计算机科学
COMPUTER SCIENCE

基于“嵩山”超级计算机系统下HHL算法的模拟实现

谢浩山, 刘晓楠, 赵晨言, 刘正煜

引用本文

谢浩山, 刘晓楠, 赵晨言, 刘正煜.基于“嵩山”超级计算机系统下HHL算法的模拟实现[J]. 计算机科学, 2023, 50(6): 74-80.

XIE Haoshan, LIU Xiaonan, ZHAO Chenyan, LIU Zhengyu. Simulation Implementation of HHL Algorithm Based on Songshan Supercomputer System [J]. Computer Science, 2023, 50(6): 74-80.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.220500108

基于“嵩山”超级计算机系统下 HHL 算法的模拟实现

谢浩山[1]　刘晓楠[1]　赵晨言[1]　刘正煜[2]
1 数学工程与先进计算国家重点实验室(信息工程大学)　郑州 450000
2 郑州大学计算机与人工智能学院　郑州 450000
(15237933568@163.com)

摘　要　量子计算是一种遵循量子力学规律来调控量子信息单元进行计算的新型计算模式,而量子算法由一系列量子门组合而成,其实现形式为量子线路。量子线路是对量子比特进行操作的线路,以量子比特为基本的存储单元,将量子逻辑门连接在一起来实现特定的计算功能。文中在“嵩山”超级计算机上利用 MPI+OpenMP 混合并行编程模型,实现了将大规模量子线路拆分到不同节点上进行构建,加快了线路的构建速度,并且在 CPU 集群系统上具有良好的可拓展性。针对节点间通信问题,设计了序列化和反序列化函数,以保证节点间数据的传输,并且根据各节点所分配任务量间存在的指数级差异,设计了一种拆分任务量、各节点轮循处理的优化方式,实现了节点间的负载均衡。最后在超级计算机 CPU 集群上成功实现了大规模的量子相位估计线路的构造,相较于单节点取得了 8.63 的加速比,并通过 HHL 算法验证了所设计的并行相位估计子模块的正确性,为大规模 HHL 算法在超算平台上的实现提供了参考。

关键词:量子相位估计;CPU 集群;MPI;HHL 算法;负载均衡
中图法分类号　TP385

Simulation Implementation of HHL Algorithm Based on Songshan Supercomputer System

XIE Haoshan[1],LIU Xiaonan[1],ZHAO Chenyan[1] and LIU Zhengyu[2]
1 State Key Laboratory of Mathematical Engineering and Advanced Computing,Information Engineering University,Zhengzhou 450000,China
2 School of Computer and Artificial Intelligence,Zhengzhou University,Zhengzhou 450000,China

Abstract　Quantum computing is a new computing mode that follows the laws of quantum mechanics to regulate and control quantum information units to perform calculations,while the quantum algorithm is composed of a series of quantum gates whose realized form is quantum circuit. Quantum circuits are circuits that operate on qubits,using qubits as the basic storage unit to connect quantum logic gates to achieve specific computing functions. This paper uses the MPI+OpenMP hybrid parallel programming model on the “Songshan” supercomputer to realize the construction of large-scale quantum circuits by splitting them into different nodes,which speeds up the construction of circuits. For the communication problem between nodes,serialization and deserialization functions are designed to ensure the transmission of data between nodes. Aiming at the exponential difference in the amount of tasks allocated by each node,an optimized method of splitting the task amount and round-robin processing of each node is designed to achieve load balance between nodes. Finally,the construction of a large-scale quantum phase estimation circuit is successfully implemented on the supercomputer CPU cluster. Compared with a single node,the speedup ratio of 8.63 is achieved. The HHL algorithm is used to verify the correctness of the designed parallel phase estimation sub-module,which provides a reference for the implementation of large-scale HHL algorithm on the supercomputing platform.

Keywords　Quantum phase estimation,CPU cluster,MPI,HHL algorithm,Load balancing

1　引言

量子计算具有经典计算技术难以企及的并行计算能力和信息携带量,在某些计算任务上,相比经典计算,量子计算已经展现出巨大的优势。1985 年,Deutsch[1] 提出了量子图灵机模型,并且首次验证了量子计算的并行性。1994 年 Shor[2] 基于 Fourier 变换提出了大数质因子分解算法,该算法利用量子计算的并行性可以快速分解出大数的质因子。1996,Grover[3] 提出了量子搜索算法,能够对无结构数据进行二次加速。2009 年,Harrow 等[4] 利用哈密顿量模拟和相位估计等技术首次提出了求解线性方程组的量子算法,该算法常被研究人员称为 HHL 算法。HHL 算法在求解线性方程组方面

到稿日期:2022-05-12　返修日期:2022-10-10
基金项目:国家自然科学基金(61972413,61701539)
This work was supported by the National Natural Science Foundation of China(61972413,61701539).
通信作者:刘晓楠(prof.liu.xn@foxmail)

相比经典算法具有指数级的加速，已被广泛应用于量子机器学习领域。这些量子算法在特定问题上相比经典算法有很大的优势。

在经典计算机上进行量子计算模拟时，所需的时间和空间开销随模拟比特数的增长呈指数级增加，因此其在单个处理器上只能模拟小规模的量子线路。量子线路拆分技术在许多研究工作中已经被证明是缩小问题规模和利用多设备并行计算的可行和有效方法[5]。文献[6]将量子电路划分成可以独立模拟的子电路，然后重新组合以获得所需的结果，并通过实验完成了 43～49 比特线路的切割，量子线路的拆分扩大了可模拟量子比特的规模并加快了任务的处理速度。文献[7]提出了一种动态规划算法，利用该算法可以找到最优的划分方案将目标线路分为 3 个部分，使模拟更易于优化和并行化，从而提升了求解时间的性能。拆分后的子线路分摊了计算整体线路的计算量与内存，增加了可模拟电路的深度。大规模的量子计算模拟中，使用高性能集群能够很好地分摊指数级的存储开销，各个处理器可以并行处理仿真时指数级的运算[8]。超级计算机则为量子计算的模拟提供了合适的硬件平台。文献[9]利用多进程和多线程在超级计算机上模拟了 33 比特规模的线路，其设计模式大大提高了通信比，能够模拟更大规模的线路。2017 年，Hner 等[10]在超级计算机上模拟了 45 比特的量子电路，使用了 8192 个节点。文献[11]在 4096 核超级计算机上用 C++编程语言实现了模拟器，通过 MPI 实现了进程间的通信，对 QAOA 算法进行了大规模模拟，取得了良好的加速效果。基于超算集群的量子计算模拟主要涉及两个关键问题：任务拆分和不同节点间的通信[12]，即如何将量子线路拆分为多个子线路并将其分配到不同的节点上进行计算，以及如何实现节点间的信息传输并最大限度地降低通信开销。

本文针对 HHL 算法的子模块量子相位估计设计了在不同节点下进行线路构建的方法。利用高级编程语言 C++和量子编程软件开发包 Qpanda2.0 实现算法的具体功能以及量子线路的构建，并将其移植到“嵩山”超级计算机平台上，分析该算法中可以并行计算的热点部分，针对可分割的量子线路，将其分解成若干独立的子任务并分配给不同的节点处理。文中利用 MPI+OpenMP 混合编程模型来实现任务粗粒度划分、节点间通信和进程内任务的细粒度划分，进而在“嵩山”实验平台上实现了在不同节点间进行大规模的量子线路构建，并针对不同节点间处理器和内存资源利用不充分导致的资源浪费和时间消耗问题进行了负载均衡的优化。

本文第 2 节介绍了“嵩山”超级计算机的架构以及 MPI+OpenMP 这种能够结合分布式内存结构和共享式内存结构的混合编程模式；第 3 节介绍了 HHL 的算法原理和算法中子程序相位估计模块的线路构成，分析了如何将线路拆分到集群的不同节点上构建以及如何处理节点间通信传输并通过实验进行验证；第 4 节介绍了大规模量子相位估计验证分析，设计了一种轮循节点的优化方式，使各个节点的资源得到充分利用，实现了负载均衡，并将其应用到 HHL 算法中，通过 HHL 算法来实现解线性方程组问题，利用解的正确性来验证本文所设计的并行模式的可靠性。

2 嵩山超级计算机体系结构

嵩山超级计算机平台的构筑体系是一种基于超融合自适应并行处理体系的超算平台架构，该体系包含混合架构的计算子系统、高速互联的网络子系统以及海量分布式统一存储子系统，其核心计算芯片由海光 1 号 X86 处理器和海光 DCU 1 号加速器构成。在软件方面，嵩山超级计算机包括完善的编程模型、编译系统、并行开发运行资源的应用开发基础环境，提供了适应算法、数据结构和体系结构均可变的协同自动调优环境。

2.1 单节点架构

单节点内基于混合架构的计算子系统采用处理器+加速器结构，其架构为国产海光一号 X86 处理器和 DCU 加速器的高性能计算节点，单节点内存容量为 128 GB/256 GB，并且采用的海光一号 X86 处理器包含 32 核，稳定运行 3.0GHz 以上，其核心数、访存性能和 I/O 扩展能力十分先进。数据通信方面分别利用 CPU-Direct 和 DCU-Direct 来提升 CPU 跨节点通信的均衡性和 DCU 跨节点的通信性，其网络子系统如图 1 所示。

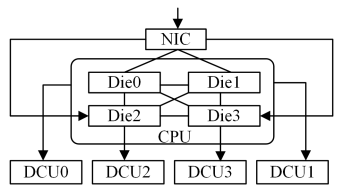

图 1 网络子系统通信

Fig. 1 Network subsystem communication

2.2 MPI+OpenMP 混合编程模型

目前两种重要的并行编程模型为共享内存编程模型和消息传递编程模型，OpenMP 和 MPI 分别为这两种编程模型的代表。OpenMP 程序的运行模式为 Fork-Join：程序以单线程开始执行，并在某些程序点派生出若干额外线程，而这些线程在执行完并行代码后又合并在一起。该编程模型能够有效利用 CPU 核心的计算能力，是一种基于线程的细粒度并行模型。但其缺陷在于其只能作用于共享内存中，这意味着如果利用 OpenMP 来实现某种程序的并行执行，则这种程序的执行规模将被限制在单台计算机上，无法实现程序的更大规模运行，因此这种共享内存编程模型的可扩展性较差[13]。MPI 消息传递并行编程模型是在集群分布式存储上十分流行的一种编程模型，其实现了进程与进程之间的通信，具有很好的可移植性，但是编程较为困难，对设计者要求较高[14]。MPI 并行程序由多个进程执行，每个进程均可执行不同的代码，然后通过显式地发送和接收消息来实现进程间的数据交换，因为每个进程均有独立的地址空间，其相关数据变量对其他进程不可见，因此必须通过消息传递操作实现进程间的交互，是一种基于粗粒度的进程级并行。MPI+OpenMP 混合并行编程模型能够充分发挥两种并行编程模型的优势，实现程序的更理想化并行[15]。这种分级的并行模型结合了分布式内存

结构和共享式内存结构的优势，通过在节点内使用共享存储模型，在节点外采用消息传递模型对并行程序进行多级求解，提供了节点间和节点内的两级并行，实现了进程级的粗粒度并行和线程级的细粒度并行，其混合模型的结构如图 2 所示。

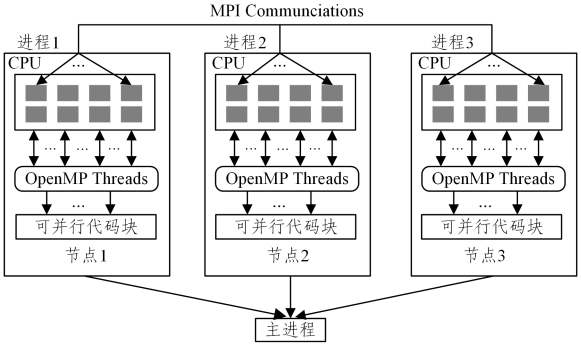

图 2 MPI+OpenMP 混合编程模型

Fig. 2 MPI+OpenMP hybrid programming model

3 HHL 算法及其子模块

在求解线性方程组方面，HHL 算法与经典算法相比在特定情况下具有指数加速效应，在天气预报、经济学、生物工程、计算科学等领域有着重要的意义。HHL 算法线路图如图 3 所示。

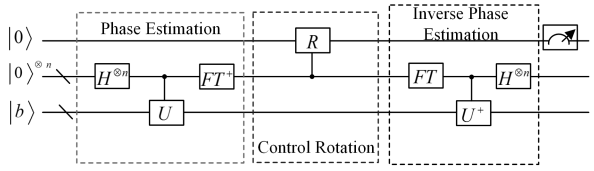

图 3 HHL 算法线路图

Fig. 3 HHL algorithm circuit diagram

HHL 算法线路图主要包含 3 个子模块：

(1)量子相位估计[16]对初始比特应用相位估计子程序，这是在特定基上分解量子态的一般步骤。初始化量子态$|b\rangle$，此时整个系统的初始状态为$|0\rangle^{\otimes n}|b\rangle$。当相位估计子程序结束后，此时整个系统的状态为$\sum_j \beta_j |\lambda_j\rangle|\mu_j\rangle$。

(2)受控旋转。该操作是为了实现$|\lambda_j\rangle \rightarrow \lambda_j^{-1}|\lambda_j\rangle$。

(3)逆量子相位估计。利用逆量子相位估计，将$|\lambda_j\rangle$所在的寄存器还原为$|0\rangle$，此时整个系统的状态为：

$$\sum_j \left(\sqrt{1-\frac{c^2}{\lambda_j^2}}|0\rangle+\frac{c}{\lambda_j}|1\rangle\right)\beta_j|0\rangle|\mu_j\rangle$$

其中，量子相位估计模块的作用就是快速估计一个酉矩阵特征值$e^{2\pi i\varphi}$的相位φ，由于酉矩阵的特征值都是模为 1 的复数，因此对酉矩阵而言，其特征值和相位基本是对等的。相位估计主要由两个阶段实现：1)提取特征值的相位置于量子态的概率辐中；2)利用量子傅里叶逆变换实现将概率辐中的相位置于量子态的基态中，最终输出估计的相位，由相位可以进一步求出输入矩阵的特征值。假设对于φ的近似估计精度为2^{-n}，文献[17]提供了一个等式：$t=n+\lceil \log\left(2+\frac{1}{2\varepsilon}\right)\rceil$，其中 t 是相位估计算法实际输出的比特数，n 为寄存器的比特数，该等式代表在第一个寄存器中使用 n 个量子位来保证输出的准确性，n 越大精度越高，冗余比特 $t-n$ 的作用是提高成功输出高精度的相位值的概率。量子相位估计的线路结构如图 4 所示。

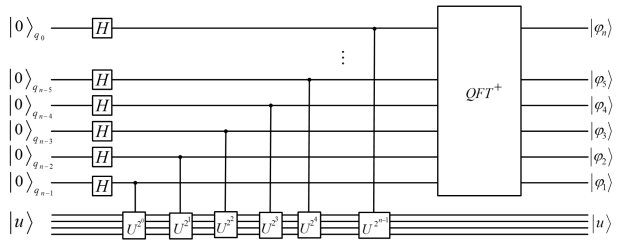

图 4 量子相位估计线路图

Fig. 4 Quantum phase estimation circuit diagram

由图 4 可以观察到相位估计主要由 Hadamard 变换、受控相位变换和量子傅里叶逆变换 3 部分组成，其中耗费时间较多的是受控相位门构建。当构建超过 10 比特的量子线路时，随着模拟比特数的增加，其构建受控相位门的操作呈指数级增加，这就造成所消耗的内存和运行时间呈指数级增加，因此在单个节点内进行线路的构建，所模拟的规模受到了极大的限制。

本节致力于将所提算法从单节点体系结构转移到分布式体系结构以达到扩大模拟规模且提升速度的目的，这需要完全重写部分模拟代码，设计通信模式，处理串行程序和并行程序之间的关系，并且需要解决跨平台验证问题。本文在单节点共享内存体系上通过 C++高级编程语言来模拟相位估计算法的具体实现，该过程包括实现量子线路的构建、状态的转换以及结果的处理。在实现串行执行的基础上，尝试将量子线路拆分为不同的子线路，以缩小任务规模，进而部署在多节点上执行。针对量子线路拆分问题，所选取的进行拆分的线路模块应当是该部分线路的构造较为耗费时间并且对该部分线路的操作不影响其他线路的构建，相对独立，保证线路拆分到不同节点上执行时不产生相互依赖。其次需要处理节点间通信问题，即从进程将分配的子线路构建完成后，如何将数据完整地发送到主进程所在的节点以及保存在主节点设置的缓冲区中。当主进程所在节点接收到全部数据时，如何按照顺序将缓冲区中的数据取出并对子线路进行拼接。最后需要对没有设置并行执行的模块进行处理，将其全部交给某个节点串行执行，进而构造完整、正确的量子线路。本文所设计的线路拆分模式可应用至其他量子算法的线路拆分与构建，在经典模拟的基础上，将量子线路拆分到不同的节点进行构建，进而加快线路的构建速度，提升算法的执行效率。其 MPI+OpenMP 混合编程思路如下：

(1)在单节点中进行量子线路构建的加速，只能通过 OpenMP 编程对适合并行的模块进行多线程操作，其加速效果受限于节点的核心数与内存。进一步的加速可以通过设计减少频繁创建、销毁线程的过程中对系统资源的消耗，该方式最大的局限在于只能在共享内存下使用，没有可拓展性。因此，可以先处理节点间任务的并行，随后在节点内再设计合适的并行代码块。

(2)利用 MPI 实现多节点任务处理的关键在于如何进行任务拆分和分配以及节点间的通信。本文代码设计根据进程号 rank、总进程数 proces、总量子比特 nums 和每个进程分配的量子比特数 proce_each 来限定每个节点所分配的任务执行区间，各个节点上的任务分配完毕后，针对节点内可以并行处理的代码块采用 OpenMP 进行多线程处理，实现节点间和节点内粗细粒度并行。

(3)针对各个节点资源利用不充分的问题，改进了任务的分配方式，采用轮循节点的方式，将某个节点上的任务量轮循地分配给空闲的节点，充分利用了节点的资源，并且解决了数据序列化和反序列化造成的时间消耗问题。

实验首先分析线路的构建过程，采用对受控量子比特进行平均分配的方法将不同的量子比特分配给各个节点处理，以 0 号进程为主进程，负责构建 H 门、接收从进程发送的数据并组装总线路。从进程负责将构建的子线路发送给 0 号进程。因为 QPanda2.0 量子编程软件使用的库函数 QCircuit 不是拷贝不变(Trivially Copyable)类型，所以如果各个进程不针对构造好的线路进行处理，数据就无法在进程间传递。进程间通信传递的对象是有严格要求的，除了基本数据类型，其他对象若想传递，必须进行可序列化。因此必须对构造好的线路进行序列化，将其转化为二进制数据后发送给主进程。主进程针对接收的数据进行反序列化，将二进制数据还原为 QCircuit 线路对象进行组装。故针对通信问题需要构建序列化函数 serialize，将构建好的线路转化为 byte 序列发送到主进程，反序列化函数 deserialize 将接收到的 byte 序列还原为线路对象。因为比特数是被平均分配给不同的节点处理，而比特索引从小到大，所以不同节点上处理的任务量不同，发送给主进程的数据量也不相同。因此可以利用 MPI 提供的函数 MPI_Gatherv()收集不同长度的数据块。该函数允许每个进程发送的数据块长度不同，并且主进程可以任意排列数据块在接收缓存中的位置。整体代码设计完毕后，尝试对受控比特数为 8 的量子相位估计算法在 4 个节点下进行实验测试，执行完毕后主进程成功组装出完整、正确的线路。文中提供了两个节点构造的部分子线路，如图 5、图 6 所示。

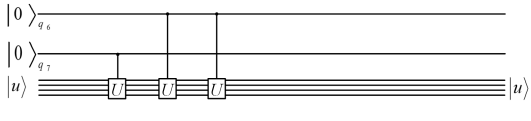

图 5　6～7 Quibts 子线路

Fig. 5　6～7 Quibts sub circuit diagram

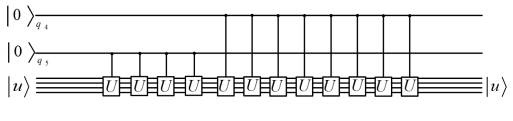

图 6　4～5 Quibts 子线路

Fig. 6　4～5 Quibts sub circuit diagram

当各个节点上的进程把构造好的线路发送给主进程后，主进程针对发送来的数据进行反序列化，将其转化为构造好的线路进行组装，组装后的线路如图 7 所示。

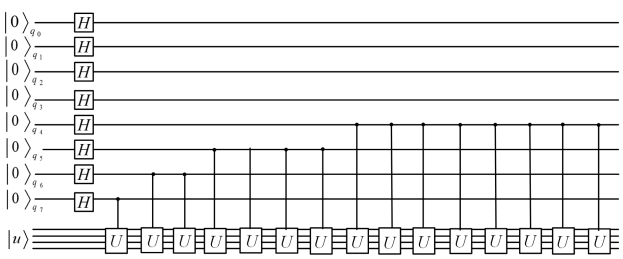

图 7　4～7 Quibts 线路图

Fig. 7　4～7 Quibts circuit diagram

从图 7 可以发现，实验成功地在不同节点上对子线路进行了构建，并通过主进程拼接了正确的量子线路。总线路的成功构建，证明了实验的可行性。随后在“嵩山”超级计算机系统上进行大规模的算法实验，以 25 比特为基础，进行了 25 比特以上的一系列实验。首先在单节点进行实验，计算了构建相位门所消耗的时间，进一步将其部署在 4 个节点上，测得构建相位门需要的时间，结果如表 1 所列。

表 1　25～30 Quibts 耗时对比

Table 1　25 to 30 Qubits time consuming comparison

(单位：s)

Quibts	Single node time/s	Four nodes time/s
25	78.674	119.524
26	185.263	243.526
27	384.563	471.256
28	856.251	1 036.224
29	1 952.438	2 137.125
30	4 152.451	4 456.243

对比单节点和 4 节点下进行线路构建所消耗的时间发现，随着量子比特数增加，在 4 节点下构建量子线路所消耗的时间多于单节点下的时间消耗。针对这种情况，对代码进行分析测试，总结出在多节点下构建量子线路所消耗时间反而增加的原因如下：

(1)利用 MPI 编程模型设计的并行代码涉及节点间的通信，而通信也需要消耗时间，并且由于所使用的库类型比较特殊，因此额外针对通信设计了序列化和反序列的函数，每次节点间在进行数据发送和接收的同时需要对数据进行序列化与反序列化。25 比特量子线路的构建过程中，数据转换所需消耗时间约为 34 s，随着模拟比特位数的增加，指数级数据的转换消耗时间也将增加。

(2)文中针对节点间数据传输的过程测试了几组数据，记录了每个节点将构建好的线路转化为二进制数据后发送给主进程的数据字节数，如表 2 所列。

表 2　节点数据传输量

Table 2　Data transfer volume of nodes

(单位：byte)

Quibts	Node 0	Node 1	Node 2	Node 3
8	200	344	1 208	4 664
10	216	436	1 432	23 128
12	344	1 416	10 824	86 088

从表 2 中可以观察到，部分节点处理的任务量非常大，这是造成加速效果不理想的主要原因，实验设计的代码针对量子比特数采用平均分配，这种分配方式没有充分有效地利用

节点资源。以 28 比特为例，利用 4 个节点进行线路的构建，每个节点需分配 7 个比特，则第一个节点需构建$2^0 \sim 2^6$个受控相位门。最后一个节点则需构建$2^{21} \sim 2^{27}$个受控相位门，可以直观地发现，最后一个节点的任务量基本超过了前三个节点的任务量，尤其是第一、第二个节点，它们所分配的任务量能够在很短的时间完成，之后节点就处于空闲状态。基于这种情况，即使将任务拆分到多个节点上处理，加速效果往往也无法达到理想状态，无法发挥出多节点并行的优势，且第二个原因一定程度上也是第一个问题形成的原因。因为负载不均衡，需要进行序列化的数据全部集中在任务量大的节点上，指数级的数据序列化和反序列消耗了大量时间，而任务量少的节点已经处于空闲状态。如果能将任务量合理地分配给多个节点执行，不仅能够充分利用各节点的计算资源，达到任务加速处理的目的，同时数据转化速度也能够大大提升。针对第二个原因，本文设计了一种优化方式，将计算量大的任务轮循地拆分给所有空闲的工作节点，实现节点间的负载均衡。

4 节点优化

4.1 负载均衡优化

首先，我们对设置的进程数做了要求，除了主进程外，工作进程须为偶数个。当构建的量子线路规模较小时，只需要设置一个工作进程来处理。当线路规模较大时，设置工作进程数为2^n，此时工作进程的索引 *process_index* 为 $1 \sim 2^n$。以量子比特数 *quibt_num* 与工作进程数 *num_of_worker* 之差 m 作为界限。第一阶段：当处理的比特索引 i($0 \sim quibt\_num - 1$) $< m$ 时，所有的任务都交给同一个节点上的进程处理，构建$2^0 \sim 2^i$个相位门。第二阶段：随着比特索引增加，当 $i = m$ 时，使工作进程索引自增，之后的构建线路操作交给节点号为 *process_index* +1 上的进程处理，此时对需要构建的相位门个数 2^i 进行拆分以及工作进程索引号自增取余，把分割后的子任务轮循地交给不同的进程处理。随着比特索引 i 的增大，变量 *num_loop_index* 递增，需构建的相位门个数2^i分割的份数则按照等比数列的模式增加，分别为 1，2，4，8，16，…。2^i越大，所拆分的份数越多，节点轮循的次数也增加，从而实现了将节点上计算量大的任务分配给其他节点执行，充分地利用了节点计算和内存资源，针对第二阶段的伪代码如下。

```
1. for( i←0 to quibt_num )
2. {
3. …
4.    /* 根据比特索引、比特总数和工作进程数之间的关系，设计
      num_loop_index */
5.    for(int k ← 0 to (1≪num_loop_index))
6.    {
7.    /* k 代表构建2^i个相位门需要轮循几次 */
8.    if(rank==(process_index + 1))
9.      {
10.     /* 构建相位门 2^(i−num_loop_index)，序列化后发送给主进程 */
11.     }
12.   process_index++
13.   process_index← process_index %
14. num_of_worker
15.   }
16. }
```

对代码进行测试，以每个工作进程向主进程发送的数据字节数为依据，如表 3 所列。

表 3 优化后节点数据传输量

Table 3 Data transfer volume of each node after optimization

Quibts	Node0/byte	Node1/byte	Node2/byte	Node3/byte
8	1642	1642	1483	1642
10	6124	6296	6296	6296
12	24954	24954	23805	24954

由表 3 可以看出每个节点处理的任务量基本处于平衡状态，说明本文方案成功优化了负载不均衡问题，实现了节点内资源的充分利用。

节点间的任务并行处理完毕后，可以尝试针对节点内的任务进行多线程操作。相对而言，节点内多线程的操作较为简单，文中利用 OpenMP 并行模型实现并行操作。首先寻找节点内适合并行的代码块。分析代码后发现最底层的相位门构建主要通过 for 循环构建，当确定比特索引 i 后，for 循环执行区间为 $0 \sim 2^i$，每次循环构建一个 U 门，当循环执行完毕后总共构建了2^i个 U 门也就是U^{2^i}，并且所有相位门都作用于同一个目标比特，因此该代码块是完全独立的，不会因为线程竞争造成计算错误，故选定该代码块进行节点内的多线程并行。“嵩山”超级计算机单节点拥有 32 核，虽然后期需要构建指数级的 U 门，但是采用 MPI+OpenMP 混合结构，依然能够在很大程度上提升加速效果。随后利用实现了负载均衡和节点内多线程优化后的代码进行实验，分别在“嵩山”超级计算机上实现 25，26，27，28，29，30 比特的多节点线路构建，其结果如表 4 所列。

表 4 负载均衡下时间消耗

Table 4 Time consumed in load balancing

Quibts	Four nodes time/s
25	46.432
26	115.236
27	172.256
28	206.526
29	296.352
30	481.256

对比表 1 和表 3 可以观察到量子线路的构建速度有了很大的提升，通过实验证明了文中针对量子线路的拆分、多节点并行构建所设计的代码具有良好的并行效果。表 1 和表 3 两次实验的相对加速比如图 8 所示。

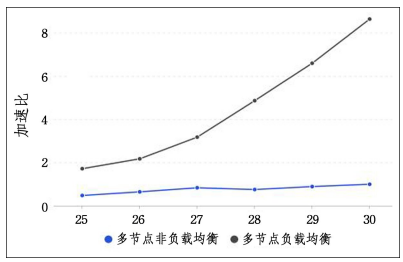

图 8 优化前后相对加速比

Fig. 8 Relative acceleration ratio before and after optimization

随后又针对多节点下数据的转化时间进行了实验测试，

其相对加速比如表 5 所列。

表 5 数据处理加速比

Table 5 Data processing acceleration ratio

Quibts	Nodes	Serialization acceleration ratio
25	15	7.96
26	17	8.78
27	19	8.64
28	21	9.53
29	23	9.87
30	25	9.47

由表 5 中的数据可知，随着节点数的增加，序列化数据并行处理速度显著提升，其时间消耗也将被控制在正常范围之内，不再因为某个节点上需要处理指数级的数据而造成大量的时间消耗，同时也说明各节点实现了负载均衡并且充分利用了节点内资源。

4.2 正确性测试

量子相位估计作为 HHL 算法的一个子程序，在算法中承担着计算矩阵特征值的作用，是 HHL 算法中一个十分关键的步骤。本文接下来尝试将构建的量子相位估计并行版本的代码块作为 HHL 的算法的子模块，利用获得的解的保真度来验证所设计的并行代码的正确性，实验利用 HHL 算法来解 8×8 矩阵，不考虑小规模模拟实验中并行消耗的时间多于串行消耗的时间。HHL 算法伪代码如算法 1 所示。

算法 1 HHL 算法

```
Input:A,b
Output:result
1. auto machine ← initQuantumMachine(CPU)
2. auto prog ← QProg()
3. QCircuit circuit ← CreateEmptyCircuit()
4. QCircuit ←QPE()
5. QCircuit←build_CR()
6. QCircuit←QPE_Inverse()
7. QCircuit hhl_circuit←HHL_circuit(A,b,machine)
8. result ← x
9. Return result
```

HHL 算法中，在相位估计阶段，对于输入的 Hermitian 矩阵，需要通过哈密顿模拟[18]将其制备为酉算子 $U=e^{iAt}$，然后将酉算子的相位信息存储到量子比特上，该过程通过量子比特来表示酉算子的特征值信息，解的准确性依赖于分配给相位估计的量子比特来表示的特征值，量子比特数目越多，相位估计的精度越高。因此相位估计算法的准确性受到表示矩阵特征值的量子比特的数量限制。本次实验需要对 HHL 算法中的相位估计进行一些处理，HHL 算法中，在相位估计阶段进行受控相位门构建时，需要将酉算子 U 分解为一系列的基本门，然后将其添加到线路当中，并且每个 U 门受控的指数都不相同，分解的一系列基本门也有所不同，不再是直接将相位门通过 for 循环进行 2 的幂次构建，因此需要对基于 OpenMP 的多线程细粒度任务并行进行优化。我们尝试在设计的代码中针对每个节点上所划分的任务区间进行多线程的改进。每次执行构建任务时，都会给函数传入控制比特的索引，当区间中的任务执行结束且发送给主进程后，在主进程中将构建好的线路按照控制比特索引大小进行排序，还原正确的拆分线路顺序，最后添加到总线路当中。

本节利用 4×4 线性方程组进行模拟实验。选取系数矩阵 $\boldsymbol{A}$ 为：

$$\frac{1}{4}\begin{pmatrix} 15 & 9 & 5 & -3 \\ 9 & 15 & 3 & -5 \\ 5 & 3 & 15 & -9 \\ -3 & -5 & -9 & 15 \end{pmatrix}$$

b 为：

$(0.5\ \ 0.5\ \ 0.5\ \ 0.5)^{\mathrm{T}}$

对于该线性方程，可以计算出准确的经典解为：

$$(-0.03125\ \ 0.21875\ \ 0.34375\ \ 0.40625)^{\mathrm{T}} \tag{1}$$

首先构建 HHL 算法线路，对于解 4×4 方程组的 HHL 算法线路构建需要 7 个比特，其中目标寄存器用于表示 $\boldsymbol{b}$ 需要 2 个比特，第一寄存器需要 4 个比特以及 1 个附属比特。经过多次测试实验，HHL 算法的输出结果为：

$$(-0.05247\ \ 0.35271\ \ 0.53272\ \ 0.41356)^{\mathrm{T}} \tag{2}$$

因为 HHL 算法解线性方程计算的是与解相关算子的期望值，而不是解本身，经典算法返回的是精确解，而 HHL 算法只能返回近似解。HHL 算法在哈密顿模拟操作阶段、相位估计阶段以及控制旋转阶段都容易产生精度误差。接下来针对 8×8 矩阵求解，对于解 8×8 方程组的 HHL 算法构建需要 13 个比特，其中目标寄存器用于表示 b 需要 3 个比特，第一寄存器采用特征值放大方法时，需要 9 个比特以及 1 个附属比特。首先在 matlab 上求得矩阵的经典解为式(3)，模拟 HHL 算法输出结果为式(4)。

$$(0.5000\ \ 1.0000\ \ 1.5000\ \ 1.7059\ \ 1.6176\ \ 1.5882\ \ 1.5882\ \ 1.5000)^{\mathrm{T}} \tag{3}$$

$$(0.1431\ \ 0.5871\ \ 2.3301\ \ 1.7234\ \ 3.9689\ \ 1.8141\ \ 2.5271\ \ 1.3249)^{\mathrm{T}} \tag{4}$$

从式(2)和式(4)可以看出，HHL 算法求得的解与经典解成正比，证明了代码改进的正确性。在量子编程软件利用高级编程语言 C++实现的 HHL 算法中，需要对输入的矩阵进行一系列复杂的操作。首先对于矩阵不是 Hermitian 矩阵的情况，需要先将其转化为 Hermitian 矩阵，之后再进行酉算子的转化以及通过计算将其分解，最后才能执行线路的构建，进行整体的计算。因此不仅 HHL 算法的 3 个子程序会产生精度的误差，对矩阵进行处理的过程中也会产生误差，这些问题对解的期望值影响较大。另一方面，在经典计算机上进行算法的实验，其线路的构建较为复杂，在量子云平台实现 4×4 线路的规模为 353，而在量子编程软件中实现的 4×4 线路构建，将矩阵分解为一系列基本门，其线路规模就达到了 976，而 8×8 矩阵分解的线路规模高达 7246。因此，虽然本文设计的并行代码在小规模的计算中体现出的加速效果并不明显，但如果要实现更大规模的矩阵求解，就需要考虑如何对输入的大规模矩阵进行一系列的处理以满足构建量子线路的要求。

结束语 本文针对大规模量子线路构建问题，利用 MPI+OpenMP 混合编程模型设计了并行代码，并将其部署在“嵩山”超级计算机上执行，将拆分后的任务交由不同的节点处理

以实现构建线路的加速。随后，面对节点资源利用不充分的问题，设计了一种分割任务量、轮循节点处理的优化方式，再次将具体节点上的任务量进行了拆分，使得任务量大的节点能够将任务拆分给其他空闲节点执行，从而实现了负载均衡。最后用 HHL 算法验证了本文针对量子相位估计模块所做的并行改进的正确性。但文中未能对更大规模的 HHL 算法线路进行构建以进一步验证并行效率，我们将在未来工作中继续深入研究。

参 考 文 献

[1] DEUTSCH D. Quantum theory the Church-Turing principle and the universal quantum computer[J]. Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences, 1985, 400(1818): 97-117.

[2] SHOR P W. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer[J]. SIAM Review, 1999, 41(2), 303-332.

[3] GROVER L K. A fast quantum mechanical algorithm for database search[C]// Proceedings of the twenty-eighth annual ACM symposium on Theory of computing. 1996: 212-219.

[4] HARROW A W, HASSIDIM A, LLOYD S. Quantum Algorithm for Linear Systems of Equations[J]. Physical Review Letters, 2009, 103(15): 150502.

[5] HUANG C J, ZHANG F, NEWMAN M, et al. Classical simulation of quantum supremacycircuits[J]. arXiv: 2005. 06787, 2020.

[6] PEDNAULT E, GUNNELS J A, NANNICINI G, et al. Pareto-Efficient Quantum Circuit Simulation Using Tensor Contraction Deferral[J]. arXiv: 1710. 05867, 2017.

[7] LI R L, WU B J, YM S, et al. Quantum supremacy circuit simulation on Sunway TaihuLight[J]. IEEE Transactions on Parallel and Distributed Systems, 2019, 31(4): 805-816.

[8] MAUCH V, KUNZE M, HILLENBRAND M. High performance cloud computing[J]. Future Generation Computer Systems, 2013, 29(6): 1408-1416.

[9] LAROSE R. Quantum Physics, Computer Science-Distributed, Parallel, and Cluster Computing[J]. arXiv: 1801. 01037, 2018.

[10] HNER T. Steiger D S. 0. 5 petabyte simulation of a 45-qubit quantum circuit[C]// the International Conference for High Performance Computing, Networking, Storage and Analysis. 2017: 1-10.

[11] ZHAO Y Q, LI R G, JIANG J Z, et al. Simulation of quantum computing on classical supercomputers with tensor-network edge cutting[J]. Physical Review A, 2021, 104(3): 032603.

[12] YU Z C, LI Y Z, LIU L, FENG S Z. A review of quantum computing simulation and optimization methods[J]. Computer Engineering, 2022, 48(1): 1-11.

[13] ILIAS K, VASSILIOS V D. Experiences with task-based programming using cluster nodes as OpenMP devices[J]. arXiv: 2205. 10656, 2022.

[14] YU Q X. A Unified Programming Model for Heterogeneous Computing with CPU and Accelerator Technologies[J]. arXiv: 2204. 06864, 2022.

[15] LU F X. Research on key technology of parallel Computing for CPU/GPU Heterogeneous Architecture[D]. Changsha: National University of Defense Technology, 2014.

[16] WOSSNING L, ZHAO Z, PRAKASH A. Quantum linear system algorithm for dense matrices[J]. Physical Review Letters, 2018, 120(5): 050502.

[17] NIELSEN M A, CHUANG I L. Quantum Computation and Quantum Information[J]. Mathematical Structures in Computer Science, 2002, 17(6): 1115-1116.

[18] SOMMA, ROLANDO D. A Trotter-Suzuki approximation for Lie groups with applications to Hamiltonian simulation[J]. Journal of Mathematical Physics, 2016, 57(6): 467-488.

XIE Haoshan, born in 1998, postgraduate. His main research interest is quantum computer.

LIU Xiaonan, born in 1977, Ph.D, associate professor, master supervisor, is a member of China Computer Federation. His main research interests include quantum computer and high-performance parallel computation.

（责任编辑：何杨）