计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2021.20210368
58(9): 1875-1896, 2021

一种面向含噪中尺度量子技术的量子-经典异构计算系统

付 祥[1] 郑宇真[1] 苏 醒[2] 于锦涛[3] 徐炜遐[1] 吴俊杰[1]
[1](国防科技大学计算机学院量子信息研究所兼高性能计算国家重点实验室 长沙 410073)
[2](国防科技大学计算机学院 长沙 410073)
[3](数学工程与先进计算国家重点实验室 郑州 450001)
(xiangfu@quanta.org.cn)

A Heterogeneous Quantum-Classical Computing System Targeting Noisy Intermediate-Scale Quantum Technology

Fu Xiang[1], Zheng Yuzhen[1], Su Xing[2], Yu Jintao[3], Xu Weixia[1], and Wu Junjie[1]
[1](*Institute for Quantum Information & State Key Laboratory of High Performance Computing, College of Computer Science and Technology, National University of Defense Technology, Changsha* 410073)
[2](*College of Computer Science and Technology, National University of Defense Technology, Changsha* 410073)
[3](*State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou* 450001)

Abstract Quantum computers promise to accelerate solving problems that are intractable by classical computers, such as prime factorization and quantum chemistry simulation. It has been demonstrated that a single quantum system can integrate more than fifty noisy solid-state qubits and surpass contemporary classical computers in specific computing tasks, marking the arrival of the noisy intermediate-scale quantum (NISQ) era. As more and more qubits can be integrated into a single system, how to integrate qubits with control hardware, software development environment, and classical computing resources to obtain a complete and usable quantum computing system is a problem that needs to be further clarified. By comparing both the control and execution of quantum and classical computing, this paper proposes a heterogeneous quantum-classical system targeting the NISQ technology. Taking a typical NISQ algorithm (the iterative phase estimation algorithm) as an example, this paper introduces the whole process of executing a quantum algorithm and related software and hardware, including the high-level programming language, compiler, quantum software and hardware interface, and control microarchitecture. On top of it, this paper discusses the challenges confronting each layer in the NISQ era. This paper aims to provide a general introduction of quantum computing systems to readers (especially beginners of quantum computing) from an engineering perspective, hoping to promote people's understanding of the overall architecture of quantum computing systems in the NISQ era and stimulate more related research.

Key words quantum computing; quantum programming language; quantum compilation; quantum computer architecture; noisy intermediate-scale quantum (NISQ)

收稿日期:2021-04-08;**修回日期**:2021-06-24

基金项目:国家自然科学基金项目(61902410);高性能计算国家重点实验自主课题(202001-01,202101-24)
This work was supported by the National Natural Science Foundation of China (61902410) and the Autonomous Project of the State Key Laboratory of High Performance Computing (202001-01, 202101-24).

通信作者:吴俊杰(junjiewu@nudt.edu.cn)

摘 要 量子计算有望加速解决经典计算难以解决的问题,如质因子分解、量子化学模拟等.已有单个量子系统可集成大于50个含噪声的固态量子比特,并在特定的计算任务上超越了经典计算机,标志含噪中尺度量子(noisy intermediate-scale quantum, NISQ)计算时代的到来.随着人们可在单个系统中集成越来越多的量子比特,如何将量子比特与控制硬件、软件开发环境、经典计算资源集成得到完整可用的量子计算系统,是一个有待进一步明确的问题.对比了量子计算与经典计算在控制及执行上的异同,并在此基础上提出了面向NISQ时代的量子-经典异构系统.以一个典型的NISQ算法(迭代相位估计算法)为例,介绍了量子算法从软件描述到硬件执行的整体流程,及与该过程相关的高级程序设计语言、编译器、量子软硬件接口和硬件等.在此基础上,讨论了流程中各个层次在NISQ时代面临的挑战.旨在从工程实现的视角,从宏观层面为读者(尤其是量子计算初学者)介绍量子计算系统,希望可以促进人们对NISQ时代下量子计算系统整体结构的理解,并激发更多相关研究.

关键词 量子计算;量子程序设计语言;量子编译;量子计算体系结构;含噪中尺度量子

中图法分类号 TP38

2019年谷歌通过利用53个超导量子比特执行随机量子线路采样任务[1],2020年中国科学技术大学通过利用76光子执行玻色采样任务[2],展示出量子计算机在某些特定问题上超越经典计算机的能力,实现了量子优越性[3-4],标志着含噪中尺度量子(noisy intermediate-scale quantum, NISQ)时代[5]的到来.NISQ技术指量子系统中可集成50至数百个含噪声量子比特的技术.NISQ时代下,量子比特的相干时间非常短暂,且量子操作的错误率仍然可观.以超导量子比特为例,已实证的适合大规模集成的超导量子比特相干时间仅有数十至数百微秒[1,6-10];虽然单量子比特门的错误率可低于1%,但两量子比特门和量子测量的错误率尚难以在每个量子比特(对)上均低于1%[1,11].受限于量子比特的数量、相干时间和量子操作的错误率,NISQ技术无法通过量子容错技术[12]支持长时间的量子状态演化来求解实际的问题.

为了加速量子计算的落地,一种思路是在比量子比特尚未退相干之前完成量子算法所需的量子状态演化,从而使量子噪声或错误不至于积累到完全破坏计算结果.已有工作提出了一系列有望在近期得到实证的量子算法,如变分量子本征值求解(variational quantum eigensolver, VQE)[13]、迭代相位估计(iterative phase estimation, IPE)[14-16]、量子近似优化算法(quantum approximate optimization algorithm, QAOA)[17]、变分量子模拟(variational quantum simulation, VQS)[18]等.这些算法运行过程中,量子比特每次从初始化到最终测量所需的时间相对较短,降低了对量子比特相干时间的要求,为利用NISQ技术解决实际问题提供了可能.

这些算法的关键共同点是深度结合了量子计算与经典计算.例如,VQE算法使用量子处理器测量参数化量子态$|\psi(\theta_k)\rangle$在目标哈密顿量$\mathcal{H}$下的能量E_k,使用经典处理器运行优化算法,根据E_k搜索下一组参数θ_{k+1},并通过二者间的迭代逼近$\mathcal{H}$的基态$|\psi_g\rangle$,从而估计$\mathcal{H}$的基态能量E_g.另一个例子是IPE算法.IPE算法执行过程中,量子测量的结果会被用于实时计算数百纳秒后的一个旋转门的角度[14,16].如果说VQE算法仅需将量子计算与离线的经典计算结合起来(经典计算发生时,量子状态不需要得到维持),那么IPE算法则需要实时(或在线)地结合量子计算和经典计算(经典计算发生于量子状态演化过程中).量子-经典异构计算是NISQ时代量子应用的第1个特点.

NISQ时代量子应用的第2个特点是在应用层需直接控制量子计算硬件的部分执行细节.一方面,由于NISQ量子比特天然含噪的特性,用于校准量子比特和量子操作相关参数的量子实验会定期反复地进行.这些实验需直接控制施加在量子比特上的模拟波形,或精确控制操作发生的时序.另一方面,由于NISQ时代下的量子计算机性能有限,特定的量子计算机用户会希望直接控制量子操作的波形和时序,从而面向具体硬件优化量子程序达到最优性能.实现量子-经典异构计算并支持用户在应用层控制量子计算硬件的部分执行细节,需要程序设计语言、编译器、体系结构等多个层次的支持及各层次之间的配合.

明确量子计算机的系统结构是指导量子软硬件设计并将其集成为一个系统的必要条件.基于线路模型,之前的工作提出了若干量子计算机系统结构

设计构想,讨论量子计算从软件到硬件的层次划分及层次间的协同方式[19-23].但这些层次结构主要面向大规模容错量子计算机,强调通过层层抽象及逐级优化,使量子算法以容错的方式在硬件上执行.这些设计构想中,有些虽然提到了量子计算与经典计算的结合,但并未具体地讨论量子计算资源和经典计算资源的组织方式.同时,这些构想中不考虑或较少考虑如何将与波形和时序等硬件相关的低级控制能力以一种可编程的方式暴露给顶层软件.因此,现有的量子计算系统结构设计构想在指导 NISQ 系统的实现上有所不足.本文比较量子计算与经典计算的异同,并以工程化视角回顾量子计算系统的层次结构.面向 NISQ 时代的量子-经典异构计算,本文以青果框架[24]为例概述量子计算软硬件系统的组织方式,并分别介绍高级量子程序设计语言、量子编译器、量子软硬件接口、量子控制微体系结构等层次的主要任务.最后,我们简要探讨程序设计语言、编译和控制体系结构等方面在 NISQ 技术条件下面临的挑战.

1　背　　景

本节介绍量子计算的基本数学原理.在此基础上,以 IPE 为例介绍量子算法的基本构成并以超导量子比特为例介绍量子比特的物理实现及其控制.

1.1　量子计算基础

比特是经典计算机的基本单元,一个比特在确定的时刻只能处于 2 种互斥的基本状态(0,1)中的一种.量子比特(quantum bit, qubit)是量子计算机的基本单元.量子比特有 2 种基本状态$|0\rangle$和$|1\rangle$.与经典比特不同,量子比特能够同时处于$|0\rangle$和$|1\rangle$的叠加态中,表示为$|\psi\rangle=\alpha|0\rangle+\beta|1\rangle$,或者一个长度为 2 的向量$(\alpha,\beta)^{\mathrm{T}}$,其中$\alpha,\beta\in\mathbb{C}$且满足归一化条件$|\alpha|^2+|\beta|^2=1$.这种现象称之为量子叠加.在归一化条件下,$|\psi\rangle$可被重写为$\mathrm{e}^{\mathrm{i}\gamma}\left(\cos\frac{\theta}{2}|0\rangle+\mathrm{e}^{\mathrm{i}\varphi}\sin\frac{\theta}{2}|1\rangle\right)$.由于在物理上没有可观测的效果,全局相位$\mathrm{e}^{\mathrm{i}\gamma}$可被忽略.这样,单个量子比特的状态可形象化地表示为 Bloch 球面上的一个点,如图 1 所示.

量子计算中,n 个量子比特可处于2^n个基本状态的叠加中,即$|\psi\rangle=\sum_{i=0}^{2^n-1}c_i|i\rangle$,其中$c_i\in\mathbb{C}$且$\sum_{i=0}^{2^n-1}|c_i|^2=1$.换言之,$n$ 个量子比特可同时存储2^n个数据,相较于 n 个经典比特只能存储一份确定的 n 比特数据,量子系统的存储空间维度随量子比特数目 n 的增加而指数增长,这是实现量子计算优越性的第 1 个理论基础.注意,量子比特可以纠缠在一起,此时,状态$|\psi\rangle=\sum_{i=0}^{2^n-1}c_i|i\rangle$无法分解为独立的量子比特状态的积的形式.

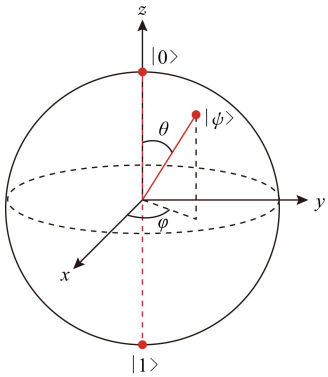


Fig. 1　The Bloch sphere that represents a single qubit state

图 1　可表示单个量子比特状态的 Bloch 球面

经典计算中,与、或、非等逻辑门是完备的,可实现任意经典计算逻辑.量子操作包括量子门和量子测量,可用于变换量子比特的状态.作用在 n 个量子比特上的量子门的效果可使用$2^n\times2^n$的酉矩阵来描述.酉矩阵是指其逆矩阵为其共轭转置的矩阵,即$\boldsymbol{U}\boldsymbol{U}^{\dagger}=\boldsymbol{I}$.因为在硬件实现上相对容易,单量子比特门与两量子比特门是算法设计中最常用的量子门.单量子比特门是 Bloch 球面上的一个旋转操作$\boldsymbol{R}_{\hat{\boldsymbol{n}}}(\theta)$,可表达为 2×2 的酉矩阵:

$$\boldsymbol{R}_{\hat{\boldsymbol{n}}}(\theta)=\cos\left(\frac{\theta}{2}\right)\boldsymbol{I}-\mathrm{i}\sin\left(\frac{\theta}{2}\right)(n_x\boldsymbol{X}+n_y\boldsymbol{Y}+n_z\boldsymbol{Z}),\tag{1}$$

其中$\hat{\boldsymbol{n}}=(n_x,n_y,n_z)(|\hat{\boldsymbol{n}}|^2=1)$是旋转轴,$\theta$ 是旋转角度,而

$$\boldsymbol{I}=\begin{pmatrix}1&0\\0&1\end{pmatrix},\ \boldsymbol{X}=\begin{pmatrix}0&1\\1&0\end{pmatrix},$$

$$\boldsymbol{Y}=\begin{pmatrix}0&-\mathrm{i}\\\mathrm{i}&0\end{pmatrix},\ \boldsymbol{Z}=\begin{pmatrix}1&0\\0&-1\end{pmatrix}$$

是标准泡利(Pauli)门.例如,$\boldsymbol{R}_x(\pi)$,$\boldsymbol{R}_y(\pi)$和$\boldsymbol{R}_z(\pi)$表示绕 x,y 和 z 轴旋转 π 角度的操作.而 Hardamard 门$\boldsymbol{H}=\frac{\sqrt{2}}{2}\begin{pmatrix}1&1\\1&-1\end{pmatrix}$则是绕$(\sqrt{2},\sqrt{2},0)/2$轴旋转 π 角度.2 个最常见的两量子比特门是受控非门

(Controlled-NOT, CNOT)和受控相位门(Controlled-Phase, CPhase),当受控相位门不指定相位时,默认为 π,即等价于 CZ 门.它们对应的矩阵分别为

$$\boldsymbol{CNOT}\equiv\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{pmatrix},\ \boldsymbol{CZ}\equiv\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&-1\end{pmatrix}.$$

当量子比特门作用在多个量子比特系统中的少数量子比特上时,其矩阵可使用该门对应的矩阵与单位矩阵的张量积来描述.如将单量子比特门 $\boldsymbol{U}_s$ 作用在四量子比特系统中的第 3 个量子比特上时,描述其整体效果的矩阵为 $\boldsymbol{U}=\boldsymbol{I}\otimes\boldsymbol{I}\otimes\boldsymbol{U}_s\otimes\boldsymbol{I}$,其中$\otimes$表示克罗内克(Kronecker)积.初始状态为$|\psi_i\rangle$的量子系统在量子门 $\boldsymbol{U}$ 的作用下将转变为状态

$$|\psi_o\rangle=\boldsymbol{U}|\psi_i\rangle. \tag{2}$$

对量子比特进行测量时,会使得整个系统的量子状态发生坍缩.以测量 n 量子比特系统中的第 1 个量子比特为例,若测量前的状态为

$$|\psi_{\text{before}}\rangle=\Big(\sum_{j=0}^{2^{n-1}-1}c_{j,0}|j\rangle\Big)|0\rangle+\Big(\sum_{j=0}^{2^{n-1}-1}c_{j,1}|j\rangle\Big)|1\rangle, \tag{3}$$

则系统测量之后会以 $p_b=\sum_{j=0}^{2^{n-1}-1}|c_{j,b}|^2$ 的概率坍缩为状态

$$|\psi_{\text{after}}\rangle=\frac{1}{\sqrt{p_b}}\sum_{j=0}^{2^{n-1}-1}c_{j,b}|j\rangle|b\rangle, \tag{4}$$

其中 $j\in\{0,1,\cdots,2^{n-1}-1\}$且 $b\in\{0,1\}$.

对比$|\psi_{\text{before}}\rangle$和$|\psi_{\text{after}}\rangle$可知,对具有 n 个量子比特的量子系统中的少数量子比特施加门操作或进行测量时,可使得表征此量子系统的 2^n 个系数同时发生变化,展现出极高的并行性,这是实现量子计算优越性的第 2 个理论基础.

量子系统中巨大的存储空间,结合量子操作内秉的高度并行性,可使得量子计算机在解决特定问题时比经典计算机更为高效.

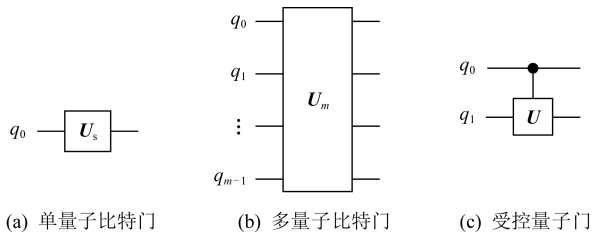
1.2 通用量子比特门

任意经典函数可通过一定的方法(如卡诺映射[25])被分解为 $S_{\text{UC}}=${与,或,非}3 个门.因此我们说 S_{UC} 是经典通用计算门.与此类似,如果一组量子门可通过组合的方式,以任意精度近似到任何 m 量子比特操作 $\boldsymbol{U}_m$,其中 $m\geqslant1$,则这组量子门被称为通用的[26].对于如 1.5 节所述的超导量子比特而言,一个常用的通用量子门集合是 $S_{\text{UQ}}=\{\boldsymbol{R}_x,\boldsymbol{R}_y,\boldsymbol{CZ}\}$,其中 $\boldsymbol{R}_x$ 和 $\boldsymbol{R}_y$ 分别代表绕 x 轴和 y 轴的任意角度的旋转.例如,CNOT 门可以使用 S_{UQ}构造,如式(5)所示:

$$\boldsymbol{CNOT}_{c,t}=\boldsymbol{R}_y(\pi/2)_t\cdot\boldsymbol{CZ}\cdot\boldsymbol{R}_y(-\pi/2)_t. \tag{5}$$

1.3 量子线路

类似于经典电路可描述硬件对经典信息的处理过程,一个量子线路可用于描述量子操作作用在量子比特上的过程.在量子线路中,每个量子比特用一条水平线来表示,施加在该量子比特上的操作用一个块来表示.例如,单个量子比特门 $\boldsymbol{U}_s$ 和多量子比特操作 $\boldsymbol{U}_m$ 可以用图 2(a)和图 2(b)所示的量子线路来表示.连接黑色圆点的方块表示受控门,如图 2(c)所示.



(a) 单量子比特门 (b) 多量量子比特门 (c) 受控量子门

Fig. 2 Quantum circuit

图 2 量子线路

量子线路中,率先作用在量子比特上的量子门被放置在随后作用的量子门的左边.图 3 描述了如式(5)所示的 CNOT 门分解.右侧量子线路利用 2 个单量子比特门和一个 CZ 门实现了 CNOT 的分解.由于 $\boldsymbol{R}_y(-\pi/2)$最先作用在受控量子比特 t 上,所以它出现在分解后的量子线路中的最左侧.图 3 左图中,黑点连接十字圈表示作用在 2 个量子比特上的 CNOT 门;右图中,2 个连接的黑点表示 CZ 门.需要注意的是,数学表达式中率先作用在量子比特上的量子门出现在随后作用的量子门的右侧(式(5)),这与量子线路中量子门出现的顺序是相反的,如图 3 所示:

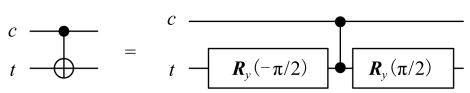


Fig. 3 A quantum circuit example of CNOT gate decomposition

图 3 CNOT 门分解的量子线路示例

1.4 量子算法

在求解某些量子计算问题时,结合量子计算和经典计算可以显著降低对量子比特数目与相干时间的要求.例如,量子相位估计算法(QPE)是量子模拟[13]和 Shor 质因子分解算法[27]等多个量子计算应用的

关键子程序;作为 QPE 的一种实现方式,IPE 算法通过使用经典计算指令实时处理量子比特的测量结果,可有效地降低对量子比特数量和相干时间的要求.本文将以 IPE 作为算法范例,介绍量子-经典异构算法的整体执行流程,说明量子应用从高级语言描述转换为量子软硬件接口表示,并如何在实际量子计算硬件上执行.本节简要介绍 IPE 算法的原理,关于该算法的详细讨论可参考文献[14-15].

假设酉算子 $\boldsymbol{U}$ 有一个本征态 $|u\rangle$,且对应的本征值为 $e^{i\theta}$,即

$$\boldsymbol{U}|u\rangle=e^{i\theta}|u\rangle, \tag{6}$$

相位估计算法的目标是在提供本征态 $|u\rangle$ 以及算子 $\boldsymbol{U}$ 的黑盒实现的前提下完成对 θ 的估计.除用于存储本征态 $|u\rangle$ 的量子比特外,IPE 算法仅需引入一个辅助量子比特,如图 4 所示.IPE 算法的最终输出是 θ 的 m 比特近似估计值 $\tilde{\theta}=2\pi\cdot 0.c_1c_2\cdots c_m$,其中 $0.c_1c_2\cdots c_m$ 是值 $\sum_{i=1}^{m}c_i\cdot 2^{-i}$ 的二进制表示.IPE 算法的主体是一个迭代 m 次的循环,每次迭代执行图 4 第②步中的量子线路.该量子线路包含 2 个 $\boldsymbol{H}$ 操作和夹在其中间的受控 $\boldsymbol{U}^{2^{k-1}}$ 和 $\boldsymbol{R}_z(\theta_k)$,以及一个量子测量.对辅助量子比特进行的测量会返回估计结果中的一位 c_k.在算法执行过程中,k 从 m 递减至 1;第 k 次迭代中的角度参数 θ_k 需通过经典计算从之前迭代的测量结果计算得到:

$$\theta_k=(-0.c_{k+1}c_{k+2}\ldots c_m)\cdot\pi, \tag{7}$$

其中,$k\in\{1,2,\cdots,m-1\}$ 且 $\theta_m=0$.需要注意的是,在算法执行过程中,应使用经优化过的受控 $\boldsymbol{U}^{2^{k-1}}$ 操作,而不是将受控 $\boldsymbol{U}$ 操作执行 2^{k-1} 次.否则,算法的效率和结果的保真度将大大降低.

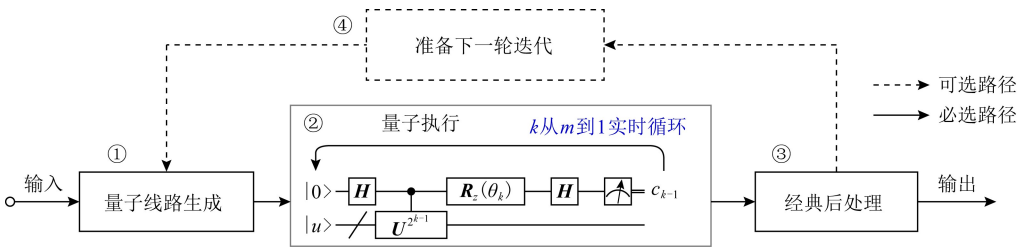


Fig. 4　Flow chart of the iterative phase estimation (IPE) algorithm

图 4　IPE 算法的流程图

1.5　量子比特的物理实现及控制

量子计算需要在满足特定条件的物理系统上执行.目前,已有一些工作归纳并总结了物理系统在执行通用量子计算时必须满足的条件.其中最著名的是 2000 年 DiVincenzo 所总结的 DiVincenzo 准则[28],包含 5 个条件:

1) 可扩展的、由表征良好(well-defined)的量子比特组成的物理系统;

2) 该系统具备将量子比特初始化到一个简单量子态的能力;

3) 量子比特拥有较长的相干时间,且远大于单个量子操作的持续时间;

4) 该系统可在量子比特上施加一组通用的量子门;

5) 该系统可完成对量子比特的测量.

2012 年诺贝尔物理学奖授予美国科学家 David J. Wineland 和法国科学家 Serge Haroche,以表彰他们于 20 世纪 80 年代在测量、操控单量子系统上所做出的开创性贡献[29].自此之后,人们研究了多种量子比特的实现技术方案,如超导量子电路[30-32]、光量子[33-35]、离子阱[36]、拓扑量子系统[37-38]、自旋量子比特[39-40]、核磁共振[41]、钻石色心[42-43]、中性原子[44-47]等.一些技术已被证明具有内在的扩展性问题,如核磁共振[48].一些技术还处于早期发展阶段,如基于马约拉纳的拓扑量子计算.在撰写本文时,最有希望的技术是超导量子比特和离子阱,这 2 种技术都已被证明符合 DiVincenzo 准则[1,49-51].本文将主要基于超导量子比特讨论面向 NISQ 的量子-经典异构系统.但是,本文关于量子-经典异构系统的大部分观点也可适用于其他类型的量子比特,如离子阱.不过读者须注意,不同的量子比特实现技术使用的控制硬件和软硬件接口可能不同.

图 5 展示了一种名为 Transmon 的超导量子比特的扫描电镜图像.Transmon 是一种集总参数的非线性 LC 谐振电路,由一个电容器与一对提供非线性电感的约瑟夫森结并联组成.我们使用该电路的基态和第一激发态分别作为量子比特的 $|0\rangle$ 态和 $|1\rangle$ 态.通过使用近端通量偏置线(端口 P_f)控制 2 个约瑟夫森结之间的环路的通量,可以在纳秒级时间尺度上对跃迁频率 f_Q 在几千兆赫兹的范围内调整.

共平面波导谐振器(R)通过电容耦合至量子比特和测量馈线(从输入端口 P_i 到输出端口 P_o).外部电子控制设备经通量偏置线(P_f)和微波偏置线(P_m)施加脉冲信号产生作用在量子比特的量子门,产生一定频率的微波信号从 P_i 至 P_o 穿透测量馈线,通过 R 完成对量子比特状态的测量.

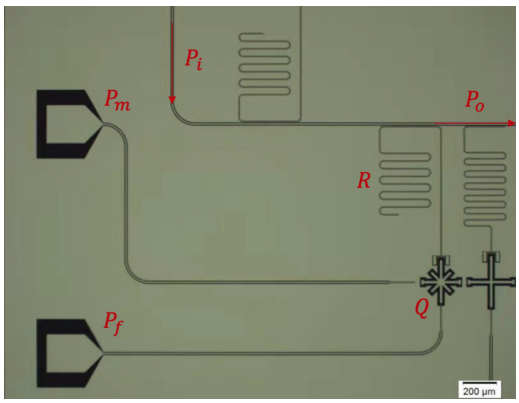
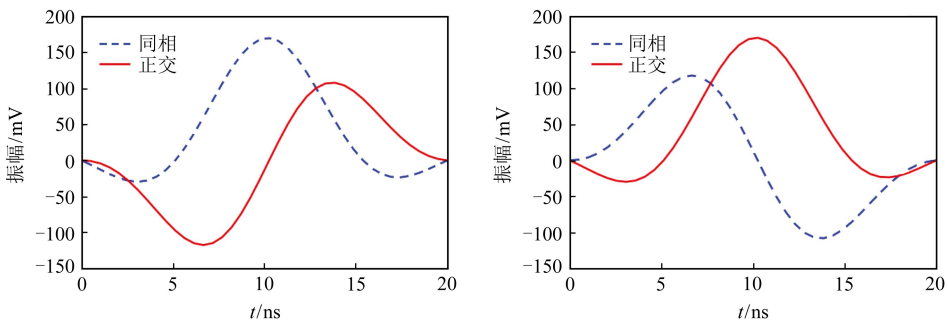


Fig. 5 The SEM image of a Transmon qubit (Q)

图 5 Transmon 量子比特(Q)的扫描电镜图像

R 的基频 f_R 根据量子比特状态的不同而不同.利用这一特性,可实现对量子比特的测量.频率在 f_R 附近的测量电磁脉冲信号(通常持续 300 ns 至 2 μs)在穿过测量馈线时询问量子比特的状态,并将其投影到$|0\rangle$或$|1\rangle$.通过对由 P_o 端口输出的信号依次进行解调、数字化、积分以及与阈值进行比较,可以推断量子比特的测量结果.

单量子比特门通过向量子比特施加频率为 f_Q、时长通常为 20 ns 的微波脉冲来实现.一般使用任意波形发生器(arbitrary waveform generator, AWG)生成微波脉冲的包络信号,然后使用 I-Q 混频器基于载波单边带调制将该信号加载到频率 f_Q.图 6 中显示了绕 x 轴和 y 轴旋转操作的包络信号的同相(I)和正交(Q)分量.该图中,I,Q 分量均已包含了频率为 −50 MHz 的单边带调制分量.该信号中,包络和载波的相位决定对应旋转操作的旋转轴在 Bloch 球赤道平面上的位置,脉冲幅度决定旋转的角度.标准的校准流程可以确定特定脉冲的幅度(如 π 和 π/2)并校正混频器的缺陷.需注意的是,用于进行量子门操作的微波脉冲需以精确的时序作用在量子比特上.例如,图 6 中所示的 x 旋转操作的包络信号已经包含了一个固定的 −50 MHz 单边带调制分量,如果 x 旋转操作施加的时间比正确的时间晚 5 ns,那么最后将产生一个绕 y 轴——而不是绕 x 轴——的旋转操作.最后,通过使用量子门分解技术,如 Solovay-Kitaev 算法[26]或重复直至成功[52],任意单量子比特门可被分解为绕 x 轴和 y 轴的旋转.



(a) $\boldsymbol{R}_x(\pi)$的同相和正交包络 (b) $\boldsymbol{R}_y(\pi)$的同相和正交包络

Fig. 6 In-phase and quadrature envelopes of $\boldsymbol{R}_x(\pi)$ and $\boldsymbol{R}_y(\pi)$ pulses including −50 MHz single-sideband modulation

图 6 包括 −50 MHz 单边带调制分量的 $\boldsymbol{R}_x(\pi)$和 $\boldsymbol{R}_y(\pi)$脉冲的同相和正交包络

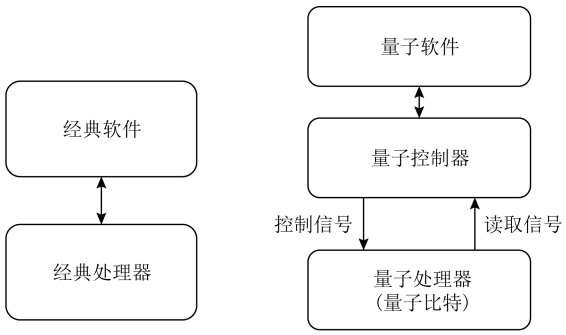
2 量子计算与经典计算的比较

2.1 存储、运算和控制的异质性

传统的数字处理器由存储单元、运算单元和控制单元组成,它们保存或者处理二进制格式的数字信号.存储单元(如寄存器或高速缓存)、运算单元(如算术逻辑单元)以及控制单元均可使用相同的基本元件——晶体管——来实现.存储、处理和控制之间的同质性使存储单元、运算单元和控制单元可自然地集成在单个芯片上.考虑到经典处理器的控制逻辑一般通过二进制指令提供,经典软件生成的指令可直接对接经典硬件,即中央处理器(CPU),如图 7(a)所示.

量子计算与经典计算不同.量子比特即是数据存储的单元,又是数据处理的单元.量子比特利用量子

叠加与量子纠缠来存储数据.当量子操作被施加到量子比特上时,数据在量子态下进行演化.数据存储和处理的格式均基于量子叠加与量子纠缠.量子比特的控制媒介一般是模拟电磁脉冲信号,通常由定制电子学设备产生.所以量子计算中,存储、处理和控制是异质的.这种异质性导致量子控制器需要独立于量子芯片进行设计.量子软件的输出也需要通过量子控制器完成对量子芯片的控制,如图 7(b)所示.



(a) 经典处理器的控制方式　(b) 量子处理器的控制方式

Fig. 7　The difference of the way of controlling between classical processors and quantum processors

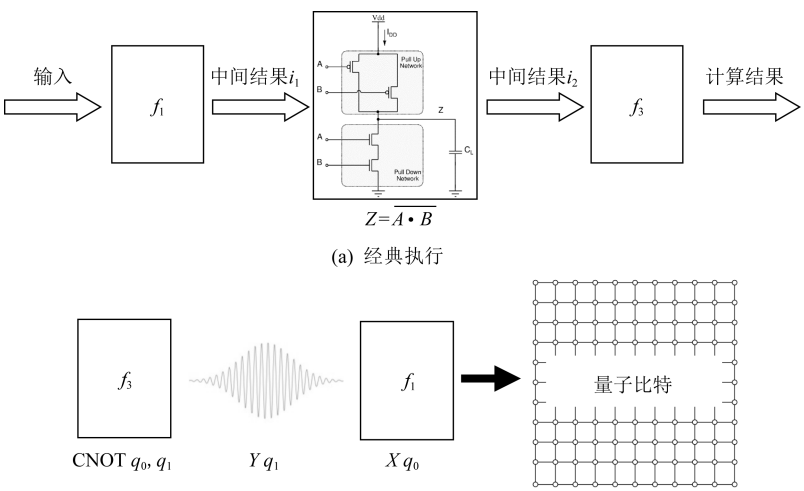
图 7　经典处理器与量子处理器控制方式的差异

2.2　存内计算与非冯·诺依曼体系结构

基于晶体管的经典处理器的计算过程中,数据从存储单元传送到运算单元的输入端口,经过运算单元的处理后,再从运算单元的输出端口传送回存储单元.由晶体管实现的运算单元在空间上是固定的;而数据以电平信号的方式在存储器和运算单元之间流动.这说明经典计算过程中,数据在空间中是运动的,而对数据的操作在空间中是静止的,如图 8(a)所示.经典处理器中的存储和处理一般是分离的.

与经典计算机不同,固态量子比特(如超导量子比特)以物理结构的形式固定在量子芯片上,所以量子比特中保存的数据是固定不动的.在执行量子操作时,外部控制器产生模拟控制信号并施加在量子比特上,控制量子比特的数据在本地完成翻转得到运算结果.量子计算中,对数据的操作在空间中是运动的,而数据在空间中是静止的,如图 8(b)所示.量子比特既是存储信息的单元,又是处理数据的单元,因此,量子计算是存内计算(process in memory, PIM)的一个典范.

意识到数据的存储和处理相分离这个特点,冯·诺依曼提出一个存储程序计算机的系统结构,即冯·诺依曼体系结构.冯·诺依曼体系结构由存储器、运算器、控制器以及输入、输出设备构成.根据所执行的指令,控制器从特定存储单元中取出数据,转交给算术逻辑单元进行计算,最后将计算结果写回存储器中.



(a) 经典执行

(b) 量子执行

Fig. 8　The difference of data flow between classical computing and quantum computing

图 8　经典计算与量子计算中数据流动的差异

冯·诺依曼体系结构是经典计算机体系结构的圭臬,显著地影响了以往量子计算机体系结构的相关研究[53-59].随着冯·诺依曼体系结构面临的存储墙问题日益突显、NISQ 技术的发展使得越来越多的中小规模量子算法可以运行在硬件上,人们逐渐注意到量子计算机存算一体的特性.与此同时,研究人员开始意识到,量子计算机或许并不需要基于冯·诺依曼体系结构来构建.

3 量子计算系统层次

3.1 量子计算系统层次结构

由于大规模系统往往是复杂的,清晰的层次结构及层次间的接口定义有助于控制构建量子计算机的复杂度.工程化实现量子计算机需要解决的核心问题是:量子计算系统中的抽象层次结构及各层间的接口应如何定义?

基于QRAM模型,我们之前的工作[21,60-61]从高层抽象的视角提出了一个量子计算系统层次结构,如图9所示:

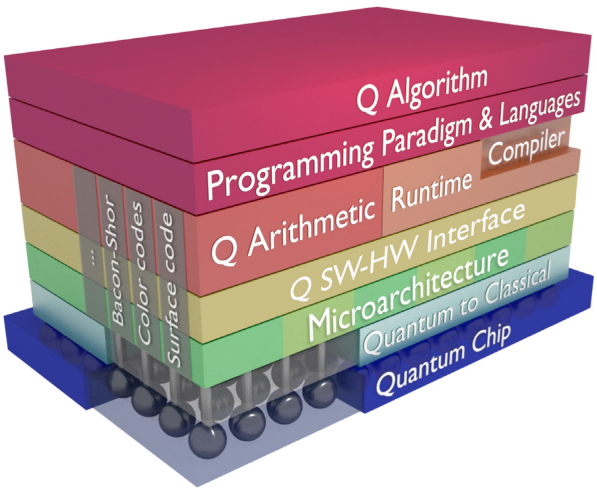


Fig. 9 Overview of quantum computer system stack based on the circuit model[60]

图9 基于线路模型的量子计算系统层次结构[60]

人们设计各类量子算法用于解决特定问题.一个合格的量子算法,其复杂度应优于解决同一问题的最佳经典算法[26].根据量子随机访问机器(quantum random access machine, QRAM)模型[62],量子算法的设计可结合使用量子计算资源和经典计算资源.计算资源包括存储资源(如经典寄存器、存储器和量子寄存器等)和可使用的操作(如对经典数据的算术逻辑操作与对量子比特的门操作和测量操作等).解决具体问题的量子算法采用高级量子程序设计语言(Programming Paradigm & Languages)来描述.为支持量子-经典异构量子算法,一个量子应用程序可能使用一种或多种经典的或量子的高级程序设计语言.编译器(compiler)采用该描述作为输入,执行可逆逻辑综合、量子门的分解、量子比特的映射和调度、量子线路的优化,并输出量子软硬件接口(Q SW-HW Interface)格式的程序表示.量子软硬件接口可由量子指令集和波形定义接口组成.量子控制微体系结构(microarchitecture)执行属于软硬件接口中的指令.量子控制体系结构支持精确的时序控制,可实时将量子指令翻译为时序精确的控制信号.最后,基于特定的量子技术,如超导量子比特、离子阱等,控制信号被转换成通过软硬件接口定义的模拟波形,并通过量子经典(quantum to classical)接口发送到量子芯片(quantum chip).

如图9系统堆栈的左侧面所示,当量子算法的执行时间较长而需要使用量子纠错[12]时,量子编译器、量子指令集和控制微体系结构需添加额外的支持.此时,高级量子程序设计语言中使用的量子比特和量子操作一般是逻辑量子比特和逻辑量子操作.量子编译器在进行与硬件无关的线路优化之后,需根据使用的量子纠错编码,将逻辑量子操作翻译为作用在物理量子比特上的物理量子操作.量子控制微体系结构还需进行实时量子错误检测和校正[63-64].它利用辅助量子比特对数据量子比特进行量子校验,生成错误症状(error syndrome)、根据错误症状推断可能的错误、记录这些错误信息或在必要时触发合适的量子门来纠正量子错误.

3.2 量子经典混合异构计算系统

虽然图9所示层次结构明确了量子算法从设计到最终的硬件运行所需经过的各个阶段,但并未讨论如何组织和管理量子和经典软硬件资源从而完成量子-经典异构计算.针对该问题,面向NISQ技术水平,我们提出青果(Quingo)量子-经典异构系统框架用于集成并管理量子-经典软件和硬件[24],如图10所示.

青果旨在为量子-经典异构应用及量子实验提供一个灵活、精简的编程框架,并能够将这些应用映射到NISQ硬件上执行.青果框架包含量子程序、编译器、硬件平台和运行时系统等4个主要组成部分.青果定义的量子程序六阶段生命周期模型描述了4个主要组成部分间的交互.

青果框架下,一个完整的量子程序生命周期可分为6个阶段.第1步,程序编写.用户首先使用包含了经典主程序和量子内核的混合程序来描述量子应用.经典主程序描述经典计算并通过编程环境提供的接口调用量子内核.主程序使用经典程序设计语言(如Python或C语言)编写.量子内核使用量子程序设计语言(如Quingo)描述发生在量子协处理器上的计算任务.第2步,经典编译.经典编译器(如GNU

Compilation Collection(GCC))编译经典主程序,并生成经典二进制文件.若经典主程序使用解释型语言描述,如 Python,则第 2 步可以跳过.第 3 步,经典预处理.(编译后的)经典主程序随后交由经典主机运行,直到调用量子内核为止.此时,调用量子内核所需的参数全部确定,进而可启动第 4 步——量子编译.这一步中,量子编译器编译量子程序,并可利用第 3 步提供的内核调用参数,使用例如部分执行等编译优化技术,优化量子内核,生成量子指令程序、量子操作对应的波形及其他配置信息.第 5 步,量子执行.量子协处理器根据第 4 步生成的波形和配置信息等,配置量子控制体系结构,然后执行量子指令程序控制量子比特进行状态翻转完成量子核心计算过程.量子核心计算结束后,返回计算结果.第 6 步,经典后处理.经典主程序继续运行,对量子内核的计算结果进行后处理.如果有必要,第 3～6 步可反复执行多次,从而产生足够好的计算结果.为了支持上述运行过程,青果硬件平台需包括 1 个经典主机、1 个量子协处理器和 1 个可供主机和协处理器访问的共享内存.运行时系统是青果必不可少的软件环境,其地位类似于经典计算机的操作系统.它包含 5 个主要的部分:①用于配置运行环境的系统配置器;②向经典主程序提供的量子内核调用接口;③用于访问不同量子硬件的后端接口;④支持主程序和量子内核间的数据传递的参数转换器;⑤用于管理量子程序阶段并在各阶段触发相应行为(如量子编译和量子执行)的阶段管理器.关于青果框架的更多设计细节,感兴趣的读者可参考文献[24].

1.4 节和 1.5 节已分别介绍了顶层量子算法和底层物理量子比特实现及其控制对量子-经典接口的需求.本文接下来采用一种自顶向下的方法,于第 4～7 节分别介绍面向 NISQ 技术的量子程序设计语言、量子编译器、量子软硬件接口和量子控制微体系结构在整个计算系统中的功能定位、关键技术或设计思路和已有的实现等.同时,在每节最后,我们将以 1.4 节中的 IPE 算法为例,介绍量子算法如何从高级程序设计语言描述一步步被转换为在硬件上可执行的表示以及其在硬件上的执行过程.

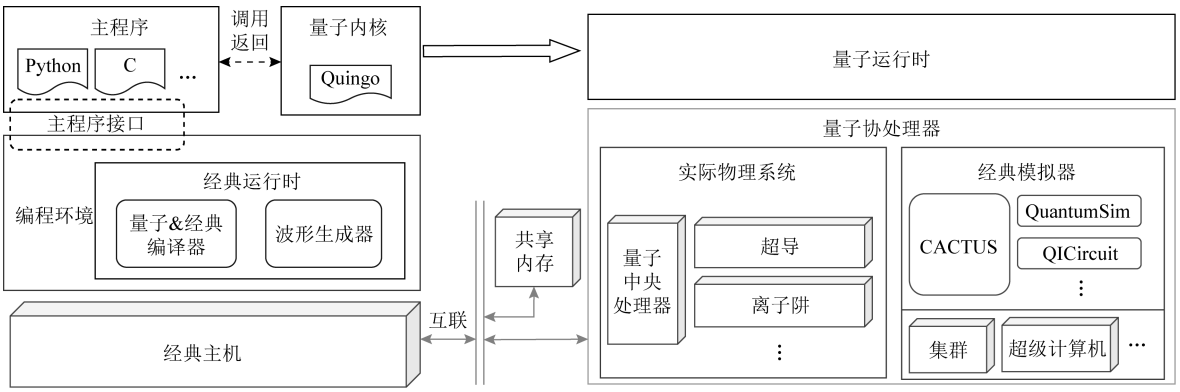


Fig. 10　Quingo heterogeneous quantum-classical computing architecture

图 10　青果量子-经典异构体系结构

4　量子程序设计语言

程序设计语言位于量子计算系统层次结构中的第 2 层,其核心功能是提供灵活而高效的方式来描述量子算法,并在必要时允许用户描述算法在运行时的部分底层细节,从而实现量子应用的最优化执行.基于此定义,量子程序设计语言应该具备 3 个层次的特性.

1) 完备的表达能力.所谓完备,是指语言通过对量子计算系统中可用的软硬件资源提供合适的抽象,从而向用户提供描述任意基于 QRAM 模型的量子算法的能力.具体而言,则包括:①完备的类型系统以描述可用的软硬件资源;②完备的基本操作集合以描述任意计算操作.量子程序设计语言不仅要支持量子类型与量子操作,还应支持经典类型与经典操作.需要注意的是,这种完备性并非必须由单个语言提供,也可以通过 2 种甚至多种程序设计语言协同配合以完成对量子-经典异构算法的描述.例如使用类似于 OpenCL[65] 的编程框架,可将量子算法表述为一个经典主机程序和若干量子内核函数的组合.经典主机程序使用经典程序设计语言(如 Python,C++)编写,描述量子算法中所需的经典计算逻辑,并调用量子内核.量子内核负责描述可被

量子硬件加速的特定计算任务.可使用嵌入在经典程序设计语言中的库如 Qiskit[66],QPanda[67]等或独立的量子程序设计语言[23,68]如 Quingo[24],Q♯[69]编写.

2) 高抽象层次语法与低级控制机制.由于 NISQ 技术的限制,量子程序设计语言在采用高抽象层次语法的同时,还需向用户提供对硬件进行低级控制的机制,例如,对量子操作采用的波形与操作发生的时序进行编程控制.现有的高级量子程序设计语言包括 Scaffold[70], PyQuil[71], Cirq[72], ProjectQ[73], Q ♯[69], OpenQL[74], Quipper[75] 与 Quingo[24]等.其中,Scaffold,Q♯与 Quingo 是独立的量子程序设计语言,而其他语言则嵌入在一门经典程序设计语言(如 Python,C++,Haskell 等)中,所以是嵌入式领域特定语言(embedded Domain-Specific Language,eDSL).eDSL 的主要优势在于可以复用宿主语言的编译器与程序库.eDSL 的不足之处在于需采用元编程(meta-programming)的方式构造量子线路或描述实时经典计算逻辑,该过程不直观、易出错,尤其是无法较好地支持实时的分支和循环等程序结构.不少高级量子程序设计语言(如 Quingo,PyQuil)对低级控制机制开展了探索性的支持.

3) 面向领域的特殊语法支持.量子程序设计语言本质上是一类领域特定语言,通常在语言层面提供了面向量子计算这一特定领域的语法工具.量子算法中存在若干常用编程模式,包括量子比特资源管理、并行量子操作、受控量子门(controlled gate)、量子门求逆(reversed gate)与反计算(uncomputation)等.现有的量子程序设计语言对上述编程模式有不同程度的支持,这些语法特性不仅可以简化算法的描述,减少编程错误,还可以辅助编译器执行代码分析与优化.

在青果框架下,1.4 节中描述的 IPE 算法主程序使用 Python 语言描述(如图 11),量子内核使用 Quingo 语言描述(如图 12),二者协作完成 IPE 算法的完整执行过程.主机程序重复 n 次调用(图 11 第 7 行)量子内核函数(图 12 第 9 行),每次调用均得到参数 θ 的一个估计值(图 12 第 12 行),最终取 n 次估计的平均值作为最终估计值 $\bar{\theta}$.量子内核函数 *ipe* 接收一个 int 类型输入并返回一个 double 类型的结果(图 12 第 8 行).在该函数中,using 结构申请了 2 个量子比特(图 12 第 12 行),它们将在程序流离开 using 结构时自动释放(图 12 第 36 行).图 12 第 17~35 行的 for 循环结构描述了 IPE 算法的子量子线路——一个由量子比特初始化、$\boldsymbol{H}$ 操作、受控 $\boldsymbol{U}^{2^{k-1}}$ 操作、$\boldsymbol{R}_z(\theta_k)$操作、$\boldsymbol{H}$ 操作与量子测量构成的操作序列.

```
1  # host.py: IPE算法的经典主程序
2  from qgrtsys import if_quingo
3  ''' 调用Quingo内核来估计给定黑盒的相位。'''
4  def ipe(m: int, n: int) -> float:
5    res = 0
6    # 重复$n$次，最终结果取平均值。
7    for i in range(n):
8      # 每次调用返回一个$m$比特的theta估计值；
9      if not if_quingo.call_quingo("ipe.qu", "ipe", m):
10         raise SystemError("The execution of "
11         "the quantum kernel fails.")
12     res += if_quingo.read_result()
13   return res / n
```

Fig. 11 The classical host program of IPE algorithm

图 11 迭代相位估计(IPE)算法的经典主程序

```
1  // kernel.qu: 迭代相位估计(IPE)算法的量子内核.
2
3  import operations.*
4  import config.json.*
5
6  // 输入：待估计相位的目标位数
7  // 输出：待估计相位的值
8  operation ipe(m: int) : double {
9      double theta = 0.0;       // = theta_k / PI
10
11     // 申请两个比特
12     using (ancilla: qubit, eigenstate: qubit) {
13         // 准备本征态 |1>
14         if (!measure(eigenstate)) {
15             Rx(eigenstate, PI);
16         }
17         for (int k = m - 1; k >= 0; k -= 1) {
18             // 重置辅助量子比特为 |0>
19             if (measure(ancilla)) {
20                 Rx(ancilla, PI);
21             }
22             H(ancilla);
23
24             // 施加量子操作 受控$U^(2^i)$
25             control(ancilla, oracle(eigenstate, k));
26             Rz(ancilla, -PI * theta);
27             H(ancilla);
28
29             // 更新所估计的相位值
30             if (measure(ancilla)) {
31                 theta = theta / 2.0 + 0.5;
32             } else {
33                 theta /= 2.0;
34             }
35         }
36     }
37     return PI * theta;
38 }
```

Fig. 12 The Quingo kernel program of IPE algorithm

图 12 IPE 算法的 Quingo 内核程序

5 量子编译器

量子编译器[76]负责解析高级程序设计语言描述的量子程序,对量子程序进行与硬件无关或相关的优化过程,并生成量子硬件可执行的格式.

与面向冯·诺依曼体系结构的经典编译器相比,量子编译器有若干特别或不同之处.

1) 量子编译器作为一款经典软件运行在经典处理器上,但生成运行在量子协处理器上的低级量子程序.因此量子编译天然地是一个交叉编译的过程.

2) 量子编译的主要操作对象不同.经典编译器操作和优化的主要对象是数据.因此,在条件允许的情况下,经典编译器可采用诸如常量传播的方法来优化代码.但如第 2 节所述,量子计算是典型的存内计算,量子程序的主体是描述施加在量子比特上的量子操作序列.此外,由于量子状态是不可克隆的且量子比特中保存的信息对编译器不可见,因此量子编译器操作和优化的对象主要是量子操作而非量子比特中的数据.需说明的是,量子编译的优化过程高度依赖量子操作序列之间的等价性,在优化量子操作序列时难以使用如常量传播等经典优化技术.

3) 面向 NISQ 技术的量子编译器需要考虑量子操作的时序.一方面,经典编译过程中,通常可假设数据能够在存储单元中保存无限长的时间,而量子编译器则必须始终应对 NISQ 量子比特相干时间非常短暂这个挑战.这导致量子编译器须通过量子比特的映射和调度、量子操作的调度等优化方法来尽可能缩短量子程序执行的绝对时长.另一方面,一些量子编译器会通过物理系统在不同的哈密顿量下演化的等价性来优化量子波形[77].这个过程中,量子操作的时序是一个高度相关的参量.

4) 量子编译器需要考虑更多物理实现细节,比如物理系统所支持的通用量子门集合、量子比特间的连接拓扑结构、量子操作的时序等.这扩展了程序的优化空间,但也使得优化过程更具有挑战.

5) 因为在 NISQ 时代量子比特数量少且相干时间短等特点,人们更愿意在量子编译上付出更多的努力对量子操作序列进行优化.量子编译优化的过程中,一个常见的做法是在每次物理实验之前重新编译量子程序,这使得许多经典输入数值从未知变为已知[23].在此基础上,量子编译器可以去除一些分支结构,进行经典数值的常量传播、循环展开等优化,最大限度挖掘优化的可能.这些优化有时被称为部分执行(partial execution)[78].

多数量子编译器的工作过程可以大致划分为 2 个阶段,如图 13 所示.首先,编译器前端解析量子程序并创建中间表示(intermediate representation, IR).然后,编译器后端施加与平台无关或相关的多种变换,包括门分解、逻辑综合、量子线路、量子比特的调度和映射等.最后,编译器输出量子硬件可执行的格式,多为量子汇编.编译器在保证逻辑正确的前提下,通过优化来减少所需的量子比特数和量子操作数量(特别是开销较大或错误率较高的门,如施加在 Transmon 上的两量子比特门[79]),从而减少所需的硬件资源、缩短量子程序执行时间并提高计算结果的保真度.

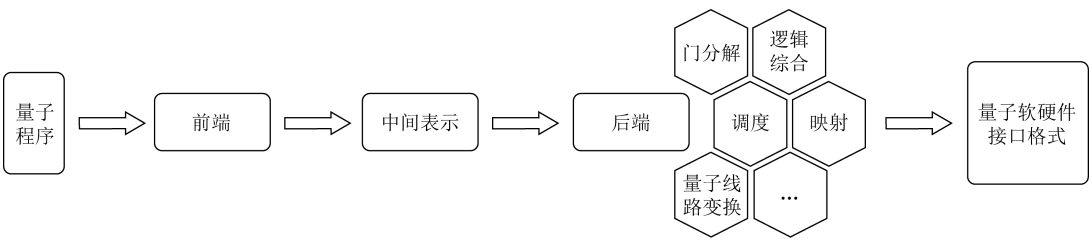


Fig. 13　The workflow of quantum compilers

图 13　量子编译器的工作流程

中间表示是编译所依赖的数据结构.一般而言,后端每种变换的输入和输出都为中间表示.有的量子编译器直接将抽象语法树(abstract syntax tree, AST)作为中间表示[80],有的使用量子操作命令列表[81].另一种比较流行的中间表示方法是有向无环图(directed acyclic graph, DAG)[82].和传统编译器中使用的依赖关系图不同的是,量子编译 DAG 中某些门可以交换顺序,这是由量子计算的特点所决定的[77].DAG 能够较好地表达量子线路,但在表达、分析和变换量子-经典混合代码上有一定的困难.因而这些编译器仅能处理固定的量子线路,其中不得包含分支、循环等经典控制结构(如图 12 中的 if 语句).为了适应量子-经典异构计算,人们尝试扩展经典编译器的中间表示,如使用 Low-Level Virtual Machine(LLVM)[83]或 Multi-Level IR(MLIR)[84],来表示量子程序[85-87].这方面的研究才刚刚开始,尚未出现广泛使用的标准.

后端是量子编译器的主体,往往要反复进行多种变换,其中常见的包括 6 类.

1) 门分解.物理量子系统所支持的量子门操作

种类和精度都比较有限.编译器需将用户程序中使用的其他量子门操作分解为系统支持的门操作.对于单量子比特门,已经有算法能够给出指定精度下最优的分解方法[88].对于多量子比特门,相关研究也取得了很大进展[89-90].门分解不仅使得各种量子算法可以运行在实际的物理平台上,还可能带来额外的优化空间.

2) 逻辑综合.逻辑综合指的是使用特定的一组门的组合来实现用户指定的经典计算逻辑.由于量子计算为可逆计算,所以量子编译中涉及的主要是可逆逻辑综合[91-92].特别地,量子程序中常用的受控门有时需要使用逻辑综合的方式实现[79].

3) 量子线路变换与化简.根据量子操作的特性,编译器可对量子线路进行变换和化简.例如,已知 $\boldsymbol{HH}\equiv\boldsymbol{I}$,就可以去掉连续出现的 2 个 $\boldsymbol{H}$ 门而不会影响量子线路的执行结果.虽然这样直接优化的机会并不常见,但借助门分解和量子线路等价变换可以创造类似的优化条件.由于求解最优量子线路的复杂度为 QMA 完全(不低于 NP 完全)[93],现有的量子线路优化方法多基于启发式算法[94-95].

4) 映射.量子线路中包含的逻辑量子比特需要映射到物理平台中.平台中物理比特数量有限,而且往往存在诸多约束,比如两量子比特门操作只能施加到一些特定的量子比特对.如果程序中的两量子比特门作用的量子比特没有被映射到可施加两量子比特门的物理量子比特上,编译器就不得不在量子线路中添加额外的操作,比如 SWAP,将这 2 个比特移动到约束允许的位置[96].可见,这种情况引发的额外开销与映射量子比特的位置有关.因为求解最优映射为 NP 完全问题[97],所以现有的映射算法也多是启发式的[97-101].

5) 调度.针对 NISQ 技术量子相干时间较短的问题,量子编译过程中会特别注意量子操作的时序调度.比如文献[82]中使用类似于尽可能迟(as late as possible, ALAP)算法调度量子线路中的门操作,减少了多个门操作的间隔时间,故而能提高程序的保真度.有的物理平台不允许同时向不同量子比特施加不同操作,那么编译器就需要据此对量子线路重新进行调度.

6) 量子最优控制.量子最优控制算法通过物理系统在不同的哈密顿量下演化的等价性,试图寻找从当前量子状态变换到目标量子状态的最优演化路径.一个常用的方法是梯度下降方法,比如 GRAPE (GRadient Ascent Pulse Engineering)算法[102],以提高程序运行的保真度.算法得到的结果是量子控制波形,而不是一般意义上的量子芯片指令[77].

值得说明的是,以上变换并非孤立存在.比如量子门的分解影响到线路的优化,调度与映射往往紧密相关.编译器需要从整体的角度优化用户程序,综合考虑物理平台在时间和空间上的约束,以输出较为理想的结果.

随着量子计算日益升温,量子编译器的研究也逐渐活跃起来.首先,几乎每种量子编程语言都有对应的编译器,比如 Scaffold[70](ScaffCC[78]),Quil[103](QuilC[104]),Q♯[69],Quipper[75],Quingo[24] 等.其他编译器则包括 Qiskit[66],ProjectQ[73],Cirq[72],SQIR[105],Strawberry Fields[106],staq[80],t|ket⟩[107] 等,它们编译的量子程序多数使用 Python 描述.

在图 14 中,我们借助微软 QIR[86] 风格的中间表示来说明图 12 中的编译过程.首先,第 1 行显示的是函数头部声明,基本和量子程序中的写法一致.第 2 行的标号表示一个基本块的开始.每个基本块中包含若干顺序执行的语句.第 3 行为局部变量%theta 分配内存空间,并在第 4 行初始化为 0.注意,中间表示

```
1
2  define double @__Quantum__Qir__ipe(i32 %m) {
3  entry:        ;初始化局部变量
4    %theta = alloca double, align 8
5    store double 0.000000e+00, double* %theta, align 8
     %ancilla = call %Qubit* @__quantum__qubit_allocate()
6    %eigenstate = call %Qubit*
       @__quantum__qubit_allocate()
7  ...
8
9    %5 = call %i1 @__quantum__m(%Qubit* %ancilla)
10   br i1 %5, label %then0__2, label %continue__2
11 then0__2:
12   call void @__quantum__rx(%Qubit* %ancilla, double
       3.141592654e+00)
13   br label %continue__2
14 continue__2:
15   call void @__quantum__h(%Qubit* %ancilla)
16   ;最终被优化为:  call void @__quantum__ry(%Qubit* %
       ancilla, double 1.570796327e+00)
17 ...
18
19   %8 = load double, double* %theta, align 8
20   %9 = fmul double -3.141592654e+00, %8
21   call void @__quantum__rz(%Qubit* %ancilla, double %
       9)
22   call void @__quantum__h(%Qubit* %ancilla)
23   ;最终被优化为:
24   ; call void @__quantum__rx(%Qubit* %ancilla, double
       -1.570796327e+00)
25   ; call void @__quantum__rx(%Qubit* %ancilla, double
       %9)
26   ; call void @__quantum__rx(%Qubit* %ancilla, double
       -1.570796327e+00)
27   ; call void @__quantum__ry(%Qubit* %ancilla, double
       -1.570796327e+00)
28 ...
29
30   // 释放量子比特
31   call void @__quantum__qubit_release(%Qubit* %
       eigenstate)
32   call void @__quantum__qubit_release(%Qubit* %
       ancilla)
33   ret double %20
34 }
```

Fig. 14 The IR of the code snippets of IPE algorithm

图 14 IPE 算法代码片段的中间表示

一般采用静态单赋值(static single assignment, SSA)形式,即每个变量只允许在一条语句中被赋值,以便于后续的分析优化.因此在为%theta 赋值时,要使用指针,而不能直接赋值.第 5 行和第 6 行调用函数@__*quantum*__*qubit*_*allocate* 分配 2 个逻辑量子比特.

然后,为了说明平台无关的优化,我们重点关注图 12 中第 19～22 行的翻译结果.图 14 第 9 行对应量子程序中的 measure 操作.其结果在第 20 行被用作分支语句 br 的条件.无论哪个分支被执行,在进入 continue__2 后,%ancilla 一定处于$|0\rangle$态,而 $\boldsymbol{H}|0\rangle \equiv \boldsymbol{R}_y(\pi/2)|0\rangle$,因此可以使用开销较小的 $\boldsymbol{R}_y(\pi/2)$门来替代第 22 行的 $\boldsymbol{H}$ 门,以提高程序的性能.

接下来,我们利用图 12 中 26,27 两行说明平台相关的优化.由于 Transmon 量子比特较少直接使用 $\boldsymbol{R}_z$ 门,所以编译器要将其转变为 $\boldsymbol{R}_x$ 和 $\boldsymbol{R}_y$ 门,依据的原理为 $\boldsymbol{R}_z(\theta)=\boldsymbol{R}_x(\pi/2)\boldsymbol{R}_y(\theta)\boldsymbol{R}_x(-\pi/2)$.由于 $\boldsymbol{H}\equiv\boldsymbol{R}_y(-\pi/2)\boldsymbol{R}_x(\pi)$,所以 $\boldsymbol{H}$ 门分解得到的第 1 个操作 $\boldsymbol{R}_x(\pi)$可与 $\boldsymbol{R}_z(\theta)$分解得到的最后一个操作 $\boldsymbol{R}_x(\pi/2)$合并为 $\boldsymbol{R}_x(3\pi/2)$,也即 $\boldsymbol{R}_x(-\pi/2)$.综上,源程序中的 $\boldsymbol{R}_z$ 与 $\boldsymbol{H}$ 两个门可以转换为图 12 第 24～27 行的 4 个量子门.

最后,@__*quantum*__*qubit*_*release* 函数将 2 个量子比特释放.在本例中,编译器将%ancilla 和%eigenstate 这 2 个逻辑量子比特映射到编号为 0 和 1 的 2 个物理量子比特,因为它们支持两量子比特门.

6　量子软硬件接口

经量子编译器优化之后的量子程序,最终被转换成为量子软硬件接口(Q SW-HW Interface)定义的接口格式.量子软硬件接口向量子软件(如编译器等)提供一组灵活、易用的硬件编程接口,使量子程序可运行在量子硬件上.量子软硬件接口包含模拟信号的定义方式和量子指令集 2 部分.前者用于定义实现量子操作或量子线路的模拟信号.后者用于支持结构化的量子程序描述、以精确的时序触发模拟信号的产生以及实现基于测量结果的反馈控制等.

如 1.5 节所述,对超导量子比特的量子操作通过施加微波脉冲实现.一方面,由于不同的实验环境选择使用不同的电子设备,且各类电子设备可能使用不同的模拟信号数据格式,因此不同的实验环境使用的模拟信号数据格式不尽相同.另一方面,作用在基于不同技术的量子比特或相同技术但不同参数的量子比特上的量子操作所需的模拟控制信号细节不尽相同,如频率、幅度、相位等.这导致人们在交流使用的控制信号时,更多地在数学的层面,而不是直接基于模拟信号的格式进行描述.为了增加交流的效率,IBM 于 2018 年提出 OpenPulse,提供了一种模拟信号的定义格式[108].

与模拟信号的定义相比,使用指令来表示量子操作能够提供更抽象的量子计算语义与更精简的低层次量子程序描述.通过指令的执行实时地控制模拟信号的产生,还可以有效地减少量子软硬件间传递的数据量[60].随着量子程序规模的增长,这种方案更具有可扩展性.结合使用经典指令,量子二进制可支持结构化的量子程序描述和基于量子测量的反馈控制.这有助于提高量子二进制表达能力且减小二进制代码的体积.量子-经典异构系统中,经典处理器具备强大的经典计算能力.由于经典处理器与量子协处理器二者间的通信延迟较大,经典处理器可用于执行不要求实时性的经典计算任务.因此,量子指令集中包含的经典指令主要用于程序流的控制和寄存器的更新.

在量子控制系统尚未充分发展时,人们已启动了对量子汇编的研究.这些构成了后续量子指令集研究的基础.Nielsen 和 Chuang 最早提出量子汇编(quantum assembly, QASM),其目标是提供量子线路的文本描述格式,用于所著教材《量子计算和量子信息》[26]中量子线路图的绘制.此后,Svore 等人将 QASM 作为量子编译器使用的一种低级中间表示[76].在 QASM 的基础上,研究人员不断提出不同的量子汇编,如 QASM-HL[78]、OpenQASM[109]、Quil[103]、带反馈的量子汇编(f-QASM)[110]、通用汇编(cQASM)[111]、可执行的量子汇编(eQASM)[112],从不同的维度增强了量子汇编的表达能力,包括结构化的量子程序描述[78,103,109,111-112]、基于量子测量结果的反馈控制[103,109-110,112]、量子-经典指令的混合执行[103,112]、单操作多量子比特执行[112]、精确的时序控制[112]、硬件可执行性[112]等.虽然历史上汇编语言作为二进制程序的可读化表示而出现,现有绝大部分量子汇编仅仅是量子程序的低层次表示,不能或难以在硬件上直接执行.eQASM 是第 1 个可执行的量子汇编,它定义了 1 套简单的映射规则,可将汇编指令映射为可在 QuMA 微体系结构[60]上执行的二进制指令.同时,eQASM 可支持全面的程序流控制、精确的时序控制等,可较好地描述绝大部分的量子算法和一系列常用量子实验[112-114].本文将以 eQASM 为例介绍量子程序的软硬件接口表示.

eQASM 的设计面向量子-经典异构计算，eQASM 仅用于描述发生在量子协处理器上的计算过程.eQASM 程序由量子指令和辅助经典指令混合组成.量子指令分为 3 类：1)设置量子操作目标寄存器的指令(SMIS 和 SMIT)；2)指定量子操作间的时间间隔的指令(QWAIT 和 QWAITR)；3)基于超长指令字的量子操作指令(quantum bundle).关于 eQASM 的更多设计思路和技术细节可参考文献[112,115].

经量子编译器编译后，图 12 中的程序被翻译为一组波形，对应常用量子操作，如 $\boldsymbol{R}_x$，$\boldsymbol{R}_y$ 和 $\boldsymbol{CZ}$ 等，以及一个量子指令程序.这里主要介绍量子指令程序.图 12 中的 for 循环可被翻译成图 15 所示的 eQASM 程序①(受限于文章篇幅，未显示受控黑盒对应的量子指令).eQASM 为了支持单量子操作多量子比特执行，量子操作使用间接寻址，即量子操作会从量子操作目标寄存器(如第 1 行和第 2 行设置的 s0 和 s1 寄存器)中读取要操作的目标量子比特.图 15 中第 13～19 行实现 ancilla 量子比特的初始化.0 号量子比特首先被测量(第 13 行)，在测量操作开始 600 ns，即 30 个周期后(假设每个周期 20 ns)，FMR 指令将测量结果读取到通用寄存器 r9 中.若测量结果不为 0，则在又 40 ns 之后通过 2，RX s0，3.141592654 操作(第 18 行)将量子比特翻转为 $|0\rangle$ 态.该指令开始处的 2 表示该操作开始于前一条指令的 2 个周期之后.图 12 第 32～38 行利用基于测量结果的程序流控制，它们被翻译为图 15 第 30～36 行.需要注意的是，生成的 eQASM 代码中，f1 的值(即 theta)先被除以 2(第 32 行)，然后仅在测量结果为 1 时，f1 被加上 0.5.图 15 第 40 行更新循环计数器，当计数器仍然大于或等于 0 时，就跳转到循环头部重新开始(第 41 行).

```
1  SMIS s0, {0} # 初始化
2  SMIS s1, {1}
3  LDI r0, 0
4
5  # 更多代码...
6
7  FCVTSW f1, r0   # f1 <- 0.0
8
9
10 # 开始for循环
11 LDI r8, 7 # k = m - 1
12
13 LOOP_START: # 基于反馈的init
14 MEASZ s0
15 QWAIT 30
16 FMR r9, q0
17 BNE r9, r0, RESET_DONE
18 2, RX s0, 3.141592654
19 RESET_DONE:
20
21 2, RY s0, 1.570796327  # H|0> = Ry(PI/2)|0>
22
23 # ...
24 # 此处跳过实现受控黑盒的诸多指令
25 # ...
26
27 FMULS f2, f1, -3.141592654
28 # Rz(theta) = Rx(PI/2) * Ry(theta) * Rx(-PI/2)
29 2, RX s1, -1.570796327
30 1, RY s1, f2
31 1, RX s1, -1.570796327
32 2, RY s1, -1.570796327  # H = Ry(-PI/2) * Rx(PI)
33
34 FDIVS f1, f1, 2.0
35 MEASZ s0
36 QWAIT 30
37 FMR r9, q0
38 BEQ r9, r0, ADDED_HALF
39 FADDS f1, f1, 0.5 # 更新theta
40 ADDED_HALF:
41
42 SUB r8, r8, 1 # 更新计数器
43 BGE r8, r0, LOOP_START
44 # for循环结束
45
46 # 更多代码...
```

Fig. 15 The eQASM code of IPE algorithm

图 15 IPE 算法的 eQASM 实现代码示例

7 量子控制微体系结构

量子控制微体系结构(microarchitecture)执行量子软硬件接口格式描述的量子程序，产生时序精确的模拟信号控制量子比特的状态更新，识别量子比特的测量结果，并完成经典寄存器的更新和程序流的控制.如果量子软硬件接口使用了量子指令，则量子控制微体系结构需将量子指令实时翻译为时序精确的控制信号.基于特定的量子比特实现技术(如超导量子比特)，控制信号被转换为所需的模拟电磁脉冲，并通过量子-经典接口发送到量子芯片.量子-经典(quantum to classical)接口负责微体系结构中的数字信号与量子比特所需的模拟信号之间的转换.量子-经典接口与量子芯片(quantum chip)的设计取决于所使用的量子比特实现技术.

如 1.5 节所述，超导量子处理器的输入和输出信号都是复杂的模拟信号.对量子比特的操作(输入信号)是通过发送模拟脉冲来执行的，而量子比特的测量结果存在于量子处理器的输出模拟信号中.一种常用的产生量子控制脉冲的方法是使用 AWG.在执行量子算法之前，需要将量子操作对应的脉冲校准并将每个脉冲的样本振幅值阵列上传至 AWG 的存储器中.持续时间为 T_d 的脉冲需要存储器存储同相和正交分量的 $N_s=2\cdot T_d\cdot \boldsymbol{R}_s$ 个样本，其中 $\boldsymbol{R}_s$ 是采样率，通常在 1 GSample/s 左右.样本垂直分辨率的典型值在 10～16 位之间.为了识别量子比特 q 的

① 代码 4 使用的是一个扩展版的 eQASM，添加了对参数化的旋转操作、浮点寄存器和浮点指令的支持.

测量结果,通常使用模数转换器将包含了测量结果信息的模拟信号 $V_a(t)$ 数字化,并进行积分和阈值比较:

$$S_q=\int V_a(t)W_q(t)\mathrm{d}t,\text{且 }M_q=\begin{cases}1,\text{若 }S_q>T_q,\\0,\text{其他},\end{cases}\tag{8}$$

其中,$W_q(t)$ 和 T_q 分别是 q 的校准权重函数和阈值,一般由量子实验人员进行配置.S_q 是积分结果,M_q 是最终测量结果(0 或 1).

回顾近 20 年来对超导量子比特使用的测量控制(测控)设备,我们认为已有的面向超导量子比特的测控系统大体可分为 2 代.第一代超导量子比特的控制系统主要由模拟设备构成,包括任意波形发生器和数据采集卡等.用户通过将所需产生的控制信号上传到 AWG 的内存决定 AWG 的模拟输出进而完成量子操作.数据采集卡将测量结果信号数字化之后,使用软件离线地或硬件实时地分析量子测量结果.这种控制系统的不足有 2 点:1)输出的波形需在量子应用执行前完全准备就绪,并在算法执行启动时开始输出.2)随着量子应用规模的增加,上传的波形数据量急剧增大,降低了量子应用的执行效率,这导致波形的输出难以动态调整.因此,这种测控系统难以实现基于量子比特的测量结果的反馈控制.这种控制模式在之前的工作中得到了广泛的应用,如文献[50,116-118].

针对 2 点不足,尤其是为了满足实验对反馈控制的需求,人们开始研究基于 FPGA 定制控制硬件[119-123],并逐步发展为基于指令集的量子控制系统[16,60,112,124-127].在这类控制系统中,基于 FPGA 的定制控制逻辑在运行过程中生成时序精确的控制信号,用于实时触发模拟控制信号的产生和测量结果的识别.我们将基于定制数字逻辑(尤其是使用指令集)的量子控制系统称为第 2 代量子控制系统.

第 2 代控制系统使用的指令集一般缺少明确的量子计算语义.这些指令语义在较低的硬件层次上,如将在内存某地址中保存若干数量的样本在特定周期之后开始转换成模拟波形输出,如 APS2 系统中的 WAVEFORM 指令[124].但也有部分第 2 代量子控制系统使用的指令集可具备明确的量子计算语义,如可支持 eQASM 指令集的 QuMA 微体系结构.本节将以 QuMA 为例,说明量子指令在控制微体系结构上的运行过程.关于 QuMA 的更多实现细节,请参考文献[60,112].

如图 16 所示,QuMA 微体系结构由经典流水线和量子流水线组成.经典流水线从指令存储器中逐条获取并处理指令.所有辅助经典指令都在经典流水线内完成执行,执行时主要完成经典寄存器的更新和程序流的控制.量子指令取出之后被转发到量子流水线进行处理.量子流水线主要实现 3 部分的功能:

1) 译码、寻址.在获得一条量子指令后,量子流水线需识别该指令对应的操作是时序控制、量子操作或配置量子操作目标寄存器.如果是量子操作,则需要读取量子操作的目标量子比特.

2) 时序控制.量子操作需要以精确的时序施加在量子比特上.时序控制单元用于决定量子操作的发射时间.

3) 模拟数字信号间的转换.量子流水线发射出来的是数字控制信号.这些控制信号需要实时地被转换成模拟脉冲信号,实现对量子比特的控制.如果对应的量子操作是测量,则还需要触发对应的硬件开始识别量子比特测量的结果.

QuMA 微体系结构的存储器至少包含指令存储器、波形存储器和数据存储器 3 部分.指令存储器用来装载量子编译器生成的原程序的二进制格式指令.波形存储器用于存储对应于量子操作的波形

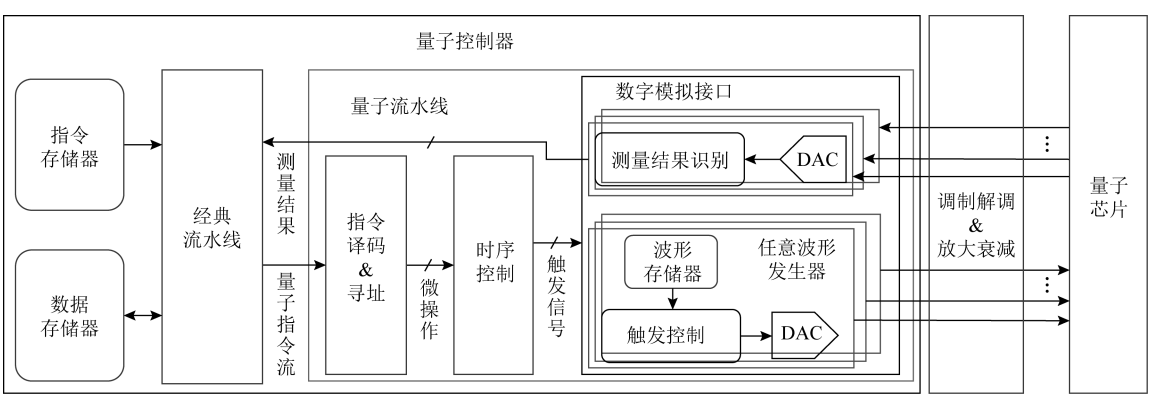


Fig. 16　QuMA quantum control microarchitecture

图 16　QuMA 系列量子控制微体系结构

信息.数据存储器保存量子协处理器运行所需的初始化数据及运行结果.

需要注意的是,经典微体系结构关注的是数据流的移动,而量子控制微体系结构关注的则是如何生成量子操作流.前者操作的是数据,后者操作的是量子操作.这与第 2 节中的讨论是一致的.

我们以图 15 为例说明一个量子程序在 QuMA 微体系结构上的执行过程.为了在硬件上执行量子程序,用户通过 1 台经典计算机(一般称为上位机)将生成的量子指令程序和波形分别上传到指令存储器和波形存储器中,并触发 QuMA 开始运行.QuMA 启动执行之后,经典流水线一条一条地从指令存储器中读取指令.经典指令在经典流水线中完成执行,并在执行过程中更新经典寄存器、读取数据存储器和控制程序流(即更新程序计数器).量子指令被经典流水线转发给量子流水线进行处理.在量子指令译码后,用于更新量子操作目标寄存器的指令在寻址阶段完成执行.时序控制指令被发送到时序控制单元,其效果是为下 1 条量子操作指令预约触发时间.如 QWAIT 30 会给下 1 条量子操作预约指令执行的开始时刻,该时刻与上 1 个被指定的时刻间隔 30 个周期.量子操作指令在译码之后,通过寻址单元获得待操作的量子比特,并指导地址解码单元(如译码器)控制该量子操作进入对应的任意波形发生器或测量结果识别单元中.时序控制单元指定量子操作的触发时刻.之后,量子操作会以数字触发信号的格式在目标时刻被发射给 AWG 或测量识别模块,触发产生对应的模拟脉冲信号施加在量子比特上控制量子状态的翻转,或启动测量结果的识别.测量结果识别单元识别量子比特的测量结果之后,会自动将该结果返回给经典流水线,供后续指令的处理.

8 挑 战

量子计算仍然处于高速发展的阶段,量子系统的不同层次仍然面临着诸多挑战.本节依次简要讨论量子程序设计语言、量子编译器、量子软硬件接口和量子控制微体系结构等层次面临的相关挑战.

8.1 量子程序设计语言

如引言中所述,NISQ 量子算法有 2 个主要特点:1)量子-经典异构计算;2)从应用层对硬件进行低级控制.现有的量子程序设计语言对上述 2 个特点的支持均存在不足.

在对量子-经典异构算法的支持上,主要问题有 2 个:1)对程序流控制结构的忽视,导致语言只能描述单一量子线路,如 Cirq[72] 语言.对于 Q#,Quingo 等独立语言,在语言层面增加对流控结构的支持是比较直观的,而对于基于经典语言的 eDSL 来说,不得不采用元编程的方式向程序中显式地插入流控结构,这会使得流控的描述很不直观,降低算法结构的清晰程度以及代码的可读性.2)难以实现包括基于测量结果的反馈控制在内的量子-经典计算实时在线交互,(例如图 12 中 19～21 行与 30～34 行).导致这一问题的根本原因是在语言层面没有清晰地界定离线与在线这 2 种经典-量子计算交互方式,对于这一挑战,文献[24]开展了一些有意义的探索.

在低级控制机制的支持方面,面临的主要挑战是如何在高级语言中实现与硬件相关的低级操作,同时对高级语言可移植性不产生显著影响.当前,Qiskit,PyQuil 与 Quingo 分别以不同的方式对时序控制进行了支持,但在时序描述的灵活性与通用性方面存在较大的差异.在量子操作波形的描述方面,IBM 提出的 OpenPulse[108] 试图提供一种解决方案,但 OpenPulse 是一种低抽象层次的描述工具,当前还不存在任何一种高级量子语言做出这方面的尝试,因此仍是一个开放性的问题.

8.2 量子中间表示

量子编译器使用的中间表示直接影响到量子编译器的能力边界,以及优化遍的实现难易程度.因此,人们在设计 IR 时,一方面使得编译器方便优化量子程序外,另一方面则力求使其表达能力尽可能完备.现有的量子中间表示主要关注如何表达量子线路、量子线路中使用的二进制控制门(binary-controlled gate)以及量子-经典指令的混合执行等.

除量子-经典异构算法外,量子实验也是一大类重要的 NISQ 应用.而在这些实验中,大多需要显示地调整量子操作的时序及使用到的波形.高级量子程序设计语言在控制量子实验时可提供极高的灵活性,表现出极大的潜力,如在 PycQED[128] 量子测控软件环境中使用的 OpenQL 语言[74].这些实验要求量子编译器能够表达并操作量子操作的低层次特性,如时序及使用的波形等.另一方面,已有研究将量子最优控制引入量子编译优化的流程之中,可提高量子程序执行结果的保真度[77].但现有量子中间表示无法表达量子操作的时序及使用的波形等.如何设计量子编译器的中间表示,以支持量子-经典异构计算并能够表达量子操作的时序和使用的波形,仍然是一个开放的问题.

8.3　量子软硬件接口

指令集体系结构是经典计算机软硬件间的接口,它承接了 2 方面的职能.1)它是硬件的编程接口.经典计算机的软件可通过不同的指令组合充分利用经典计算机提供的计算能力.2)它提供了经典程序在低层次描述上的可移植性.针对特定体系结构生成的二进制程序可运行在支持同一体系结构的不同硬件实现上,如基于 x86_64 指令集的程序可运行在如 Intel Core 系列和 AMD Ryzen 系列等不同的处理器上.

但在量子计算领域,硬件编程接口和低级描述层面的可移植性这 2 个功能被分开在不同的层次上.如第 7 节所述,现有的硬件编程接口一般通过波形和定制指令集构成.虽然 eQASM 具备明确的量子计算语义,但更多的硬件控制系统所使用的指令集并不具备相关支持,如 BBN 公司和 IBM 公司使用的控制系统[16,124].

为了提供量子程序的可移植的低层次描述,人们对量子汇编开展了大量的研究.越来越多的量子汇编语言(如 QASM-HL,OpenQASM,Quil,cQASM)在其规范中引入了函数调用、循环等高级语言特性,这离硬件接口这个目标渐行渐远.

硬件编程接口和低级描述层面的可移植性这 2 种功能的分离,一方面使得具体的量子系统需要额外定制一套低层次编译器或翻译软件将量子汇编翻译为控制硬件可执行的格式.这在工程实现上引入了额外的工作量,增加量子系统维护人员的负担.另一方面,这种分离也导致量子计算硬件系统无法形成统一的接口标准.未遵循统一标准开发的不同量子控制硬件系统在功能上可能不尽相同,低层次的量子汇编程序难以保证在不同硬件上的可执行性.例如,虽然 Quil 支持实时经典操作,但现有许多量子控制硬件系统无法完全支持 Quil 描述的带经典逻辑的量子程序.

如何实现硬件编程接口和可移植的程序低层次描述的统一,是一个尚未解决的问题.这对于 NISQ 时代实现不同量子计算系统的联合发展具有实际的工程意义.

8.4　与历史无关的量子操作波形产生

量子软硬件接口使用指令触发量子操作对应的波形的一个隐含前提是,量子操作的脉冲波形在每次触发时是确定的.如多次施加 $\boldsymbol{R}_x(\pi)$ 门在一个 Transmon 量子比特上,可以固定地使用图 6 左侧所示波形.然而,这个假设对于不同类型的量子比特或不同类型的量子操作而言未必始终成立.如在 Transmon 量子比特上施加 CZ 操作时,一般需要在通量偏置线上短暂地施加一个直流方波信号.该信号从任意波形发生器产生至到达量子芯片的 P_f 端口,一般会经过一个低通滤波器.该滤波器不仅会改变信号的形状,还会产生一个长拖尾.为了抵消信号形变和之前的信号产生的长拖尾的影响,现有的实验中一般会利用软件提前对信号进行预处理,并使用 AWG 输出预处理的信号,使得最终到达 P_f 端口的信号是一个方波信号.这导致 AWG 为每个 CZ 操作预存的波形是不一样的,从而使得基于测量的实时反馈控制难以实现.

这构成了基于指令集的量子控制系统所面临的巨大挑战.为了解决这个问题,研究人员开始研究相关技术使得静态上传到任意波形发生器中的波形是固定的.相关研究主要包含 2 种方法:1)AWG 在输出波形之前,利用无限响应滤波器进行实时的波形预处理,使得量子比特所见波形为预期波形.目前瑞士苏黎世仪器公司生产的高密度任意波形发生器(HDAWG)已实现了该功能.2)使用新型控制波形,使该波形经过滤波器之后可同样地更新量子比特状态,但不会产生长拖尾效应[129].目前 2 种方法在小规模的量子芯片得到了验证.前者在一个波形 500 ns 之后仍然会产生拖尾,而后者生成的受控相位门的保真度仅可达 99.1%[129].能够生成与历史无关的量子控制波形是实现基于指令集并支持基于测量结果反馈控制的控制系统的必要条件.如何实现与历史无关的波形产生是量子控制体系结构面临的一个巨大挑战.

8.5　分布式量子控制微体系结构

现有的量子控制微体系结构中的数字处理器、AWG 和测量识别模块一般通过定制 FPGA 实现.受限于 FPGA 片上存储大小、输入输出管脚数量、指令执行过程中的量子操作发射速率[112]等因素,实验中观察到平均每块 FPGA 可控制的量子比特数量小于 10.已有的量子系统可集成 60 多个量子比特[10],且 NISQ 系统有望在几年内集成几百乃至上千个固态量子比特.因此,使用分布式的量子控制系统是实现 NISQ 系统控制的不二选择.分布式的量子控制系统中,很可能需要在多个微处理器内同时执行多个二进制,协作完成对大量量子比特的控制.分布式的量子控制系统需要将同一个量子应用拆分为多个二进制程序,每个二进制程序用于操作一组量子比特.由于量子算法允许任意量子比特之间的反馈控制,反馈控制不仅需要在控制系统的不同设备之间进行传递信息(主要是测量结果),还需要

实现多个指令流间的周期级同步.如何将一个量子应用拆分为多个二进制程序,支持任意量子比特间的任意反馈并保证指令流之间周期级的同步,仍然是一个有待解决的问题.

9 结 论

随着迈入含噪中尺度量子技术时代,量子软件与量子硬件对接并支持量子-经典异构计算的需求日益明显.通过对比量子计算与经典计算的异同,我们发现量子计算是典型的存内计算.

由于未知的量子状态不可克隆,量子计算在程序设计语言、编译、体系结构等方面比经典计算更加关注对量子操作——而非数据——的描述、优化和产生.这使得量子计算机的系统结构设计未必需要基于冯•诺依曼体系结构.从工程实现的视角,本文介绍了量子-经典异构计算的宏观系统层次结构,讨论了如何组织和管理量子、经典软硬件资源,以支持量子-经典异构计算.本文讨论了量子程序设计语言、量子编译器、量子软硬件接口和量子控制微体系结构的功能定位、关键技术或设计思路和已有的实现,并以 IPE 算法为例介绍了一个量子计算算法从高级程序设计语言描述到硬件执行的整体流程.本文讨论了量子程序设计语言、量子编译器、量子软硬件接口和量子控制微体系结构等层次在支持量子-经典异构计算、时序控制、量子操作波形控制等方面面临的挑战.针对量子计算系统中的不同层次,本文在相应的章节提供了若干参考文献,以帮助有兴趣的读者深入了解对应领域.我们希望本文可以促进人们对 NISQ 时代下量子计算系统整体结构的理解,并激发更多相关研究.

致谢 作者感谢朱晓波教授提供的超导量子比特的扫描电镜照片(图 5).

参 考 文 献

[1] Arute F, Arya K, Babbush R, et al. Quantum supremacy using a programmable superconducting processor [J]. Nature, 2019, 574(7779): 505-510

[2] Zhong Hansen, Wang Hui, Deng Yuhao, et al. Quantum computational advantage using photons [J]. Science, 2020, 370(6523): 1460-1463

[3] Preskill J. Quantum computing and the entanglement frontier [J/OL]. arXiv preprint, arXiv, 2012: 1-18 [2021-05-17]. http://arxiv.org/abs/1203.5813

[4] Guo Chu, Liu Yong, Xiong Min, et al. General-purpose quantum circuit simulator with projected entangled-pair states and the quantum supremacy frontier [J]. Physical Review Letters, 2019, 123(19): 190501

[5] Preskill J. Quantum computing in the NISQ era and beyond [J]. Quantum, 2018, 2: 79-98

[6] Ye Yangsen, Ge Ziyong, Wu Yulin, et al. Propagation and localization of collective excitations on a 24-qubit superconducting processor [J]. Physical Review Letters, 2019, 123(5): 050502

[7] Gong Ming, Chen Mingcheng, Zheng Yarui, et al. Genuine 12-qubit entanglement on a superconducting quantum processor [J]. Physical Review Letters, 2019, 122(11): 110501

[8] Wu Yulin, Bao Wansu, Cao Sirui, et al. Strong quantum computational advantage using a superconducting quantum processor [J]. arXiv preprint arXiv:2106.14734, 2021

[9] Kjaergaard M, Schwartz M E, Braumuller J, et al. Superconducting qubits: Current state of play [J]. Annual Review of Condensed Matter Physics, 2020, 11(1) 369-395

[10] Gong Ming, Wang Shiyu, Zha Chen, et al. Quantum walks on a programmable two-dimensional 62-qubit superconducting processor [J]. Science, 2021, 372(6545): 948-952

[11] Song Chao, Xu Kai, Li Hekang, et al. Generation of multicomponent atomic Schrödinger cat states of up to 20 qubits [J]. Science, 2019, 365(6453): 574-577

[12] Terhal B M. Quantum error correction for quantum memories [J]. Reviews of Modern Physics, 2015, 87(2): 307-346

[13] Peruzzo A, Mcclean J, Shadbolt P, et al. A variational eigenvalue solver on a photonic quantum processor [J]. Nature Communications, 2014, 5(1): 4213-4219

[14] Dobšíček M, Johansson G, Shumeiko V, et al. Arbitrary accuracy iterative quantum phase estimation algorithm using a single ancillary qubit: A two-qubit benchmark [J]. Physical Review A, 2007, 76(3): 030306

[15] Svore K M, Hastings M B, Freedman M. Faster phase estimation [J]. Quantum Information and Computation, 2014, 14(3-4): 306-328

[16] Corcoles A D, Takita M, Inoue K, et al. Exploiting dynamic quantum circuits in a quantum algorithm with superconducting qubits [OL]. [2021-05-17]. http://arxiv.org/abs/2102.01682

[17] Farhi E, Goldstone J, Gutmann S. A quantum approximate optimization algorithm applied to a bounded occurrence constraint problem [J]. arXiv preprint arXiv: 1411.4028, 2014

[18] Yuan Xiao, Endo S, Zhao Qi, et al. Theory of variational quantum simulation [J]. Quantum, 2019, 3: 191-231

[19] Jones N C, Van Meter R, Fowler A G, et al. Layered architecture for quantum computing [J]. Physical Review X, 2012, 2(3): 031007

[20] Van Meter R, Horsman C. A blueprint for building a quantum computer [J]. Communications of the ACM, 2013, 56(10): 84-93

[21] Fu Xiang, Riesebos L, Lao Lingling, et al. A heterogeneous quantum computer architecture [C] //Proc of 2016 ACM Int Conf on Computing Frontiers. New York: ACM, 2016: 323-330

[22] Gambetta J M, Chow J M, Steffen M. Building logical qubits in a superconducting quantum computing system [J]. npj Quantum Information, 2017, 3(1): 2-8

[23] Chong F T, Franklin D, Martonosi M. Programming languages and compiler design for realistic quantum hardware [J]. Nature, 2017, 549(7671): 180-187

[24] Fu Xiang, Yu Jintao, Su Xing, et al. Quingo: A programming framework for heterogeneous quantum-classical computing with NISQ features [J]. arXiv preprint, arXiv: 2009.01686, 2020

[25] Karnaugh M. The map method for synthesis of combinational logic circuits [J]. Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics, 1953, 72(5): 593-599

[26] Nielsen M A, Chuang I L. Quantum Computation and Quantum Information [M]. Cambridge, UK: Cambridge University Press, 2010

[27] Shor P W. Algorithms for quantum computation: Discrete logarithms and factoring [C] //Proc of the 35th Annual Symp on Foundations of Computer Science. Piscataway, NJ: IEEE, 1994: 124-134

[28] DiVincenzo D P. The physical implementation of quantum computation [J]. Fortschritte der Physik, 2000, 48(9-11): 771-783

[29] The Royal Swedish Academy of Sciences. Nobel prize 2012 information: Measuring and manipulating individual quantum systems [EB/OL]. 2012 [2021-05-18]. https://s3. eu-de. cloud-object-storage. appdomain. cloud/kva-image-pdf/assets/globalassets-priser-nobel-2012-fysik-sciback_fy_12.pdf

[30] Nakamura Y, Pashkin Y A, Tsai J S. Coherent control of macroscopic quantum states in a single-Cooper-pair box [J]. Nature, 1999, 398(6730): 786-788

[31] Blais A, Huang Renshou, Wallraff A, et al. Cavity quantum electrodynamics for superconducting electrical circuits: An architecture for quantum computation [J]. Physical Review A—Atomic, Molecular, and Optical Physics, 2004, 69(6): 062320

[32] Koch J, Terri Y M, Gambetta J, et al. Charge-insensitive qubit design derived from the Cooper pair box [J]. Physical Review A—Atomic, Molecular, and Optical Physics, 2007, 76(4): 042319

[33] Knill E, Laflamme R, Milburn G J. A scheme for efficient quantum computation with linear optics [J]. Nature, 2001, 409(6816): 46-52

[34] Qiang Xiaogang, Zhou Xiaoqi, Wang Jianwei, et al. Large-scale silicon quantum photonics implementing arbitrary two-qubit processing [J]. Nature Photonics, 2018, 12(9): 534-539

[35] Qiang Xiaogang, Wang Yizhi, Xue Shichuan, et al. Implementing graph-theoretic quantum algorithms on a silicon photonic quantum walk processor [J]. Science Advances, 2021, 7(9): eabb8375

[36] Raizen M G, Gilligan J M, Bergquist J C, et al. Ionic crystals in a linear Paul trap [J]. Physical Review A, 1992, 45(9): 6493-6501

[37] Nayak C, Simon S H, Stern A, et al. Non-Abelian anyons and topological quantum computation [J]. Reviews of Modern Physics, 2008, 80(3): 1083-1159

[38] Sarma S D, Freedman M, Nayak C. Majorana zero modes and topological quantum computation [J]. npj Quantum Information, 2015, 1(1) : 15001

[39] Loss D, Divincenzo D P. Quantum computation with quantum dots [J]. Physical Review A, 1997, 57(1): 120-126

[40] Kane B E. A silicon-based nuclear spin quantum computer [J]. Nature, 1998, 393(6681): 133-137

[41] Oliveira I, Sarthour J R R, Bonagamba, et al. NMR Quantum Information Processing [M]. Netherlands: Amsterdam: Elsevier, 2011

[42] Weber J R, Koehl W F, Varley J B, et al. Quantum computing with defects [J]. Proceedings of the National Academy of Sciences of the United States of America, 2010, 107(19): 8513-8518

[43] Childress L, Hanson R. Diamond NV centers for quantum computing and quantum networks [J]. MRS Bulletin, 2013, 38(2): 134-138

[44] Saffman M, Walker T G, Mølmer K. Quantum information with Rydberg atoms [J]. Reviews of Modern Physics, 2010, 82(3): 2313-2363

[45] Saffman M. Quantum computing with atomic qubits and Rydberg interactions: Progress and challenges [J]. Journal of Physics B: Atomic, Molecular and Optical Physics, 2016, 49 (20): 202001

[46] Morgado M, Whitlock S. Quantum simulation and computing with tydberg qubits [J]. arXiv preprint, arXiv:2011.03031, 2020

[47] Wu Xiaoling, Liang Xinhui, Tian Yaoqi, et al. A concise review of Rydberg atom based quantum computation and quantum simulation [J]. Chinese Physics B, 2021, 30(2): 020305

[48] Suter D, Mahesh T S. Spins as qubits: Quantum information processing by nuclear magnetic resonance [J]. Journal of Chemical Physics, 2008, 128(5): 052206

[49] Debnath S, Linke N M, Figgatt C, et al. Demonstration of a small programmable quantum computer with atomic qubits [J]. Nature, 2016, 536(7614): 63-66

[50] Riste D, Poletto S, Huang Mengzi, et al. Detecting bit-flip errors in a logical qubit using stabilizer measurements [J]. Nature Communications, 2015, 6(1): 6983

[51] Córcoles A D, Magesan E, Srinivasan S J, et al. Demonstration of a quantum error detection code using a square lattice of four superconducting qubits [J]. Nature Communications, 2015, 6(1): 6979-6988

[52] Paetznick A, Svore K M. Repeat-until-success: Non-deterministic decomposition of single-qubit unitaries [J]. Quantum Information and Computation, 2014, 14(15-16): 1277-1301

[53] Oskin M, Chong F T, Chuang I L. A practical architecture for reliable quantum computers [J]. Computer, 2002, 35(1): 79-87

[54] Metodi T S, Thaker D D, Cross A W, et al. A quantum logic array microarchitecture: Scalable quantum data movement and computation [C] // Proc of the 38th Annual IEEE/ACM Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 2005: 305-316

[55] Thaker D D, Metodi T S, Cross A W, et al. Quantum memory hierarchies: Efficient designs to match available parallelism in quantum computing [C] //Proc of the 33rd Annual Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2006: 378-389

[56] Chi E, Lyon S A, Martonosi M. Tailoring quantum architectures to implementation style: A quantum computer for mobile and persistent qubits [J]. ACM SIGARCH Computer Architecture News, 2007, 35(2): 198-209

[57] Isailovic N, Whitney M, Patel Y, et al. Running a quantum circuit at the speed of data [J]. ACM SIGARCH Computer Architecture News, 2008, 36(3): 177-188

[58] Whitney M G, Isailovic N, Patel Y, et al. A fault tolerant, area efficient architecture for Shor's factoring algorithm [C] //Proc of the 36th Annual Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2009: 383-394

[59] Mariantoni M, Wang Haohua, Yamamoto T, et al. Implementing the quantum von Neumann architecture with superconducting circuits [J]. Science, 2011, 334(6052): 61-65

[60] Fu Xiang, Rol M A, Bultink C C, et al. An experimental microarchitecture for a superconducting quantum processor [C] //Proc of the 50th Annual IEEE/ACM Int Symp on Microarchitecture. New York: ACM, 2017: 813-825

[61] Fu Xiang. Quantum control architecture: Bridging the gap between quantum software and hardware [D]. Delft: Delft University of Technology, 2018

[62] Knill E. Conventions for quantum pseudocode [R]. Los Alamos: Los Alamos National Laboratory, 1996

[63] Fowler A G. Minimum weight perfect matching of fault-tolerant topological quantum error correction in average $O(1)$ parallel time [J]. Quantum Information and Computation, 2015, 15(1-2): 145-158

[64] Tomita Y, Svore K M. Low-distance surface codes under realistic quantum noise [J]. Physical Review A, 2014, 90(6): 062320

[65] Stone J E, Gohara D, Shi Guochun. OpenCL: A parallel programming standard for heterogeneous computing systems [J]. Computing in Science & Engineering, 2010, 12(3): 66-73

[66] IBM. Qiskit: An open-source SDK for working with quantum computers at the level of pulses, circuits, and algorithms [OL]. (2020) [2021-05-17]. https://github.com/QISKit

[67] ORIGIN. QPanda 2 [OL]. [2021-05-17]. https://github.com/OriginQ/QPanda-2

[68] Heim B, Soeken M, Marshall S, et al. Quantum programming languages [J]. Nature Reviews Physics, 2020, 2(12): 709-722

[69] Svore K, Geller A, Troyer M, et al. Q# enabling scalable quantum computing and development with a high-level dsl [C] //Proc of the Real World Domain Specific Languages Workshop. New York: ACM, 2018: 1-10

[70] Abhari A J, Faruque A, Dousti M J, et al. Scaffold: Quantum programming language [R]. Princeton: Princeton University Department of Computer Science, 2012

[71] Rigetti. A Python library for quantum programming using Quil [OL]. [2021-05-17]. https://github.com/rigetti/pyquil

[72] Google. Cirq: A Python library for writing, manipulating, and optimizing quantum circuits and running them against quantum computers and simulators [OL]. [2021-05-17]. https://github.com/quantumlib/Cirq

[73] Steiger D S, Häner T, Troyer M. ProjectQ: An open source software framework for quantum computing [J]. Quantum, 2018, 2: 49-61

[74] Khammassi N, Ashraf I, Someren J, et al. OpenQL: A portable quantum programming framework for quantum accelerators [J]. arXiv preprint, arXiv:2005.13283, 2020

[75] Green A S, Lumsdaine P L F, Ross N J, et al. Quipper: A scalable quantum programming language [C] //Proc of the 34th ACM SIGPLAN Conf on Programming Language Design and Implementation. New York: ACM, 2013: 333-342

[76] Svore K M, Aho A V, Cross A W, et al. A layered software architecture for quantum computing design tools [J]. Computer, 2006, 39(1): 74-83

[77] Shi Yunong, Leung N, Gokhale P, et al. Optimized compilation of aggregated instructions for realistic quantum computers [C] //Proc of the 24th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2019: 1031-1044

[78] JavadiAbhari A, Patil S, Kudrow D, et al. ScaffCC: Scalable compilation and analysis of quantum programs [J]. Parallel Computing, 2015, 45: 2-17

[79] Häner T, Steiger D S, Svore K, et al. A software methodology for compiling quantum programs [J]. Quantum Science and Technology, 2018, 3(2): 020501

[80] Amy M, Gheorghiu V. staq—A full-stack quantum processing toolkit [J]. Quantum Science and Technology, 2020, 5(3): 034016

[81] Steiger D S. Software and algorithms for quantum computing [D]. Zurich: ETH Zurich, 2018

[82] Zhang Yu, Deng Haowei, Li Quanxi, et al. Optimizing quantum programs against decoherence: Delaying qubits into quantum superposition [C] //Proc of 2019 Int Symp on Theoretical Aspects of Software Engineering (TASE). Piscataway, NJ: IEEE, 2019: 184-191

[83] Lattner C, Adve V. LLVM: A compilation framework for lifelong program analysis transformation [C] //Proc of the 2004 Int Symp on Code Generation and Optimization (CGO'04). Piscataway, NJ: IEEE, 2003: 75-86

[84] Lattner C, Amini M, Bondhugula U, et al. MLIR: A compiler infrastructure for the end of Moore's law [J]. arXiv preprint, arXiv:2002.11054, 2020

[85] Litteken A, Fan Y C, Singh D, et al. An updated LLVM-based quantum research compiler with further OpenQASM support [J]. Quantum Science and Technology, 2020, 5(3): 034013

[86] Microsoft. QIR [OL]. [2021-05-17]. https://github.com/microsoft/qsharp-language/tree/main/Specifications/QIR

[87] Mccaskey A, Nguyen T. A MLIR dialect for quantum assembly languages [OL]. [2021-05-17]. http://arxiv.org/abs/2101.11365

[88] Ross N J, Selinger P. Optimal ancilla-free Clifford approximation of z-rotations [J]. Quantum Information Computation, 2016, 16(11-12): 901-953

[89] Amy M, Maslov D, Mosca M, et al. A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits [J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2013, 32(6): 818-830

[90] Heyfron L E, Campbell E T. An efficient quantum compiler that reduces T count [J]. Quantum Science and Technology, 2018, 4(1): 015004

[91] Toffoli T. Reversible computing[G]//LNCS 85: Proc of Int Colloquium on Automata, Languages, and Programming. Berlin: Springer, 1980: 632-644

[92] Miller D M, Maslov D, Dueck G W. A transformation based algorithm for reversible logic synthesis [C] //Proc of the 40th Annual Design Automation Conf. New York: ACM, 2003: 318-323

[93] Janzing D, Wocjan P, Beth T. "Non-identity-check" is QMA-complete [J]. International Journal of Quantum Information, 2005, 3(3): 463-473

[94] Nam Y, Ross N J, Su Yuan, et al. Automated optimization of large quantum circuits with continuous parameters [J]. npj Quantum Information, 2018, 4(1): 1-12

[95] Zhou Xiangzhen, Li Shanjiang, Feng Yuan. Quantum circuit transformation based on simulated annealing and heuristic search [J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2020, 39(12): 4683-4694

[96] Zulehner A, Paler A, Wille R. Efficient mapping of quantum circuits to the IBM QX architectures [C] //Proc of 2018 Design, Automation Test in Europe Conf Exhibition (DATE). Piscataway, NJ: IEEE, 2018: 1135-1138

[97] Siraichi M Y, Santos V F dos, Collange S, et al. Qubit allocation [C] //Proc of the 2018 Int Symp on Code Generation and Optimization. New York: ACM, 2018: 113-125

[98] Li Gushu, Ding Yufei, Xie Yuan. Tackling the qubit mapping problem for NISQ-era quantum devices [C] //Proc of the 24th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2019: 1001-1014

[99] Lao Lingling, van Wee B, Ashraf I, et al. Mapping of lattice surgery-based quantum circuits on surface code architectures [J]. Quantum Science and Technology, 2018, 4(1): 015005

[100] Lao Lingling, van Someren H, Ashraf I, et al. Timing and resource-aware mapping of quantum circuits to superconducting processors [J/OL]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2021.[2021-05-17]. https://ieeexplore.ieee.org/abstract/document/9349092?casa_token=jRwPmAdJwtsAAAAA:PXYYvPevAYLKl2rcrclAIWjcZ8mNEBG-bIn_lRy4baqAYcp30ja7TVJOcqzaBMAZR5lWktCHAlA

[101] Li Shanjiang, Zhou Xiangzhen, Feng Yuan. Qubit mapping based on subgraph isomorphism and filtered depth-limited search [J/OL]. IEEE Transactions on Computers, 2020. [2021-05-17]. https://ieeexplore.ieee.org/abstract/document/9194320?casa_token=LL5gmdcZBr4AAAAA:ZdtrFPGX67n0MtzTdujAs3HjF_NFJqwamnAM4jDvtFTtnWAF_EIiowIiV62afqGJ0p4cYIhpJ_0

[102] De Fouquieres P, Schirmer S G, Glaser S J, et al. Second order gradient ascent pulse engineering [J]. Journal of Magnetic Resonance, 2011, 212(2): 412-417

[103] Smith R S, Curtis M J, Zeng W J. A practical quantum instruction set architecture [OL]. [2021-05-17]. http://arxiv.org/abs/1608.03355

[104] Smith R S, Peterson E C, Skilbeck M G, et al. An open-source, industrial-strength optimizing compiler for quantum programs [J]. Quantum Science and Technology, 2020, 5(4): 044001

[105] Hietala K, Rand R, Hung S H, et al. A verified optimizer for quantum circuits [J]. Proceedings of the ACM on Programming Languages, 2021, 5(POPL): 1-29

[106] Killoran N, Izaac J, Quesada N, et al. Strawberry fields: A software platform for photonic quantum computing [J]. Quantum, 2019, 3: 129-155

[107] Sivarajah S, Dilkes S, Cowtan A, et al. t|ket⟩: A retargetable compiler for NISQ devices [J]. Quantum Science and Technology, 2020, 6(1): 014003

[108] McKay D C, Alexander T, Bello L, et al. Qiskit backend specifications for OpenQASM and OpenPulse experiments [J]. arXiv preprint, arXiv:1809.03452, 2018

[109] Cross A W, Bishop L S, Smolin J A, et al. Open quantum assembly language [J]. arXiv preprint, arXiv:1707.03429, 2017

[110] Liu Shusen, Wang Xin, Zhou Li, et al. Q|SI⟩: A quantum programming environment [G] //LNCS 11180: Proc of Symp on Real-Time and Hybrid Systems. Berlin: Springer, 2018: 133-164

[111] Khammassi N, Guerreschi G G, Ashraf I, et al. cQASM v1.0: Towards a common quantum assembly language [J]. arXiv preprint, arXiv:1805.09607, 2018

[112] Fu Xiang, Riesebos L, Rol M A, et al. eQASM: An executable quantum instruction set architecture [C] //Proc of 2019 IEEE Int Symp on High Performance Computer Architecture (HPCA). Piscataway, NJ: IEEE, 2019: 224-237

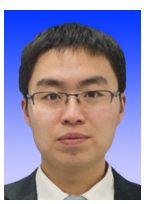
[113] Sagastizabal R, Bonet-Monroig X, Singh M, et al. Experimental error mitigation via symmetry verification in a variational quantum eigensolver [J]. Physical Review A, 2019, 100(1): 010302

[114] Bultink C C, O'Brien T E, Vollmer R, et al. Protecting quantum entanglement from leakage and qubit errors via repetitive parity measurements [J]. Science Advances, 2020, 6(12): eaay3050
[115] Fu Xiang. CC-Light eQASM architecture specification [R]. Delft: Delft University of Technology, 2020
[116] Dicarlo L, Chow J M, Gambetta J M, et al. Demonstration of two-qubit algorithms with a superconducting quantum processor [J]. Nature, 2009, 460(7252): 240-244
[117] Kelly J, Barends R, Fowler A G, et al. State preservation by repetitive error detection in a superconducting quantum circuit [J]. Nature, 2015, 519(7541): 66-69
[118] Song Chao, Xu Kai, Liu Wuxin, et al. 10-Qubit entanglement and parallel logic operations with a superconducting circuit [J]. Physical Review Letters, 2017, 119(18): 180511
[119] Ristè D, Bultink C C, Lehnert K W, et al. Feedback control of a solid-state qubit using high-fidelity projective measurement [J]. Physical Review Letters, 2012, 109(24): 240502
[120] De Lange G, Riste D, Tiggelman M J, et al. Reversing quantum trajectories with analog feedback [J]. Physical Review Letters, 2014, 112(8): 080501
[121] Riste D, Dukalski M, Watson C A, et al. Deterministic entanglement of superconducting qubits by parity measurement and feedback [J]. Nature, 2013, 502(7471): 350-354
[122] Reilly D J. Engineering the quantum-classical interface of solid-state qubits [J]. npj Quantum Information, 2015, 1(1): 15011
[123] Salathé Y, Kurpiers P, Karg T, et al. Low-latency digital signal processing for feedback and feedforward in quantum computing and communication [J]. Physical Review Applied, 2018, 9(3): 034011
[124] Ryan C A, Johnson B R, Ristè D, et al. Hardware for dynamic quantum computing [J]. Review of Scientific Instruments, 2017, 88(10): 104703
[125] Hu Ling, Ma Yuwei, Cai Weizhou, et al. Demonstration of quantum error correction and universal gate set on a binomial bosonic logical qubit [J]. arXiv preprint, arXiv: 1805.09072, 2018
[126] Ofek N, Petrenko A, Heeres R, et al. Extending the lifetime of a quantum bit with error correction in superconducting circuits [J]. Nature, 2016, 536(7617): 441-445
[127] Xiang Liang, Zong Zhiwen, Sun Zhenhai, et al. Simultaneous feedback and feedforward control and its application to realize a random walk on the bloch sphere in an xmon-superconducting-qubit system [J]. Physical Review Applied, 2020, 14(1): 014099
[128] Rol M A, Dickel C, Asaad S, et al. DiCarloLab PycQED_py3 [OL]. (2019) [2021-05-18]. https://github.com/DiCarloLab-Delft/PycQED_py3, 2019.18
[129] Rol M A, Battistel F, Malinowski F K, et al. Fast, high-fidelity conditional-phase gate exploiting leakage interference in weakly anharmonic superconducting qubits [J]. Physical Review Letters, 2019, 123(12): 120502

Fu Xiang, born in 1990, PhD, assistant professor. His main research interests include quantum computer architecture, quantum programming languages and quantum compilation.
付 祥,1990 年生.博士,助理研究员.主要研究方向为量子计算体系结构、量子程序设计语言及量子编译.

Zheng Yuzhen, born in 1996. Master candidate. His main research interests include quantum computer architecture, quantum compilation and quantum computing algorithm. (yuzhen.zheng@quanta.org.cn)
郑宇真,1996 年生.硕士研究生.主要研究方向为量子计算体系结构、量子编译以及量子计算算法.

Su Xing, born in 1989, PhD, assistant professor. His main research interests include high performance computing, compiling optimization and quantum programming language.
苏 醒,1989 年生.博士,助理研究员.主要研究方向为高性能计算、编译优化与量子程序设计语言.



Yu Jintao, born in 1987. PhD, assistant. His main research interests include in-memory computing, quantum compilation, and domain-specific languages.
于锦涛,1987 年生.博士,助教.主要研究方向为存内计算、量子编译、领域专用语言.

Xu Weixia, born in 1963, PhD, professor. Member of CCF. His main research interests include computer architecture, high-performance microprocessor design, artificial intelligence, and neuromorphic computation.
徐炜遐,1963 年生.博士,研究员.CCF 会员.主要研究方向为计算机体系结构、高性能微处理器设计、人工智能、神经形态计算.

Wu Junjie, born in 1981, PhD, professor. Distinguished member of CCF. His main research interests include quantum computing and quantum chip.
吴俊杰,1981 年生.博士,研究员.CCF 杰出会员.主要研究方向为量子计算、量子芯片.