arXiv:2303.00123v1 [quant-ph] 28 Feb 2023

QCLAB++：在图形处理器上模拟量子电路

Roel Van Beeumen[1], Daan Camps[2], Neil Mehta[2]

[1]劳伦斯伯克利国家实验室应用数学与计算研究部，美国加利福尼亚州伯克利[2]劳伦斯伯克利国家实验室国家能源研究科学计算中心，美国加利福尼亚州伯克利

摘要

我们介绍了 qclab++，这是一款轻量级、完全模板化的 C++ 软件包，可用于图形处理器加速的量子电路模拟。该代码没有外部依赖项，且图形处理器内核通过 OpenMP 卸载生成，因此具有很强的可移植性。qclab++旨在实现高性能和数值稳定性，其针对单量子比特门、受控单量子比特门和双量子比特门的模拟算法均经过高度优化。此外，我们还介绍了一个面向 MATLAB 的量子电路工具箱 qclab，其语法仿照 qclab++。这让用户能够灵活、便捷地使用类似 MATLAB 的脚本语言研究量子算法，同时在需要时获得高性能图形处理器加速。因此，qclab++库具备独特的功能组合。我们将 qclab++ 中的 CPU模拟器与通过 OpenMP 生成的图形处理器内核进行比较，观察到加速比超过 $40\times$。此外，我们还在 NERSC 的 Perlmutter 系统上进行了一系列基准测试，将 qclab++ 与 cirq-qsim 和 qibo 等其他电路模拟软件包进行比较，结果表明其具有很强的竞争力。

1 引言

量子计算领域正在迅速发展，近期量子硬件方面的进展便是明证。基于超导 [12] 和离子阱 [14] 的量子处理器在实验上实现纠错方案的初步尝试，是该领域令人鼓舞的重要里程碑。其他量子比特实现方式，例如中性原子[16]，，也展现出巨大潜力。

目前正在开发的大多数量子硬件平台都支持量子计算的电路模型。因此，量子算法研究人员很希望能够使用先进的计算工具，在量子硬件开发期间辅助研究。除了使用量子处理器之外，量子电路模拟器也因多种原因而备受关注。首先，研究人员可以借助它对新算法进行原型设计和研究，而不必耗费宝贵的量子资源来调试和测试想法。这些工作都可以通过在经典硬件上进行模拟来完成。其次，由于目前可用的量子硬件仍受噪声影响，因此无法研究那些所需电路深度超出设备相干性极限的算法。许多值得研究的算法，例如分子科学领域中的算法 [10]，都需要深电路。

这推动了多种量子模拟器的开发，它们各自具有不同的优势和强项。Google Quantum AI 开发了 Cirq 工具包 [4] 及其 qsim 模拟器 [13]；IBM 推出了搭配 Aer 模拟器后端的 Qiskit [11]；而 PennyLane [1] 则是 Xanadu 推出的可微量子编程框架。Qibo [7, 6]，（内含用于量子模拟的即时编译器）和 Quest [9]也开展了类似工作。上述软件包中，除 Quest 外，均提供 Python API 来生成量子程序。

最近，NVIDIA 推出了*cuQuantum* SDK [8]，其中包含一系列用于执行量子门模拟的图形处理器内核。许多现有的量子模拟软件包，例如 Cirq、Qiskit、PennyLane 和 Qibo，已开始支持 cuQuantum SDK，以便在 NVIDIA 硬件上运行电路模拟。

在这项工作中，我们介绍了两种名为`qclab++`[1] [15] 和`qclab`[2] [2]的量子电路编程与模拟替代框架。前者是一个轻量级、完全模板化的C++库，支持通过OpenMP 卸载实现图形处理器加速的状态向量模拟；后者则是一个 MATLAB工具箱，具有易于使用的接口，能够充分利用MATLAB IDE 的优势。两个库提供了类似的编程接口，见图 1，这降低了将模拟代码从一个框架转换到另一个框架的门槛，并支持更流畅的工作流程：先在 MATLAB中对想法进行原型设计，再使用`qclab++`在图形处理器上进行大规模模拟。下面以量子傅里叶变换电路 [5, 3] 的图 1为例进行说明。值得注意的是，两种实现非常相似，易于相互转换。

```
% number of qubits
n = 5;

% quantum circuit
circ = qclab.QCircuit(n);

% blocks
for i=0:n-1
  % Hadamard
  circ.push_back( H(i) );
  % diagonal blocks
  for j=2:n-i
    ctrl = j+i-1;
    th = -2*pi/2^j;
    circ.push_back( ...
      CP(ctrl, i, 1, th) );
  end
end

% swaps
for i=0:floor(n/2)-1
  circ.push_back( ...
    SWAP(i, n-i-1) );
end
```

```
// number of qubits
int n = 5 ;

// quantum circuit
qclab::QCircuit<std::complex<double>> circ(n);

// blocks
for (int i=0; i<n; i++) {
  // Hadamard
  circ.push_back( std::make_unique<H>(i) );
  // diagonal blocks
  for (int j=2; j<=n-i; j++) {
    int ctrl = j+i-1;
    double th = -2*pi/(1<<j);
    circ.push_back(
      std::make_unique<CP>(ctrl, i, th) );
  }
}

// swaps
for (int i=0; i<n/2; i++) {
  circ.push_back(
    std::make_unique<SWAP>(i, n-i-1) );
}
```

图 1：MATLAB 使用 qclab 工具箱实现的量子傅里叶变换电路 [5, 3]（左），以及使用 qclab++ 实现的等效 C++ 代码（右）。

我们的模拟器属于强模拟器，会跟踪波函数中的所有振幅，随着量子电路对波函数进行修改而更新。对于量子系统，这类模拟问题的计算量会随量子比特数呈指数级增长，并随电路中的门数呈线性增长。强模拟的主要优势在于，我们可以在模拟的每一步获取每个振幅，而且所有计算在数值精度范围内都是精确的。主要缺点是波函数的规模呈指数级增长，这限制了经典计算能够模拟的系统规模。这个限制大约在 40 到 50 个量子比特。利用张量网络或波函数的稀疏表示进行近似模拟的方法可以处理更大的系统，但无法以相同的精度近似希尔伯特空间中的每个量子态。

本文其余部分安排如下。在第2节中，我们将介绍经过优化的

[1] qclab++: https://github.com/QuantumComputingLab/qclabpp
[2] qclab: https://github.com/QuantumComputingLab/qclab

单量子比特门、受控单量子比特门和双量子比特门的状态向量模拟器算法。在第3节中，我们将讨论实现细节，以及用于加速OpenMP卸载的qclab++。在第4节中，我们比较qclab++ 的 CPU 和图形处理器实现，并将其与其他经过图形处理器加速的状态向量模拟软件包进行基准测试。

2 高效量子门模拟

将酉量子门作用于 U 给定的量子态 $|\phi\rangle \in \mathbb{C}^{2^n}$ （图 2）在数学上对应于矩阵-向量乘法（$matvec$）

$$|\psi\rangle = (I_l \otimes U \otimes I_r)\,|\phi\rangle\,, \tag{1}$$

其中，$\otimes$ 表示克罗内克积，I_l，I_r 是维度适当的单位矩阵。朴素的量子电路模拟器实现会显式计算式（1）中的克罗内克积，并需要 2 层嵌套 for 循环。

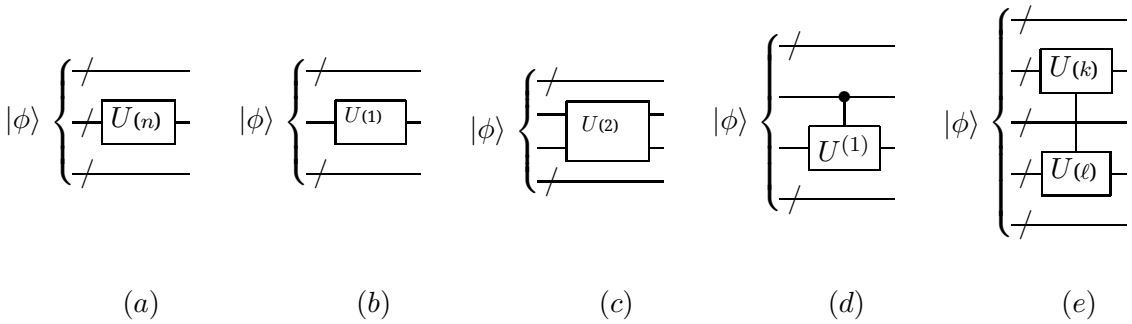


图2：将 (a) 一个稠密的 n 量子比特量子门、(b) 一个单量子比特量子门、(c) 一个双量子比特量子门、(d) 一个受控单量子比特量子门，以及 (e) 一个作用于非相邻量子比特的 $k+\ell$ 量子门应用于量子态 $|\phi\rangle$。

本节介绍更高效的量子门模拟算法，这些算法仅由一个简单的 for 循环与用于索引计算的位运算组成。因此，我们将采用以下定义给出的大端序二进制约定。

定义1（二进制表示）。我们将 $j \in \mathbb{N}$ 的二进制表示定义如下：$0 \le j \le 2^n - 1$

$$j = [\mathtt{j}_0\mathtt{j}_1\cdots\mathtt{j}_{n-1}] = \mathtt{j}_0\cdot 2^{n-1} + \mathtt{j}_1\cdot 2^{n-2} + \cdots + \mathtt{j}_{n-1}\cdot 2^0,$$

其中 $\mathtt{j}_i \in \{0,1\}$，对于 $i = 0$，....., $n-1$。

2.1 单量子比特门

We 从最简单的一类量子门开始，它们仅作用于单个量子比特

$$U := \begin{bmatrix} u_{0,0} & u_{0,1} \\ u_{1,0} & u_{1,1} \end{bmatrix}. \tag{2}$$

如图 3中不同颜色所示，单量子比特门的状态向量模拟可分解为一系列 2×2 矩阵-向量运算

$$\begin{bmatrix} \psi_{\mathtt{i}_0\mathtt{i}_1\cdots\mathtt{i}_{q-1}\,0\,\mathtt{i}_{q+1}\cdots\mathtt{i}_{n-1}} \\ \psi_{\mathtt{i}_0\mathtt{i}_1\cdots\mathtt{i}_{q-1}\,1\,\mathtt{i}_{q+1}\cdots\mathtt{i}_{n-1}} \end{bmatrix} = \begin{bmatrix} u_{0,0} & u_{0,1} \\ u_{1,0} & u_{1,1} \end{bmatrix} \begin{bmatrix} \phi_{\mathtt{i}_0\mathtt{i}_1\cdots\mathtt{i}_{q-1}\,0\,\mathtt{i}_{q+1}\cdots\mathtt{i}_{n-1}} \\ \phi_{\mathtt{i}_0\mathtt{i}_1\cdots\mathtt{i}_{q-1}\,1\,\mathtt{i}_{q+1}\cdots\mathtt{i}_{n-1}} \end{bmatrix}, \tag{3}$$

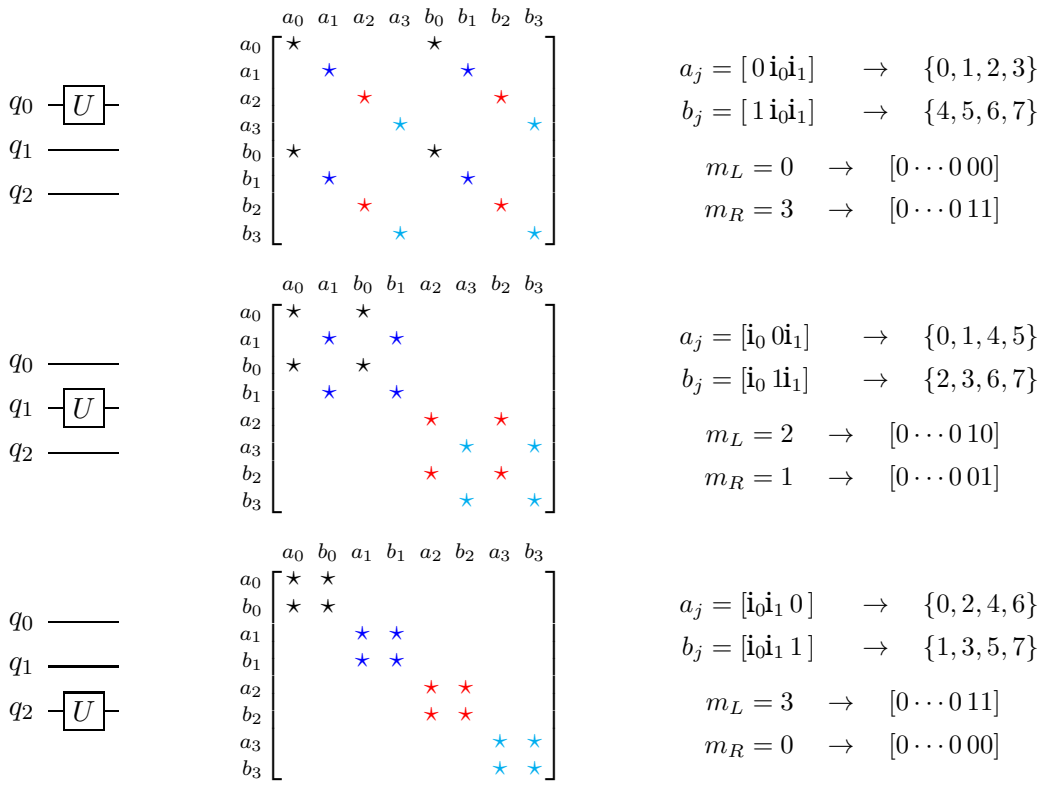


图3：应用单量子比特门 U 的示意图。

其中， q 是门 U 所作用的量子比特。根据 q，式（3）中的索引

$$\begin{aligned}[\mathbf{i}_0\mathbf{i}_1\cdots\mathbf{i}_{q-1}\,0\,\mathbf{i}_{q+1}\cdots\mathbf{i}_{n-1}] &=: a_j,\\ [\mathbf{i}_0\mathbf{i}_1\cdots\mathbf{i}_{q-1}\,1\,\mathbf{i}_{q+1}\cdots\mathbf{i}_{n-1}] &=: b_j,\end{aligned} \qquad j = 0,1,\ldots,2^{n-1}-1, \tag{4}$$

可以彼此相邻，也可以相隔至多 2^{n-1}。现在我们将介绍一种仅基于位运算来计算（4）的高效方法。

首先，我们注意到 a_j 和 b_j 的二进制表示可以分为 3 部分，即分别为 0（和 1）左侧的比特、0（和 1）比特，以及 0（和 1）右侧的比特。不失一般性，我们可以选取比特 $\mathbf{i}_\ell$，使其对应 $0 \le \ell \le n-1$ 和 $\ell \ne q$，因此

$$j = [\mathbf{i}_0\mathbf{i}_1\cdots\mathbf{i}_{q-1}\,|\,\mathbf{i}_{q+1}\cdots\mathbf{i}_{n-1}]. \tag{5}$$

接下来，使用以下左右位掩码

$$m_L := [0\cdots0\,|\,\overbrace{1\cdots1}^{q}\,|\,0\cdots0] = \left(2^{n-1}-1\right) - \left(2^{n-q-1}-1\right), \tag{6}$$

$$m_R := [0\cdots0\,|\,0\cdots0\,|\,\underbrace{1\cdots1}_{n-q-1}] = 2^{n-q-1}-1, \tag{7}$$

我们可以计算 将每个 a_j 表示为 j 与 m_R 按位与的结果，再加上 j 与 m_L 按位与的结果左移后的值 ，

$$a_j = (j \,\&\, m_R) + [(j \,\&\, m_L) \ll 1], \tag{8}$$

相应的 b_j 索引由以下求和得出

$$b_j = a_j + 2^{n-q-1}. \tag{9}$$

一般单量子比特门的向量模拟算法见 算法 1。注意，对于（2）中的某些元素等于 0 的单量子比特门，第 5 行和第 6 行可以简化。例如，泡利-X 门就是一个交换操作

$$\begin{aligned}\psi[a_j] &= \phi[b_j],\\ \psi[b_j] &= \phi[a_j],\end{aligned}$$

以及泡利-Y 门

$$\begin{aligned}\psi[a_j] &= -i\phi[b_j],\\ \psi[b_j] &= \quad i\phi[a_j].\end{aligned}$$

注意，对于泡利-Z 门，只需更新一半的向量元素（b_j）

$$\begin{aligned}\psi[a_j] &= \quad \phi[a_j],\\ \psi[b_j] &= -\phi[b_j].\end{aligned}$$

算法 1：应用单量子比特门

输入：输入态 $|\phi\rangle$。输出：输出态 $|\psi\rangle = U|\phi\rangle$。1 右掩码：$m_R = 2^{n-q-1} - 1$2 左掩码：$m_L = 2^{n-1} - 1 - m_R$对于 $j = 0$，1，...，$2^{n-1} - 1$ 执行3 $a_j = (j \,\&\, m_R) + [(j \,\&\, m_L) \ll 1]$4 $b_j = a_j + 2^{n-q-1}$

5 $\psi[a_j] = u_{0,\,0}\phi[a_j] + u_{0,\,1}\phi[b_j]$

6 $\psi[b_j] = u_{1,\,0}\phi[a_j] + u_{1,\,1}\phi[b_j]$结束

2.2 受控单量子比特门

现在我们考察 受控单量子比特门，并区分以下 4 种不同情况



(10)

其中 q_c 是控制量子比特， q_t 是目标量子比特。如图 4所示，这些受控门只会更新一半的元素。具体更新哪些元素取决于q_c 和 q_t的实际值。

与标准单量子比特门类似，受控单量子比特门的状态向量模拟可分解为一系列 2×2 矩阵-向量运算

$$\begin{bmatrix}\psi_{a_j}\\ \psi_{b_j}\end{bmatrix} = \begin{bmatrix}u_{0,0} & u_{0,1}\\ u_{1,0} & u_{1,1}\end{bmatrix}\begin{bmatrix}\phi_{a_j}\\ \phi_{b_j}\end{bmatrix}, \tag{11}$$

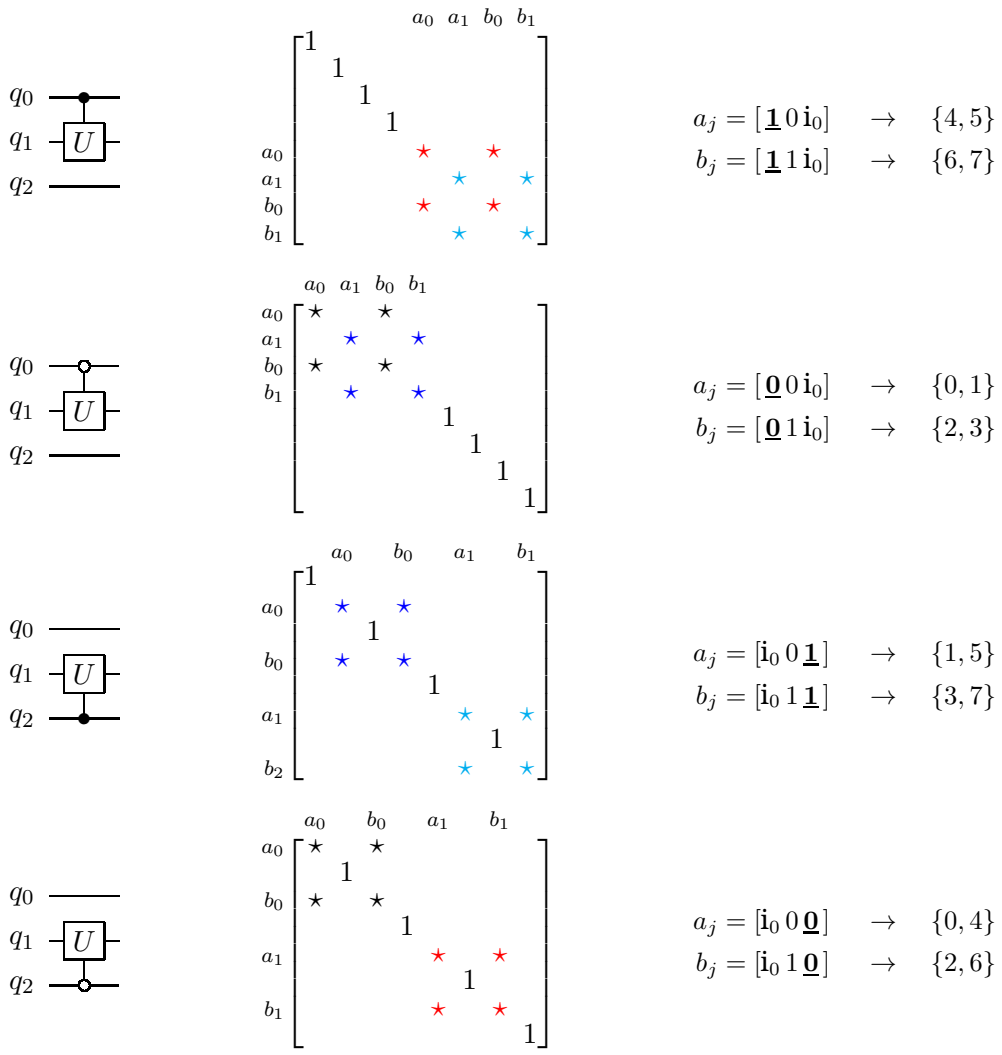


图4：应用受控单量子比特门 U 的图示。

其中我们区分（10）的左侧两种情况，即 $q_c < q_t$,

$$a_j = [\mathtt{i}_0\mathtt{i}_1\cdots\mathtt{i}_{q_c-1} * \mathtt{i}_{q_c+1}\cdots\mathtt{i}_{q_t-1}\,0\,\mathtt{i}_{q_t+1}\cdots\mathtt{i}_{n-1}],$$
$$b_j = [\mathtt{i}_0\mathtt{i}_1\cdots\mathtt{i}_{q_c-1} * \mathtt{i}_{q_c+1}\cdots\mathtt{i}_{q_t-1}\,1\,\mathtt{i}_{q_t+1}\cdots\mathtt{i}_{n-1}], \qquad j = 0,1,\ldots,2^{n-2}-1, \tag{12}$$

以及（10）的右侧两种情况，即 $q_t < q_c$,

$$a_j = [\mathtt{i}_0\mathtt{i}_1\cdots\mathtt{i}_{q_t-1}\,0\,\mathtt{i}_{q_t+1}\cdots\mathtt{i}_{q_c-1} * \mathtt{i}_{q_c+1}\cdots\mathtt{i}_{n-1}],$$
$$b_j = [\mathtt{i}_0\mathtt{i}_1\cdots\mathtt{i}_{q_t-1}\,1\,\mathtt{i}_{q_t+1}\cdots\mathtt{i}_{q_c-1} * \mathtt{i}_{q_c+1}\cdots\mathtt{i}_{n-1}], \qquad j = 0,1,\ldots,2^{n-2}-1, \tag{13}$$

其中 $* = 0$ 用于零控制门，而 $* = 1$ 用于一控制门。请注意，我们只有2^{n-2} 个不同的值对应于 j ，而 2^{n-1} 中的标准单量子比特门有4个值。L
et

$$q_0 = \min(q_c, q_t), \tag{14}$$
$$q_1 = \max(q_c, q_t). \tag{15}$$

使得 $q_0 < q_1$。接下来，我们观察到，式（a_j 和 b_j 12）和（13）中的索引可以分成 5 部分。不失一般性，我们可以选择比特 $\mathtt{i}_\ell$ 对应于 $0 \le \ell \le n-1$ 和 $\ell \ne \{q_0, q_1\}$，于是

$$j = [\mathtt{i}_0\mathtt{i}_1\cdots\mathtt{i}_{q_0-1}\,|\,\mathtt{i}_{q_0+1}\cdots\mathtt{i}_{q_1-1}\,|\,\mathtt{i}_{q_1+1}\cdots\mathtt{i}_{n-1}]. \tag{16}$$

接下来，利用以下位掩码

$$m_L := [0\cdots0\,|\,\overbrace{1\cdots1}^{q_0}\,|\,\overbrace{0\cdots0}^{q_1-q_0-1}\,|\,0\cdots0] = \left(2^{n-2}-1\right) - \left(2^{n-q_0-2}-1\right), \tag{17}$$

$$m_C := [0\cdots0\,|\,0\cdots0\,|\,1\cdots1\,|\,0\cdots0] = \left(2^{n-q_0-2}-1\right) - \left(2^{n-q_1-1}-1\right), \tag{18}$$

$$m_R := [0\cdots0\,|\,0\cdots0\,|\,0\cdots0\,|\,\underbrace{1\cdots1}_{n-q_1-1}] = 2^{n-q_1-1}-1, \tag{19}$$

we 可以将零控制的 a_j 索引计算为以下 3 项之和

$$a_j = (j \,\&\, m_R) + [(j \,\&\, m_C) \ll 1] + [(j \,\&\, m_L) \ll 2], \tag{20}$$

相应的 b_j 索引则由以下求和得出

$$b_j = a_j + 2^{n-q_t-1}. \tag{21}$$

对于一控制门，我们需要将 (20) 和 (21) 都增加 2^{n-q_c-1}。

受控单量子比特门的向量模拟算法见 算法2。与标准单量子比特门类似，对于 (2) 中包含一些零元素的受控单量子比特门，第 8 行和 第 9 行可以简化。例如，CNOT门只需交换一半的元素。

算法2：应用受控单量子比特门

输入：输入态 $|\phi\rangle$。输出：输出态 $|\psi\rangle = U|\phi\rangle$。

1 右掩码：$m_R = 2^{n-q_1-1}-1$ 2 中心掩码：$m_C = 2^{n-q_0-2}-1-m_R$ 3 左掩码：$m_L = 2^{n-2}-1-m_R-m_C$ 对于 $j = 0, 1, \ldots, 2^{n-2}-1$ 执行 4 $a_j = (j \,\&\, m_R) + [(j \,\&\, m_C) \ll 1] + [(j \,\&\, m_L) \ll 2]$ 如果控制态 $= 1$ 则

6 $a_j = a_j + 2^{n-q_c-1}$ 7 $b_j = b_j + 2^{n-q_c-1}$

结束 8 $\psi[a_j] = u_{0,\ 0}\phi[a_j] + u_{0,\ 1}\phi[b_j]$

9 $\psi[b_j] = u_{1,\ 0}\phi[a_j] + u_{1,\ 1}\phi[b_j]$ 结束

2.3 双量子比特门

作用于量子比特 q_0 和 q_1 的一般双量子比特门的状态向量模拟可分解为一系列 4×4 矩阵-向量运算。对于索引计算，我们同样可以使用位掩码（17）–（19），相应的算法见 算法3。

算法3：应用双量子比特门

输入：输入态 $|\phi\rangle$。

输出：输出态 $|\psi\rangle = U|\phi\rangle$。

1 Right mask: $m_R = 2^{n-q_1-1} - 1$

2 Center mask: $m_C = 2^{n-q_0-2} - 1 - m_R$

3 Left mask: $m_L = 2^{n-2} - 1 - m_R - m_C$

for $j = 0, 1, \ldots, 2^{n-2} - 1$ **do**

4 $\quad a_j = (j \,\&\, m_R) + [(j \,\&\, m_C) \ll 1] + [(j \,\&\, m_L) \ll 2]$

5 $\quad b_j = a_j + 2^{n-q_0-1}$

6 $\quad c_j = a_j + 2^{n-q_1-1}$

7 $\quad d_j = a_j + 2^{n-q_0-1} + 2^{n-q_1-1}$

8 $\quad \psi[a_j] = u_{0,0}\phi[a_j] + u_{0,1}\phi[b_j] + u_{0,2}\phi[c_j] + u_{0,3}\phi[d_j]$

9 $\quad \psi[b_j] = u_{1,0}\phi[a_j] + u_{1,1}\phi[b_j] + u_{1,2}\phi[c_j] + u_{1,3}\phi[d_j]$

10 $\quad \psi[c_j] = u_{2,0}\phi[a_j] + u_{2,1}\phi[b_j] + u_{2,2}\phi[c_j] + u_{2,3}\phi[d_j]$

11 $\quad \psi[d_j] = u_{3,0}\phi[a_j] + u_{3,1}\phi[b_j] + u_{3,2}\phi[c_j] + u_{3,3}\phi[d_j]$

end

一种特殊且常用的双量子比特门是 SWAP门

$$U = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{22}$$

在此情况下， 4×4 该操作仅对应于交换 2 个元素。因此，算法3中的第 8 至第 11 行得到

$$\begin{aligned} \psi[a_j] &= \phi[a_j], \\ \psi[b_j] &= \phi[c_j], \\ \psi[c_j] &= \phi[b_j], \\ \psi[d_j] &= \phi[d_j]. \end{aligned}$$

值得注意的是，算法3 允许对非相邻量子比特应用任意双量子比特门，如图 2（e）所示。

2.4 多量子比特门

1 个和 2 个量子比特门的态矢量模拟算法可以直接推广到多量子比特门。对于每增加一个量子比特，我们需要定义 2 个额外的位掩码，以便通过位运算高效计算索引。

然而，对于较小的单量子比特门或双量子比特门的多重控制版本，性能提升的空间最大。这是因为每增加一个控制量子比特，态矢量中需要更新的元素数量就会减半。例如，如图5所示，双重控制单量子比特门只会更新态矢量中四分之一的元素。托福利门（也称为双重控制非门）是最常见的多量子比特门。通过交换其作用的态矢量元素，还可以进一步优化其实现。

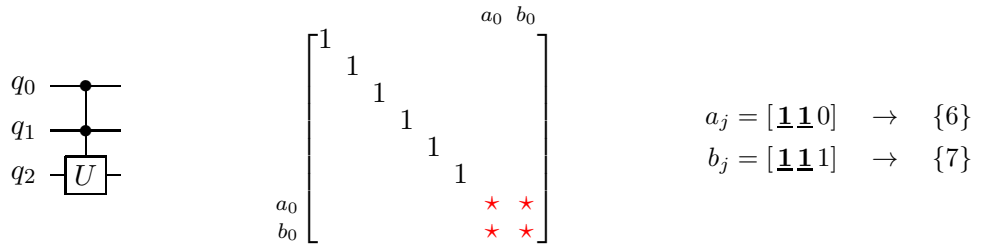


图5：应用双控制单量子比特门的图示 U .

3 实现细节

qclab++是一个面向对象、完全模板化的 C++ 软件包，用于创建和表示量子电路。qclab++可用于快速制作原型并测试量子算法，从而实现算法的快速开发与探索。qclab++没有任何外部依赖，并通过 openQASM 提供输入输出功能，因此与量子硬件兼容。

为了使 qclab++保持轻量化并具有高度可移植性，所有并行化（包括 CPU 和图形处理器）均通过 OpenMP 完成。采用 OpenMP 进行图形处理器卸载的另一个优势是，第 2 节中的算法在 CPU 和图形处理器上的实现完全相同，唯一的区别在于 OpenMP #pragma 语句。这一点如图6所示，其中展示了应用 Pauli-X 门的 CPU 和图形处理器实现。请注意，这两个实现仅在 for 循环上方的#pragma语句上有所不同，其余代码完全相同。

```cpp
template <typename T>
void apply(int nbQubits, int qubit, T* x) {

  // number of iterations
  int jmax = 1 << (nbQubits-1);

  // bit masks
  int mR = (1 << (nbQubits-qubit-1)) - 1;
  int mL = (1 << (nbQubits-1)) - 1 - mR;

  // loop over 2x2 matvecs
  #pragma omp parallel for
  for (int j = 0; j < jmax; j++) {
    int aj = (j & mR) + ((j & mL) << 1);
    int bj = aj + 1 << (nbQubits-qubit-1);
    std::swap(x[aj], x[bj]);
  }

}
```

```cpp
template <typename T>
void apply_device(int nbQubits, int qubit, T* x) {

  // number of iterations
  int jmax = 1 << (nbQubits-1);

  // bit masks
  int mR = (1 << (nbQubits-qubit-1)) - 1;
  int mL = (1 << (nbQubits-1)) - 1 - mR;

  // loop over 2x2 matvecs
  #pragma omp target teams distribute parallel for
  for (int j = 0; j < jmax; j++) {
    int aj = (j & mR) + ((j & mL) << 1);
    int bj = aj + 1 << (nbQubits-qubit-1);
    std::swap(x[aj], x[bj]);
  }

}
```

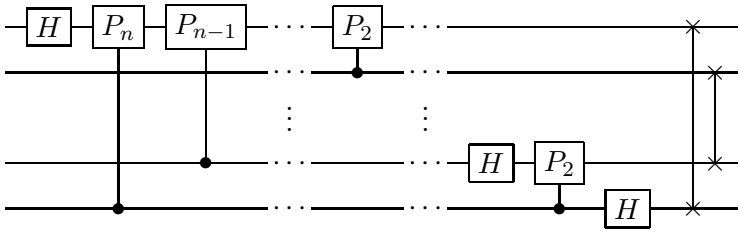
图6：qclab++ 中 Pauli-X门的 CPU（左）与图形处理器（右）实现。请注意，它们唯一的区别在于 for 循环上方的 #pragma 语句。

4 数值实验

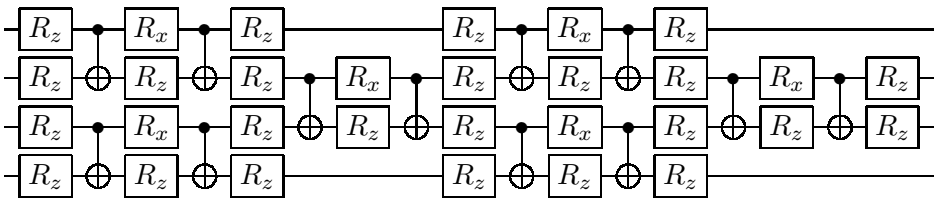
本节所展示的所有实验均在 NERSC Perlmutter 超级计算机上进行。Perlmutter 的每个 GPU 节点均配备 4 个 NVIDIA A100 40GB GPU 和一颗 AMD

EPYC 7763 CPU。这些节点通过 HPE Slingshot-11 互连网络连接。单个 GPU 的内存足以存储一个采用复数双精度的 31 量子比特态矢量，或一个采用复数单精度的 32 量子比特态矢量。我们首先将 qclab++CPU 实现与其 GPU 实现进行比较。接着，我们将 qclab++与其他现有的 GPU 加速状态向量模拟软件包进行基准测试，例如使用 cuQuantum 后端的 cirq-qsim，以及分别使用 cupy 和 cuQuantum 后端的 qibo。

我们选择两类可扩展的基准测试电路来开展实验，见图 7。第一类是量子傅里叶变换电路，由单量子比特哈达玛门、连接电路中每一对量子比特的受控相位门组成，并以一系列交换门收尾。第二类是用于一维最近邻 TFXY 自旋链的特罗特时间演化电路。该电路由多层单量子比特旋转构成，各层之间交替穿插着作用于最近邻量子比特的受控非门层。



(a) 量子傅里叶变换电路



(b) 哈密顿量模拟电路

图 7：电路概 览：用于对各种量子电路模拟器进行基准测试的电路 .

4.1 qclab++：CPU 与图形处理器对比

在第一项实验中，我们比较了 qclab++中的 CPU 和图形处理器实现。量子傅里叶变换电路和哈密顿量模拟电路的计时结果分别列于 图 8 和9中。这两幅图均展示了单精度和双精度下的运行时间及加速比。

我们观察到，对于超过 22 个量子比特的系统，GPU 内核在双对数图上呈现完美的线性扩展。每增加一个量子比特，模拟时间就会翻倍。CPU 模拟的运行时间扩展规律不那么规整，这可能是由非统一内存访问域等内存访问效应导致的。此外，图形处理器模拟的性能显著优于 CPU 实现，加速比超过 $40\times$。请注意，两组基准测试电路的结果在定性上具有可比性。主要差异在于，与量子傅里叶变换电路相比，哈密顿量模拟电路在 19 至 25 个量子比特范围内展现出更高的 GPU 加速比。

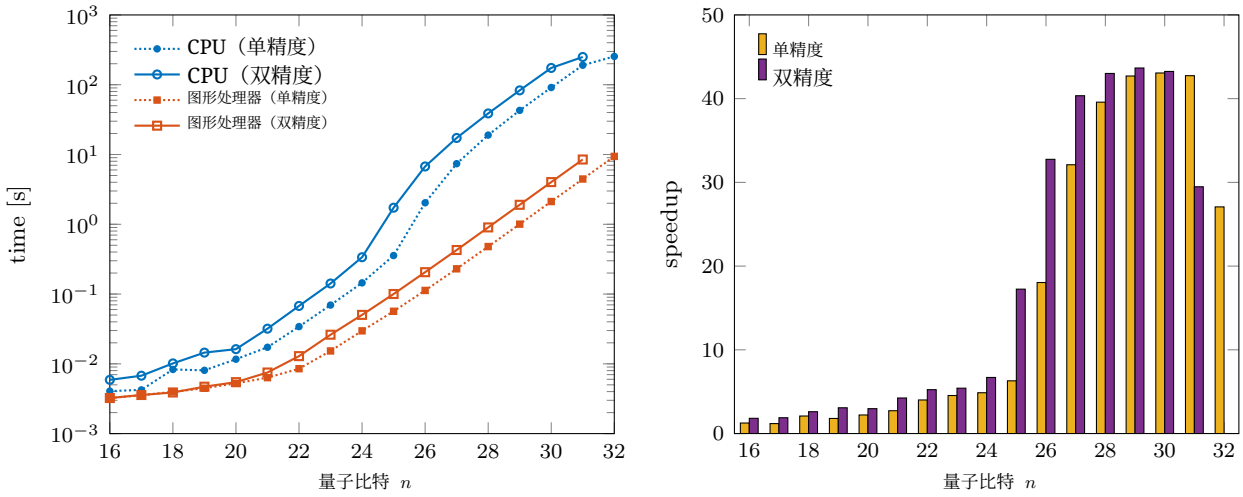


图 8：qclab++：量子傅里叶变换电路的 CPU 与图形处理器对比（Perlmutter）。

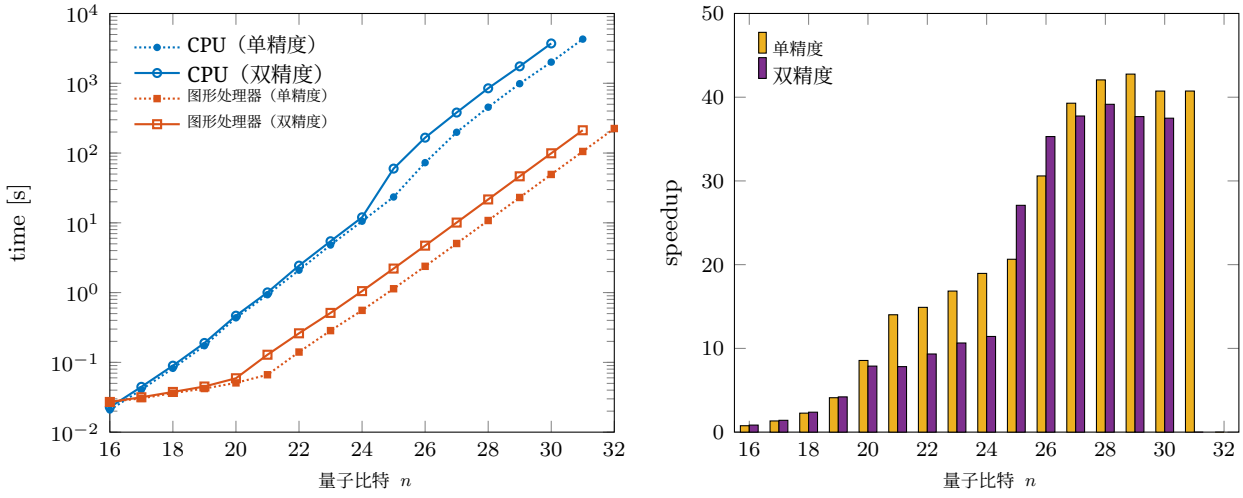


图 9：qclab++：Perlmutter 上哈密顿量模拟电路的 CPU 与图形处理器对比。

4.2 GPU 基准测试

在第二项实验中，我们将 qclab++的 GPU 状态向量模拟器与 cirq-qsim 提供的 cuQuan- tum 后端以及 qibo 进行比较，其中 qibo 使用 cupy 和 cuQuantum 后端。单精度模拟的运行时间汇总于 图 10。最显著的差异出现在量子比特数较少的范围：量子傅里叶变换电路为 16 至 25 个量子比特，哈密顿量模拟电路为 16 至 21 个量子比特。在这一范围内，qclab++相比其他模拟器具有显著优势。这种差异最可能的原因是，qclab++ 运行的是经过编译的C++代码，与其他模拟器使用的 Python 接口相比，其开销更低。量子比特数较大时，Python 带来的开销变得可以忽略，所有模拟器的性能也都相近。我们得出结论： qclab++在具备若干独特优势的同时，也能与其他模拟器相媲美。

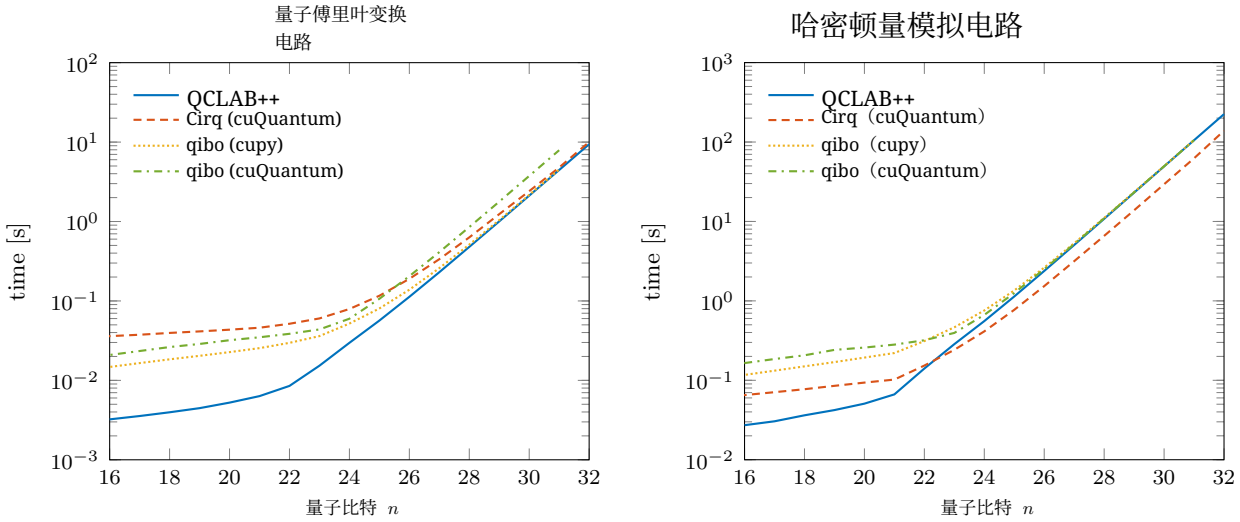


图 10：在 Perlmutter 上进行 GPU 基准测试（单精度）。

致谢

本研究使用了国家能源研究科学计算中心的资源。该中心是美国能源部科学办公室用户设施，由美国能源部科学办公室根据合同编号 DE-AC02-05CH11231 提供支持；本研究使用的 NERSC 资助编号为 ASCR-ERCAP0024463。

参考文献

[1] V. Bergholm 等人。Pennylane：混合量子-经典计算的自动微分，2018年。doi:10.48550/arXiv.1811.04968。[2] D. Camps 和 R. Van Beeumen。Quantumcomputinglab/qclab：Qclab v0.1.2，2021年8月。doi:10.5281/zenodo.5160555。[3] D. Camps、R. Van Beeumen 和 C. Yang。重新审视量子傅里叶变换。《应用数值线性代数》，28(1):e2331，2021年。doi:10.1002/nla.2331。

[4] Cirq 开发者。Cirq，2022年12月。作者完整列表见 GitHub：https://github .
com/quan- tumlib/Cirq/graphs/contributors。doi:10.5281/zenodo.7465577。[5] D.
Coppersmith。用于量子整数分解的近似傅里叶变换，1994年。[6] S. Efthymiou、M.
Lazzarin、A. Pasquale 和 S. Carrazza。通过即时编译进行量子模拟。Quantum，6:814，
2022年9月。doi:10.22331/q-2022-09-22-814。[7]S. Efthymiou、S. Ramos-Calderer、C.
Bravo-Prieto、A. P´erez-Salinas、D. Garc´ıa-Mart´ın、A. Garcia-Saez、J. I. Latorre 和 S.
Carrazza。Qibo：一种通过硬件加速进行量子模拟的框架。《量子科学与技术》，7(1):015018，
2021年12月。doi:10.1088/2058-9565/ac39f5。[8] L. Fang、ahehn nv、hhbayraktar 和 sa
m stanwyck。Nvidia/cuquantum：cuquantum Python v22.11.0.1，2023年1月。
doi:10.5281/zenodo.7523366。[9] T. Jones、A. Brown、I. Bush 和 S. C. Benjamin。QuEST 与
量子计算机的高性能模拟。《科学报告》，9(1)，2019年7月。
doi:10.1038/s41598-019-47174-9。[10]H. Liu、G. H. Low、D. S. Steiger、T. H¨aner、M.
Reiher 和 M. Troyer。量子计算在分子科学领域的前景。《材料理论》，6(1)，2022年3月。
doi:10.1186/s41313-021-00039-z。[11]Qiskit 开发者。Qiskit：量子计算开源框架，2021年。
doi:10.5281/zenodo.2573505。[12] 量子人工智能团队。通过扩展表面码逻辑量子比特来抑制量
子错误。《自然》，614(7949):676–681，2023年2月。doi:10.1038/s41586-022-05434-1。[13]
量子人工智能团队及合作者。qsim，2020年9月。doi:10.5281/zenodo.4023103。[14] C.
Ryan-Anderson 等人。在五量子比特码和颜色码上实现容错纠缠门，2022年。
doi:10.48550/arXiv.2208.01863。[15] R. Van Beeumen 和 D. Camps。
Quantumcomputinglab/qclabpp：Qclab++ v0.1.2，2021年8月。
doi:10.5281/zenodo.5160682。[16] H. J. Williams。多用途中性原子进军量子电路。《自然》，
604(7906):429–430，2022年4月。doi:10.1038/d41586-022-01029-y。