使用 Python控制流进行囚禁离子的实时混合量子-经典计算

Tobias Schmale*[1]，Bence Temesi*[2]，Niko Trittschanke*，Nicolas Pulido-Mateo†‡，Ilya Elenskiy ¶，Ludwig Krinner†‡，Timko Dubielzig†，Christian Ospelkaus†‡，Hendrik Weimer*§，Daniel Borcherding*[3]

*汉诺威莱布尼茨大学理论物理研究所，德国汉诺威 Appelstraße 2，邮编 30167
†汉诺威莱布尼茨大学量子光学研究所，德国汉诺威 Welfengarten 1，邮编 30167
‡德国联邦物理技术研究院，德国布伦瑞克 Bundesallee 100，邮编 38116§柏林工业大学理论物理研究所，德国柏林 Hardenbergstraße 36，邮编 10623
¶布伦瑞克工业大学电气测量技术与电气工程基础研究所，Hans-Sommer Strasse66,38106 布伦瑞克，德国

arXiv:2303.01282v1 [quant-ph] 2 Mar 2023

摘要——**近年来，结合量子计算与经典计算的混合算法数量持续增加。这两种计算方法能够相互提升彼此的性能，从而有望实现更先进、优于各自纯粹形式的算法。为了适应这类新型代码，需要构建合适的环境，使量子与经典硬件能够协同工作。**

对于许多这类混合流程而言，量子计算机的相干时间构成了一个自然的时间限制，因此尽可能减少经典开销至关重要。然而，对于离子阱量子计算机来说，这一因素的限制要比超导技术小得多，因为相关时间尺度为秒，而非微秒。事实上，较长的相干时间使我们能够开发一种方案，使用 Python 这类解释型编程语言实时控制量子计算。尤其是，与超导量子比特不同，无需预先编译所有指令。这使混合算法的实现保持简洁，也让用户能够受益于现有丰富的 Python 库。

为了证明这种解释型量子—经典计算（IQCC）方法切实可行，我们提供了真实案例，并在贴近实际的基准测试中对其进行评估。

索引词—**量子计算、混合量子—经典计算、囚禁离子**

I. 引言

近年来，量子计算作为一个领域稳步发展，因为它有望比经典计算机快几个数量级地解决问题。理论上已证明某些任务 [1, 2, 3, 4], ，甚至已在量子硬件 [5, 6] 上演示，这些任务比经典方案更快；然而，在实际问题中实现量子优势仍有待实现。

尽管要实现量子优势仍需在硬件层面进一步创新，但也可以通过结合经典计算来提升量子计算能力。这一点在当前含噪中等规模量子（NISQ）时代尤为重要，因为现有的量子硬件都受到含噪量子门和量子比特数量的限制。各种算法都展现了这种量子-经典混合计算模型的优势：迭代相位估计（IPE）通过重复使用一个辅助量子比特并结合中间的经典计算，比量子相位估计算法（QPE）所需的量子比特少得多 [7]。量子纠错需要使用经典查找表，根据综合征测量结果确定纠错协议。变分量子算法（VQA）等其他算法则通过经典方式求解优化问题，以减少量子电路中的纠缠门数量 [8]。关于量子-经典混合算法的概述，请参见 [9]。

然而，在许多量子硬件方案中，实现此类混合算法在技术上极具挑战性，因为物理量子比特的相干时间对量子设备与经典设备之间的通信时间构成了天然限制。对于超导量子比特或自旋量子比特等系统，这一时间限制约为微秒[10] ——这意味着任何经典逻辑都必须编译后在硅上以纳秒级延迟执行，才能访问量子设备 [11]。这给量子软件开发人员带来了负担，他们必须以兼容这些限制的方式编写算法。

在许多情况下，人们提出了全新的量子编程语言来解决这一问题。IBM 最近为其基于 Python 的软件开发工具包 QISKIT 添加了简单的经典控制流，以支持基于 OpenQASM 3 的量子计算过程中进行简单的经典计算 0 [12]。作为更复杂的替代方案，人们开发了诸如 Q# 这类使用中间表示 QIR 的编译型语言 [13] ，以及基于 C/C ++ [14] 的 QCOR 语言。

另一方面，囚禁离子具有远超前述方案的相干时间，甚至可达数分钟 [15, 16]。因此，迄今提出的一些量子-经典编程解决方案显得不成比例，因为它们解决的问题

[1]tobias.schmale@itp.uni-汉诺威.de[2]
bence.temesi@itp.uni-汉诺威.de[3]
daniel.borcherding@itp.uni-汉诺威.de

对于囚禁离子量子计算机而言，这个问题并不存在。本文利用这种较长的相干时间，提出一种基于标准 Python控制流 的解释型量子-经典计算（IQCC）新用户界面。这使量子软件开发者能够受益于 Python语言的简洁性，以及丰富的可用库和软件框架。此外，尽管存在许多专用量子语言，Python 仍然是开发静态纯量子算法最受欢迎的语言之一。因此，我们的方案可以利用所有这些现有成果，并将其扩展到动态实时量子-经典软件。

我们的 Python 控制流框架能够实现当今各类重要的量子-经典算法，从快速量子纠错到变分量子算法。我们通过评估并测试多个真实应用案例，并将其与即将问世的囚禁离子量子计算机（如 QVLS-Q1 芯片 [17]）预期的执行速度进行比较，证明了我们的软件配置切实可行。

II. 概念

A. 用户 API

为结合量子计算与经典计算，我们使用现有的 Python 软件包来生成量子电路对象，并将其与标准 Python 控制流结合起来。这样，用户既能够生成量子电路对象，又能充分利用 Python 的灵活性以及外部软件包，通过经典计算增强量子算法。下文以软件开发工具包 QISKIT [18]，为例，说明如何生成量子电路对象。不过，任何能够将量子电路导出为 OpenQASM 2.0 语言的其他 Python 量子电路软件包也同样适用。

在我们的框架中，量子-经典程序是一个 Python 脚本，它通过调用函数 qpu.run(quantum_circuit)，实时触发量子设备上量子电路（QISKIT 对象 quantum circuit）的执行。混合算法的结果可以通过在结果对象上调用函数 upload() 上传到数据库。图 1 展示了量子-经典程序的主要结构，其中包含两次量子处理器调用（第 5 行和第 9 行），以及用于经典计算的占位符；这些经典计算也会生成待执行的量子电路对象。第 11 行将结果对象上传。

B. 示例

使用 [19]，中的术语，我们区分实时混合计算和近实时混合计算：前者在量子比特的相干时间内进行经典计算，后者则在每次运行前重置量子计算机。这里，我们展示一些示例，说明我们的用户界面适用于这两种混合计算。

一个简单的实时量子—经典计算示例是图 2 [20]，所示的三量子比特位翻转码。

```
1  来自 qvls 导入qpu, upload
2
3  # 插入经典计算
4
5  结果 = 量子处理器.运行(量子_量子电路_1)
6
7  # 插入更多经典计算
8
9  结果 = 量子处理器.运行(量子_量子电路_2)
10
11 上传(结果)
12
```

图 1：混合算法示例：该算法会触发量子设备实时执行量子电路（第 5 行和第 9 行）。占位符表示可添加经典计算的位置，这些计算最终会生成量子电路对象。最后，结果对象会上传到数据库。

在此示例中，先制备状态 $|000\rangle$，然后对第一个量子比特施加 Pauli X错误（第 6 行）。随后，使用两个辅助量子比特执行奇偶校验（或“稳定子”）测量 Z_0Z_1 和 Z_1Z_2（第 8–14 行）。接着，根据稳定子测量结果，通过经典条件纠正错误（第 16–23 行），使最终态应为 $|000\rangle$，且混合程序应返回相应的字典 {“000”： 1}（第 29 行）。

```
1  来自 qvls 导入qpu, upload
2  来自 qiskit 导入QuantumCircuit
3
4  qc = QuantumCircuit(5, 3)
5
6  qc.x(0)
7
8  量子计算。受控非门(0, 3)
9  量子计算。受控非门(1, 3)
10 qc.测量(3,0)
11
12 量子电路。控制非门(1,4)
13 量子计算。控制非门 (2, 4)
14 量子电路。测量(4,1)
15
16 结果 = 量子处理器.run(qc)
17
18 if '001' 在 结果中:
19     qc.x(0)
20 否则如果'010' 在 结果中:
21     量子计算。x(2)
22 elif '011' 在 结果:
23     量子计算。
24     x(1)
25 量子电路.测量_全部()
26
27 结果 = 量子处理器.运行(qc)
28
29 上传(结果)
```

图 2：使用我们的 Python控制流 实现的三量子比特位翻转码。

我们的下一个示例展示了用户如何在硬件上运行近实时混合作业，例如变分量子算法。这些算法通常可以概括为以下几个高层步骤：

```
从 qvls 导入qpu, upload
从 SciPy优化模块 导入minimize

# 创建变分电路
变分电路_量子电路 = 创建_circuit()

# 符号编译
量子电路 = qpu.compile(ansatz_circuit)

定义 成本_函数(参数):
    # 测量并计算成本函数
    结果 = 后端.运行(量子电路, 参数)
    返回 计算_成本(结果)

# 使用经典优化器寻找解
结果 = 最小化(成本_函数, 初始_参数)
上传(results)
```

图 3：近实时混合作业示例，此处为一个变分量子算法，它通过量子计算机反复测量成本函数，并在经典计算中优化参数。

1) 为当前问题创建参数化试探电路。 2) 选择一个成本函数，并在每次评估时使用量子计算机对其进行测量。 3) 使用经典计算机通过某种方法（基于梯度或其他方法）更新参数，从而优化测得的成本函数。

图 3 中的代码片段概述了如何执行此类变分量子算法。除了前面提到的步骤，我们还可以注意到以下几点：我们使用 qpu.compile() 对参数化量子电路进行转译和编译，且只需执行一次，因此在优化过程中只需赋入更新后的参数值，从而避免冗余编译。此外，在第 16 行中，我们可以看到，如何轻松使用从自己选择的 Python 软件包中获取的任意优化器，例如 SciPy [21]。

对于实时和近实时混合作业，我们将在第 III 节中给出具体且具有物理意义的示例，并对其进行基准测试。

C. 软件与服务器架构

本节将讨论实现第 II-A 节中介绍的用户 API 所需的服务器架构，该 API 用于实时和近实时量子-经典计算。对于实时情况，混合算法的总运行时间为 t_{total}，其主要要求是算法必须在量子硬件中离子的相干时间 $t_{\text{coherence}}$ 内完成，即需要满足 $t_{\text{total}} < t_{\text{coherence}}$。对于任何混合算法，我们都可以通过降低系统设置的开销来缩短总运行时间。具体而言，我们希望确保混合算法的总运行时间主要由量子部分 t_{quantum} 决定，而不是由经典计算和其他经典部分 $t_{\text{classical}}$ 决定。这样可以确保更宝贵的量子资源得到高效利用。

因此，我们使用经典开销来量化针对特定算法的系统效率，其定义为$t_{\text{classical}}/t_{\text{total}}$，其中 $t_{\text{total}} = t_{\text{classical}} + t_{\text{quantum}}$。

当量子计算机可供未知用户使用时，还会出现另一项要求：在这种情况下，需要确保量子—经典作业中的不受信任的 Python 代码能够安全执行。因此，我们为服务器架构确立了以下目标：

1）性能：尽量降低混合算法的经典开销$t_{\text{classical}}/t_{\text{total}}$。
2）安全性：确保不受信任的 Python 代码安全执行。

为满足这些要求，我们选择了由量子处理器部分（现场可编程门阵列和囚禁离子）、一台中介计算机（“实验控制系统”，可与量子处理器通信）以及另一台称为中央处理器的计算机组成的服务器架构。中央处理器在沙箱环境（例如 Docker 容器）中运行不受信任的 Python 代码。在这种架构中，中介负责协调量子处理器与中央处理器之间的通信：一方面，它可以从中央处理器接收汇编指令（在我们的例子中为 TIASM 指令 [17]），将其转换为现场可编程门阵列指令，并转发至量子处理器的现场可编程门阵列部分；另一方面，它可以从量子处理器接收测量结果，并转发给中央处理器。所有这些组件都位于同一局域网中。原则上，中介和中央处理器的软件可以在同一台物理设备上运行，但我们选择进一步保护量子处理器，避免其受到不受信任的 Python 代码执行的影响。

这是将混合计算的服务器架构划分为各部分以满足安全约束的最简单方法。下文将说明，这种划分也足以实现性能目标。图 4 展示了近实时和实时算法中网络内所有参与方之间的通信，以及预期耗时。

III. 评估

本节旨在展示实时和近实时 IQCC 在囚禁离子量子计算机上实现的可行性，也就是说，证明符合第 II-C 节列出的所有标准。这里最重要的是证明，混合算法的执行速度受量子处理器的速度限制，而非受经典开销限制。作为囚禁离子量子计算机的具体示例，我们选择了即将推出的 QVLS-Q1 芯片及其编译器，并在此基准测试中使用 [17]。

A. 用于基准测试的示例

为此，我们运行了三种具有现实应用价值的混合量子算法：

作为第一个示例，我们考虑迭代相位估计（IPE）。该算法可以估算给定本征态对应的幺正操作的特征值；它也很有意义，因为它是其他更大型算法（例如 Shor算法 [1]）中的一个子程序。对于我们的设置而言，这是一个相关的测试案例，因为每次量子处理器调用所执行的量子线路都非常短，在我们的示例中只包含 $\sim$ 10 个原生门，这些门执行完后，中央处理器就需要作出另一项经典决策。因此，这是对我们设置最不利的情形，因为量子处理单元时间较短，必然意味着网络时间与量子处理单元时间之比更大，进而可能导致更大的

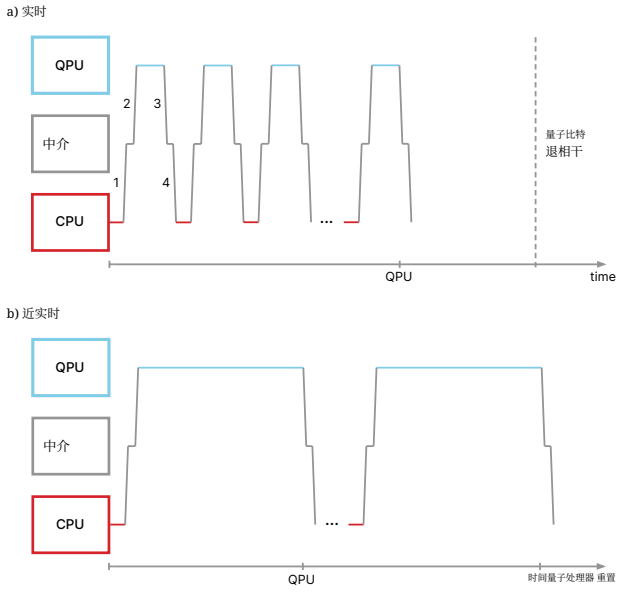


图 4：混合算法执行期间中央处理器、中介和量子处理器（现场可编程门阵列和囚禁离子）所耗时间的对比（横线）。数字标示不同的通信步骤：1）将 TIASM 从中央处理器发送至中介；2）将脉冲信息从中介发送至量子处理器；3）将结果信息从量子处理器发送至中介；4）将测量结果从中介发送至中央处理器。请注意，实时情形下，混合算法会在量子比特退相干之前完成；而在近实时情形下，量子计算机会在每个周期后重置。

经典开销。有关 IPE 的更多详细信息，以及使用超导量子比特进行动态实现的内容，请参见附录 B [11] 。

接下来，我们讨论量子纠错（QEC）。量子纠错可能是实现量子优势的关键。大多数实现（例如本文采用的 Steane码）都依靠奇偶校验测量，在量子计算过程中检测错误症状，并使用经典查找表确定纠正检测到的错误所需的量子操作。由于逻辑量子比特嵌入到许多物理量子比特中，每次量子处理器调用的工作量显著增加；在我们的案例中，两次经典决策之间约需执行 100–200 个量子门。更多详情请参阅附录 C，并参阅 [22] ，了解首次动态实验实现。

最后，为了评估变分量子算法在我们的设置中的运行情况，我们使用变分量子本征求解器（VQE）[23]，为 7 个量子比特制备一维横向伊辛模型的基态。这种混合算法对于在近期量子计算机上进行量子模拟可能很有价值，因为它所需的含噪量子资源较少。VQE 通过改变变分电路中的参数来寻找哈密顿量的基态，使系统的能量（成本函数）最小化。参数在经典计算机上更新，而能量则通过量子硬件测量哈密顿量的期望值来评估。有关此处所选 VQE 实现的更多详情，请参阅附录 D。

操作	假定持续时间
单量子比特门	0.1 毫秒 [25]
双量子比特门	1 毫秒 [25]
离子冷却	1 毫秒 [16]
离子搬运	0.1 毫秒 [26]
测量	0.1 毫秒 [16]

表 I：各种量子操作所假定的持续时间。假定每次双量子比特门之前都进行一次离子冷却操作，因此双量子比特门的有效时间为 2 ms。不过，所有这些数值都不应被视为精确值，而只能视为数量级，因为确切数值将取决于具体的硬件实现和校准。

所有这些示例均可在 [24], 下下载，作为本地环境中的最简可运行示例。请注意，用户 API 的具体细节可能会发生变化。

B. 基准测试策略

在进行基准测试时，我们将算法运行时间的以下组成部分纳入考量：

1）*QPU* 执行时间： 虽然我们目前还无法使用正常运行的 QPU，但可以将即将推出的 QVLS-Q1 芯片 [17]所对应的已编译量子线路指令，与先前研究中给出的门、测量和离子移动持续时间相结合，从而估算在 QPU 上执行量子线路的运行时间。表 I 列出了囚禁离子芯片各种指令的假定持续时间。这些持续时间被视为既定条件，优化这些时间不在我们软件栈的控制范围内。基于此，我们计算每项任务的平均量子处理单元时间（图 5 中的“QPU - Computation”）。

2）经典计算时间：这部分分为以下几个组成部分：编译，即将原生门映射到囚禁离子芯片上、执行离子路由，并生成量子处理器 [17] 的汇编指令。其次，是在经典计算机上花费的其余时间，即图 5 中的“中央处理器 - 计算”。这包括在两次量子处理器调用之间执行用户 Python 代码，也就是混合量子算法的经典部分。

3）实验控制系统开销：这仅包括解析汇编指令所需的时间，因为这是我们目前唯一能够获取的指标。具体而言，不包括信号生成方面的任何开销。不过，我们预计，与其他组成部分相比，这些开销可以忽略不计。

4）网络贡献：如“服务器架构”II-C 节所述，我们将用户提交的 Python 代码与实验控制系统隔离开来。这会带来若干网络延迟因素，我们将其分解如下：

- 网络延迟：这是在通信双方运行于同一主机时，通过 TCP 连接传输量子处理器指令所产生的纯延迟（图 4 中第 1 至 4 项的总和）。
- 局域网开销：这是将通信双方中的一方移至同一网络上的另一台独立主机时，所增加的总网络延迟。

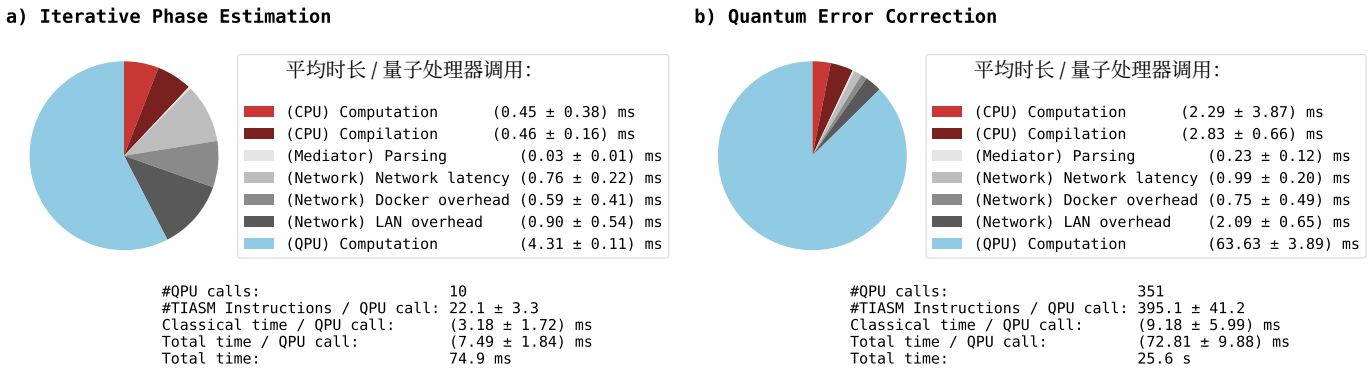


图 5：图中展示了量子纠错和迭代相位估计示例的运行时间构成。误差线表示整个作业的标准差，以说明现实世界算法所进行的量子处理器调用在同一算法中并非都相同。各项构成的详细说明见第 III-B 节。

我们承认，这个数值在很大程度上取决于网络配置。为了尽可能模拟预期的未来实验配置，我们仅通过一个交换机，将两台机器连接到千兆比特每秒以太网上。

•沙箱开销：这是将用户提交的 Python 代码隔离在 Docker 容器中时增加的总网络延迟。

C. 基准测试结果

图 5 展示了迭代相位估计和量子纠错示例中所有这些组件的贡献。在这两个示例中，量子处理器对总运行时间的贡献最大。

对于迭代相位估计，我们实现了 $t_{\text{classical}}/t_{\text{total}} \approx {}^{3.18\text{毫秒}}_{7.49\text{毫秒}} \approx 42.5\,\%$，如果忽略局域网的影响（使用单台机器时便是如此），则可降至 $\approx 34.6\,\%$。尽管此处的经典开销相当显著，但并不妨碍该实现的可行性。迭代相位估计算法仍然足够短（在我们的案例中为 74。9 毫秒），远低于数秒的相干时间 [27]。因此，经典开销导致量子处理单元时间“浪费”的绝对时长非常短，IQCC 实现也因此具有实用性。

对于量子纠错，$t_{\text{classical}}/t_{\text{total}} \approx {}^{9.18\text{毫秒}}_{72.81\text{毫秒}} \approx 12.7\,\%$，而在单主机场景下为 $\approx 10.1\,\%$。因此，算法的总运行时间受量子处理器限制；即使完全消除经典开销，运行时间最多也只能缩短 10。1%。

对于变分量子本征求解器，绘制饼图并不像之前的示例那样有意义，因为量子处理器时间本身就占据了图表的绝大部分。原因在于，优化过程中需要通过多次测量采样来求得系统能量；也就是说，每次量子处理器调用都必须运行量子电路数十次到数千次（采样次数）。在之前的实时算法中，每次量子处理器调用只需运行一次量子线路，因此量子与经典时间尺度相当。但在这里，量子处理器时间比经典时间长几个数量级，$t_{\text{classical}}/t_{\text{total}} < 1\%$。具体而言：测得的经典开销为 $10 - 100\text{ms}$（包括转译时间和

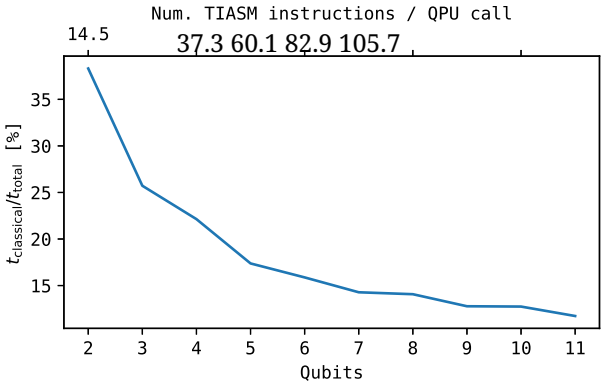


图6：图中展示了 IPE 示例中的经典开销随量子比特数的变化情况。在这里，量子比特越多，每次量子处理器调用所需的硬件指令就越多，因此时序图中的所有部分耗时都会增加。然而，经典开销与总时间之比仍会下降，也就是说，算法越复杂，IQCC 就越可行。

局域网开销），累计 QPU 时间为数秒，甚至数十秒。请注意，可以采用量子错误缓解技术来提高结果的准确性。在这种情况下，经典开销可能会比我们基准测试示例中的更显著。随着量子处理器操作变得更加复杂，经典开销的增长速度低于量子计算机中的耗时。我们通过对作用于不同数量量子比特的相关幺正算子的迭代相位估计示例进行基准测试，展示了这一点。这会增加所有经典环节的耗时，也会延长量子处理器的运行时间。图6展示了二者的比值，证实了预期趋势，并表明该设置能够随量子比特数量增加而扩展。

因此，根据我们的基准测试，我们认为，使用 Python控制流在囚禁离子量子计算机上实现时，这些类型的算法都具有可行性。

IV. 结论与展望

我们介绍了用于实现实时和近实时混合量子

算法的用户 API 和服务器架构，该架构利用囚禁离子的长相干时间，实现基于 Python 的解释型控制流。通过相关的混合示例，我们展示了由此产生的经典开销很小：对于变分量子算法，开销不到 1 %；对于量子纠错，开销约为 10%。此外，我们还证明了需要量子与经典硬件更加紧密协同的任务（例如迭代相位估计）同样可行。最后，在要求最高的示例中，随着混合算法所用量子比特数量增加，经典开销反而会降低。因此，我们证明，即使离子阱硬件的量子体积不断提升，解释型混合工作流仍然是可行的选择。

在未来的工作中，我们计划研究 OpenQASM3 [12] 和 IBM 动态电路 [11] 中的理念如何帮助进一步降低剩余开销。我们设想，未来用户可以在编译型控制流和解释型控制流之间进行选择，并能够在同一个量子算法中流畅地在二者之间切换。

致谢

本工作由 Quantum Valley Lower SaxonyQ1 项目（QVLS-Q1）通过大众基金会和下萨克森州科学与文化部资助，并获得德国卓越战略——EXC-2123 QuantumFrontiers——390837967 的资助。

参考文献

[1] Peter W Shor。《量子计算算法：离散对数与因数分解》。载于：第*35*届计算机科学基础年会论文集。IEEE，1994年，第124–134页。[2] Lov K Grover。《用于数据库搜索的快速量子力学算法》。载于：第二十八届*ACM*计算理论研讨会论文集。1996年，第212–219页。[3] Sergey Bravyi、David Gosset和Robert K¨onig。《浅层电路的量子优势》。载于：*Science*362.6412（2018年），第308–311页。[4] Patrick Rebentrost、Masoud Mohseni和Seth Lloyd。《用于大数据分类的量子支持向量机》。载于：*Physical Review Letters*113.13（2014年），第130503页。[5] Frank Arute等。《利用可编程超导处理器实现量子霸权》。载于：*Nature*574.7779（2019年），第505–510页。[6] Diego Rist`e等。《机器学习中量子优势的演示》。载于：*npj QuantumInformation*3.1（2017年），第1–5页。[7] Robert B. Griffiths和Chi-Sheng Niu。《用于量子计算的半经典傅里叶变换》。载于：*Phys. Rev. Lett.*76（1996年4月17日），第3228–3231页。[8] Marco Cerezo等。《变分量子算法》。载于：*Nature Reviews Physics*3.9（2021年），第625–644页。

[9] Thomas Lubinski等。《通过实时执行推进混合量子–经典计算》。载于：*Frontiers in Physics*（2022年），第671页。[10] Youwei Zhao等。《利用超导量子比特实现纠错表面码》。载于：*Phys.Rev. Lett.*129（2022年7月3日），第030501页。[11] A. D. C´orcoles等。《在使用超导量子比特的量子算法中利用动态量子电路》。载于：*Physical Review Letters*127.10（2021年8月31日），第100501页。[12] Andrew Cross等。《OpenQASM 3：更广泛、更深入的量子汇编语言》。载于：*ACM*量子计算汇刊3.3（2022年），第1–50页。[13]Alan Geller。量子中间表示（*QIR*）简介。devblogs.microsoft.com/qsharp/introducing-quantum-intermediate-representation-qir/。2021年。[14] Tiffany M. Mintz等。《QCOR：异构量子-经典计算模型的语言扩展规范》。载于：*J. Emerg. Technol. Comput.Syst.*16.2（2020年3月）。[15] Pengfei Wang等。《估计相干时间超过一小时的单离子量子比特》。载于：*Nature Communications*12.1（2021年1月11日），第233页。[16] J. M. Pino等。《囚禁离子量子CCD计算机架构的演示》。载于：*Nature*592.7853（2021年4月8日），第209–213页。[17] Tobias Schmale等。《囚禁离子量子计算机的后端编译器阶段》。载于：*2022 IEEE*量子软件国际会议（*QSW*）。2022 IEEE量子软件国际会议（QSW）。西班牙巴塞罗那：IEEE，2022年7月，第32–37页。[18] Gadi Aleksandrowicz等。《Qiskit：用于量子计算的开源框架》。载于：访问日期：*2022*年*3*月16（2019年）。[19] Gambetta J. Faro I Johnson B。重新思考量子系统，以实现更快速、更高效的计算。research.ibm.com/blog/near-real-time-quantum-compute。2022年。[20] Michael A. Nielsen和Isaac L. Chuang。《量子计算与量子信息：十周年纪念版》。剑桥大学出版社，2010年。[21] Pauli Virtanen等。《SciPy 1.0：Python科学计算的基础算法》。载于：*NatureMethods*17（2020年），第261–272页。[22] C. Ryan-Anderson等。《实时容错量子纠错的实现》。载于：*PhysicalReviewX* 11.4（2021年12月23日），第041058页。[23] Alberto Peruzzo等。《光子量子处理器上的变分本征值求解器》。载于：*Nature Communications*5.1（2014年），第1–7页。[24] Daniel Borcherding等。混合算法。gitlab.uni-hannover.de/QVLS/HybridAlgorithms。2023年。[25] G. Zarantonello等。《稳健且资源高效的微波近场纠缠 $^{9}Be^{+}$ 门》。载于：*Phys. Rev. Lett.* 123（2019年12月26日），p. 260503.

[26] WilliamCodyBurton等。在射频保罗阱中将多物种离子晶体输运通过一个结。2022年。

[27] 亨宁·哈恩。“在可扩展表面电极离子阱中，使用 9Be+ 离子的双量子比特微波量子逻辑门”。发表于：(2019)。与 Technische Informationsbibliothek (TIB) 和 Technische Informationsbibliothek (TIB) 合作。出版方：汉诺威：莱布尼茨汉诺威大学机构知识库。[28] 塞永·西瓦拉贾等。“t|ket〉：面向含噪声中等规模量子设备的可重定向编译器”。发表于：《量子科学与技术》6.1 (2020)，第 014003 页。[29] 亚当·佩茨尼克和本·W·赖哈特。“23量子比特戈莱码的容错辅助量子比特制备与噪声阈值下界”。发表于：《量子信息与计算》12.11–12 (2012年11月)，第 1034–1080 页。[30] 克里斯托弗·钱伯兰和安德鲁·W·克罗斯。“使用标志量子比特进行容错魔态制备”。发表于：《*Quantum*》3 (2019年5月)，第 143 页。[31] James C Spall。“用于高效优化的同时扰动方法概述”。发表于：《约翰·霍普金斯大学应用物理实验室技术文摘》19.4 (1998)，第 482–492 页。[32] 戴夫·韦克、Matthew B Hastings 和马蒂亚斯·特罗耶。“实用量子变分算法的进展”。发表于：《物理评论*A*》92.4 (2015)，第 042303 页。

附录

A. 电路转译的影响

这里进行基准测试的所有示例都假设量子线路已经使用原生门。这并不是问题，因为总可以在执行前转译量子线路，而且 QISKIT 或 TKET 等现代软件框架甚至支持以符号方式进行转译。 [28]

不过，这样做需要额外进行一步预先电路转译。我们通过提供一个负责处理此步骤的装饰器，让操作变得简单，例如：

```
@预先转译
定义量子电路_部分(alpha, beta)
    qc = QuantumCircuit(2)
    qc.rzz(alpha, 0, 1)
    量子计算。rx(beta,  0)

量子电路 = 量子电路_部分(alpha=1,  beta=2)
qpu.运行(量子电路)
```

此处，函数中 circuit 部分定义的量子电路会预先完成电路转译，后续对该函数的调用都能非常高效地返回由原生门组成的量子电路，而无需再次进行电路转译。

如果不使用这一方法，电路转译就必须在混合算法的相干循环中进行；即使使用 TKET 等快速的、基于 C++ 的软件包，也可能耗时数十毫秒。对于迭代相位估计示例，这会额外增加 $\sim$ 3 毫秒的开销；对于量子纠错示例，则会增加 ~ 15 毫秒。

B. 迭代相位估计算法

在迭代相位估计算法中，根据 $U|\psi\rangle = \mathrm{e}^{\mathrm{i}\phi}|\psi\rangle$，通过迭代计算幺正算子 U 和本征态 $|\psi\rangle$ 的相位 ϕ 的二进制展开的不同部分，来估计该相位。对于第三节中的示例，我们估计了相位门的相位ϕ

$$P(\phi) = \begin{pmatrix} 1 & 0 \\ 0 & \mathrm{e}^{\mathrm{i}\phi} \end{pmatrix} \tag{1}$$

对于本征态 $|1\rangle$，迭代相位估计算法中的门主要来自受控 U 门的连续执行

$$CU = I \otimes |0\rangle\langle 0| + U \otimes |1\rangle\langle 1| . \tag{2}$$

对于受控相位门 $CP(\phi)$，可以使用以下恒等式来减少每次量子处理器调用所需的门数量：

$$CP(\phi)^n = CP(n\phi). \tag{3}$$

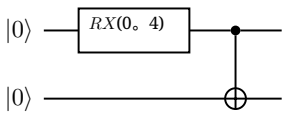
这样，每次量子处理器调用最终只需使用 ~ 10 个原生门。完整实现请参见 [24]。

*C. Steane*码

Steane码是一种量子纠错码，它需要七个物理量子比特来表示一个逻辑量子比特，并且能够纠正任意单量子比特错误。在第三节的示例中，我们使用了 [29] 的编码电路，以及 [30] 的 T 门实现，并稍作调整，将 T 定义为：

$$T = \begin{pmatrix} 1 & 0 \\ 0 & \mathrm{e}^{\mathrm{i}\pi/4} \end{pmatrix} . \tag{4}$$

为了展示经典计算的需求有多频繁，我们实现了一个混合程序，用于执行以下逻辑量子电路



在两个逻辑量子比特上。由于 Steane码只能执行 Clifford + T 门，我们使用递归次数为二的 Solovay-Kitaev算法，将示例量子电路转译为 137 个逻辑 Clifford + T 门。此外，按照 Steane码容错实现的要求 [20]，每次执行逻辑门后，我们都会测量两个逻辑量子比特上的全部六个稳定子。最终，每次量子处理器调用约需执行 100–200 个原生门，以展示 Steane码带来的经典开销。有关实现的所有细节，请参阅 [24] 。

*D. VQE*算法

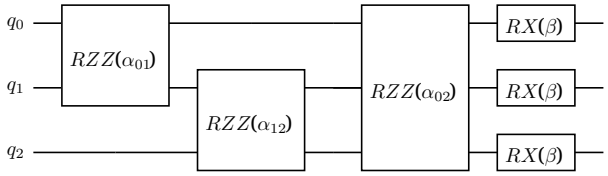
这里，我们会详细介绍所使用的 VQE，并进一步说明变分量子算法的不同要素如何影响基准测试结果。我们使用 VQE 制备了以下一维横向伊辛哈密顿量的基态：

$$H = -\sum_{\langle i,j \rangle} \sigma_i^z \sigma_j^z - h \sum_i \sigma_i^x, \tag{5}$$

其中，$\{\sigma^\mu\}$ 为泡利矩阵，h 为耦合强度，并且我们只考虑最近邻相互作用$\langle \cdot, \cdot \rangle$。由于哈密顿量只有两组不对易项，因此每次评估能量时，我们只需测量量子电路两次。但很多时候，我们需要处理更复杂的哈密顿量，或需要进行更多次测量的成本函数，这会使QPU时间进一步增加，甚至超过我们在结果中看到的情况。

选择经典优化器也会直接影响量子处理器调用次数。在模拟中，我们使用了基于梯度的同时扰动随机近似（SPSA）优化器 [31]。它的一大特点是，只需对成本函数评估两次，就能近似计算梯度，而且评估次数与量子电路中的参数数量无关。因此，与其他优化器相比，它是最节省量子处理器调用的选择。

最后，变分算法能否收敛到解取决于变分电路的设计。在我们的例子中，哈密顿量变分试探（HVA） [32]通过对哈密顿量（1）中的各项进行参数化来制备变分态，即一个变分层由 $RZZ(\alpha)$ 和 $RX(\beta)$ 门组成，其中α, $\beta \in [0, 2\pi)$。对于三个量子比特，HVA 层如下所示：



在 HVA 中，我们仅为彼此不对易的门设置不同的层参数，例如量子电路中的 RZZ 门。理想情况下，我们希望将参数数量保持在较低水平，以避免过多的成本函数评估，但同时参数数量又要足够多，以确保解包含在参数空间中。然而，根据问题中的物理性质、纠缠或系统规模，变分层数，即参数数量，可能会显著增加。所有这些都可以进一步抑制经典开销，相较于量子处理单元时间而言更是如此。