InQuIR：互连量子计算机的中间表示

Shin Nishio * 和 Ryo Wakizaka †
*信息学系，综合研究大学院大学（高级研究大学院大学）跨学科科学学院，日本东京
†京都大学信息学研究生院，日本京都

arXiv:2302.00267v1 [quant-ph] 1 Feb 2023

摘要——各种物理限制约束了单个量子处理器中可实现的量子比特数量，因此有必要通过量子互连连接多个量子处理器。虽然已有多个面向互连量子计算机的编译器实现，但尚无适合作为其编译目标的表示形式。缺少这种表示形式会妨碍已编译程序的重用，也使得难以对分布式量子程序的复杂行为进行形式化推理。我们提出 InQuIR，这是一种能够表达分布式量子系统中的通信和计算的中间表示。InQuIR 具有形式语义，使我们能够精确描述分布式量子程序的行为。我们给出了用 InQuIR 编写的示例，以说明分布式程序中出现的问题，例如死锁。我们提出了一条使用类型系统进行静态验证、以应对此类问题的路线图。我们还提供了 InQuIR 软件工具，并评估了各种条件下量子电路的计算成本。我们的工具可在 https://github.com/team-InQuIR/InQuIR 获取。

I. 引言

近年来，量子器件的规模和精度迅速提升，从而实现了多个量子算法、量子纠错等的小规模实验。相比之下，由于稀释制冷机尺寸、硅晶圆尺寸以及布线困难等物理限制，单个处理器上可实现的量子比特数量存在上限。因此，有必要通过量子互连连接多个量子处理器，构建量子计算机集群，以执行大规模量子信息处理。

对于分布式系统上的量子共识 [7] 和盲量子计算 [8] 等分布式任务，量子信息处理中量子计算机之间所需的通信很容易明确描述。当量子通信被明确描述时，诸如 QMPI [9] 这样的量子通信接口便发挥着重要作用。然而，在大多数量子信息处理中，量子程序并非为分布式系统编写。此外，编写满足分布式系统各种约束的量子程序，或考虑避免通信瓶颈的策略，都很困难。不满足这些固有约束的程序可能导致整个计算失败。

为解决这些问题，人们提出了几种分布式量子计算的编译方法，并通过执行时间 [10], 电路深度 [11],量子纠缠资源 [12]对其进行了评估。然而，由于这些方法分别在各自的架构和约束条件下评估性能，因此难以在相同环境中进行比较。此外，由于缺乏适合描述分布式量子程序的编程语言，它们并未以 QIR（量子中间表示） [13] 这类可复用的程序格式发布其成果。

A. 贡献

本文提出了 InQuIR，这是一种面向互连量子计算机的中间表示。我们的主要贡献如下：

- 我们定义了 InQuIR 的形式语义，其指令集足以描述分布式量子程序的复杂行为。
- 我们给出了若干用 InQuIR 编写的示例，帮助我们理解分布式量子程序如何在量子互连上运行。这些示例包括分布式系统中固有的运行时错误，例如量子比特内存耗尽，以及处理器之间相互通信时发生的死锁。
- 我们提供了一种使用 InQuIR 对各种分布式量子程序进行资源估算的软件工具。我们还实现了一个玩具编译器，可将量子程序的传统表示形式（例如 OpenQASM）转换为 InQuIR，但我们的工具并不依赖于特定的编译器。
- 我们提出了一条路线图，旨在引入静态分析，使 InQuIR 程序更加可靠。具体而言，我们解释了如何利用类型系统估算资源消耗，并处理死锁等运行时错误。

本文其余部分安排如下：第二节介绍量子互连，即本研究的背景。第三节介绍相关工作，包括分布式编程语言和用于分布式量子计算的库。第四节讨论 InQuIR 编程语言的语言设计。第五节介绍 InQuIR 编程语言的形式语法和语义，并给出若干示例。第六节介绍用于评估 InQuIR 程序的软件工具。第七节介绍 InQuIR 静态验证的路线图。第八节

讨论如何改进 InQuIR，使其适用于实际架构，并展望未来工作；最后，第九节总结全文。

II. 预备知识

A. 量子互连

许多先进的经典高性能计算（HPC）系统 [14]–[16] 利用多个处理器和互连，以实现高于单个处理器的性能。类似的方法也可能适用于量子计算机。这种量子计算系统可以通过提供一种称为量子互连的量子通信信道来实现 [6]。量子互连可以将一个节点中的量子态传输到另一个节点，和/或在多个量子处理器之间分发纠缠资源，例如 EPR态、GHZ态 [17], 簇态 [18], 和图态 [19] 。这有望实现使用大量量子比特的量子计算。另一方面，人们担心量子互连会成为瓶颈，导致计算速度下降 [12]。为解决这一问题，已有多种编译器 [20]–[24]被研究，用于将量子电路分配到分布式架构中，从而减少系统中由量子通信造成的瓶颈。

在许多情况下，用于量子计算的数据量子比特（物质量子比特）和参与方之间的通信量子比特（飞行量子比特）使用不同物理自由度中的量子比特。将量子比特转换为另一种物理系统中的量子比特的器件称为量子接口 [25]，在最近的实验 [26]–[34]中，其转换效率和保真度均取得了显著提升。预计高质量的量子互连将在不久的将来实现。

量子互联网 [35], [36] 也是连接多台量子计算机的方法之一，但它在若干方面不同于量子互连。量子互联网的物理系统采用远距离量子通信，主要通过光进行，因此必须应对光学系统固有的故障。这需要采用光子量子纠错码 [37]–[39]和量子中继器 [40]等方法。此外，各节点之间同步和经典通信的成本高于互连，而且需要复杂的多方通信协议 [41]–[43]。由于这些困难，预计量子互联网比量子互连更难实现。

B. 遥数据与遥门

要在多个处理器上运行量子程序，必须对位于不同处理器上的量子比特执行远程操作。我们可以利用纠缠来实现远程操作；这些纠缠可通过量子互连在直接连接的量子处理器之间生成。消耗纠缠来实现此类操作有两种方式：(1) 遥数据 通过量子隐形传态传送量子比特 [44]；以及 (2) 遥门 通过门隐形传态实现 [45]。在第一种方法中，在应用多量子比特门之前，一个量子比特

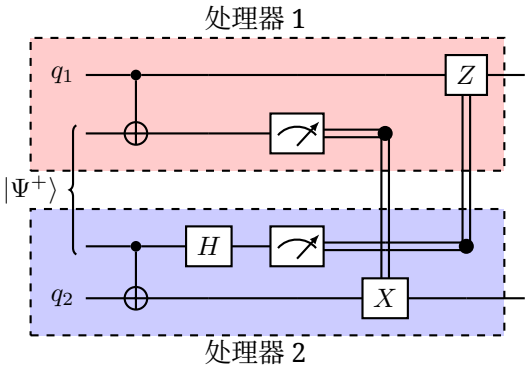


图 1. 远程 CX 门。

会被传送到另一个量子比特所在的处理器上，然后在本地应用该门。在第二种方法中，通过门隐形传态远程对目标量子比特应用多量子比特门，因此量子比特的位置不会改变。例如，图 1 展示了通过门隐形传态实现远程 受控非门 [46]。分布式量子编译器可以恰当地运用这些策略，生成高效的量子程序。

一般来说，量子处理器的连接图并不总是完全连接的。也就是说，其中一些处理器之间没有直接连接。在这种情况下，可以通过消耗多个纠缠，在相距较远的处理器之间建立纠缠。这一过程称为纠缠交换 [47]，如图 2 所示。在这个示例中，三个处理器呈线性连接，电路通过消耗两个纠缠，在两端的处理器之间建立纠缠。最后，我们可以利用纠缠交换生成的纠缠，在相距较远的量子处理器之间实现远程操作。

III. 相关工作

A. 分布式计算编程语言

Dahlberg 等人提出了 NetQASM [48]，这是 OpenQASM [49] 的变体之一，专为量子互联网设计。NetQASM 采用与平台无关的指令集，其中包括纠缠生成和纠缠分发，并支持编写可在量子互联网上运行的通用量子协议。NetQASM 与 InQuIR 的区别在于，NetQASM 主要面向量子互联网设计，而 InQuIR 则面向量子互连设计。此外，InQuIR 具有形式化定义的语义，而 NetQASM 没有。形式语义对于讨论和推理分布式量子程序的行为至关重要。

Gay 和 Nagarajan 通过扩展通信量子进程（CQP） [50] ，为 π演算 [51] 添加量子信息处理原语。他们还将 CQP 应用于量子通信协议（如 BB84 [52], [53]）的形式化验证。虽然 CQP 适合描述量子互联网中的量子通信协议，但它并非为控制

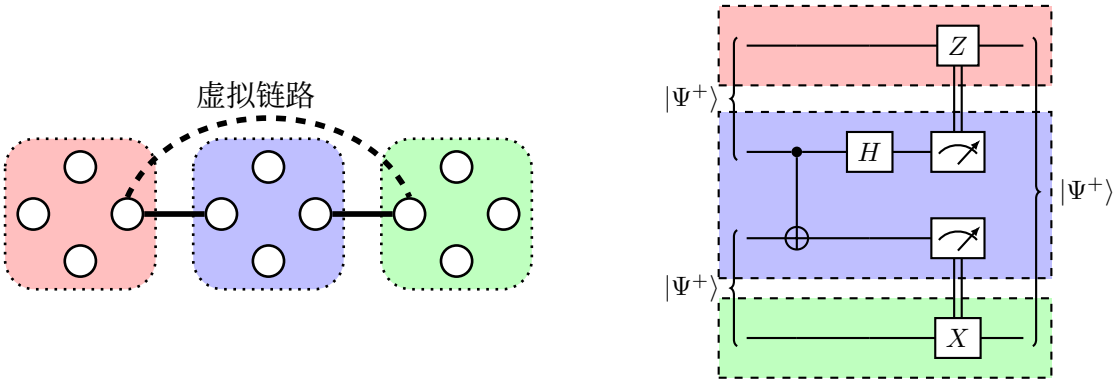


图 2. 量子处理器的拓扑结构（左）和纠缠交换过程（右）。

互连量子处理器而设计的指令集。我们应开发一种面向量子计算机集群的表示，并实现对分布式量子算法的形式化推理。

Dπ-演算 [54]是用于分布式计算的 π-演算的扩展。与 π-演算不同，Dπ-演算根据进程运行的位置区分进程。定位进程也有助于描述量子互连上的量子程序，因为每个量子处理器可能拥有不同的计算资源，例如可用量子比特的数量、与其他处理器的连接情况，以及指令的执行成本。因此，我们设计了 InQuIR，将其作为带有量子原语的 Dπ-演算扩展。

B. 分布式量子计算框架

Häner 等人提出了用于实现分布式量子算法的 QMPI [9] 。虽然我们可以广泛使用 QMPI 来实现分布式量子程序，但它不适合作为分布式量子编译器输出目标的表示，因为 QMPI 只是一个框架，而非编程语言。他们还提出了 SENDQ 模型，用于评估分布式量子算法的性能。该模型有助于分析分布式量子程序，包括 InQuIR。

近年来，各种量子互联网模拟器 [55]–[60] 已经问世，可用于验证通信协议并模拟噪声。值得注意的是，NetQASM 可在 SimulaQron [55] 和 NetSquid [58]上进行模拟。这些模拟器有望用于设计 InQuIR 的后端模拟器。

IV. 语言设计

在考虑互联量子计算机的中间表示时，InQuIR 的设计基于以下模型：

使用 EPR 对进行多方量子交互。多个参与方之间的所有量子操作（远程门操作和遥数据）都利用 EPR 对执行。例如，使用 EPR 对进行量子隐形传态，以发送和接收量子态。也可以直接传输计算过程中使用的数据量子比特，而不使用任何纠缠态。不过，这要求量子接口具有极高的转换效率和保真度，这很难实现，尤其是在没有量子纠错码的情况下。另一方面，使用纠缠纯化等协议，可以借助纠缠资源实现高保真度的远程操作。此外，只需关注纠缠资源消耗，就能轻松地表征程序执行的成本。

架构独立性。它尽可能不依赖特定的物理系统（例如超导、囚禁离子）或指令集。它不涉及参与方内部的量子比特分配，因此不受量子比特连接性等约束的影响。这提高了使用 InQuIR 编写的分布式量子程序的可重用性，也为与量子互联网模拟器相对轻松地集成留出了空间。

经典操作与通信。在不同处理器上运行的分布式量子程序会彼此进行经典通信，以执行远程操作，包括纠缠生成、量子隐形传态和远程 CX 门。此外，布尔运算等基本经典操作对于管理量子测量的经典反馈也是必不可少的。我们将 InQuIR 设计为一种包含原语的语言，使我们能够灵活地表达此类经典操作与通信。因此，InQuIR 可用于描述基于测量的分布式量子计算模型 [61]。

异步与并发进程。 在互联计算机中，分布式程序通常包含多个进程，它们在不同的处理器上并发运行。因此，InQuIR 必须提供一种原语来表达此类并发进程。此外，InQuIR 采用异步执行模型；并发进程以异步方式运行。与异步模型相比，同步模型可能会增加每条指令的执行时间，因为它需要所有处理器之间进行同步，或者由一个经典控制器管理整个系统。尤其是在异构系统中，如果每个处理器采用不同的架构或代码空间，运行速度最慢的处理器就会成为整个系统的瓶颈

。因此，异步执行模型适合描述分布式系统的行为。如果我们使用 InQuIR 编写量子通信协议，异步模型也适用于量子互联网。

V. InQuIR 的语法与语义

A. 语法与操作语义

首先，我们给出由图 3 中所示语法定义的 InQuIR 语法。

数字 p 表示一个参与方（或处理器）。（经典）会话 s 用于参与方之间发送和接收经典数据。标签 l 用于标识所通信数据的类型以及通信操作的对象，后文将对此进行说明。会话和标签分别可视为对应于（消息传递接口）MPI 库中的通信器和标记 [62]。

在 InQuIR 中，量子比特分为两类：通信量子比特和数据量子比特。通信量子比特在处理器之间的量子通信中发挥着至关重要的作用，第二节将对此进行说明。其他所有部分中的量子比特均为数据量子比特。数据量子比特集合和通信量子比特集合分别表示数据量子比特和通信量子比特的集合。我们也分别将数据量子比特和通信量子比特写作 q 和 q。

进程 P 是 InQuIR 程序的一个单元，包含一系列操作。系统 S 表示一个完整的 InQuIR 程序，由并发运行的定位进程组成。定位进程 $[\![P]\!]_p$ 表示进程P 在参与方 p 上执行。系统的组合$S_1 \mid S_2$ 表示 S_1 和 S_2 并发且异步地求值。InQuIR 允许 S_1 和 S_2 包含位于同一位置的进程。换言之，具有 $p_1 = p_2$ 的并发系统 $[\![P_1]\!]_{p_1} \mid [\![P_2]\!]_{p_2}$ 是合法的。请注意，$[\![P_1]\!]_p \mid [\![P_2]\!]_p$中$P_1$ 和P_2 的求值顺序是不确定的。

接下来，我们引入运行时状态，以定义 InQuIR 的操作语义。

定义 1 (运行时状态) InQuIR 程序的运行时状态是一个由 $[\rho, Q, E, P\ H]$ 构成的五元组，其中

- ρ 是表示量子态的密度算符，
- Q是每个参与方可用的数据量子比特集合，
- E 是参与方之间可用的通信量子比特集合，
- S 是当前正在求值的系统，并且
- H 是用于经典通信的堆。

堆 H： 会话×参与方 $\to$ 列表 (标签×值)用于管理其他参与方发送的经典数据。每个参与方都有一个通信缓冲区—$H(s, p) = l : v$，其中$l : v$ 是带标签的值序列。标签表示发送该值的用途。

现在，我们可以将 InQuIR 的操作语义定义为运行时状态上的迁移关系 $\to$。

定义 2 (操作语义) InQuIR 的操作语义由运行时状态上的迁移关系 $\to$ 定义

。换言之，当我们为运行时状态 R 和 R' 写下 $R \to R'$ 时，可以理解为 R 在一步之内迁移到 R'。具体内容见图 4。

接下来，我们将对图 3 中每种操作的语义进行非正式描述。

停止。 停止操作 **stop** 表示进程已成功完成。

开启与关闭。 会话开启操作 s =**open** $[p_1, .., p_n]$ 会与处理器$p_1, .., p_n$开启一个新会话s。处理器使用会话发送或接收经典数据，例如量子测量的结果。开启新会话时，会在当前堆中创建 n 通信缓冲区，以存储通过该会话发送的数据。相反，关闭操作 **close(**s**)**会关闭会话s，并丢弃堆中与该会话s 以及关闭该会话的参与方p 关联的缓冲区。

量子比特初始化。 量子比特初始化 x = **init()** 从当前处理器获取一个已初始化为 $|0\rangle$ 的量子比特，并将变量 x 绑定到该量子比特。如果处理器在进行量子比特初始化时没有可用的量子比特，则会等待后文所述的释放操作返回量子比特。

不同于量子电路表示，InQuIR 不会规定量子比特在量子比特初始化时的位置。换句话说，每次量子比特初始化时分配哪些量子比特，将由更加架构特定的后端编译器或运行时调度器决定。

纠缠生成。EPR 对生成 x =**genEnt** $[p]$**()** 与另一个参与方p 生成一个 EPR 对，并将变量 x 绑定到该 EPR 对。标签 l 用于标识参与方 p 上运行的配对操作，以避免歧义，如图 5 所示。如果每个处理器上同时运行两对纠缠生成操作，则它们之间可能存在两种连接（红线和蓝色虚线）。

量子比特释放。 释放操作 **free**x 将数据量子比特或通信量子比特 x 返回处理器。该操作会针对量子比特 x对当前状态取偏迹。此时，InQuIR 认为可以安全地释放量子比特 x；也就是说，量子比特 x 与任何其他量子比特均处于非纠缠状态。通常，很难检查给定量子比特是否可以安全地重置 [63]。安全重置的保证不在 InQuIR 的范围内。

纠缠交换。 纠缠交换表达式（x_1,x_2） =**纠缠交换** （y_1,y_2）使用两个纠缠态y_1 和y_2，通过纠缠交换创建一条虚拟链路。此操作将纠缠交换过程中测量的结果绑定到变量x_1 和x_2上。图 6 展示了（x_1,x_2） =**纠缠交换** （y_1, y_2）的工作方式。为了完成整个纠缠交换，我们必须确保端点收到x_1（或x_2），以应用受控 Z（或 X）门。

测量。操作 x =**M(**$x_1, .., x_n$**)** 执行量子测量 $\{M_s\}$，并将x 绑定到经典结果，其中

$$M_s = \frac{I + (-1)^s Z_{x_1} \dots Z_{x_n}}{2} \quad (s = 0, 1)$$

参与方 $\ni p$ 会话 $\ni s$ 标签 $\ni l$ 数据量子比特 $\ni q$ 通信量子比特 $\ni q$

$$
\begin{aligned}
(\textit{Process}) \quad P ::= {} & \mathbf{stop} \mid s = \mathbf{open}\,[p_1, .., p_n]; P \mid \mathbf{close}\,(s); P \mid x = \mathbf{init}\,(); P \mid \mathbf{free}\,x; P \mid \\
& x = e; P \mid U(e_1, .., e_n); P \mid x = \mathbf{M}(e_1, .., e_n); P \mid \\
& x = \mathbf{genEnt}\,[p](l); P \mid (x_1, x_2) = \mathbf{entSwap}\,(e_1, e_2); P \mid \\
& \mathbf{if}\ e\ \mathbf{then}\ P_1\ \mathbf{else}\ P_2 \mid x_1 = \mathbf{qrecv}\,(s, l, e_2); P \mid \mathbf{qsend}\,[p](s, l, e_1, e_2); P \mid \\
& \mathbf{rcxc}\,[p](s, l, e_1, e_2); P \mid \mathbf{rcxt}\,[p](s, l, e_1, e_2); P \mid s[p]!(l : e); P \mid s?(l : x); P \\
(\textit{System}) \quad S ::= {} & \mathbf{0} \mid [\![P]\!]_p \mid (S_1 \mid S_2) \\
(\textit{Expression}) \quad e ::= {} & v \mid e_1 \wedge e_2 \mid e_1 \oplus e_2 \mid \ldots \\
(\textit{Value}) \quad v ::= {} & 0 \mid 1 \mid q \mid \overline{q} \mid x \mid \ldots \\
(\textit{Gate}) \quad U ::= {} & X \mid Z \mid H \mid T \mid CX \mid \ldots
\end{aligned}
$$

图 3. InQuIR 的语法。

量子比特隐形传态。量子比特发送操作$\mathbf{qsend}\ [p](s, l, x_1, x_2)$ 通过纠缠x_2传送数据量子比特x_1。类似地，量子比特接收操作$x_1 = \mathbf{qrecv}\,(s, l, x_2)$ 通过纠缠x_2 接收数据量子比特，并将变量x_1 绑定到该量子比特。会话s和标签l 用于量子比特隐形传态中泡利校正的经典通信。这些操作可以写成如下形式：

$$
\begin{aligned}
&\mathbf{qsend}\,[p_2](s, l, x_1, x_2) \\
&\quad \triangleq CX(x_1, x_2); H(x_1); y_1 = \mathbf{M}(x_1); y_2 = \mathbf{M}(x_2); \\
&\qquad s[p_2]!(l : y_1); s[p_2]!(l : y_2); \mathbf{free}\,x_1; \mathbf{free}\,x_2 \\
&x_1 = \mathbf{qrecv}\,(s, l, x_2) \\
&\quad \triangleq x_1 = \mathbf{init}\,(); s?(l : y_1); s?(l : y_2); \\
&\qquad Z^{y_1}(x_2); X^{y_2}(x_2); SWAP(x_1, x_2); \mathbf{free}\,x_2
\end{aligned}
$$

请注意，U^v 表示由经典值v控制的酉操作。量子比特发送操作无需等待接收方，因为量子隐形传态是一种单向通信。

远程 CX 门。远程 CX 操作$\mathbf{rcxc}\ [p](s,\ l,\ x_1, x_2)$ 和 $\mathbf{rcxt}\ [p](s,\ l,\ x_3, x_4)$ 通过消耗纠缠对 (x_2, x_4)，在远程对x_1 (控制量子比特)和x_3 (目标量子比特)应用 CX 门。这两条指令也可以写成如下形式：

$$
\begin{aligned}
&\mathbf{rcxc}\,[p](s, l, x_1, x_2) \triangleq CX(x_1, x_2); y = \mathbf{M}(x_2); \\
&\qquad s[p]!(l : y); s?(l : y'); Z^{y'}(x_1) \\
&\mathbf{rcxt}\,[p](s, l, x_1, x_2) \triangleq CX(x_2, x_1); H(x_2); y' = \mathbf{M}(x_2); \\
&\qquad s[p_1]!(l : y'); s?(l : y); X^{y}(x_1)
\end{aligned}
$$

经典通信。发送操作$s[p]!(l : v)$ 会通过会话s将经典数据v（标签为l ）发送到处理器 p。发送的数据会追加到通信缓冲区 $H(s,p)$ 的末尾。接收操作$s?(l : x)$ 从通信缓冲区中获取标签为l 的数据，并将其绑定到x 。如果缓冲区中有两个或更多标签为l 的数据，则弹出第一个数据。如果目标数据尚未发送，此操作将等待该数据。

分支。分支操作$\mathbf{if}\ e\ \mathbf{then} P_1\ \mathbf{else}\ P_2$ 计算e 并根据计算结果选择P_1 或P_2 。记法$e \Downarrow v$ 表示e 的计算结果为值v。

接下来，我们介绍操作语义中使用的记号。

$[v/x]$ 表示在进程和表达式中用v代换x。形式上，代换定义如下：

$$
\begin{aligned}
[v/x]y &\triangleq \begin{cases} y & (x \neq y) \\ v & (x = y) \end{cases} \\
[v/x]\,y = e; P &\triangleq \begin{cases} y = [v/x]e; [v/x]\,P & (x \neq y) \\ y = [v/x]e; P & (x = y) \end{cases} \\
&\vdots
\end{aligned}
$$

当且仅当以下所有条件均成立时，我们记作 $\mathsf{get}(Q,p) = (Q', q)$：

- $q \in Q(p)$,
- $Q'(p) = Q(p) \setminus \{q\}$, and
- $\forall p'.\ p \neq p' \Rightarrow Q(p') = Q'(p')$.

当且仅当以下所有条件均成立时，我们记作 $Q' = \mathsf{ret}(Q,p\ q)$：

- $Q'(p) = Q(p) \cup \{q\}$, and
- $\forall p'.\ p \neq p' \Rightarrow Q(p') = Q'(p')$.

我们也以相同方式将 get 和 ret 用于通信量子比特 E 。当且仅当以下所有条件均成立时，我们记作 $H' = \mathsf{push}(H, s, p, l{:}\ v)$：

- $H'(s, p) = H(s, p) +\!\!+ l : v$, and
- $\forall s', p'.\ s \neq s' \vee p \neq p' \Rightarrow H(s, p) = H(s', p')$,

其中，该运算符 $+\!\!+$ 将两个列表连接起来。当且仅当以下所有条件均成立时，我们记作 $\mathsf{pop}(H, s, p, l) = (H', v)$：

$$(\textit{DataStore}) \quad Q \triangleq \textit{Participants} \to 2^{\textit{DataQubits}} \qquad (\textit{EPRStore}) \quad E \triangleq \textit{Participants} \rightharpoonup 2^{\textit{CommQubits}}$$
$$(\textit{Heap}) \quad H \triangleq \textit{Session} \times \textit{Participant} \rightharpoonup \textit{List}\ (\textit{Labels} \times \textit{Values})$$

$$[\rho, Q, E, [\![s = \mathbf{open}\,[p_1, .., p_n]; P_1]\!]_{p_1} \mid .. \mid [\![s = \mathbf{open}\,[p_1, .., p_n]; P_n]\!]_{p_n}, H]$$
$$\to [\rho, Q, E, [\![P_1]\!]_{p_1} \mid .. \mid [\![P_n]\!]_{p_n}, H \uplus \{(s, p_1) \mapsto \epsilon, .., (s, p_n) \mapsto \epsilon\}]$$
$$[\rho, Q, E, [\![\mathbf{close}\,(s); P]\!]_p, H] \to [\rho, Q, E, [\![P]\!]_p, H \setminus \{(s, p)\}]$$

$$\frac{\text{get}(Q, p) = (Q', q)}{[\rho, Q, E, [\![x = \mathbf{init}\,(); P]\!]_p, H] \to \left[\rho \otimes |0\rangle_q \langle 0|, Q', E, [\![[q/x]\,P]\!]_p, H\right]}$$

$$[\rho, Q, E, [\![\mathbf{free}\,q; P]\!]_p, H] \to [\text{Tr}_q(\rho), \text{ret}(Q, p, q), E, [\![P]\!]_p, H]$$
$$[\rho, Q, E, [\![\mathbf{free}\,\overline{q}; P]\!]_p, H] \to [\text{Tr}_{\overline{q}}(\rho), Q, \text{ret}(E, p, \overline{q}), [\![P]\!]_p, H]$$

$$[\rho, Q, E, [\![x = e; P]\!]_p, H] \to [\rho, Q, E, [\![[v/x]\,P]\!]_p, H] \qquad (\text{if } e \Downarrow v)$$

$$\frac{\forall v_i \in \textit{DataQubit} \cup \textit{CommQubit}}{[\rho, Q, E, [\![U(v_1, .., v_n); P]\!]_p, H] \to \left[U_{v_1, .., v_n} \rho U^\dagger_{v_1, .., v_n}, Q, E, [\![P]\!]_p, H\right]}$$

$$[\rho, Q, E, [\![s[p_2]!(l : v); P]\!]_{p_1}, H] \to [\rho, Q, E, [\![P]\!]_{p_1}, \text{push}(H, s, p_2, l : v)]$$

$$\frac{\text{pop}(H, s, p, l) = (H', v)}{[\rho, Q, E, [\![s?(l : x); P]\!]_p, H] \to [\rho, Q, E, [\![[v/x]\,P]\!]_{p_1}, H']}$$

$$\frac{v \in \{0, 1\} \qquad M_v = (I + (-1)^v Z_{q_1} \cdots Z_{q_n})/2}{[\rho, Q, E, [\![x = \mathbf{M}(q_1, .., q_n); P]\!]_p, H] \to \left[M_v \rho M_v^\dagger, Q, E, [\![[v/x]\,P]\!]_p, H\right]}$$

$$[\rho, Q, E, [\![\mathbf{if}\ e\ \mathbf{then}\ P_1\ \mathbf{else}\ P_2]\!]_p, H] \to [\rho, Q, E, [\![P_i]\!]_p, H] \qquad (\text{if } e \Downarrow i \wedge i \in \{0, 1\})$$

$$\frac{\text{get}(E, p_1) = (E', \overline{q}_1) \qquad \text{get}(E', p_2) = (E'', \overline{q}_2)}{[\rho, Q, E, [\![x_1 = \mathbf{genEnt}\,[p_2](l); P_1]\!]_{p_1} \mid [\![x_2 = \mathbf{genEnt}\,[p_1](l); P_2]\!]_{p_2}, H]}$$
$$\to \left[\rho \otimes |\Psi_+\rangle_{\overline{q}_1 \overline{q}_2} \langle \Psi_+|, Q, E'', [\![[\overline{q}_1/x_1]\,P_1]\!]_{p_1} \mid [\![[\overline{q}_2/x_2]\,P_2]\!]_{p_2}, H\right]$$

$$[\rho, Q, E, [\![(w_1, w_2) = \mathbf{entSwap}\,(\overline{q}_1, \overline{q}_2); P]\!]_p, H] \to [\rho, Q, \text{ret}(\text{ret}(E, p, \overline{q}_1), p, \overline{q}_2), [\![P]\!]_p, H\}\}]$$

$$\frac{Q' = \text{ret}(Q, p_1, q) \qquad E' = \text{ret}(E, p_1, \overline{q}) \qquad H' = \text{push}(H, s, p_1, p_2, l : v_1 +\!\!+ l : v_2])}{[\rho, Q, E, [\![\mathbf{qsend}\,[p_2](s, l, q, \overline{q}); P]\!]_{p_1}, H] \to \left[\text{Tr}_{q, \overline{q}}\left(U_{\text{qsend}} \rho U^\dagger_{\text{qsend}}\right), Q', E', [\![P]\!]_{p_1}, H'\right]}$$

$$\frac{\text{get}(Q, p) = (Q', q) \qquad E' = \text{ret}(E, p, \overline{q}) \qquad \text{pop}(H, s, p, l) = (H', v_1) \qquad \text{pop}(H', s, p, l) = (H'', v_2)}{[\rho, Q, E, [\![x = \mathbf{qrecv}\,(s, l, \overline{q}); P]\!]_p, H] \to \left[U_{\text{qrecv}}(\rho \otimes |0\rangle_q \langle 0|) U^\dagger_{\text{qrecv}}, Q', E', [\![[q/x]\,P]\!]_p, H''\right]}$$

$$[\rho, Q, E, [\![\mathbf{rcxc}\,[p_2](s, l, q_1, \overline{q}_1); P_1]\!]_{p_1} \mid [\![\mathbf{rcxt}\,[p_1](s, l, q_2, \overline{q}_2); P_2]\!]_{p_2}, H]$$
$$\to \left[CX_{\,q_1\,q_2} \rho CX^\dagger_{\,q_1\,q_2}, Q, \text{ret}(\text{ret}(E, p_1, \overline{q}_1), p_2, \overline{q}_2), [\![P_1]\!]_{p_1} \mid [\![P_2]\!]_{p_2}, H\right]$$

$$\frac{[\rho, Q, E, S_1, H] \to [\rho', Q, E, S_1', H]}{[\rho, Q, E, S_1 \mid S_2, H] \to [\rho', Q, E, S_1' \mid S_2, H]} \qquad \frac{S_1 \equiv S_2 \qquad [\rho, Q, E, S_1, H] \to [\rho', Q, E, S_1', H'] \qquad S_1' \equiv S_2'}{[\rho, Q, E, S_2, H] \to [\rho, Q, E, S_2', H]}$$

$$S \mid \mathbf{0} \equiv S \qquad S_1 \mid S_2 \equiv S_2 \mid S_1 \qquad S_1 \mid (S_2 \mid S_3) \equiv (S_1 \mid S_2) \mid S_3 \qquad [\![\mathbf{stop}]\!]_p \equiv \mathbf{0}$$

Fig. 4. The operational semantics of InQuIR.

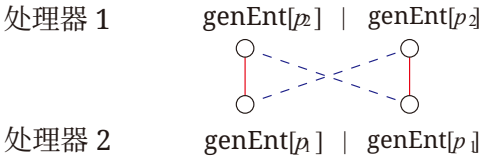


图 5. 有歧义的纠缠生成。

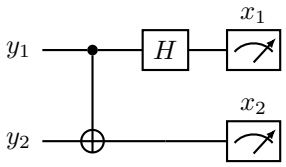


图 6. $(x_1,\ x_2) = $**纠缠交换**$(y_1,\ y_2)$ 的示意图。

- $H(s,\ p) = \overline{l_1 : v_1} + l : v + \overline{l_2 : v_2}$ 并且不存在满足以下条件的 $v' :\ l : v' \in l_1 : v_1$,
- $H'(s,\ p) = \overline{l_1 :\ v_1} + \overline{l_2 :\ v_2}$, 且
- $\forall s', p'. s \neq s' \vee p \neq p' \Rightarrow H(s, p) = H(s', p')$.

最后，我们定义一种卡住状态，它对 InQuIR 程序的安全性起着至关重要的作用。

定义 3 (卡住状态) 如果 $P \neq \mathbf{0}$，并且不存在运行时状态 $[\rho', Q', E', P', H']$ 使得 $[\rho, Q, E, P\ \ H] \to [\rho', Q', E', P', H']$，那么运行时状态 $[\rho, Q, E, P\ \ H]$ 称为卡住状态。

卡住状态是不理想的，因为它无法完成所有进程。正如后文所述，分布式程序可能因计算资源有限而到达这种状态。因此，分布式量子编译器必须转换量子程序，以确保输出不会陷入卡住状态。卡住状态这一概念使我们能够讨论分布式量子程序的安全性。也就是说，如果分布式量子程序不会到达卡住状态，就可以安全地执行。这一概念对于 InQuIR 的静态验证非常重要；静态验证会检查我们能否安全地运行 InQuIR 程序。

B. 示例

示例 1（纠缠交换）第一个示例展示了如何在相距遥远的处理器之间通过纠缠交换应用远程 CX 门。

$$
\begin{aligned}
E &\triangleq \{(p_1, p_2) \mapsto \{\overline{q}_1\}, (p_2, p_1) \mapsto \{\overline{q}_2\}, \\
&\qquad (p_2, p_3) \mapsto \{\overline{q}_3\}, (p_3, p_2) \mapsto \{\overline{q}_4\}\} \\
P_1 &\triangleq s = \mathbf{open}\,[p_1, p_2, p_3]; x_2 = \mathbf{genEnt}\,[p_2](l_1); \\
&\qquad s?(l : w_1); Z^{w_1}(x_2); \mathbf{rcxc}\,[p_3](s, l_3, q_1, x_2) \\
P_2 &\triangleq s = \mathbf{open}\,[p_1, p_2, p_3]; z_1 = \mathbf{genEnt}\,[p_1](l_1); \\
&\qquad z_2 = \mathbf{genEnt}\,[p_3](l_2); \\
&\qquad (w_1, w_2) = \mathbf{entSwap}\,(z_1, z_2); \\
&\qquad s[p_1]!(l : w_1); s[p_2]!(l : w_2) \\
P_3 &\triangleq s = \mathbf{open}\,[p_1, p_2, p_3]; y_2 = \mathbf{genEnt}\,[p_2](l_2); \\
&\qquad s?(l : w_2); X^{w_2}(y_2); \mathbf{rcxt}\,[p_1](s, l_3, q_2, y_2) \\
S &\triangleq [\![P_1]\!]_{p_1} \mid [\![P_2]\!]_{p_2} \mid [\![P_3]\!]_{p_3}
\end{aligned}
$$

为简洁起见，此处省略了最后的操作 **停止**，后文所有示例也同样如此。在此示例中，我们假设量子比特 q_1、q_2 已经初始化。此外，处理器 p_1,p_2,p_3 呈线性连接，并且每个连接处恰好有一个通信量子比特可用。表 I 展示了该程序的求值步骤。

示例 2（屏障）尽管 InQuIR 的进程以异步方式运行，但你可能希望让 InQuIR 进程的某个部分同步运行。例如，可以设想所有处理器在每条指令执行时都彼此同步。InQuIR 可以通过其异步原语实现同步通信。我们通过实现 **屏障** $[p_1, p_2](s)$ 来说明这一点；这是会话 s 上处理器 p_1 和 p_2 之间的一种屏障操作。三个或更多处理器的情况也可以用类似方式实现。

$$
\begin{aligned}
&[\![\mathbf{barrier}\,[p_1, p_2](s); P_1]\!]_{p_1} \mid [\![\mathbf{barrier}\,[p_1, p_2](s); P_2]\!]_{p_2} \\
&\quad \triangleq [\![s[p_2]!(l : v); s?(l : x_1); P_1]\!]_{p_1} \\
&\qquad\qquad \mid [\![s?(l : x_2); s[p_1]!(l : x_2); P_2]\!]_{p_2}
\end{aligned}
$$

，其中 l、v 和 x_i 都是虚设变量。我们可以用同样的方式表达任意非阻塞操作的阻塞版本。

示例 3（量子比特耗尽）下一个示例说明了如何通过量子隐形传态移动数据量子比特。

$$
\begin{aligned}
P_1 &\triangleq x_1 = \mathbf{genEnt}\,[p_2](l); \mathbf{qsend}\,[p_2](s, l, q_1, x_1) \\
P_2 &\triangleq x_2 = \mathbf{genEnt}\,[p_1](l); \\
&\qquad x_3 = \mathbf{qrecv}\,(s, l, x_2); CX(q_2, x_3) \\
S &\triangleq [\![P_1]\!]_{p_1} \mid [\![P_2]\!]_{p_2}
\end{aligned}
$$

程序 S 通过将 q_1 传送到处理器 p_2，并应用本地 CX 门，对远程量子比特 q_1 和 q_2 应用受控非门。与远程 CX 操作不同，**qsend** 操作要求接收方 p_2 准备一个新的量子比特，以接收传送来的量子数据。如果处理器 p_2 中的数据量子比特已全部占用（即 $Q(p_2 = \emptyset)$），程序将无法接收该量子比特。换言之，程序将进入卡住状态。我们将这个问题称为量子比特耗尽。

表 I. S 的执行步骤。每行包含各步骤正在运行的进程，以及评估该进程后通信量子比特和堆的变化。

正在运行的进程	E （之后，仅列出变化）	H （之后，仅列出变化）
初始状态	$p_1 \mapsto \{\overline{q}_1\}, p_2 \mapsto \{\overline{q}_2, \overline{q}_3\}, p_3 \mapsto \{\overline{q}_4\}$	$\emptyset$
$[\![s = \textbf{打开}\,[p_1, p_2, p_3]]\!]_{\ p_1}$ | .. | $[\![s = \textbf{打开}\,[p_1, p_2, p_3]]\!]_{\ p_3}$		$(s,\ p_1) \mapsto \epsilon,\ (s,\ p_2) \mapsto \epsilon,\ (s,\ p_3) \mapsto \epsilon$
$[\![x_2 = \textbf{genEnt}\,[p_2](l_1)]\!]_{\ p_1}$ | $[\![z_1 = \textbf{genEnt}\,[p_1](l_1)]\!]_{\ p_2}$	$p_1 \mapsto \emptyset,\ \ p_2 \mapsto \{\overline{q}_3\}$	
$[\![z_2 = \textbf{genEnt}\,[p_3](l_2)]\!]_{\ p_2}$ | $[\![y_1 = \textbf{genEnt}\,[p_2](l_2)]\!]_{\ p_3}$	$p_2 \mapsto \emptyset, p_3 \mapsto \emptyset$	
$[\![(w_1, w_2) = \textbf{纠缠交换}(q_2, \overline{q}_3)]\!]_{\ p_2}$ （结果为 $w_1 = v_1, w_2 = v_2$）	$p_2 \mapsto \{\overline{q}_2, \overline{q}_3\}$	
$[\![s[p_1]!(l_1 : v_1)]\!]_{\ p_2}$		$(s, p_1) \mapsto l_1 :\ v_1$
$[\![s[p_3]!(l_2 : v_2)]\!]_{\ p_2}$		$(s,\ p_3) \mapsto l_2 :\ v_2$
$[\![s?(l_1 : w_1)]\!]_{\ p_1}$ | $[\![s?(l_2 :\ w_2)]\!]_{\ p_3}$		$(s,\ p_1) \mapsto \epsilon,\ (s,\ p_3) \mapsto \epsilon$
$[\![Z^{v_1}(\overline{q}_1)]\!]_{\ p_1}$ | $[\![X^{v_2}(\overline{q}_4)]\!]_{\ p_3}$		
$[\![\textbf{rcxc}\ [p_3](s, l_3, q_1, \overline{q}_1)]\!]_{\ p_1}$ | $[\![\textbf{rcxt}\ [p_1]\ (s,\ l_3,\ q_2,\ \overline{q_4})\ ]\!]_{\ p_3}$	$p_1 \mapsto \{\overline{q}_1\}, p_3 \mapsto \{\overline{q}_4\}$	

示例 4（死锁）最后一个示例展示了一个更实际的问题，即并发程序中的循环依赖所导致的死锁。

$$
\begin{aligned}
E &\triangleq \{p_1 \mapsto \{\overline{q}_1\}, p_2 \mapsto \{\overline{q}_2, \overline{q}_3\}, p_3 \mapsto \{\overline{q}_4\}\} \\
P_1 &\triangleq x_1 = \textbf{genEnt}\,[p_2](l_1); \cdots \\
P_2 &\triangleq y_1 = \textbf{genEnt}\,[p_1](l_1); y_2 = \textbf{genEnt}\,[p_3](l_2); \\
&\quad\ (w_1, w_2) = \textbf{entSwap}\,(y_1, y_2); \cdots \\
P_3 &\triangleq z_1 = \textbf{genEnt}\,[p_2](l_2); \cdots \\
S_1 &\triangleq [\![P_1]\!]_{p_1} \mid [\![P_2]\!]_{p_2} \mid [\![P_3]\!]_{p_3} \\
P_4 &\triangleq x_2 = \textbf{genEnt}\,[p_2](l_3); \cdots \\
P_5 &\triangleq y_3 = \textbf{genEnt}\,[p_1](l_3); y_4 = \textbf{genEnt}\,[p_3](l_4); \\
&\quad\ (w_3, w_4) = \textbf{entSwap}\,(y_3, y_4); \cdots \\
P_6 &\triangleq z_2 = \textbf{genEnt}\,[p_2](l_4); \cdots \\
S_2 &\triangleq [\![P_4]\!]_{p_1} \mid [\![P_5]\!]_{p_2} \mid [\![P_6]\!]_{p_3} \\
S &\triangleq S_1 \mid S_2
\end{aligned}
$$

系统 S_1 和 S_2 尝试以相同的方式在处理器 p_1 和 p_3 之间进行纠缠交换。我们可以将S 理解为一个程序，其中两个不同的分布式程序S_1 和 S_2 运行在同一组互连量子计算机上。$P_i (i = 1, \ldots, 6)$ 的执行顺序未作规定，因此 S 可能会按照以下执行步骤继续执行：

1) $[\![P_1]\!]_{p_1}$ 和 $[\![P_2]\!]_{p_2}$ 生成一个 EPR 对，并将 q_1从 E^- 中移除。2) $[\![P_5]\!]_{p_2}$ 和 $[\![P_6]\!]_{p_3}$ 生成一个 EPR 对，并将 q_4从 E 中移除。3) 尝试在 P_2 和 P_3 之间（或在 P_4 和 P_5之间）生成一个 EPR 对，以建立用于纠缠交换的链。

在步骤 3 之前，p_1 和 p_2 之间唯一的通信量子比特 q_1已被 S_1^- 占用，因此系统 S_2 必须等待 S_1 将其归还给 E。同样，S_1 也必须等待 S_2 将 q_4 归还给E。S_1 和 S_2 都在等待对方的进程完成，因此陷入僵局。

我们将在第 VII 节讨论如何处理示例 3 和示例 4 这类问题。

VI. InQuIR 程序的资源分析

本节将说明，InQuIR 可以作为分布式量子编译器的目标语言，用于估算其编译后程序的性能。为此，我们实现了若干软件工具，包括 InQuIR 编译器和资源分析器，并在各种条件下对多个量子电路进行了基准测试。

A. 编译策略

由于纠缠生成是分布式量子计算中开销最大的操作之一，量子编译器必须安排数据量子比特的位置，使量子通信的数量尽可能少。由于开发新型优化方法超出了本文的讨论范围，我们基于一种直接的策略编译量子程序：

- 按顺序将量子比特分配给量子处理器，并且
- 将任何远程受控非门 转换为**rcxc** 和**rcxt** ，并进行纠缠交换。

请注意，InQuIR 不依赖特定的编译器，而我们的工具提供了玩具编译器。

B. 评估方法

在本研究中，我们根据各种指标计算了量子程序的成本。具体而言，本文使用 InQuIR 模拟器展示了以下指标：

- E计数、E 深度，
- C计数、C 深度，
- 估计时间，以及
- 各处理器在每个时刻剩余操作的数量。

E计数是纠缠生成的次数。E 深度是仅考虑

纠缠生成依赖关系时关键路径的长度。这些指标常用于量子互连，因为纠缠生成的成本非常高。同样，C 计数和 C 深度是用于估算经典通信成本的指标。

备注 1 我们的 E 深度计算方法与常见方法略有不同。在分析中常用的量子电路表示中，量子比特以线路表示，因此可以清楚地看出每条指令使用的是哪个特定量子比特。另一方面，InQuIR 并未指定在初始化量子比特或生成纠缠时使用哪个量子比特，而量子比特的选择可能会改变 E 深度的计算结果。有多个选项时，我们的模拟器会按照优先使用最早创建的量子比特这一策略进行计算。

执行成本是对在所有处理器上完成处理所需执行时间的估算。此处，我们根据近期实验数据 [64], [65]以及采用跨蒙特量子比特和微波-光学（M2O）转换器的 [67]方案中，关于可验证纠缠生成 [66] 的理论提案，为每种操作采用以下数值。

- 单量子比特门：30 ns
- 本地 CX 门：60 ns
- 测量：240 ns
- 经典通信：30 ns
- 纠缠生成：1000 ns

根据需要，可以通过提供 JSON 文件来自由自定义每个处理器的这些值。通过为不同处理器设置不同的成本，我们可以分析异构架构。

每个时刻各处理器尚未执行的操作数量，使我们能够分析各处理器处理其进程的效率。我们的工具根据模拟器在执行每项操作时记录的时间戳，直观展示这一数量随时间的变化。该指标可用于分析分配给各处理器的任务数量，以及它们在各个时刻的忙碌程度。

我们在以下架构上，针对处理器之间的各种网络拓扑，计算了上述计算成本：

- 线性连接的处理器，
- （3D）超立方体，以及
- 2D 环面。

我们使用小型单体量子电路进行了实验，这类电路广泛用于验证量子编译器 [68], [69][1]。

C. 评估结果

详细的评估结果如表 II 和图 7 所示。接下来，我们将说明从评估结果中可以获得哪些有助于提升程序性能的反馈。

[1]文献中的数据集 [68]源自 RevLib [70]

在表 II 的上半部分，我们可以看到，在线性连接的处理器中，通信量子比特数量变化时，计算成本会如何变化。这里，我们用 $(Q,\ E)\ \times M$ 表示具有 M 个处理器、每个处理器配有 Q 个数据量子比特和 E 个通信量子比特的架构。从 (2，2) $\times 8$ 到 (2，4)$\times 8$，各项成本几乎减半。然而，从 (2，4)$\times 8$到 (2，6)$\times 8$，这些数值并未变化。这些结果表明，优化计算成本需要多少通信量子比特。

在表 II 的下表中，我们可以看到处理器之间的连接性如何影响计算成本。结果表明，连接性越密集，计算成本就越低，因为纠缠生成成本可以分摊。通过比较这些数值，我们可以为要运行的算法选择合适的架构。

如图 7 所示，我们以 4gt12-v1_89 和 adr4_197 电路为例，展示如何根据各节点剩余操作数量随时间的变化，获取改进编译算法的提示。对于 4gt12-v1_89，可以看出，在计算过程中，处理器 0 有时会完全停止工作。例如，程序开始执行后，在 20 至 40μs 期间，它一直处于停止状态。对于 adr4_197，可以看出分配给各处理器的操作数量差异很大。这些结果表明，通过改进量子比特到处理器的分配方式，有望降低总执行成本。否则，有些量子比特会被频繁操作，而另一些则不会，这也为量子比特分配提供了很好的参考。例如，可以优先将使用频率较高的量子比特分配给性能更高的处理器。

请注意，在这两张图中，有些处理器完全没有执行任何操作。这是因为程序中声明了一些量子比特，但实际上并未使用。

如我们所见，我们可以使用 InQuIR 来估算各种条件下的程序执行成本。确定未来运行大型量子算法所需的计算资源十分重要。InQuIR 可作为分布式量子程序资源估算的软件基础。

VII. InQuIR 的静态验证

如第五节所述，InQuIR 程序在执行过程中可能会陷入停滞。在实际设备上运行量子程序的执行成本仍然很高，因此我们希望在不运行程序的情况下，也就是通过静态方式，检查给定程序是否会导致此类错误。InQuIR 具有形式语义，因此我们可以引入静态验证技术，例如类型系统和抽象解释。我们尚未成功构建一个完备的验证系统，因此下面将简要介绍如何通过类型系统来保证给定程序的安全性。

类型系统的目标是通过证明类型安全来验证给定程序的安全性。类型安全通常表述如下：类型正确的程序不得进入不理想的

表 II. 评估结果。我们使用了 [68], [69] 中给出的部分数据集进行实验。我们在三种不同的处理器拓扑结构上线性、三维立方体和环面对数据集进行了编译。我们用 $(Q,\ E) \times M$ 表示具有 M 个处理器的架构，其中每个处理器有 Q 个数据量子比特和 E 个通信量子比特。符号 D_E 和 D_C 分别表示 E 深度和 C 深度。 N 是程序中声明的量子比特数量。

电路名称	N	E计数	C计数	线性：(2,2) × 8			线性：(2,4) × 8			线性：(2,6) × 8		
				D_E	D_C	成本[纳秒]	D_E	D_C	cost	D_E	D_C	成本 [纳秒]
adr4_197	16	5308	10616	1020	3150	1562850	510	2248	751630	510	2248	751630
ising_模型_16	16	140	280	10	20	13510	5	20	7280	5	20	7280
rd53_138	16	122	244	33	74	47730	17	66	23610	17	66	23610
sqn_258	16	15054	30108	2843	9606	4393910	1365	6024	2136340	1365	6024	2136340
root_255	16	31286	62572	5112	19268	8086560	2596	12878	3942970	2596	12878	3942970
4gt12-v1_89	16	224	448	68	136	97370	34	118	48780	34	118	48780
9symml_195	16	66732	133464	10512	40366	16504980	5342	25616	8025860	5342	25616	8025860
life_238	16	42796	85592	6755	26076	10628990	3432	16564	5153840	3432	16564	5153840

电路名称	N	超立方体: (2,3) × 8					环面（2，4） × 9				
		E计数	C计数	D_E	D_C	成本 [ns]	E计数	C计数	D_E	D_C	成本 [ns]
adr4_197	16	4300	8600	790	2214	1174120	3580	7160	574	1892	870770
伊辛_模型_16	16	140	280	10	20	13510	180	360	10	22	14710
rd53_138	16	122	244	31	72	44560	128	256	22	58	31870
sqn_258	16	12238	24476	2104	6600	3269030	9762	19524	1933	5334	2922090
root_255	16	22358	44716	3737	10858	5745630	18378	36756	2970	9324	4462430
4gt12-v1_89	16	224	448	48	118	70680	152	304	48	104	66720
9symml_195	16	50524	101048	7884	25472	11934160	39780	79560	8232	21696	12420720
life_238	16	32484	64968	5039	16530	7675450	25408	50816	5073	13740	7675000

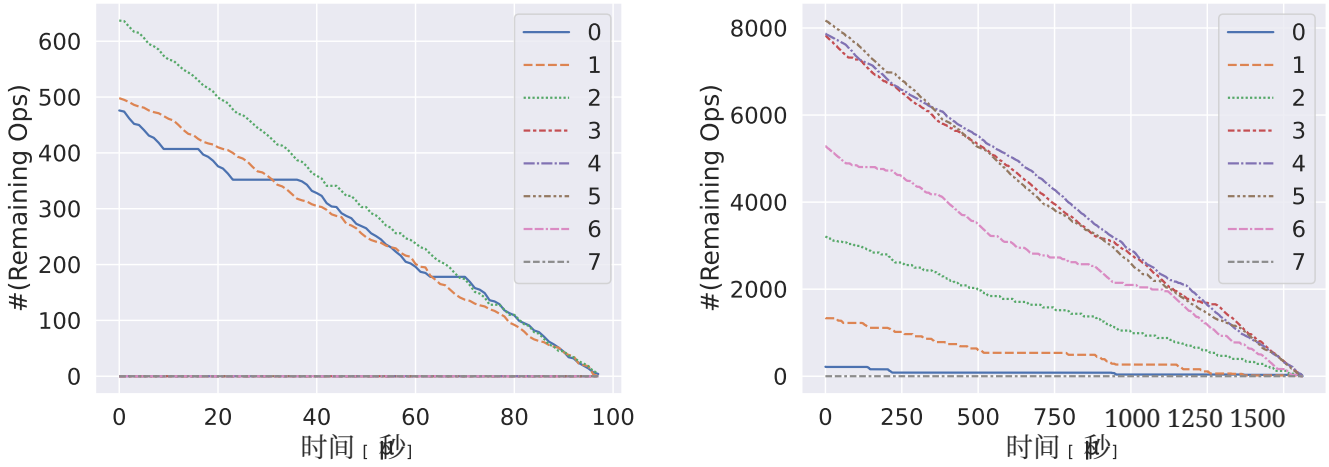


图 7. 操作数量 各处理器在每个时刻剩余的资源，处理器编号从 0 到 7。左图和右图分别表示 4gt12-v1_89和 adr4_197。它们是在 在 (2，2) × 8 线性连接架构上运行。

状态。不理想的状态取决于具体情况。例如，我们可以将卡住状态定义为不理想状态。我们需要根据想要证明的性质来形式化类型系统。

在 InQuIR 中，程序可能会因量子比特耗尽和死锁而陷入停滞，示例 3 和示例 4对此作了解释。为了避免量子比特耗尽，我们需要进行量子资源分析，计算同时使用了多少量子比特。我们可以使用线性类型 [71], 来分析量子资源；许多量子编程语言都普遍采用这种方法，以确保每个量子比特恰好被消耗一次。至于死锁检测，除了量子比特资源之外，我们还需要分析指令之间的依赖关系分析。对于两个操作 O_1 和 O_2，依赖分析会检查 O_1 是否必须在 O_2 之后执行，或者 O_1 和 O_2是否可以同时执行。由于 InQuIR 进程以异步方式运行，这项任务比单处理器同步进程的情况更为复杂。幸运的是，（多方）会话类型 [72], [73] 已被广泛用于检查经典消息传递进程的无死锁性。我们相信，类似的类型系统与线性类型系统结合后，可以很好地应用于 InQuIR，跟踪量子比特和 EPR 对的消耗情况。

VIII. 讨论与未来工作

在真实的量子计算机上运行分布式量子程序时，许多信息只有到

运行时才会知晓。例如，纠缠纯化 [74]–[76] 是一种用于生成高保真度纠缠的随机过程。此外，如果量子程序运行时间很长，计算节点可能会发生故障，或者错误率可能会因某种原因变得极高。在这种情况下，动态重新调度对于完成程序是必要的。目前，InQuIR 假设给定程序采用静态调度，因此 InQuIR 仍有扩展动态操作的空间。请注意，动态调度会使分布式量子程序的行为变得更加复杂，也更容易发生死锁等运行时错误。

InQuIR 的语义采用从前向后按顺序发出指令的策略，这与经典计算机基本相同。然而，在许多研究中，指令会在其数据依赖得到解决后立即发出；InQuIR 也采用这一策略来评估 InQuIR 程序的计算成本。后一种策略的优点是，每条指令都能尽早发出。与此同时，由于需要维护依赖图，每个执行周期的运行时成本可能会更高。目前很难说哪种方法更好，因为每个执行周期可用的时间取决于多种因素，包括未来量子计算机的架构、指令集和计算范式。

目前存在的量子器件属于含噪中等规模量子（NISQ）器件，其噪声影响不可忽视。在此类系统中使用 InQuIR 时，必须根据器件的各种约束，将其编译为高质量的可执行量子电路。特别是，为了降低量子门执行期间的噪声，编译电路时必须考虑处理器内部的量子比特连接性，使量子门数量尽可能少 [77]。此外，每个量子比特上的量子门精度并不一致，因此需要谨慎进行量子比特分配 [78], [79]。

为了在未来实现大规模量子计算，人们认为有必要采用量子纠错码进行容错量子计算（FTQC） [80] 。InQuIR 的基本结构和分析方法具有足够的通用性，可用于设计面向分布式量子系统的 FTQC 语言。然而，代码结构、指令集、微体系结构和编译策略仍需妥善制定 [81]。

InQuIR 支持无远程内存访问模型（NORA）架构的量子对应形式，且不假定存在共享内存。不过，假设有人定义了能够处理非均匀访问内存模型（NUMA）架构量子对应形式的指令集 [82],，其中包括访问共享内存的能力。在这种情况下，InQuIR 可用作此类程序的编译目标。

IX. 结论

我们提出了 InQuIR，这是一种中间表示，用于描述在多个量子处理器上进行的分布式量子计算，并增强分布式量子程序的可重用性。我们将 InQuIR 设计为一种形式化定义的编程语言，以便准确讨论分布式量子程序的行为。我们给出了几个用 InQuIR 编写的示例，展示远程操作以及死锁等不期望出现的运行时状态的实现。我们讨论了一种类型系统，用于在执行前检测此类运行时错误。此外，我们还实现了软件工具，用于评估使用 InQuIR 编写的程序的性能。

致谢

SN 和 RW 感谢 Atsushi Igarashi 和 Kae Nemoto 在整个项目期间提供的宝贵讨论。本研究部分获得 JSPS KAKENHI JP22J20882 号资助，以及 JST“面向科学技术创新的大学 fellowship 建设”项目 JPMJFS2123 号资助。

参考文献

[1] A. Adedoyin, J. Ambrosiano, P. Anisimov, A. Bärtschi, W. Casper, G. Chennupati, C. Coffrin, H. Djidjev, D. Gunter, S. Karra, 等，“面向初学者的量子算法实现”，*arXiv* 预印本 *arXiv:1804.03719*，2018年。[2] J. Chiaverini, D. Leibfried, T. Schaetz, M. D. Barrett, R. Blakestad, J. Britton, W. M. Itano, J. D. Jost, E. Knill, C. Langer, 等，“量子纠错的实现”，《自然》，第432卷，605，2004 7017，602 页 – 。[3] S. Krinner, S. Storz, P. Kurpiers, P. Magnard, J. Heinsoo, R. Keller, J. Luetolf, C. Eichler, 和 A. Wallraff，“面向百量子比特规模超导电路系统的低温装置工程设计”，*EPJ Quantum Technology*，第6卷，第1期，第2页，2019年。[4] A. Gold, J. Paquette, A. Stockklauser, M. J. Reagor, M. S. Alam, A. Bestwick, N. Didier, A. Nersisyan, F. Oruc, A. Razavi, 等，“模块化超导量子比特器件中分离硅芯片之间的纠缠”，*npj Quantum Information*，第7卷，第1期，第1–10页，2021年。[5] S. Tamate, Y. Tabuchi, 和 Y. Nakamura，“迈向大规模超导量子计算机可扩展封装与布线的实现”，*IEICE Transactions on Electronics*，2021年。[6] D. Awschalom, K. K. Berggren, H. Bernien, S. Bhave, L. D. Carr, P. Davids, S. E. Economou, D. Englund, A. Faraon, M. Fejer, S. Guha, M. V. Gustafsson, E. Hu, L. Jiang, J. Kim, B. Korzh, P. Kumar, P. G. Kwiat, M. Lonˇcar, M. D. Lukin, D. A. Miller, C. Monroe, S. W. Nam, P. Narang, J. S. Orcutt, M. G. Raymer, A. H. Safavi- Naeini, M. Spiropulu, K. Srinivasan, S. Sun, J. Vuˇckovi´c, E. Waks, R. Walsworth, A. M. Weiner, 和 Z. Zhang，“面向下一代信息技术的量子互连（quics）开发”，*PRX Quantum*，第2卷，第1–21页，2021年。[7] M. Ben-Or 和 A. Hassidim，“快速量子拜占庭协议”，载于第三十七届 *ACM* 理论计算年度研讨会论文集，第481–485页，2005年。[8] S. Barz, E. Kashefi, A. Broadbent, J. F. Fitzsimons, A. Zeilinger, 和 P. Walther，“盲量子计算的演示”，科学，第335卷，第6066期，第303–308页，2012年。[9] T. Häner, D. S. Steiger, T. Hoefler, 和 M. Troyer，“使用 QMPI 进行分布式量子计算”，国际高性能计算、网络、存储与分析会议论文集，第1–13页，2021年11月。[10] D. Cuomo, M. Caleffi, K. Krsulich, F. Tramonto, G. Agliardi, E. Prati, 和 A. S. Cacciapuoti，“面向分布式量子计算的优化编译器”，*arXiv:2112.14139 [量子物理]*,2021年12月。[11] D. Ferrari, A. S. Cacciapuoti, M. Amoretti, 和 M. Caleffi，“分布式量子计算的编译器设计”，*IEEE*量子工程汇刊，第2卷，第1–20页，2021年。[12] R. V. Meter, W. Munro, K. Nemoto, 和 K. M. Itoh，“分布式内存量子多计算机上的算术运算”，*ACM Journal on Emerging Technologies in Computing Systems (JETC)*，第3卷，第4期，第1–23页，2008年。[13] A. McCaskey 和 T. Nguyen，“用于量子汇编语言的 MLIR 方言”，*arXiv:2101.11365 [量子物理]*, 2021年1月。

[14] D. R. N. 实验室，“Frontier - HPE Cray EX235A、AMD 优化第三代 EPYC 64C 2GHz、AMD Instinct MI250X、Slingshot-11、HPE”，技术报告，美国。https://www.olcf.ornl.gov/frontier/。 [15] J. Dongarra，“ Fujitsu Fugaku 系统报告”，田纳西大学诺克斯维尔创新计算实验室，技术报告 ICLUT-20-06，2020年。 [16] EuroHPC/CSC，“Lumi 超级计算机”，技术报告，芬兰。https://www.lumi-supercomputer.eu/lumis-fullsystem-architecture-revealed/。 [17] D. Greenberger, M. Horne, 和 A. Zeilinger，“超越贝尔定理：贝尔定理、量子理论与宇宙观，编者 M. Kafatos”，多德雷赫特：Kluwer，第69卷，第69–72页，1989年。 [18] R. Raussendorf 和 H. J. Briegel，“单向量子计算机”，Physical Review Letters，第86卷，第22期，第5188页，2001年。 [19] R. Raussendorf, D. E. Browne, 和 H. J. Briegel，“簇态上的基于测量的量子计算”，Physical Review A，第68卷，第2期，第022312页，2003年。 [20] M. Zomorodi-Moghadam, M. Houshmand, 和 M. Houshmand，“ 优化分布式量子电路中的量子隐形传态成本”，International Journal of Theoretical Physics，第57卷，第3期，第848–861页，2018年。 [21] O. Daei, K. Navi, 和 M. Zomorodi，“基于门交换降低分布式量子电路中的量子隐形传态成本”，International Journal of Theoretical Physics，第60卷，第9期，第3494–3513页，2021年。 [22] E. Nikahd, N. Mohammadzadeh, M. Sedighi, 和 M. S. Zamani，“量子电路的自动化基于窗口的划分”，Physica Scripta，第96卷，第3期，第035102页，2021年。 [23] D. Dadkhah, M. Zomorodi, 和 S. E. Hosseini，“一种优化分布式量子电路的新方法”，International Journal of Theoretical Physics，第60卷，第9期，第3271–3285页，2021年。 [24] R. G Sundaram, H. Gupta, 和 C. Ramakrishnan，“量子电路的高效分布”，载于第35届分布式计算国际研讨会（DISC 2021），Schloss Dagstuhl-Leibniz-Zentrum für Informatik，2021年。 [25] K. Hammerer, A. S. Sørensen, 和 E. S. Polzik，“光与原子系综之间的量子接口”，Reviews of Modern Physics，第82卷，第2期，第1041页，2010年。 [26] R. W. Andrews, R. W. Peterson, T. P. Purdy, K. Cicak, R. W. Simmonds, C. A. Regal, 和 K. W. Lehnert，“微波与光之间的双向高效转换”，Nature physics，第10卷，第4期，第321–326页，2014年。 [27] M. Mirhosseini, A. Sipahigil, M. Kalaee, 和 O. Painter，“ 超导量子比特到光子的转换”，《自然》，第588卷，第7839期，第599–603页，2020年。 [28] B. M. Brubaker, J. M. Kindem, M. D. Urmey, S. Mittal, R. D. Delaney, P. S. Burns, M. R. Vissers, K. W. Lehnert, 和 C. A. Regal，“连续且高效的电光转换器中的光机械基态冷却”，Physical Review X，第12卷，021062，2022 2，第 p. 。 [29] L. Fan, C.-L. Zou, R. Cheng, X. Guo, X. Han, Z. Gong, S. Wang, 和 H. X. Tang，“超导腔电光学：实现超导电路与光子电路之间相干光子转换的平台”，Science advances，第4卷，第8期，第eaar4994页，2018年。 [30] R. Sahu, W. Hease, A. Rueda, G. Arnold, L. Qiu, 和 J. M. Fink，“量子增强型微波-光接口的运行”，Nature communications，第13卷，第1期，第1–7页，2022年。 [31] X. Fernandez-Gonzalvo, S. P. Horvath, Y.-H. Chen, 和 J. J. Longdell，“Er 中的腔增强拉曼异频光谱 3+: Y 2 sio 5，用于微波到光信号转换”，Physical Review A，第100卷，第3期，第033807页，2019年。 [32] T. Vogt, C. Gross, J. Han, S. B. Pal, M. Lam, M. Kiffner, 和 W. Li，“利用里德伯原子实现高效微波到光转换”，Physical Review A，第99卷，第2期，第023832页，2019年。 [33] H. J. Kimble，“腔量子电动力学中单原子与光子的强相互作用”，Physica Scripta，第1998卷，T76期，第127页，1998年。 [34] S. Girvin, M. Devoret, 和 R. Schoelkopf，“电路量子电动力学与基于电荷的超导量子比特工程”，Physica Scripta，第2009卷，T137期，第014012页，2009年。 [35] N. Gisin 和 R. Thew，“ 量子通信 ”，Nature photonics，第 1 卷，第 3期，第165–171页，2007年。 [36] H. J. Kimble，“量子互联网”，《自然》，第453卷，第7198期，第1023–1030页，2008年。 [37] D. Gottesman, A. Kitaev, 和 J. Preskill，“在振荡器中编码量子比特”，Physical Review A，第64卷，第1期，第012310页，2001年。 [38] M. H. Michael, M. Silveri, R. Brierley, V. V. Albert, J. Salmilehto, L. Jiang, 和 S. M. Girvin，“新型量子纠错玻色模式的编码”，《Physical Review X》，第6卷，031006，2016年3，第。 [39] M. Grassl、W. Geiselmann 和 T. Beth，“量子里德–所罗门码”，载于应用代数、代数算法与纠错码国际研讨会，页码231–244，施普林格，1999年。 [40] H.-J. Briegel、W. Dür、J. I. Cirac 和 P. Zoller，“量子中继器：不完美局域操作在量子通信中的作用”，Physical Review Letters，第81卷，第26期，页码5932，1998年。 [41] R. Van Meter，《量子组网》。约翰·威利父子公司，2014年。 [42] S. Wehner、D. Elkouss 和 R. Hanson，“量子互联网：未来发展愿景”，《科学》，第362卷，第6412期，页码eaam9288，2018年。 [43] W. Kozlowski、A. Dahlberg 和 S. Wehner，“量子网络协议的设计”，载于第16届新兴网络实验与技术国际会议论文集，页码1–16，2020年。 [44] C. H. Bennett、G. Brassard、C. Crépeau、R. Jozsa、A. Peres 和 W. K. Wootters，“通过经典信道和爱因斯坦–波多尔斯基–罗森信道传送未知量子态”，Physical review letters，第70卷，第13期，页码1895，1993年。 [45] D. Gottesman 和 I. L. Chuang，“量子隐形传态是通用计算原语”，《自然》，第402卷，页码390–393，1999年11月。 [46] L. Jiang、J. M. Taylor、A. S. Sørensen 和 M. D. Lukin，“基于小型量子寄存器的分布式量子计算”，Physical Review A，第76卷，第6期，页码062323，2007年。 [47] M. Zukowski、A. Zeilinger、M. A. Horne 和 A. K. Ekert，“通过纠缠交换进行‘事件就绪探测器’贝尔实验”，Physical Review Letters，第71卷，第26期，1993年。 [48] A. Dahlberg、B. van der Vecht、C. D. Donne、M. Skrzypczyk、I. te Raa、W. Kozlowski 和 S. Wehner，“NetQASM——量子互联网中量子–经典混合程序的低级指令集架构”，arXiv:2111.*09823 [*量子物理*]*, 2021年11月。 [49] A. W. Cross、L. S. Bishop、J. A. Smolin 和 J. M. Gambetta，“开放量子汇编语言”，arXiv预印本 arXiv:1707.03429，2017年。 [50] S. Gay 和 R. Nagarajan，《通信量子过程》，2004年9月。 [51] R. Milner、J. Parrow 和 D. Walker，“移动进程演算，I”，《信息与计算》，第100卷，页码1–40，1992年9月。 [52] S. Gay、R. Nagarajan 和 N. Papanikolaou，“量子协议的概率模型检验”，2005年10月。 [53] R. Nagarajan、N. Papanikolaou、G. Bowen 和 S. Gay，“量子密钥分发安全性的自动化分析”，2005年2月。 [54] M. Hennessy 和 J. Riely，“移动代理系统中的资源访问控制”，《信息与计算》，第173卷，页码82–120，2002年2月。 [55] A. Dahlberg 和 S. Wehner，“ Simulaqron——用于开发量子互联网软件的模拟器”，Quantum Science and Technology，第4卷，第1期，页码015001，2018年。 [56] B. Bartlett，“用于量子网络和信道的分布式仿真框架”，arXiv预印本 arXiv:1808.07047，2018年。 [57] R. Satoh、M. Hajdušek、N. Benchasattabuse、S. Nagayama、K. Teramoto、T. Matsuo、S. A. Metwalli、T. Satoh、S. Suzuki 和 R. Van Me-ter，“Quisp：量子互联网仿真套件”，arXiv预印本 arXiv:2112.07093，2021年。 [58] T. Coopmans、R. Knegjens、A. Dahlberg、D. Maier、L. Nijsten、J. de Oliveira Filho、M. Papendrecht、J. Rabbie、F. Rozp.edek、M. Skrzypczyk 等，“ NetSquid：一种使用离散事件模拟量子信息的网络模拟器”，《通信物理》，第4卷，第1期，页码1–15，2021年。 [59] X. Wu、A. Kolar、J. Chung、D. Jin、T. Zhong、R. Kettimuthu 和 M. Suchara，“Sequence：可定制的量子网络离散事件模拟器”，Quantum Science and Technology，第6卷，第4期，页码045027，2021年。 [60] S. DiAdamo、J. Nötzel、B. Zanger 和 M. M. Be.se，“Qunetsim：量子网络软件框架 ”，IEEE 量子工程汇刊，第2卷，页码1–12，2021年。 [61] V. Danos、E. D’Hondt、E. Kashefi 和 P. Panangaden，“分布式测量型量子计算”，2005年6月。 [62] M. P. I. Forum，“MPI：消息传递接口标准”，技术报告，美国，1994年。 [63] B. Bichsel、M. Baader、T. Gehr 和 M. Vechev，“Silq：具有安全逆计算和直观语义的高级量子语言”，载于第41届ACM SIGPLAN编程语言设计与实现大会论文集，PLDI 2020，（美国纽约）

，第 286 页–300，计算机协会 ，2020年6月。

[64] Y. Sung、L. Ding、J. Braumüller、A. Vepsäläinen、B. Kannan、M. Kjaer- gaard、A. Greene、G. O. Samach、C. McNally、D. Kim、A. Melville、B. M. Niedzielski、M. E. Schwartz、J. L. Yoder、T. P. Orlando、S. Gus- tavsson 和 W. D. Oliver，“实现高保真度 CZ 和无 ZZ 的 iSWAP 门”，*Physical ReviewX*，第 11 卷，页码 021058，2021年6月。[65] Y. Sunada、S. Kono、J. Ilves、S. Tamate、T. Sugiyama、Y. Tabuchi 和 Y. Nakamura，“通过内置 Purcell 滤波器实现与谐振器耦合的超导量子比特的快速读出与重置”，*Physical Review Applied*，第 17 卷，页码 044016，2022年4月。[66] S. Barz、G. Cronenberg、A. Zeilinger 和 P. Walther，“纠缠光子对的预示生成”，*Nature photonics*，第 4 卷，第 8 期，第 553–556 页，2010年。[67] J. Ang、G. Carini、Y. Chen、I. Chuang、M. A. DeMarco、S. E. Economou、A. Eickbusch、A. Faraon、K.-M. Fu、S. M. Girvin、M. Hatridge、A. Houck、P. Hilaire、K. Krsulich、A. Li、C. Liu、Y. Liu、M. Martonosi、D. C. McKay、J. Misewich、M. Ritter、R. J. Schoelkopf、S. A. Stein、S. Sussman、H. X. Tang、W. Tang、T. Tomesh、N. M. Tubman、C. Wang、N. Wiebe、Y.-X. Yao、D. C. Yost 和 Y. Zhou，“ 多节点超导量子计算机的架构 ”，2022 年 12 月。[68] A. Zulehner、A. Paler 和 R. Wille，“将量子电路映射到 IBM QX 架构的高效方法”，2018年6月。[69] A. Li、S. Stein、S. Krishnamoorthy 和 J. Ang，“QASMBench：用于 NISQ 评估与模拟的低级 QASM 基准测试套件”，2022年5月。[70] R. Wille、D. Große、L. Teuber、G. W. Dueck 和 R. Drechsler，“RevLib：可逆函数和可逆电路的在线资源”，载于国际多值逻辑研讨会，第 220–225 页，2008年。[71] D. N. Turner、P. Wadler 和 C. Mossin，“从前有一种类型”，载于第七届函数式编程语言与计算机体系结构国际会议论文集，FPCA '95，（美国纽约州纽约市），第 1–11 页，计算机协会，1995年10月。[72] K. Honda、N. Yoshida 和 M. Carbone，“多方异步会话类型”，*ACM SIGPLAN Notices*，第 43 卷，第 273–284 页，2008年1月。[73] HondaKohei、YoshidaNobuko 和 CarboneMarco，“多方异步会话类型”，*Journal of the ACM (JACM)*，2016年3月。[74] C. H. Bennett、G. Brassard、S. Popescu、B. Schumacher、J. A. Smolin 和 W. K. Wootters，“通过噪声信道纯化噪声纠缠并实现保真隐形传态”，*Physical Review Letters*，第 76 卷，第 722–725 页，1996年1月。[75] C. H. Bennett、D. P. DiVincenzo、J. A. Smolin 和 W. K. Wootters，“混合态纠缠与量子纠错”，*Physical Review A*，第 54 卷，第 5 期，页码 3824，1996年。[76] C. H. Bennett、H. J. Bernstein、S. Popescu 和 B. Schumacher，“通过局域操作浓缩部分纠缠”，*Physical Review A*，第 53 卷，第 4 期，页码 2046，1996年。[77] A. Holmes、S. Johri、G. G. Guerreschi、J. S. Clarke 和 A. Y. Matsuura，“量子比特连接性对量子算法性能的影响”，*Quantum Science and Technology*，第 5 卷，第 2 期，页码 025009，2020年。[78] P. Murali、J. M. Baker、A. Javadi-Abhari、F. T. Chong 和 M. Martonosi，“面向含噪中等规模量子计算机的噪声自适应编译器映射”，载于第二十四届编程语言与操作系统的体系结构支持国际会议论文集，第 1015–1029 页，2019年。[79] S. Nishio、Y. Pan、T. Satoh、H. Amano 和 R. V. Meter，“利用误差感知编译从 IBM 的20量子比特机器中提取成功”，*ACM Journal on Emerging Technologies in Computing Systems (JETC)*，第 16 卷，第 3 期，第 1–25 页，2020年。[80] D. Gottesman，稳定子码与量子纠错。加州理工学院，1997年。[81] Y. Li 和 S. C. Benjamin，“适用于任意大小模块的网络量子计算分层表面码”，*Physical Review A*，第 94 卷，第 4 期，页码 042303，2016年。[82] C. Lameter，“NUMA（非一致性内存访问）概述：由于内存控制器靠近微处理器上的执行单元，NUMA 越来越普遍。”，*Queue*，第 11 卷，51，2013 7，40第。页码 –。