are the direct evolutions of the quantum state [25], [27], [33], [34], [35] and the perturbation of stabilizer circuits [36], [37], [38], respectively. The tensor network contraction class [28], [29], [39] is the most suitable method for current flop-oriented architectures such as on the Fugaku [40] and Summit [41] supercomputers. Additionally, several hybrid algorithms [26], [31], [42] achieve strong performances when simulating or benchmarking with ~ 50 quantum bits. Other studies [27], [43] have attempted to build up the benchmarking of the quantum computational advantage based on the RCS problem using the Sunway TaihuLight supercomputer.

The boson sampling problem, introduced by Aaronson and Arkhipov [44], was the first protocol to conclusively demonstrate the quantum computational advantage. Because it is actually difficult to scale the standard boson sampling to high photon numbers due to its intrinsic photon cost, GBS [45] and GBS with threshold detection [12] were recently proposed to obtain higher-order photon numbers than those of boson sampling. Based on this branch of quantum sampling, a 76-output-photon-clicks (i.e., 76 quantum bits or 2^{76} states) quantum computer was built [14] as the largest boson-based quantum computer in the world to demonstrate the quantum computational advantage.

In terms of the classical benchmarks of boson-based methods, several algorithms have been designed, and experiments have been conducted [46], [47], [48], [49], [50]. There are also several classical benchmarks for GBS. Based on the Titan supercomputer, Brajesh Gupt *et al.* [51] proposed a benchmark for GBS with threshold detection. However, due to the limitations of the method, it can only manage approximately 22 clicks (or photons) when using the entire Titan supercomputer.

Diverging from previous studies, we establish the first classical benchmarking for GBS with threshold detection based on a calculation of the Torontonian function. The proposed method can calculate a single Torontonian for a 50-photon GBS within one day. This work thus describes the largest boson-based quantum simulation and benchmarking to support the quantum computational advantage.

3 Background

3.1 Selected Algorithm: Boson Sampling

In a typical boson sampling experiment, as shown in Fig. 1a, N indistinguishable single photons are sent to an M-port linear optical network, and the output scattered photons are detected by N single-photon detectors. The probabilities of these N-photon events are related to the permanent function of an $N \times N$ sampling matrix, which is shown to be #P-hard for classical computers. However, a large-scale experiment suffers from the formidable

Authorized licensed use limited to: ULAKBIM UASL - DOKUZ EYLUL UNIVERSITESI. Downloaded on October 28,2022 at 12:39:49 UTC from IEEE Xplore. Restrictions apply.

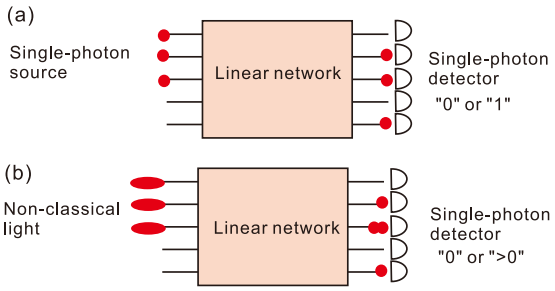


Fig. 1. (a) Standard boson sampling; (b) GBS with threshold detection.

challenge of preparing N perfect single photons [46], [48], [49], [52], [53], [54], [55], [56].

As shown in Fig. 1b, a practical improvement was proposed to change the input single photons to nonclassical Gaussian light and use single-photon detectors without photon-number resolution to register N-click events, which is called GBS with threshold detection [11], [12], [57], [58]. In this case, the probability of N-click events is the Torontonian function of a $2N \times 2N$ sampling matrix, which is expected to still be exponentially hard for classical computers.

In a GBS experiment [12], [59], [60], before photon detection, the output Gaussian state is described by a covariance matrix $\sum$ that can be approximately expressed as

$$\sum = \frac{2-\eta}{2} I + \frac{\eta}{2} VSS^H V^H, \tag{1}$$

in which η is a constant around 1. The definitions of S and V can be found in the appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2021.3111185/.

The sampling matrix A is then defined as $A = I - \sum^{-1}$, and the probability $p(S)$ of an N-click event $S = s_1, s_2, \ldots, s_M$ is determined by

$$p(S) = \langle S|\rho_{\sum}|S\rangle = \frac{Tor(A_{\{S\}})}{\sqrt{det(\sum)}}, \tag{2}$$

where $\rho_{\sum}$ is the output quantum state of a covariance matrix $\sum$, S is the threshold click pattern with $s_k \in 0, 1$ and $\sum_{k=1}^{M} = N$, and $A_{\{S\}}$ is the $2N \times 2N$ submatrix of A by selecting the kth and $(M+k)$th row and column when $s_k = 1$. The matrix function $Tor(\cdot)$ is the Torontonian function, which is defined as

$$Tor(A) = \sum_{Z \in P_N} (-1)^{N-|Z|} \frac{1}{\sqrt{|det(I - A_Z)|}}, \tag{3}$$

where P_N is the power set of $1, 2, \ldots, N$, and thus, there are 2^N determinant terms in the summation [12]. The computational complexity of the Torontonian function is $O(2^N)$, which is the same as the permanent function in standard boson sampling with N photons.

In this work, the major task is to calculate the Torontonian function, which has exponential complexity. According to the formulation, the summation loop is suitable for large-scale parallelism, and the determinant is very computationally intensive that needs to be carefully optimized.

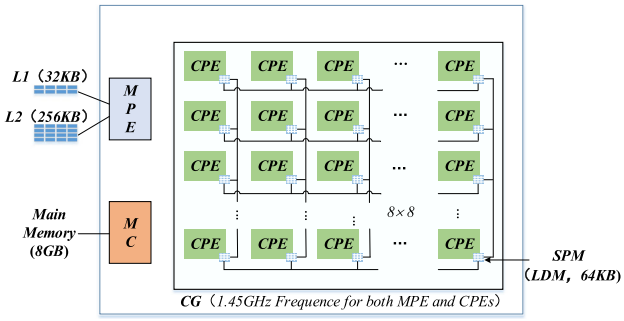


Fig. 2. Block diagram of a CG.

3.2 Sunway Architecture

To achieve the best simulation performance, this work uses the Sunway TaihuLight supercomputer as the classical platform. The system consists of 40,960 many-core processors called SW26010s and can provide a peak performance of 125 PFlops and a sustainable performance of 93 PFlops.

As shown in Fig. 2, each SW26010 processor consists of four core groups (CGs). Each CG has a *memory controller* (MC) with 8 GB main memory, a *management processing element* (MPE) and 64 *computing processing elements* (CPEs). Both MPE and CPEs has the frequency of 1.45 GHz. The MPE is a fully functional 64-bit RISC general-purpose core that a customized linux kernel with a busybox is equipped onto, while the CPE is a simplified 64-bit RISC computing-oriented core that only runs a bare machine daemon communicating with MPE and handles necessary jobs, such as loading ELFs, handling interrupts. MPE contains a 32-KB L1 data cache and 256-KB L2 instruction/data cache. CPE has a private 16-KB L1 instruction cache and 64-KB *scratch-pad memory* (SPM, also called the *local data memory* (LDM)).

Data exchange mechanism is important for such a heterogenuous architecture. *Direct memory access* (DMA) is the primary method for CPEs to access data from global memory, and helps achieve MPE-CPEs communication. Additionally, the effective inter-CPEs communication is enabled by register communication, which provides direct data exchanges between CPEs in the same row or column.

The specific instruction sets of the Sunway architecture provide futher potentials for accelerating the Torontonian function. Here we list some specific instructions that will be used in later sections. As shown in Table 1, the Sunway architecture has 256-bit large integer instruction sets. The first five instructions (multiplication, addition, subtraction and shifting) belong to the 256-bit large integer instruction sets. Note that the multiplication instruction is not a straightforward 256-bit version; instead, it performs an unsigned multiplication of two 128-bit unsigned numbers and obtains a full 256-bit result.

The remainder of the instructions in Table 1 are also very useful for vector instructions. The parameters of instruction `vshff` are a, b and $mask$. a and b are two 256-bit registers that contain four 64-bit numbers. $mask$ provides information on how to construct a new register from a (the first two digits of $mask$) and b (the last two digits of $mask$). Fig. 3a shows an example: based on the four digits of $mask$, 0 and 2 of A (corresponding value a0 and a2), and position 0 and 1 of B (corresponding value b0 and b1) are selected to construct a new register.

Authorized licensed use limited to: ULAKBIM UASL - DOKUZ EYLUL UNIVERSITESI. Downloaded on October 28,2022 at 12:39:49 UTC from IEEE Xplore. Restrictions apply.

TABLE 1
Main Instructions in This Work

Instruction	Description	Latency
umulqa	128-bit unsigned complete multiplication	6
uaddo_carry	256-bit unsigned addition	2
usubo_carry	256-bit unsigned subtraction	2
sllow	256-bit logical left shifting	2
srlow	256-bit logical right shifting	2
vshff	shuffle based on two vectors and a mask	1
vsellt	vector "less than" conditional selection	1

The conditional selection instruction `vsellt` requires the three parameters c, a, and b, which are all 256-bit registers composed of four 64-bit numbers. c is the conditional register, used to make choices between a and b. As shown in Fig. 3b, if one of the 64-bit numbers in c is below 0, then the corresponding 64-bit number of the result comes from the corresponding bits of a and vice versa.

4 Precision Requirement Analysis

Before designing the parallel framework for better computational performance and parallel efficiency, the precision requirement of the quantum simulation shall be analyzed and satisfied. This section points out the shortcoming of using only standard precision types to calculate the Torontonian function and analyzes the minimum precision requirement using mathematic methods. The fastest precision strategy with the fewest bits can be designed based on the analysis of minimum precision requirement.

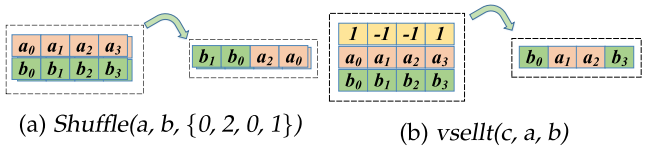
4.1 Motivation

For real-world data, the result of the Torontonian function is a small decimal, which results in a high-precision requirement. For example, when we calculate a practical matrix of size 45, the result is 1.44×10^{-25}. Thus, at least 25 decimal digits (roughly equivalent to 83 binary bits) are required to achieve adequate accuracy in the determinant calculation. Based on the IEEE 754 standard [61], the double-precision type, which has only 53 binary bits of significant precision, cannot sufficiently represent these numbers. Therefore, a customized multiple-precision type should be designed for the Torontonian function.

In this section, we analyze the precision requirement of the Torontonian function, which depends on the upper and lower bounds of the absolute value of the intermediate calculating results. Several mathematic derivations and approximations are used to estimate the bounds. Additionally, the proposed approximation formulae are also a critical and essential part of the adaptive precision framework that will be introduced afterwards.

4.2 Mathematical Analysis

Based on Equation (1) and the analysis in the Appendix attached in the end, available in the online supplemental material, the matrix $A = I - \sum^{-1}$ can be derived and rewritten as



(a) *Shuffle(a, b, {0, 2, 0, 1})* (b) *vsellt(c, a, b)*

Fig. 3. Example (a) vshff instruction, and (b) vsellt instruction.

$$\mathbf{A} = \left(\begin{array}{c|c} mI & -lU^2 \\ \hline -lU^{H^2} & mI \end{array}\right), \tag{4}$$

where m and l are constants only related to r and η described at Equation (1); while U is a random unitary matrix whose values multiplied by l can be proved much smaller than m, which makes the diagonal elements dominant in the determinant. The upper-left and lower-right parts of the sampling matrices are diagonal matrices with diagonal elements less than 1, which gives this kind of matrices some mathematical properties similar to the diagonal matrices.

As for the boundary of the intermediate results for such kind of matrix, the upper bound of the intermediate result of Gaussian elimination is the absolute reciprocal of $det\ B$ (where B denotes to $I - A$), which is intuitive for such a nearly diagonal matrix. This is explained in the Appendix part attached in the end, available in the online supplemental material. Meanwhile, according to the physical characteristics of Gaussian boson sampling, the lower bound is reached by the final result $Tor(A)$.

The upper bound $det\ B$ can be easily calculated with little cost. However, as an NP problem, the final result $Tor(A)$ is difficult to be calculated directly because of the huge cost. Thus, in order to obtain the lower bound, an approximation of $Tor(A)$ should be taken. As Hermitian positive definite matrices, the determinant of B_Z, which denotes to $I - A_Z$ (where Z is a subset of the power set of $0, 2, \ldots, N-1$), can be expanded recursively by the Laplace expansion. With these recursive expansions, $det\ B_Z$ can be rewritten as

$$\begin{aligned} \mathbf{det\ B_Z} &= b_{00}\ det\ \widetilde{B_Z} - x^H adj\ \widetilde{B_Z} x \\ &= det\ \widetilde{B_Z}\left(b_{00} - x^H \widetilde{B_Z}^{-1} x\right) \\ &= \prod_{i=0}^{2N-1}\left(b_{ii} - x_i^H \widetilde{B_{Z_i}}^{-1} x_i\right), \end{aligned} \tag{5}$$

where $\widetilde{B_Z}$ denotes to the submatrix of B_Z by removing the first row and column. B_{Z0} is equivalent to B_Z, and B_{Zi} is equivalent to the $\widetilde{B_{Zi-1}}$, i.e., the submatrix without the first i rows and columns. Similarly, x_i denotes the subvector of x by removing the first i elements.

According to this expression, we can begin the approximation. Considering that the diagonal parts of A or each B_Z are all multiple of I, we can assume that the non-diagonal elements make only few contributions to $det\ B_Z$. The matrix-vector multiplication $x_i^H \widetilde{B_{Z_i}}^{-1} x_i$ can be approximated as the vector multiplication $\frac{x_i^H x_i}{b}$ with a constant b. Then, its numerator $x_i^H x_i$ can be approximated in a linear form as $k|Z|$, which treats each x_i as the same value $\sqrt{k}$. Consequently, the expression can be simplified as

$$\mathbf{det\ B_Z} \approx \left(b - \frac{k|Z|}{b}\right)^{2|Z|}. \tag{6}$$

Authorized licensed use limited to: ULAKBIM UASL - DOKUZ EYLUL UNIVERSITESI. Downloaded on October 28,2022 at 12:39:49 UTC from IEEE Xplore. Restrictions apply.

Substituting this value back into $Tor(A)$ yields a much simpler expression of $Tor(A)$

$$\mathbf{Tor(A)} \approx -1^N + \sum_{i=1}^{N} (-1)^{N-i} \binom{N}{i} \left(b - \frac{ik}{b}\right)^{-i}. \tag{7}$$

The expression is also too complex; thus, we change the fraction $\frac{ik}{b}$ to $\frac{Nk}{b}$ to make it suit to the binomial theorem. With this approximation, we can obtain the final form as

$$\mathbf{Tor(A)} \approx \left(\left(b - \frac{Nk}{b}\right)^{-1} - 1\right)^N. \tag{8}$$

In real-world experiments, $Tor(A)$ is always below $det\ B$; thus, the lower bound can be estimated by this expression.

In real-world experiments, the range of b is 0.84 ± 0.06. Because the diagonal elements affect the determinant of B most strongly, the range of b can be used to determine the range of $det\ B$, which is determined to reach its minimum, 10^{-12} at $N = 50$. After the analysis, the upper bound can be estimated as 10^{12}.

Conversely, if we use a looser formula of Equation (8)

$$\mathbf{Tor(A)} \approx (b^{-1} - 1)^N, \tag{9}$$

which can be guaranteed to be below the real result, the lower bound will reach 10^{-51} at $N = 50$.

Based on the bounds, the range of the intermediate results lies within 10^{-51} and 10^{12}, which can be represented by a large fixed-point number. According to the range, around $12 + |-51| = 63$ significant digits and 12 integer dights are required. A fixed-point precision type with 256 bits is absolutely enough for the Torontonian calculation.

5 Multiple Fixed-Point Precision Design

This section provides the entire precision design of the classical simulation of Gaussian boson sampling. In general, a fixed-point number is composed of an integer in complements form with a scaling factor. The first two subsections introduce our detailed implementations of the fixed-point type based on the Sunway-specific large integer instruction set described in Section 3.2. Moreover, to select the proper fixed-point precision configurations (the scaling factor and the bit length) automatically, an adaptive precision framework based on the mathematical analysis in Section 4 is proposed in the end of this section.

5.1 256-Bit Operations

Based on the calculation of the Torontonian function, five operators are required: addition, subtraction, multiplication, reciprocal and reciprocal square root. Addition and subtraction can be directly implemented by `uaddo_carry` and `usubo_carry` instructions, respectively. While reciprocal and reciprocal square root require marginally complex designs.

Multiplication, which requires several instructions and must be calculated $O(N^3)$ times, is the bottleneck and should be designed carefully. The large integer instruction set only provides the multiplication between two 128-bit unsigned numbers. Constructing a 256-bit signed multiplication based on the 128-bit unsigned multiplication is the major challenge of the multiplication design. A direct method is to convert negative numbers to positive numbers by introducing *IF* statements, which causes a pipeline stall when branch prediction fails. Fortunately, an efficient method without *IF* statements can be designed by exploiting properties of complement.

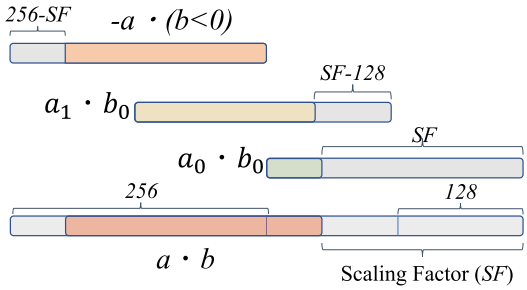


Fig. 4. Scaling factor implemented by shifting.

A 256-bit multiplication between two 256-bit negative numbers a and b is considered as an example to explain the following idea. The complements of a and b can be expressed as $(2^{256} - |a|)$ and $(2^{256} - |b|)$ because they are negative. Then, the unsigned multiplication can be represented and expanded below

$$\begin{aligned} a \times_U b &= (2^{256} - |a|) \times_U (2^{256} - |b|) \\ &= 2^{512} - (|a| + |b|) \times_U 2^{256} + |a| \times_U |b| \\ &= 2^{512} + (a + b) \times_U 2^{256} + a \times_S b \\ &= 2^{512} + (a + b) << 256 + a \times_S b, \end{aligned}$$

where $\times_S$ indicates the signed multiplication and $\times_U$ indicates the unsigned multiplication. Consequently, $a \times_S b$ can be expressed as $a \times_U b - (a + b) << 256$, in which the term 2^{512} can be truncated for a 512-bit number. Considering the four combinations of positive and negative, the signed multiplication can be expressed as

$$\begin{aligned} a \times_S b = a \times_U b &- (a << 256) \cdot (b < 0) \\ &- (b << 256) \cdot (a < 0), \end{aligned} \tag{10}$$

$a \times_U b$ can be further rewritten as $(a_1 \times_U b_1) << 256 + (a_1 \times_U b_0 + a_0 \times_U b_1) << 128 + a_0 \times_U b_0$ for 256-bit multiplication due to the limitation of bit length, where a_1 and a_0 are the upper and lower 128 bits of a, respectively.

The remaining problem is to handle the computation of *scaling factor* (*SF*). For the consideration of efficiency, scaling is achieved by a shift operation. Fig. 4 describes the shifting factor of each term in the expansion of $a \times_S b$. In the figure, the orange bar, which indicates $a \cdot (b < 0)$, $b \cdot (a < 0)$, and $a_1 \cdot b_1$, has a left-shifting factor of $256 - SF$, and the yellow and green bars have right-shifting factors of $SF - 128$ and SF, respectively.

Combining all the derivations above, Algorithm 1 describes the pseudo code of the proposed 256-bit signed multiplication based on the large integer instruction set. Line 2 - 3 use `vshff` instructions to obtain a_1 and b_1 from a and b. Line 4 uses `vshff` instructions to obtain the sign bit a_s of a from a. Line 6 is used to calculate $a \cdot (b < 0)$ using the `vsellt` instruction. Line 8 - 11 are used to calculate

Authorized licensed use limited to: ULAKBIM UASL - DOKUZ EYLUL UNIVERSITESI. Downloaded on October 28,2022 at 12:39:49 UTC from IEEE Xplore. Restrictions apply.

four 128-bit multiplications. Line 12 - 14 and Line 17 manage left and right shifting. The left lines with respect to `uaddo_carry` and `usubo_carry` are used to sum all terms to obtain the final result. Thus, one 256-bit signed multiplication requires 19 instructions.

Algorithm 1. 256-Bit Signed Multiplication

Require: 256-bit signed number a and b
1: **function** mul256a, b
2: $a_1 \leftarrow \text{VSHFF}(0, a, 0\text{x}0\text{e})$ ▷ SHUFFLE(0, a, {2, 3, 0, 0})
3: $b_1 \leftarrow \text{VSHFF}(0, b, 0\text{x}0\text{e})$ ▷ SHUFFLE(0, b, {2, 3, 0, 0})
4: $a_s \leftarrow \text{VSHFF}(0, a, 0\text{xff})$ ▷ SHUFFLE(a, a, {3, 3, 3, 3})
5: $b_s \leftarrow \text{VSHFF}(0, b, 0\text{xff})$ ▷ SHUFFLE(b, b, {3, 3, 3, 3})
6: $a_x \leftarrow \text{VSELLT}(b_s, a, 0)$
7: $b_x \leftarrow \text{VSELLT}(a_s, b, 0)$
8: $c \leftarrow \text{UMULQA}(a, b)$
9: $c_{10} \leftarrow \text{UMULQA}(a_1, b)$
10: $c_{10} \leftarrow \text{UMULQA}(a, b_1)$
11: $c_{11} \leftarrow \text{UMULQA}(a_1, b_1)$
12: $c \leftarrow \text{SRLOW}(c, SF)$
13: $c_{10} \leftarrow \text{SRLOW}(c_{10}, SF - 128)$
14: $c_{01} \leftarrow \text{SRLOW}(c_{01}, SF - 128)$
15: $c_{11} \leftarrow \text{USUBO\_CARRY}(c_{11}, a_x)$
16: $c_{11} \leftarrow \text{USUBO\_CARRY}(c_{11}, b_x)$
17: $c_{11} \leftarrow \text{SLLOW}(c_{11}, 256 - SF)$
18: $c \leftarrow \text{UADDO\_CARRY}(c, c_{10})$
19: $c \leftarrow \text{UADDO\_CARRY}(c, c_{01})$
20: $c \leftarrow \text{UADDO\_CARRY}(c, c_{11})$
21: **return**c
22: **end Function**

For reciprocals and reciprocal square roots that are not bottlenecks, bitwise-based trial division approaches are used and carefully optimized at the instruction level.

5.2 128-Bit Operations

Research on precision reduction has become a new trend in scientific computing recently because increasing existing commercial hardware supports effective and efficient low-precision components that were designed for deep learning originally. By reasonably reducing the precision, the performance of scientific computing can be markedly improved without affecting correctness.

The proposed idea can be used to calculate the Torontonian function from two reasons. One reason is that the requirement of 256 bits is not strict. The requirement derives from the analysis of upper and lower bounds of some extreme cases in Section 4. However, real-world data rarely reaches these bounds, and thus 256-bit precision is too expensive and wasteful for these data. The other reason is that performance can be improved by reducing the precision from 256 bits to 128 bits, since the specific large integer instruction set intrinsically supports 128-bit multiplication. Thus, 128-bit operations are supported in the proposed precision design to provide a higher performance option with reduced but sufficient precision.

Equation (10) can also be used to construct the 128-bit signed multiplication algorithm. Compared to the 256-bit multiplication, it is no longer necessary to construct the term $a \times_U b$ with four multiplications, which are costly. Algorithm 2 describes the 128-bit multiplication at the instruction level, which requires only 9 instructions and is thus approximately twice as faster as the 256-bit version.

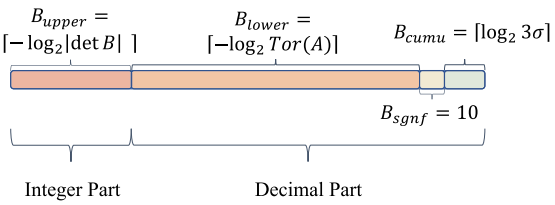


Fig. 5. Four parts of the *estimated bit length* (EBL) calculation. The $log_2 x$ in the figure is used to calculate the number of required bits according to the estimated value.

Algorithm 2. 128-Bit Signed Multiplication

Require:128-bits signed number a and b
1: **function** mul128a, b
2: $c \leftarrow \text{UMULQA}(a, b)$
3: $a_s \leftarrow \text{VSHFF}(a, 0, 0\text{x}55)$ ▷ SHUFFLE(0, a, {1, 1, 1, 1})
4: $b_s \leftarrow \text{VSHFF}(b, 0, 0\text{x}55)$ ▷ SHUFFLE(0, b, {1, 1, 1, 1})
5: $a_x \leftarrow \text{VSELLT}(b_s, a, 0)$
6: $b_x \leftarrow \text{VSELLT}(a_s, b, 0)$
7: $c_1 \leftarrow \text{UADDO\_CARRY}(a_x, b_x)$
8: $c \leftarrow \text{SRLOW}(c, SF)$
9: $c_1 \leftarrow \text{SLLOW}(c_1, 128 - SF)$
10: $c \leftarrow \text{USUBO\_CARRY}(c, c_1)$
11: **return**c
12: **end Function**

However, the algorithms designed for 128-bit reciprocals and reciprocal square roots are similar to the 256-bit versions with several trivial modifications.

5.3 Adaptive Precision Framework

After addressing the precision implementations, the remaining work is to find a way to determine the configurations of the fixed-point type, i.e., the scaling factor and the bit length. The scaling factor, which only depends on the upper bound of the intermediate computing result, can be obtained during the progress of estimating the bit length. Thus we mainly focus on the selection of bit length.

According to the proposed precision implementations, there are two options for the bit length — either 128 bits or 256 bits. To select one of them, the estimated requirement of number of precision bits, which is called the *estimated bit length* (EBL), shall be calculated. The 128- or 256-bit precision type can be selected by comparing EBL and 128. If EBL is greater than 128, the 256-bit precision type should be selected and vice versa. EBL must avoid any false positive cases, i.e., there is no case where EBL is less than 128 but 256-bit precision type is required.

Fig. 5 describes each part of the estimate formula of EBL. To calculate EBL, the integer part that depends on the upper bound B_{upper} should be determined first. To avoid false positive, B_{upper} must be able to represent the strict upper bound. Any intermediate computing results that are greater than B_{upper} will lead to overflow errors. The decimal part is comprised of three estimated parts B_{lower}, B_{sgnf}, and B_{cumu} should be calculated. B_{lower} denotes the estimated lower bound, which indicates the number of leading zeros after the decimal point of the Torontonian function. To

Authorized licensed use limited to: ULAKBIM UASL - DOKUZ EYLUL UNIVERSITESI. Downloaded on October 28,2022 at 12:39:49 UTC from IEEE Xplore. Restrictions apply.

avoid false positive, the B_{lower} needs to be an estimated value less than the actual value. B_{sgnf} denotes the number of significant digits that is fixed at 10 (which is the number of binary bits of 3 decimal dights) in this work due to the scientific requirement. B_{cumu} denotes the cumulative error caused by the sum of many truncated $det\ B$.

The bound analysis in Section 4 provide algorithms for B_{upper} and B_{lower}. For B_{upper}, we can directly count the bit number of the reciprocal of $det\ B$ rather than using an estimated value. As mentioned, $det\ B$ is a strict upper bound and thus B_{upper} is qualified as the integer part of EBL. For B_{lower}, because the cost of calculating Torontonian function is too high, Equation (8) is used to provide the estimated lower bound. Note that Equation (8) is supposed to be a lower bound estimate of Torontonian function. Therefore, the two formulae from Section 4 satisfy the requirement of EBL calculation.

For the cumulative error B_{cumu}, probability analysis is used. Each truncation forms an uniform distribution whose expectation is 0, and standard deviation σ_u can be simply approximated as $EBL/2\sqrt{3}$. With the central limited theorem, the sum of $2^N N^2$ truncation that is independent and identically distributed can be approximately expressed as a normal distribution, of which the standard deviation σ is $\sigma_u\sqrt{2^N N^2}$. Based on the 3σ principle, the error in the range of 3σ has a correct rate of 99.74%. Therefore, the cumulative error B_{cumu} can be calculated by 3σ.

According to the EBL, the precision type can be finally determined. As mentioned, the bit length of the precision type depends on the relationship between EBL and 128. Besides, the scaling factor is equal to B_{upper}.

6 Parallel Framework

Since the precision problem is solved by a customized precision method mentioned in previous two sections, this section focuses on obtaining an effective parallel framework for the computational kernel described in Equation (3) based on the selected precision method. Generally, in order to achieve the highest floating-point efficiency, the issues of workload partition and memory access must be carefully addressed. Our parallel framework is able to distribute the workloads almost evenly (Section 6.1), and eliminate the memory access in the computing kernel (Section 6.2).

6.1 Load-Balanced Partition Strategy

For the Torontonian function described in Equation (3), the best dimension to partition is the enumeration of Z due to its embarrassing parallelism. To describe the proposed strategy, we map each Z onto a number mask "$mask_Z$" ranging from 1 to $2^N - 1$. If $i \in Z$, the ith bit in the binary representation of $mask_Z$ is "1" and vice versa.

To balance the load, we classify all masks into N groups based on $|Z|$ (i.e., the number of "1" in the $mask_Z$). Matrices A_Z within a group have the same size, and different groups are calculated in order. Because computation time only depends on the size of these matrices, the proposed strategy can obtain a nearly perfect load balance.

Algorithm 3 describes the pseudo code of our parallel framework of Torontonian function based on the proposed partition strategy. Line 3 indicates the enumerating of the N groups that we have classified; in Line 4, the masks assigned to a process or a thread are determined; in Line 6, the calculating matrix A_Z is generated by the current mask and the input matrix A; Line 7 is the determinant calculation, which is the most expensive part of the calculation; Line 8 is the updating of the result; and in Line 9, the next mask is derived from the current mask, and then, the next loop begins.

Algorithm 3. Calculate $Torontonian(A)$

Require: Matrix A
Ensure: A is Hermitian positive definite
1: **function** $TorontonianA$
2: $result \leftarrow 0$
3: **for** $i_{|Z|} = 1 \rightarrow N$ **do**
4: get $mask_Z$ and $mask_ZEnd$ of the process or the thread
5: **while** $mask_Z \neq mask_ZEnd$ **do**
6: $A_Z \leftarrow \mathrm{GEN\_A_Z}(mask_Z)$
7: $det \leftarrow \mathrm{GET\_DETERMINANT}(I - A_Z)$
8: $result \leftarrow result + (-1)^{N-i_{|Z|}} \frac{1}{\sqrt{|det|}}$
9: $mask_Z \leftarrow \mathrm{GET\_NEXT\_MASK}(mask_Z)$
10: **end while**
11: **end for**
12: **return** $result$
13: **end Function**

6.2 Zero-Memory-Access Storage Strategy

According to Algorithm 3, two matrices need to be stored during the calculation of Torontonian function. One is the input matrix of Torontonian function A, which is used to generate the calculating matrices $I - A_Z$ based on the enumerated masks (Line 6); the other is the calculating matrix $I - A_Z$, which needs to perform a determinant calculation with customized precision (Line 7). When N is 50 in 256-bit precision mode, each complex number requires $64\ B$ storage space. The total storage requirement of input or calculating matrix is $(2N) * (2N) * 64\ B = 625\ KB$, far exceeding the $64\ KB$ LDM capacity. Therefore, this part aims to reduce the storage requirement of two matrices to put them into LDM entirely and thus achieve zero memory access in the main loop (Line 3-11 of Algorithm 3).

Matrix $I - A_Z$ participates in the customized-precision calculations and thus each element of it needs a full precision storage. We use the Hermitian symmetric and distribute the matrix into several CPEs to reduce LDM space requirements for the calculating matrix. Based on Hermitian symmetry, half of the LDM requirement can be reduced but the LDM space remains insufficient. Distributing the matrix can further save the space. But additional data exchange overhead among CPEs are introduced as well. Fortunately, on the Sunway architecture, the inter-CPEs communication based on the register communication mechanism is very efficient, so that the overhead is greatly minimized.

Fig. 6 shows the utilization of register communication when the matrix is scattered to two CPEs. Each matrix row is alternately assigned to two CPEs. In the upper panel, the row with a pivot is located at CPE 0. In this round, CPE 0 sends the row with a pivot to CPE 1 via register communication. Then, each CPE multiplies this row by a proper scalar and adds the scaled row to other rows to finish reduction. In the next round of row reduction, as shown in the lower

Authorized licensed use limited to: ULAKBIM UASL - DOKUZ EYLUL UNIVERSITESI. Downloaded on October 28,2022 at 12:39:49 UTC from IEEE Xplore. Restrictions apply.

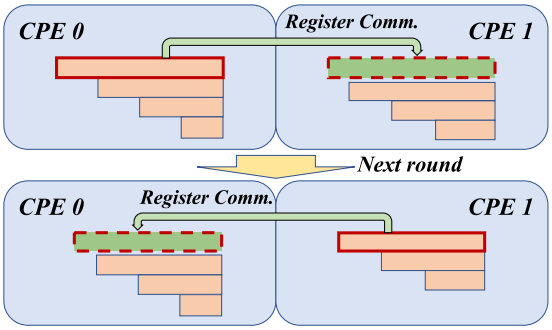


Fig. 6. Gaussian Elimination based on register communication. The upper and lower panels represent two adjacent rounds of reduction. Each block indicates a matrix row. Because the matrix is an upper triangular matrix due to Hermitian symmetry, the lower block has fewer elements than the upper block. A block with a red box indicates a row where the pivot is located. A green arrow indicates the direction of register communication. A green block indicates a buffer for receiving the row with pivot from the other CPE.

panel, the row with a pivot is input to CPE 1, and thus, the data must be transferred from CPE 1 to CPE 0. The reduction process is the same as in the previous round.

As for matrix A, which is only used to generate the calculating matrix, we can apply lossy compression and use matrix symmetries simultaneously. As scientific measurement data, each element of matrix A contains only three to five significant digits. Therefore, a 16-bit number with an error of $\pm 1.525 \times 10^{-5}$ is sufficient to describe the measurement data, and 32-bit lossy compression is also supported for higher precision requirements. Additionally, based on the symmetries derived from physical properties, only $1/4$ elements in the matrix A must be stored.

Due to the two storage strategies, the LDM space is now sufficient for the two matrices. When N is 50, and the matrix $I - A_Z$ is distributed to eight CPEs, each CPE occupied by $625\ KB/2/8 = 39.0625\ KB$ of LDM space, where $625\ KB$ is the LDM requirement with no optimization, the division factor 2 is due to Hermitian symmetry, and 8 is due to the utilization of eight CPEs. On the other hand, when using 32-bit compression, the size of the input matrix is $(2N) * (2N) * 2 * 4/4 = 19.53125\ KB$. The total LDM requirement is approximately $39.0625\ KB + 19.53125\ KB = 58.59375\ KB$, which is below the $64\ KB$ LDM capacity limit. Therefore memory access is no longer required in the main loop.

7 Optimal Instruction Scheduling

To fully exploit the performance of the GBS algorithm on a classical supercomputer, the instruction arrangement within the kernel of the Torontonian function must be carefully considered. Unfortunately, on the Sunway architecture, the default Open64-based compiler yields a poor scheduling solution with the large integer instruction set. Existing open- or closed-source instruction scheduling tools cannot be directly or conveniently ported and used with this special architecture. Given the situation, a new method, which is available and effective on the Sunway architecture, is helpful to accelerate the calculations. Register allocation is another critical problem that must be considered. In the Torontonian function, the kernel is a multiple-precision complex multiplication that is composed of at least three real multiplications [62], which may lead to a shortage of register resources. Thus, optimal instruction scheduling may not produce the fastest solution without awareness of register usage due to undesirable and additional overhead from saving and loading registers. To optimize the two interdependent problems simultaneously, we propose a shortest-path-based method to obtain an optimal solution.

7.1 DAG-Based Data Dependence

Before introducing the shortest-path-based method, we first describe a directed acyclic graph (DAG) to indicate data dependences and several symmetric optimizations.

In the field of instruction scheduling, DAG is widely used to indicate data dependences [63], which is also suitable for this work. Fig. 7 shows the entire design of the DAG for two 256-bit fixed-point multiplications, including the basic edges that are derived from data dependences and two types of additional edges based on symmetry.

The right part of the figure describes the skeleton of the DAG based on big nodes. Each big node represents a section of the same instructions in Algorithm 1 that have similar purposes, and each edge between two big nodes implies that their instructions have several dependent relationships. The detailed dependences within a big node and between two nodes are described in the left part of the figure.

In the left part of the figure, local and detailed dependences are described. The `vshff` node from Line 2 - 3 and the `umulqa` node from Line 8 - 11 of Algorithm 1 are selected as a typical example. Each big node has several instructions from two proposed fixed-point multiplications; the number in the bracket after an instruction name indicates which fixed-point multiplication the instruction comes from. The dependencies of the two nodes are shown via the black dotted edges; only the instructions at the corresponding positions have dependences. More importantly, the primary idea of symmetric optimization is shown. In addition to

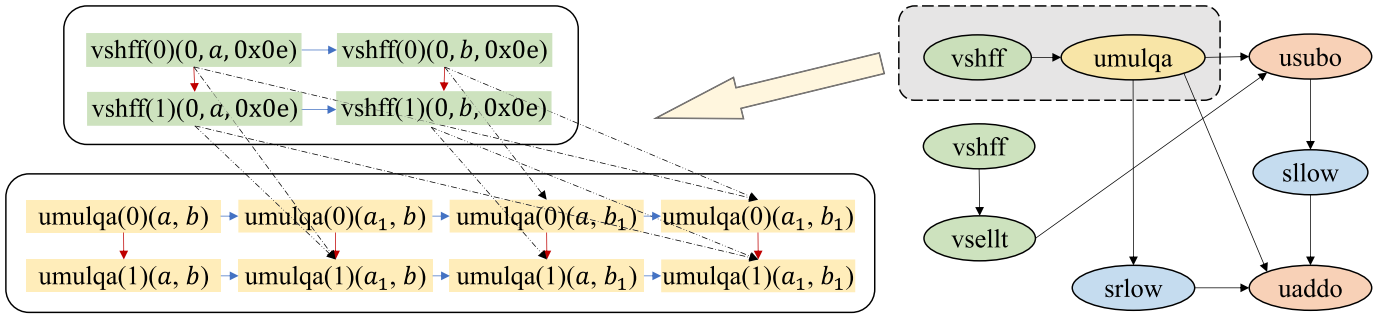


Fig. 7. Directed acyclic graph (DAG) to describe data dependences.

Authorized licensed use limited to: ULAKBIM UASL - DOKUZ EYLUL UNIVERSITESI. Downloaded on October 28,2022 at 12:39:49 UTC from IEEE Xplore. Restrictions apply.

basic dependences, two types of symmetric relationship can be added to the DAG. One derives from the symmetry of the two multipliers. Instructions related to a can always be executed before b. The other one is due to multiple multiplications. Instructions related to previous multiplication can always be executed first. In the figure, dependencies that are colored blue represent the first symmetry, and those that are colored red represent the second symmetry.

7.2 Shortest-Path-Based Solution

In this subsection, we introduce the proposed shortest-path-based solution in detail. We first build a naive graph to be used to manage the instruction scheduling. A node in the graph represents an arbitrary permutation of an arbitrary subset of instructions. There is an edge between two nodes only if the former node can arrive at the latter node by pushing back a given instruction. The weight of an edge depends on the pipeline stall caused by the newly added instruction. The shortest path from the node denoted as the empty set to a node denoted as the full set represents the optimal instruction scheduling.

Even if the naive method has the capacity to find the optimal solution, it requires markedly more time to solve the model. We merge nodes from the naive graph to simplify and optimize the model. The weight of an edge (i.e., the pipeline stall) is irrelevant to the order of instructions ahead and only related to the last few instructions that are not finished in the pipeline. Therefore, we can use a combination instead of a permutation of most instructions and only record the order of the last few instructions.

A tuple (S, M, l) can be used to represent a node, where S is the set of used instructions, M indicates the position and permutation of `umulqa` in the last five positions of instructions, and l denotes the last instruction. In the kernel of the Torontonian function, `umulqa` is the only instruction requiring greater than two clocks, which must be specifically recorded for handling data dependencies. Thus, M is used to manage data dependences stemed from `umulqa`, while l is used for other instructions below or equal to two clocks.

Based on the definition of a node, the only source s is $(\emptyset, \emptyset, \epsilon)$, which indicates that no instructions are scheduled. Its distance function used by shortest-path algorithm is $D(s) = 0$. How to determine the edges of the graph and their weight is the remaining key issue. To address the issue, we consider how to generate all successors of a node instead. For a node (S, M, l), we enumerate each unused instruction i to obtain the successor node $(S + i, M', i)$, where M' indicates the new status of `umulqa` after pushing back instruction i, and calculate the edge weight by the following three steps:

1) Checking dependences. To guarantee the availability of adding an edge, instruction i must satisfy the data dependencies represented by the DAG described in the previous subsection. Every instruction i_s in set S is checked to ensure that there is no directed edge from i to i_s in the DAG.
2) Checking data hazard. Data hazard is a major factor that causes pipeline stalls in most architectures and thus must be avoided. Unlike the first step that checks dependencies, the DAG is not used in this step. The data hazard due to `umulqa` instructions in set M is checked first. The last `umulqa` instruction contained data dependence with instruction i determines the clock cycles of pipeline stall. If all `umulqa` instructions have no dependencies, whether the instruction l has dependency and whether it requires two clocks are checked.
3) Checking the structural hazard of the write-back stage. On the Sunway architecture, the write-back stage's structural hazard is another critical factor that impedes pipeline efficiency. If a previous instruction and the current instruction will enter write-back stage in the same clock, the current instruction cannot be issued during that clock. Based on the preliminary result of pipeline stall obtained based on the data hazard, we check whether instruction i conflicts with instructions in M or l in this step. If structural hazards occurs, the issuance of i must be delayed until the conflict disappears.

With regard to the implementation, we choose the classic and effective Dijkstra algorithm, and use a hash table to record tuples and identify nodes.

7.3 Register Usage Awareness Optimization

As mentioned above, register allocation is another critical problem in compilation optimization. Fortunately, the proposed method can conveniently add the awareness of register usage and obtain the optimal register allocation.

Using many dimensions of information is a common method to manage complex multi-target optimization. To manage register usage, two types of additional information with respect to register resources should be added and recorded in the states of all nodes. One is the historical maximum usage of register resources, which indicates the requirement of register resources of a certain instruction permutation. This information can also be used to obtain the clock of the best instruction scheduling scheme under a specific register resources constrains. The other is the current usage of registers, which is used to update the historical maximum. As explained later, the records of current usage information can be omitted because it can be uniquely determined based on the set S.

Several definitions in the field of compiler technology are useful to describe this method. A `program point` is a location between two consecutive instructions. A `temporary` is live at a program point if it holds a value that will be used later. The `live range` of a temporary is the set of program points where the temporary is live.

Fig. 8 shows a simple code snippet and the live range of each temporary within it. $t2$ is considered as a typical example. In the figure, the instruction in line 1 writes the result to $t2$ and is called a *WRITE* instruction with respect to $t2$. After line 1, the live range of $t2$ begins, and two instructions read its value, which are called *READ* instructions. After line 3, the current value of $t2$ is never used, and thus the live range ends. At line 5, $t2$ is written again, and its live range begins again. Thus, the live range of a temporary t is composed of several consecutive segments of program points, and each segment is composed of a *WRITE* instruction and following several *READ* instructions with respect to t.

Due to the data dependencies represented by a DAG, for each temporary t, the order of *WRITE* instructions cannot

Authorized licensed use limited to: ULAKBIM UASL - DOKUZ EYLUL UNIVERSITESI. Downloaded on October 28,2022 at 12:39:49 UTC from IEEE Xplore. Restrictions apply.

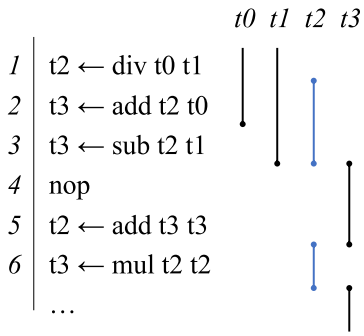


Fig. 8. Live ranges for an example code snippet.

be changed. Additionally, *READ* and *WRITE* instructions cannot be exchanged. Only *READ* instructions between two *WRITE* instructions can be rearranged, or scheduled. Thus, if the number of instructions related to t are determined, whether t is live or not is determined regardless of the scheduling of instructions. Thus, when set S is determined, the number of instructions for each temporary is determined and thus the current register usage is determined.

Now, nodes in the new graph that aim to manage register allocation can be defined as (S, M, l, r), where r represents the historical maximum of the number of register usage. For the source node $(\emptyset, \emptyset, \epsilon, r_0)$, r_0 denotes the number of registers of the input parameters. When adding a new instruction i, the current register usage c is recalculated and used to update and determine the new r' of successor node $(S + i, M', i, r')$. If c is greater than r, r' is set to c, otherwise r' is equal to r.

7.4 Effectiveness Analysis

This part presents the effectiveness analysis of the proposed instruction scheduling method. The Knuth's algorithm [62], by which a complex multiplication can be expressed by 3 multiplications and 5 additions and subtractions, is suitable for the scene where the real multiplication is much slower than real addition and subtraction. Table 2 shows the results of 128-bit and 256-bit complex multiplications based on the Knuth's algorithm.

In the table, the `Naive-Clocks` indicates the CPU clocks of a naive method, in which all multiplications and additions are performed sequentially without rearrangement. `Opt-Clocks` and `Opt-RegNums` indicate the CPU clocks and register usage of the proposed optimal method, respectively. To analyze the effectiveness, the 256-bit complex multiplication is considered as an example. The naive method requires 130 clocks, which is much larger than 62, i.e., the number of fixed-point instructions. The extra clock consumption derives from two sides: one is the pipeline stalls inside the multiplication; the other is register moving and saving/storing data to/from the stack introduced by register allocation.

The shortest-path-based method can accelerate the performance on both two sides. For the pipeline stall, the method provides a 64 clocks instruction scheduling, which only remains two stalls. For the register allocation, the optimal scheduling interleaves three fixed-point multiplications are interleaved, each of which requires totally 12 variables according to the Algorithm 1. The proposed method can provide the optimal register allocation of 14 registers for these $12 \times 3 = 36$ variables, when the scheduling is also optimal. As a result, the optimal method is approximately 2 times faster than the naive method and results in more than 95% pipeline efficiency.

TABLE 2
Results of the Shortest-Path-Based Methods on 128-bit and 256-bit Kernels

Precision	Instructions	Naive-Clocks	Opt-Clocks	Opt-RegNums
128 bits	32	68	34	9
256 bits	62	130	64	14

The results of the proposed method in the 128-bit precision mode also show the remarkable improvement.

8 Evaluation

8.1 Performance Analysis

This section discusses the performance results. Unless otherwise specified, we use 39,286 nodes (157,144 processes) of the Sunway TaihuLight supercomputer for experimentation, which is nearly the entire capacity of the supercomputer (a total of 40,960 nodes).

8.1.1 Time to Solution

Fig. 9 shows the execution time of both precision modes based on a log-transformed distribution. In the figure, two obvious parts can be found — the left nonlinear part and the right linear part. Initially, the execution time is nonlinear and grows slowly because the workload occupancy is insufficient. At arount 27 photons, the workloads become sufficient to the entire Sunway Taihulight and the shape of the execution time line changes. Due to the computation's high scalability of our method, the line is almostly linear in the right part.

To obtain a sample, the computer typically must calculate approximately 100 probabilities of the candidate samples using the Markov Chain Monte Carlo (MCMC) sampling method [50]. To calculate one Torontonian probability for a 50-click photon detecting event, the execution time in the proposed benchmark is 73,773s (approximately 20 hours) for 128-bit mode and 170,891s (approximately 2 days) for 256-bit mode, respectively.

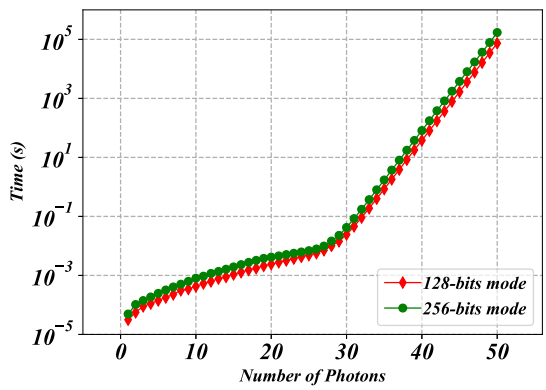


Fig. 9. Execution time for obtaining a single Torontonian. Two precision modes are tested with different number of photons on the entire Sunway TaihuLight supercomputer.

Authorized licensed use limited to: ULAKBIM UASL - DOKUZ EYLUL UNIVERSITESI. Downloaded on October 28,2022 at 12:39:49 UTC from IEEE Xplore. Restrictions apply.

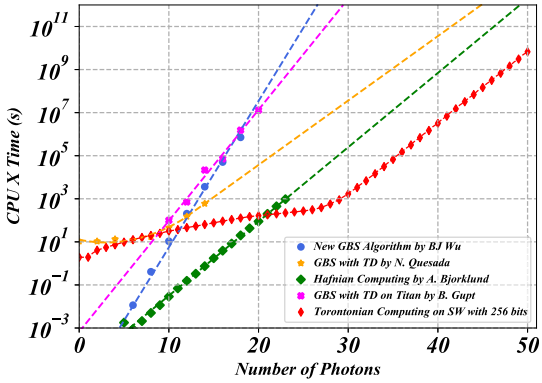


Fig. 10. $CPU \times Time$ results of various GBS classical benchmarks. Red line indicates the proposed method in 256-bit precision mode. Blue, yellow, green, and pink lines indicate the results from [64], [65], [66], and [51], respectively.

Fig. 10 shows a comparison with other methods that handle GBS. In the figure, most methods cannot process 50 photons, except for the proposed method (red line) and the results of [66] (green line). Compared to the study represented by the green line, the proposed method is hundreds of times faster. The time-to-solution results exemplify the excellent performance of the proposed method.

8.1.2 FLOPS

For FLOPS counting, we count all arithmetic operations in the Torontonian function. In general, a real space linear solver requires $2/3n^3 + O(n^2)$ *floating- or fixed-point operations* (FLO) with a size n matrix [67]. However, we must adjust the method of FLO counting due to the particularity of determinant calculation in the Torontonian function. First, the input matrix is guaranteed to be an Hermitian positive definite complex matrix, which can be solved by complex space Chomsky decomposition. Additionally, the matrix size n is typically small and thus the term $O(n^2)$ should not be omitted. Therefore, we count the operations of the determinant calculations accurately combining these two factors, as shown in Formula (11)

$$FLO_{det} = \frac{4}{3}n^3 + n^2 - \frac{4}{3}n. \tag{11}$$

To obtain the total FLO of Torontonian function, the FLO_{det} is substituted into the determinant part of Torontonian function (Equation (3)). As the determinant parts with the same N have the same FLO_{det}, summation of them can be replaced by one term FLO_{det} multiplied by a factor. So the number of summations in Equation (3) can be reduced from 2^N to N in the FLO_{tor} equation. The derivation result can be shown as follow:

$$FLO_{tor} = \sum_{i=1}^{N} \binom{N}{i} \left(\frac{32}{3}i^3 + 4i^2 - \frac{8}{3}i\right). \tag{12}$$

Fig. 11 shows the performance results of two different precisions based on the final formula. The highest sustained performance of 128-bit mode and 256-bit mode are 2.78 PFLOPS when $N = 45$ and 1.27 PFLOPS when $N = 37$; the highest peak performance is 3.67 PFLOPS when $N = 45$ and 1.85 PFLOPS when $N = 41$, respectively.

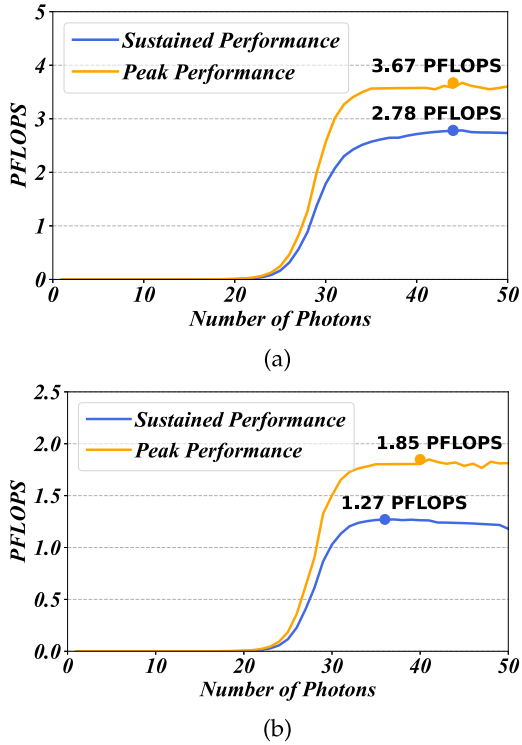


Fig. 11. Peak and Sustained Performance of (a) 128-bit precision; (b) 256-bit precision.

The highest FLOPS did not occur at $N = 50$ because when N becomes larger, LDM requirement becomes larger. Thus, as mentioned in Section 6.2, a calculating matrix is distributed to more CPEs with larger N, which results in additional overhead, as explained in Section 8.1.5. This issue is particularly severe in 256-bit mode.

8.1.3 Scalability

For strong scalability, we choose the data scale $N = 36$ and use 4,096 to 131,072 processes. Fig. 12 shows the strong scalability results that both precision modes achieve a nearly linear strong scalability.

8.1.4 Effectiveness of Each Optimization

The effectiveness of each optimization are evaluated in this subsection. *Basic*, *Storage*, and *All* are the three evaluated versions with different optimizations, respectively. *Basic* is the baseline version, which only contains a basic parallel framework including the load-balanced partition strategy and the proposed fixed-point precision operations. It has no

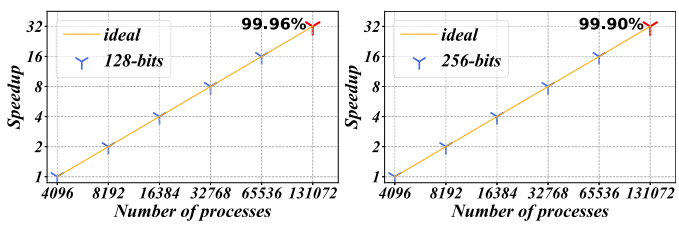


Fig. 12. Strong scalability of (a) 128 bits; (b) 256 bits.

Authorized licensed use limited to: ULAKBIM UASL - DOKUZ EYLUL UNIVERSITESI. Downloaded on October 28,2022 at 12:39:49 UTC from IEEE Xplore. Restrictions apply.

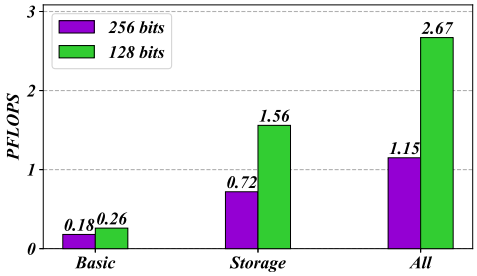


Fig. 13. PFLOPS of time cost of each version with different optimizations when $N = 50$.

deep computational and storage optimization. The matrix is not distributed at the CPE level. So each CPE needs to handle an entire matrix, which causes insufficient LDM space. Therefore, in every Gaussian elimination step, the matrix needs to be read into the LDM by DMA gradually, and transferred back to the main memory afterwards, which brings very heavy memory-access overhead. *Storage* version applies all proposed storage strategy in Section 6.2 to achieve zero-memory-access in the main loop. In this version, the computational kernel simply uses 256-bit precision operations without fine optimizations, which is also the naive version in Section 7.4. *All* is the final version with full optimizations, including the storage reduction and instruction scheduling.

Fig. 13 shows the PFLOPS of each version when N is 50. According to the figure, compared with the *Basic* version, the storage optimizations bring an overall acceleration of 4-6 times; the instruction scheduling contributes to another 1.6-1.7 times speedup based on the *Storage* version. Compared with the *Basic* version, the *All* version is 6.39 and 10.27 times fast on 128-bit and 256-bit precision mode, respectively.

8.1.5 Performance of Each Part

Fig. 14 shows the percentage of time cost of each part when $N = 50$. As shown in Algorithm 3, *GenAz* corresponds to Line 6, which is used to generate the matrix A_Z based on the current mask. *Inv*, *Sync*, *RCom*, and *Elimi* correspond to Line 7 (i.e., the function `get_determinant`). More specifically, *Inv* relates to the reciprocal calculation, while *RCom* relates to the register communication.

Sync represents the time cost of synchronization with others' CPEs, which primarily occurs when several CPEs are working together to compute an identical matrix. Only one CPE must calculate the reciprocal of the pivot, and the other CPEs simply wait. Fig. 14 shows that the time cost of

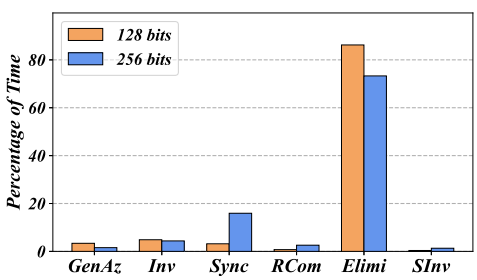
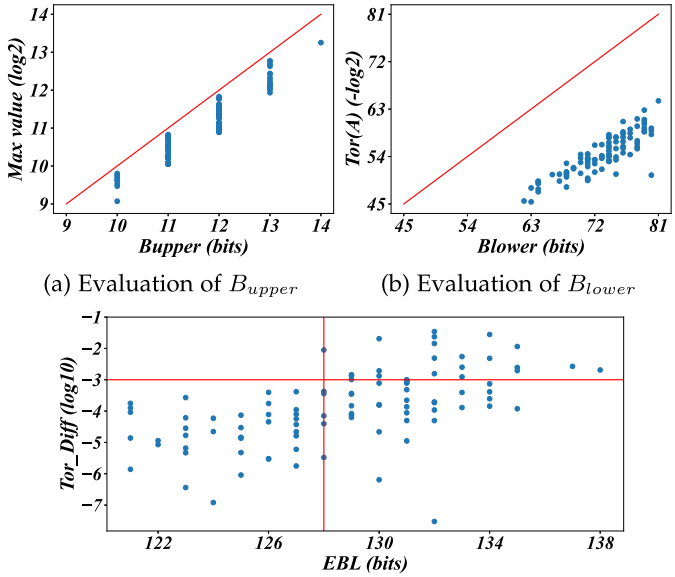


Fig. 14. Percentage of time cost of each part.



(a) Evaluation of B_{upper} (b) Evaluation of B_{lower}

(c) Evaluation of EBL

Fig. 15. Experimental results of 100 random real-world matrices with $n \in [39, 43]$.

Sync in 256 bits is much higher than that in 128 bits because the matrix is distributed to more CPEs in 256 bits; this is the primary factor that affects performance when N becomes larger.

Elimi is related to the inner loop, which is composed of $O(2^N N^3)$ times of the arithmetic operations and thus is the bottleneck that takes up $70\% \sim 90\%$ of the time.

Last, *SInv* corresponds to Line 8, which computes the reciprocal square root.

8.2 Verification for Adaptive Precision Framework

Although the availability of the proposed adaptive precision framework is guaranteed by the mathematical proof and the physical characteristics of boson sampling as analyzed in Sections 4 and 5, experiments can further describe the accuracy and efficiency of the proposed precision selection algorithm. According to Section 5.3, three factors are important for evaluating the algorithm — B_{upper}, B_{lower}, and EBL. This part provides experiments with some real-world matrices that were provided in [14]. Based on several observations and estimations, the turning point of precision selection occurs when $n = 41$. Thus, 100 real-world matrices with $n \in [39, 43]$ are chosen to set up the experiments. The results are plotted into three figures with different x- and y-axis, to present the evaluations of the three factors respectively.

Fig. 15a are used to evaluate B_{upper}, which illustrates the strict upper bound of the intermediate results. The x-axis denotes the real maximum intermediate result and the y-axis denotes the chosen upper bound B_{upper}, which can be obtained by $1/|det\ B|$. All points are located on the lower right part of the figure splitted by the red line $y = x$, which indicates that B_{upper} is less than the real result at every data point. Moreover, all data points are very close to the line $y = x$, which indicates that the estimation of maximum result is quite accurate.

Authorized licensed use limited to: ULAKBIM UASL - DOKUZ EYLUL UNIVERSITESI. Downloaded on October 28,2022 at 12:39:49 UTC from IEEE Xplore. Restrictions apply.

TABLE 3
FLOPS Compared to Other Architectures

Hardware	Library	Kernel	Bits	Peak.	FLOPS	Effi.
Sunway(SW26010)	-	$Tor(A)$	128	3.06T	93.5G	3.06%
Sunway(SW26010)	-	$Tor(A)$	256	3.06T	46.5G	1.52%
CPU(E5-2680 v3)	QD	FMA	106	0.96T	23.4G	2.43%
CPU(E5-2680 v3)	QD	FMA	212	0.96T	0.97G	0.10%
GPU(Tesla A100)	CAMPARY	FMA	106	9.50T	532G	5.6%
GPU(Tesla A100)	CAMPARY	FMA	212	9.50T	35.1G	0.37%

Fig. 15b shows the evaluation of B_{lower}. The x-axis refers to the real result of Torontonian function (calculated with 256-bit type) and the y-axis refers to the estimated result based on Equation (8). Similar to Fig. 15a, this figure shows that the estimation of B_{lower} is under expectations.

In Fig. 15c, EBL is evaluated. The x-axis refers to EBL. The y-axis *Tor_Diff* refers to the absolute of difference between the 128- and 256-bit results of Torontonian function, which represents the differences between the real results and 128-bit results. When the difference is great than 10^{-3} (i.e., below three significant decimal digits), the 128-bit precision type is insufficient for the Torontonian calculation. The horizontal red line $y = -3$ divides the figure into two parts. The lower part can be calculated with 128-bit type correctly, while the upper part cannot. Meanwhile, the vertical line $x = 128$ is the boundary of the actual selection of precision type made by the proposed algorithm. The figure is thus divided into four parts based on two metrics. The upper left part is empty, which indicates that the proposed precision selection strategy does not produce any false negative results. Consequently, the proposed method provides an accurate and efficient adaptive precision framework.

8.3 Performance Comparisons With Counterparts

This section provides performance results based on different platforms. First, while the Torontonian function has not yet been implemented on traditional platforms, such as Intel CPU and GPU, we cannot obtain straightforward comparisons of the Torontonian function. However, according to the analysis in Section 8.1.5, the most time-consuming part of Torontonian function is the Gaussian elimination part, which is dominated by numerous fused multiply-add (FMA) operations using customized multiple precision type. As a result, it is a reasonable alternative to directly use the FMA kernels (only contain multiple precision FMA operations) for traditional platforms.

In the meantime, as the large integer instruction set is not supported by traditional platforms such as x86 CPU and GPU, our proposed method is still inapplicable. Therefore, for traditional platforms, this paper achieves efficient multiple precision arithmetic operations by representing higher precision numbers with unevaluated sums of several floating-point numbers (e.g., several Doubles). In this way, two Doubles can represent a 106-bit floating-point number ($(\texttt{mantissa} + 1) \times 4$), and four Doubles can represent a 212-bit number. Details of the algorithm based on such representation can be found in some existing efforts [68], [69]. As far as we know, the most state-of-the-art multiple precision libraries are mainly based on such designs, including the QD library [69] that supports CPU, and the CAMPARY library [70] that supports both CPU and GPU. Unlike traditional high precision libraries GMP [71] and MPFR [72], QD and CAMPARY are specifically optimized for 106- and 212-bit types and are thereby more efficient for these two types.

Table 3 lists the performance results based on different platforms. The Sunway design (SW26010) directly shows the peak performance when calculating the Torontonian function, which are 93.5 GFLOPS and 46.5 GFLOPS for 128- and 256-bit precision, respectively. So the efficiency (the ratio of customized multiple precision FLOPS to double precision FLOPS), are 3.06% and 1.52%, respectively. On the other hand, QD library with version 2.3.22 and CAMPARY library with version 01.06.17 are used for CPU (E5-2680) and GPU (Tesla A100) designs, respectively. Intel compiler 2019 and CUDA V11.4 are used as for the compilers. As analyzed before, while the cost of the rest parts can be neglected, only the FMA kernels are implemented on these two platforms.

As shown in Table 3 , due to lower performance of CPU and heavier dependence of automatic vectorization of QD library, the performance of CPU design is the worst. The performance of GPU design in 106 bit mode is better than the Sunway design in 128 bit. But the Sunway design has slightly higher accuracy. The performance of Sunway design in 256 bit (46.5 GFLOPS) is 1.32 times better than the GPU design in 212 bit (35.1 GFLOPS), even though Tesla A100 is two to three generations more advanced. Note that the 256-bit mode is provided for large experiments (e.g., cases with $N \geq 50$), and is usually more important for demonstrating quantum computational advantage. In terms of the efficiency, the Sunway design in 256-bit mode (1.52%) is 4.1 times better than the GPU design (0.37%).

Consequently, the proposed benchmarking (256-bit mode) has better performance and shall be an excellent classical benchmark for GBS with threshold detection.

9 Discussion

In addition to introducing an state-of-the-art classical benchmark for GBS with threshold detection, the proposed method provides certain new insights from the perspective of computing. The proposed instruction-scheduling method is suitable for other applications with a hand-written assembly kernel that may contain specific instruction sets. With minor modifications, the proposed method can also be applied to other platforms to find the optimal scheduling.

The multiple-precision fixed-point design is one of the major contributions of this work. Based on an architecture-specific large integer instruction set, a set of fast operators is achieved to support calculation of the Torontonian function, which includes addition, subtraction, multiplication, reciprocal, and reciprocal square root. The proposed design can be easily applied to other scientific applications that require high accuracy. In addition to the Sunway architecture, any current or future machine with such a specific instruction set can benefit. In future work,

Authorized licensed use limited to: ULAKBIM UASL - DOKUZ EYLUL UNIVERSITESI. Downloaded on October 28,2022 at 12:39:49 UTC from IEEE Xplore. Restrictions apply.

we may increase the number of operators and improve their performance to establish an efficient and cross-platform fixed-point multiple-precision library based on hardware instructions.

Last, the proposed program exhibits good portability by replacing the proposed customized precision design with existing multiple-precision libraries under other platforms.

10 Conclusion

Based on the Sunway TaihuLight supercomputer, we established a state-of-the-art classical simulation of the GBS with threshold detection in demonstrating quantum computational advantage. This benchmark served as the classical counterpart of H.Zhong's study, which is a significant milestone in achieving quantum computational advantage based on photon quantum method [14].

To simultaneously achieve nearly optimal performance and sufficient accuracy, a series of methods, including a partition strategy with excellent load balancing, optimal instruction scheduling based on the shortest path algorithm, multiple-precision fixed-point design based on an architecture-specific instruction set, and adaptive precision mode selection based on upper- and lower-bound estimates, was proposed and used.

Thus, sustained performance of 2.78 PFLOPS for 128-bit precision and 1.27 PFLOPS for 256-bit precision was achieved with a proper N. The largest run enabled us to calculate one Torontonian function with a 100×100 submatrix from 50-photon GBS within 20 hours with 128-bit precision and 2 days in 256-bit precision. To our knowledge, this was the largest quantum computing simulation based on Boson Sampling by using modern supercomputers.

References

[1] A. W. Harrow and A. Montanaro, "Quantum computational supremacy," *Nature*, vol. 549, no. 7671, pp. 203–209, 2017.

[2] R. P. Feynman, "Simulating Physics with computers," *Int. J. Theor. Phys.*, vol. 21. no. 6/7, 1982.

[3] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Rev.*, vol. 41, no. 2, pp. 303–332, 1999.

[4] M. A. Nielsen and I. L. Chuang, "*Quantum Computation and Quantum Information: 10th Anniversary Edition*. New York, NY, USA: Cambridge Univ. Press, 2011.

[5] S. Aaronson and A. Arkhipov, "The computational complexity of linear optics," in *Proc. 43rd Annu ACM Symp. Theory Comput.*, 2011, pp. 333–342.

[6] J. Preskill, "Quantum computing in the NISQ era and beyond," *Quantum*, vol. 2, 2018, Art. no. 79.

[7] A. M. Dalzell, A. W. Harrow, D. E. Koh, and R. L. La Placa, "How many Qubits are needed for quantum computational supremacy?," *Quantum*, vol. 4, 2020, Art. no. 264.

[8] M. J. Bremner, A. Montanaro, and D. J. Shepherd, "Average-case complexity versus approximate simulation of commuting quantum computations," *Phys. Rev. Lett.*, vol 117, no. 8, 2016, Art. no. 080501.

[9] S. Boixo *et al.*, "Characterizing quantum supremacy in near-term devices," *Nat. Phys.*, vol. 14, no. 6, pp. 595–600, 2018.

[10] S. Aaronson and L. Chen, "Complexity-theoretic foundations of quantum supremacy experiments," 2016, *arXiv:1612.05903*.

[11] C. S. Hamilton, R. Kruse, L. Sansoni, S. Barkhofen, C. Silberhorn, and I. Jex, "Gaussian boson sampling," *Phys. Rev. Lett.*, vol. 119, no. 17, 2017, Art. no. 170501.

[12] N. Quesada, J. M. Arrazola, and N. Killoran, "Gaussian boson sampling using threshold detectors," *Phys. Rev. A*, vol. 98, no. 6, 2018, Art. no. 062322.

[13] A. P. Lund, M. J. Bremner, and T.imothy C. Ralph, "Quantum sampling problems, Bosonsampling and quantum supremacy," *npj Quantum Inf.*, vol. 3, no. 1, 2017, Art. no. 15.

[14] H. -Sen *et al.*, "Quantum computational advantage using photons," *Science*, vol. 370, no. 6523, pp. 1460–1463, 2020.

[15] J. M. Arrazola, T. R. Bromley, and P. Rebentrost, "Quantum approximate optimization with Gaussian boson sampling," *Phys. Rev. A*, vol. 98, no. 1, 2018, Art. no. 012322.

[16] K. Brádler, P.-L. D.-Demers, P. Rebentrost, D. Su, and C. Weedbrook, " Gaussian boson sampling for perfect matchings of arbitrary graphs," *Phys. Rev. A*, vol. 98, no. 3, 2018, Art. no. 032310.

[17] J. M. Arrazola and T. R. Bromley, "Using Gaussian boson sampling to find dense subgraphs," *Phys. Rev. Lett.*, vol. 121, no. 3, 2018, Art. no. 030503.

[18] M. Schuld, K. Brádler, R. Israel, and D. B. Gupt, "Measuring the similarity of graphs with a Gaussian boson sampler," *Phys. Rev. A*, vol. 101, no. 3, 2020, Art. no. 032314.

[19] S. Jahangiri, J. M. Arrazola, N. Quesada, and N. Killoran, "Point processes with Gaussian boson sampling," *Phys. Rev. E*, vol. 101, no. 2, 2020, Art. no. 022134.

[20] L. Banchi, M. Fingerhuth, T. Babej, C. Ing, and J. M. Arrazola, "Molecular docking with Gaussian boson sampling," *Sci. Adv.*, vol. 6, no. 23, 2020, Art. no. eaax1950.

[21] J. Huh, G. G. Guerreschi, B. Peropadre, J. R. McClean, and A. A.-Guzik, "Boson sampling for molecular vibronic spectra," *Nat. Photon.*, vol. 9, no. 9, pp. 615–620, 2015.

[22] J. Huh and M.-H. Yung, "Vibronic boson sampling: Generalized Gaussian boson sampling for molecular vibronic spectra at finite temperature," *Sci. Rep.*, vol. 7, no. 1, pp. 1–10, 2017.

[23] L. Banchi, N. Quesada, and J. M. Arrazola, "Training Gaussian boson sampling distributions," *Phys. Rev. A*, vol. 102, no. 1, 2020, Art. no. 012417.

[24] F. Arute *et al.*, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.

[25] E. Pednault *et al.*, "Breaking the 49-Qubit barrier in the simulation of quantum circuits," 2017, *arXiv:1710.05867*.

[26] Z.-Y. Chen, Q. Zhou, C. Xue, X. Yang, G.-C. Guo, and G.-P. Guo, "64-Qubit quantum circuit simulation," *Sci. Bull.*, vol 63, no. 15, pp. 964–971, 2018.

[27] H. De Raedt *et al.*, "Massively parallel quantum computer simulator, eleven years later," *Comput. Phys. Commun.*, vol. 237, pp. 47–61, 2019.

[28] J. Chen, F. Zhang, C. Huang, M. Newman, and Y. Shi, "Classical simulation of intermediate-size quantum circuits," 2018, *arXiv: 1805.01450*.

[29] B. Villalonga *et al.*, "Establishing the quantum supremacy frontier with a 281 Pflop/s simulation," *Quantum Sci. Technol.*, vol 5, no. 3, 2020, Art. no. 034003.

[30] M.-C. Chen *et al.*, "Quantum-teleportation-inspired algorithm for sampling large random quantum circuits," *Phys. Rev. Lett.*, vol. 124, no. 8, 2020, Art. no. 080502.

[31] E. Pednault, J. A. Gunnels, G. Nannicini, L. Horesh, and R. Wisnieff, "Leveraging secondary storage to simulate deep 54-Qubit sycamore circuits," 2019, *arXiv:1910.09534*.

[32] C. Huang *et al.*, "Classical simulation of quantum supremacy circuits," 2020, *arXiv:2005.06787*.

[33] K. De Raedt *et al.*, "Massively parallel quantum computer simulator," *Comput. Phys. Commun.*, vol. 176, no. 2, pp. 121–136, 2007.

[34] M. Smelyanskiy, N. P. D. Sawaya, and A. A.-Guzik, "Qhipster: The quantum high performance software testing environment," 2016, *arXiv:1601.07195*.

[35] T. Häner and D. S. Steiger, "5 Petabyte simulation of a 45-Qubit quantum circuit," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2017, pp. 1–10.

[36] S. Aaronson and D. Gottesman, "Improved simulation of stabilizer circuits," *Phys. Rev. A*, vol. 70, no. 5, 2004, Art. no. 052328.

[37] S. Bravyi and D. Gosset, "Improved classical simulation of quantum circuits dominated by clifford gates," *Phys. Rev. Lett.*, vol. 116, no. 25, 2016, Art. no. 250501.

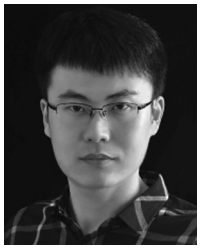
[38] R. S. Bennink *et al.*, "Unbiased simulation of near-clifford quantum circuits," *Phys. Rev. A*, vol. 95, no. 6, 2017, Art. no. 062337.

[39] B. Villalonga *et al.*, "A flexible high-performance simulator for the verification and benchmarking of quantum circuits implemented on real hardware," vol. 5, 2019, Art. no. 86.

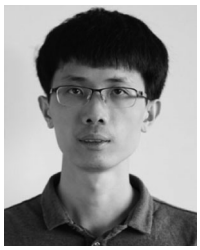
Authorized licensed use limited to: ULAKBIM UASL - DOKUZ EYLUL UNIVERSITESI. Downloaded on October 28,2022 at 12:39:49 UTC from IEEE Xplore. Restrictions apply.

[40] The Fugaku Supercomputer from RIKEN Center for Computational Science. Accessed: Aug. 2020. [Online]. Available: https://www.r-ccs.riken.jp/en/fugaku/project
[41] The Summit Supercomputer from the Oak Ridge National Laboratory. Accessed: Aug. 2020. [Online]. Available: https://www.olcf.ornl.gov/summit/
[42] I. L. Markov, A. Fatima, S. V. Isakov, and S. Boixo, "Quantum supremacy is both closer and farther than it appears," 2018, *arXiv: 1807.10749*.
[43] R. Li, B. Wu, M. Ying, X. Sun, and G. Yang, "Quantum supremacy circuit simulation on sunway Taihulight," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 4, pp. 805–816, Apr. 2020.
[44] S. Aaronson and A. Arkhipov, "The computational complexity of linear optics," *Theory Comput.*, vol. 9, no. 4, pp. 143–252, 2013.
[45] C. S Hamilton, R. Kruse, L. Sansoni, S. Barkhofen, C. Silberhorn, and I. Jex, "Gaussian boson sampling," *Phys. Rev. Lett.*, vol. 119, no. 17, 2017, Art. no. 170501.
[46] M. A. Broome *et al.*, "Photonic boson sampling in a tunable circuit," *Science*, vol. 339, no. 6121, pp. 794–798, 2013.
[47] M. Tillmann, B. Dakić, R. Heilmann, S. Nolte, A. Szameit, and P. Walther, "Experimental boson sampling," *Nat. Photon.*, vol. 7, no. 7, pp. 540–544, 2013.
[48] J. B. Spring *et al.*, "Boson sampling on a photonic chip," *Science*, vol. 339, no. 6121, pp. 798–801, 2013.
[49] A. Crespi *et al.*, "Integrated multimode interferometers with arbitrary designs for photonic boson sampling," *Nat. Photon.*, vol. 7, no. 7, pp. 545–549, 2013.
[50] A. Neville *et al.*, "Classical boson sampling algorithms with superior performance to near-term experiments," *Nat. Phys.*, vol. 13, no. 12, pp. 1153–1157, 2017.
[51] B. Gupt, J. M. Arrazola, N. Quesada, and T. R. Bromley, "Classical benchmarking of Gaussian boson sampling on the titan supercomputer," *Quantum Inf. Process.*, vol. 19, no. 8, pp. 1–14, 2020.
[52] H. Wang *et al.*, "High-efficiency multiphoton boson sampling," *Nat. Photon.*, vol. 11, no. 6, pp. 361–365, 2017.
[53] H. Wang *et al.*, "Toward scalable boson sampling with photon loss," *Phys. Rev. Lett.*, vol. 120, no. 23, 2018, Art. no. 230502.
[54] H. Wang *et al.*, "Boson sampling with 20 input photons and a 60-mode Interferometer in a 1 0 14-Dimensional Hilbert Space," *Phys. Rev. Lett.*, vol. 123, no. 25, 2019, Art. no. 250503.
[55] M. Bentivegna *et al.*, "Experimental scattershot boson sampling," *Sci. Adv.*, vol. 1, no. 3, 2015, Art. no. e1400255.
[56] H.-S. Zhong *et al.*, "12-photon entanglement and scalable scattershot boson sampling with optimal entangled-photon pairs from parametric down-conversion," *Phys. Rev. Lett.*, vol. 121, no. 25, 2018, Art. no. 250505.
[57] S. Paesani *et al.*, "Generation and sampling of quantum states of light in a silicon chip," *Nat. Phys.*, vol. 15, no. 9, pp. 925–929, 2019.
[58] H.-S. Zhong *et al.*, "Experimental Gaussian boson sampling," *Sci. Bull.*, vol. 64, no. 8, pp. 511–515, 2019.
[59] R. Kruse, C. S. Hamilton, L. Sansoni, S. Barkhofen, C. Silberhorn, and I. Jex, "Detailed study of Gaussian boson sampling," *Phys. Rev. A*, vol. 100, no. 3, 2019, Art. no. 032326.
[60] C. Weedbrook *et al.*, "Gaussian quantum information," *Rev. Modern Phys.*, vol. 84, no. 2, 2012, Art. no. 621.
[61] *IEEE Standard for Floating-Point Arithmetic*. IEEE Standard 754, 2008.
[62] D. E. Knuth, *Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Boston, MA, USA: Addison-Wesley, 2014.
[63] M. Smotherman, S. Krishnamurthy, P. S. Aravind, and D. Hunnicutt, "Efficient DAG construction and heuristic calculation for instruction scheduling," in *Proc. 24th Annu. Int. Symp. Microarchitect.*, 1991, pp. 93–102.
[64] B. Wu, B. Cheng, F. Jia, J. Zhang, M.-H. Yung, and X. Sun, "Speedup in classical simulation of Gaussian boson sampling," *Sci. Bull.*, vol. 65, no. 10, pp. 832–841, 2020.
[65] N. Quesada and J. M. Arrazola, "The classical complexity of Gaussian boson sampling," 2019, *arXiv:1908.08068*.
[66] A. Björklund, B. Gupt, and N. Quesada, "A faster Hafnian formula for complex matrices and its benchmarking on a supercomputer," *J. Exp. Algorithmics*, vol. 24, no. 1, pp. 1–17, 2019.
[67] The Top500 List. Accessed: Aug. 2020. [Online]. Available: https://www.top500.org/
[68] T. J. Dekker, "A floating-point technique for extending the available precision," *Numerische Mathematik*, vol. 18, no. 3, pp. 224–242, 1971.
[69] Y. Hida, X. S. Li, and D. H. Bailey, "Algorithms for quad-double precision floating point arithmetic," in *Proc. 15th IEEE Symp. Comput. Arithmetic*, 2001, pp. 155–162.
[70] M. Joldes, J.-M. Muller, V. Popescu, and W. Tucker, "Campary: Cuda multiple precision arithmetic library and applications. in *Proc. Int. Congress Math. Softw.*, 2016, pp. 232–240.
[71] The GMP Library. Accessed: Aug. 2020. [Online]. Available: https://gmplib.org
[72] The MPFR Library. Accessed: Aug. 2020. [Online]. Available: https://www.mpfr.org

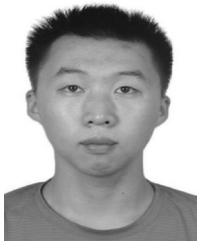
Yuxuan Li (Student Member, IEEE) received the bachelor's degree in computer science from Tsinghua University. He is currently working toward the PhD degree with the Department of Computer Science and Technology, Tsinghua University. He is the head of Climate Modeling Group, National Supercomputing Center, Wuxi. His research interests include high-performance-computing solutions to geo-science applications based on hybrid platforms such as CPUs, GPUs, and the Sunway TaihuLight system. He was the recipient of First Place Award in both ASC Student Supercomputer Challenge 2017 and the ISC Student Cluster Competition 2017.



Lin Gan (Member, IEEE) received the PhD degree in computer science from Tsinghua University. He is currently an associate professor with the Department of Computer Science and Technology and the Beijing National Research Center for Information Science And Technology, Tsinghua University. He is also the assistant director of the National Supercomputing Center, Wuxi. His research interest include HPC solutions based on state-of-the-art systems. He was the recipient of 2016 ACM Gordon Bell Prize, 2018 IEEE-CS TCHPC Early Career Researchers Award for Excellence in HPC, and the 2015 IEEE FPL Most Significant Paper Award in 25 Years.



Mingcheng Chen is currently an associate professor with the University of Science and Technology of China and the Hefei National Laboratory for Physical Sciences, Microscale Nanoscience Laboratory. His research interests include quantum information and quantum foundations.



Yaojian Chen is currently working toward the undergraduate degree with the Department of Physics, Tsinghua University. His research interests include parallel optimization designs and HPC solutions for applications such as quantum mechanics and protein reconstruction with cryo-EM on different platforms.

Haitian Lu received the bachelor's degree in Computer Science and Technology from the Dalian University of Technology. He is currently an engineer with the National Supercomputer Center, Wuxi, China. He is a member of the Parallel Optimization Department. His research interest include parallel optimization of various kinds of programs.

Authorized licensed use limited to: ULAKBIM UASL - DOKUZ EYLUL UNIVERSITESI. Downloaded on October 28,2022 at 12:39:49 UTC from IEEE Xplore. Restrictions apply.

Chaoyang Lu is currently a full professor with the University of Science and Technology of China and Hefei National Laboratory for Physical Sciences, Microscale Nanoscience Laboratory. His research interests include quantum information and quantum foundations.

Jianwei Pan is currently a full professor with the University of Science and Technology of China and Hefei National Laboratory for Physical Sciences, Microscale Nanoscience Laboratory. His research interests include quantum information and quantum foundations.

Haohuan Fu (Member, IEEE) received the PhD degree in computing from Imperial College, London. He is currently a professor with the Ministry of Education Key Laboratory for Earth System Modeling, and the Department of Earth System Science, Tsinghua University, China. He is also the deputy director of National Supercomputing Center, Wuxi, China. His research interests include high-performance computing in earth and environmental sciences, computer architectures, performance optimizations, and programming tools in parallel computing. He was the recipient of ACM Gordon Bell Prize in 2016 and 2017, and Most Significant Paper Award by FPL in 2015.

Guangwen Yang (Member, IEEE) received the PhD degree in computer science from the Harbin Institute of Technology. He is currently a professor with the Department of Computer Science and Technology and Beijing National Research Center for Information Science and Technology, Tsinghua University, China. He is also the director of National Supercomputing Center, Wuxi, China. His research interests include parallel algorithms, cloud computing, machine learning, and the earth system model. He was the recipient of ACM Gordon Bell Prize in 2016 and 2017.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/csdl.

Authorized licensed use limited to: ULAKBIM UASL - DOKUZ EYLUL UNIVERSITESI. Downloaded on October 28,2022 at 12:39:49 UTC from IEEE Xplore. Restrictions apply.