计算机科学
COMPUTER SCIENCE

基于OpenMP并行模型下HHL算法的经典模拟实现

谢浩山, 刘晓楠, 赵晨言, 何明, 宋慧超

引用本文

谢浩山, 刘晓楠, 赵晨言, 何明, 宋慧超基于OpenMP并行模型下HHL算法的经典模拟实现[J]. 计算机科学, 2022, 49(11A): 211200028-5.

XIE Hao-shan, LIU Xiao-nan, ZHAO Chen-yan, HE Ming, SONG Hui-chao. Classical Simulation Realization of HHL Algorithm Based on OpenMP Parallel Model [J]. Computer Science, 2022, 49(11A): 211200028-5.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.211200028

基于 OpenMP 并行模型下 HHL 算法的经典模拟实现

谢浩山　刘晓楠　赵晨言　何　明　宋慧超
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
(15237933568@163.com)

摘　要　量子计算天然的叠加性和纠缠性使其具有经典计算技术难以比拟的并行计算能力,基于量子计算强大的并行能力,某些已知的量子算法处理问题的速度要快于经典算法。然而现阶段由于量子计算机的研制还处于发展阶段,想要在量子计算机上进行算法实验的需求还不能被满足,因此可以在经典计算机上对量子算法进行经典模拟。HHL 算法常用来解决线性系统的方程问题,其被广泛应用在数据处理、数值计算、最优化问题等领域。本文基于经典的计算机平台,通过高级编程语言 C++对 HHL 算法进行模拟实现,并采用 OpenMP 并行编程模型对算法进行并行加速。实现了 HHL 算法模拟解决 4×4,8×8,16×16 矩阵的线性方程组,并且对算法进行了加速处理。

关键词:HHL 算法;OpenMP;并行加速;经典模拟
中图法分类号　TP385

Classical Simulation Realization of HHL Algorithm Based on OpenMP Parallel Model

XIE Hao-shan,LIU Xiao-nan,ZHAO Chen-yan,HE Ming and SONG Hui-chao
State Key Laboratory of Mathematical Engineering and Advanced Computing,Information Engineering University,Zhengzhou 450000,China

Abstract　Due to its natural superposition and entanglement,quantum computing has parallel computing capability that is incomparable to classical computing technologies. Based on the powerful parallel capability of quantum computing,some known quantum algorithms are faster than classical algorithms in processing problems. However,at this stage,because quantum computers is still in the development stage,the demand for algorithm experiments on quantum computers cannot be met. Therefore,quantum algorithms can be classically simulated on classical computers. HHL algorithm is used to solve the equation problem of linear system and it is widely used in data processing,numerical calculation,optimization problem and other fields. Based on the classic computer platform,HHL algorithm is simulated with C++,and the parallel programming model of OpenMP is used to accelerate the algorithm. Realizing the HHL algorithm simulation to solve the linear equations of 4×4,8×8,16×16 matrix and realize the acceleration of the algorithm.

Keywords　HHL algorithm,OpenMP,Parallel acceleration,Classic simulation

1　引言

线性系统是很多科学和工程领域的核心,当今很多领域对于线性方程问题的求解有很大的依赖,2009 年,Harrow 等[1]提出用量子算法解决线性系统的方程问题,该算法常被研究人员称为 HHL 算法,它常被应用于机器学习算法中,因为机器学习中的问题最终都与最优化问题的求解相关,而最优化问题常涉及线性方程组的求解。由于 HHL 算法在特定条件下相较于经典算法有很好的指数加速效果,因此被广泛应用在数据处理、数值计算、最优化问题等领域。

随着量子科技的快速发展,根据量子特有的叠加性和纠缠性设计的量子计算机得到人们的重视,其强大的并行能力带来了远超经典计算机的强大的算力,然后由于量子计算总体上还处于起步阶段,基于这一背景,想要在真实的量子计算机上进行算法的研究还不能完全实现,所以目前有效的 HHL 算法实际应用和实验实现在文献中仍然较少。对于 HHL 算法的模拟大多是在 IBM、本源等公司提供的量子云平台上进行,平台提供的免费可用的量子比特数目较少,不足以支持大规模的实验模拟,因此已有的关于 HHL 算法实验规模都是较小的,基本集中在 2×2 矩阵和 4×4 矩阵[2-4],所需要的量子比特数也不超过 10 比特。本文在经典多核计算机上利用高级编程语言 C++实现对 HHL 的算法模拟,同时我们利用并行编程模型 OpenMP 对算法进行改进,模拟实现了 4×4,8×8 和 16×16 矩阵,得到了它们在经典计算机下正常运行以及并行加速后的时间对比结果,该实验为以后进行更大规模的矩阵模拟提供了参考。在经典计算机上,任何量子逻辑操作都可以分解成一系列单量子位的逻辑操作和双量子位受控非门的组合序列[5],而这些单量子位门或双量子门又可以利用高级编程语言通过矩阵来表示。目前常用的工具有 libquantum 和 Qpanda2.0 等已经封装好的包,这些工具把量子计算中需要的门通过代码封装起来,方便我们使用。

本文第 2 节介绍了 HHL 算法的理论基础及其具体构

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.com)

成;第 3 节介绍了程序的并行原理,OpenmMP 等并行编程模型以及针对算法的热点分析;第 4 节介绍了该实验的实现以及数据对比。

2 HHL 量子算法

如果矩阵 $\boldsymbol{A}$ 每行或每列最多具有 s 个非零元,那么就将线性方程组称为 s-稀疏线性方程组。用经典算法(共轭梯度法)来解决 N 维的 s-稀疏线性方程组,需要的时间复杂度为 $O\left(Ns\kappa\log\frac{1}{\varepsilon}\right)$,这里 κ 表示系统的条件数,ε 表示近似的精度。当 $\boldsymbol{A}$ 是 Hermitian 矩阵时,用 HHL 算法解线性方程组的时间复杂度为 $O\left(\log(N)s^2\frac{\kappa^2}{\varepsilon}\right)$。经典的求解线性方程的问题为输入一个 $N\times N$ 的矩阵 $\boldsymbol{A}$ 和一个 N 维向量 $\boldsymbol{b}$,输出 N 维向量 $\boldsymbol{x}$,假设 $\boldsymbol{b}$ 和 $\boldsymbol{x}$ 是标准化的,就可以将他们映射为量子态 $|\boldsymbol{b}\rangle$ 和 $|\boldsymbol{x}\rangle$,其量子表达式如下:

$$\boldsymbol{A}|\boldsymbol{x}\rangle=|\boldsymbol{b}\rangle$$

该算法有一定的限制条件:$N\times N$ 的矩阵 $\boldsymbol{A}$ 必须是一个 Hermitian 矩阵,如果不是,$\boldsymbol{A}$ 就需要以某种方式转换成 Hermitian 矩阵,假设 $|\boldsymbol{\mu}_j\rangle$ 是 $\boldsymbol{A}$ 的特征向量,λ_j 是其对应的特征值,由谱分解定理可以知识每个 Hermitian 矩阵都有一个特征向量的标准正交基,所以矩阵 $\boldsymbol{A}$ 的谱分解可以表示为 $\boldsymbol{A}=\sum_j\lambda_j|\boldsymbol{\mu}_j\rangle\langle\boldsymbol{\mu}_j|$。

对于输入向量 $|\boldsymbol{b}\rangle$,在矩阵 $\boldsymbol{A}$ 的特征向量上展开,可以得到:

$$|\boldsymbol{b}\rangle=\sum_j\beta_j|\boldsymbol{\mu}_j\rangle$$

我们理想的最终状态就是获得 $|\boldsymbol{x}\rangle=\sum_j\frac{\beta_j}{\lambda_j}|\boldsymbol{\mu}_j\rangle$,由该式可以看出,最后要求输出的向量 $|\boldsymbol{x}\rangle$,就是在输入向量 $|\boldsymbol{b}\rangle$ 之前添加矩阵 $\boldsymbol{A}$ 的特征值倒数作为系数。所以需要进一步进行公式转换,由 $\boldsymbol{A}|\boldsymbol{\mu}_j\rangle=\lambda_j|\boldsymbol{\mu}_j\rangle$ 可得表达式 $\boldsymbol{A}^{-1}|\boldsymbol{\mu}_j\rangle=\frac{1}{\lambda_j}|\boldsymbol{\mu}_j\rangle$,由该式与 $|\boldsymbol{b}\rangle$ 的表达式可以得到:

$$|\boldsymbol{x}\rangle=\boldsymbol{A}^{-1}|b\rangle=\boldsymbol{A}^{-1}\sum_j\beta_j|\mu_j\rangle=\sum_j\frac{\beta_j}{\lambda_j}|\mu_j\rangle$$

由此可以看出,求解线性方程组的问题就转化为了求解矩阵 $\boldsymbol{A}$ 的特征值信息。

HHL 算法的的 3 个子过程是:相位估计、受控旋转、逆相位估计。一般步骤为:

(1)对初始比特应用进行相位估计,这是在特定基础上分解量子态的一般步骤。初始化量子态 $|\boldsymbol{b}\rangle$,将系数矩阵 $\boldsymbol{A}$ 通过哈密顿模拟制备制备成酉算子 $e^{i\boldsymbol{A}t}$,通过量子相位估计算法应用受控操作 $\boldsymbol{U}=e^{i\boldsymbol{A}t}$ 作用到 $|\boldsymbol{b}\rangle$ 上,将 $|\boldsymbol{b}\rangle$ 分解为 $\boldsymbol{A}$ 的特征向量线性组合的叠加。当相位估计算法作用之后 $|\boldsymbol{b}\rangle$ 状态为 $\sum_j\beta_j|\boldsymbol{\mu}_j\rangle$,此时整个系统的状态为 $\sum_j\beta_j|\boldsymbol{\mu}_j\rangle$。

(2)进行受控旋转操作,目的是将 $|\lambda_j\rangle\rightarrow\lambda_j^{-1}|\lambda_j\rangle$。为了达到这个效果,这里我们引入一个辅助量子位,根据 $|\lambda_j\rangle$ 在特征值寄存器中的值,受控门作用于辅助量子比特之后整个系统的状态为:

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

受控旋转操作可以实现有效量子信息从寄存器到量子态振幅的转移,通过附加量子比特实现了将特征值的倒数按比例提取到了对应基态的概率辐上。

(3)利用逆量子相位估计方法抵消寄存器中存储的特征值得到:

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

在第一量子位上测量,当测量为 $|1\rangle$ 时可得到 $\frac{c}{\lambda_j}\beta_j|\boldsymbol{\mu}_j\rangle$,若为 $|0\rangle$ 则重新计算。由于 HHL 算法计算的是与解相关的算子的期望值,而不是解本身,HHL 算法相对于经典算法有着指数级的加速,但经典算法可以返回精确解,而 HHL 算法只能返回近似解。而我们得到的表达式结果和所求的线性方程组的解成正比,此时只需要测量附加比特为 $|1\rangle$ 的结果即可。其线路图如图 1 所示。

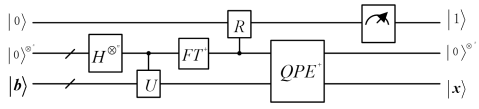

图 1 HHL 算法线路图

Fig. 1 HHL algorithm circuit diagram

3 并行编程模型

目前流行的并行编程模型有 OpenMP 和 MPI,这两种模型最大的不同之处在于 OpenMP 是一种适用于共享内存多处理器体系结构的可移植并行编程模型,它只能在单台主机上工作,我们可以充分利用多核 CPU 通过多线程将并行负载分配到计算核心来达到加速的目的;而 MPI 是多主机联网协作进行并行计算的工具,适用于分布式内存系统,它能够协调多台主机之间的并行计算,在并行规模上的可伸缩性很强。

3.1 并行编程设计

共享式内存并行的基本思路就是使用多线程。OpenMP 采用了 fork-join 执行模型,以线程为基础,通过编译指导语句显示的指导并行化,即主线程在并行区派生很多子线程,并且在并行区结束之后由主线程继续执行。其执行模型如图 2 所示。当确定适合并行化的代码块后,我们选择合适的 OpenMP 编译指导指令和运行库函数创建并行程序。OPenMP 的应用程序接口包含编译指导、运行函数库和环境变量 3 部分,我们使用应用编程接口实现线程之间的任务分配和数据共享,而编译器通过对串行程序中蕴含并行性的分析与发掘,自动生成适合并行体系结构运行的并行程序[6]。当编译器选择忽略预处理指令,或者编译器不支持 OpenMP 时,程序又退化串行程序,此时代码依然可以正常运作。

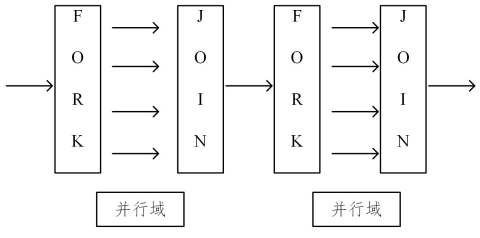

图 2 Fork-join 执行模型

Fig. 2 Fork-join execution model

OpenMP 程序刚开始启用主线程时一直串行地执行,直到遇见第一个并行域才开始并行执行,并行域表示该部分程序计算量大,需要多个处理器共同处理以提高效率和运行速度[7]。

首先对算法代码进行分析，确定适合并行的模块，而 OpenMP 最主要的并行来源是循环结构，当定位到适合并行的模块之后，我们需要合理管理算法中的私有数据和公有数据，因为多线程是异步运行的，当多个线程对同一个数据进行操作时，那么该数据就会在多个线程的作用下出现差异从而造成结果的不可知性，所以应该注意避免线程数据之间的竞争从而导致计算出现错误，在启用多线程时做好对数据源的同步。最后，如果确定的代码部分是独立的且不会因为线程竞争造成计算错误，那么就可以尝试选定该代码块作为并行处理的部分进行并行实验，通过实验结果对比该部分的并行效果。

3.2 OpenMP 指令函数

在 OpenMP 中，通过在串行代码中插入编译制导指令来实现并行，其通用格式为 # pragma omp 指令。OpenMP 常用的指令如表 1 所列。

表 1 OpenMP 参数说明

Table 1 OpenMP parameter description

指令	含义
sections	用在可能会被并行执行的代码段之前
barrier	用于并行区内代码的线程同步
parallel for	for 循环的代码将被多个线程并行执行
ordered	用于指定并行区域的循环按顺序执行
critical	用于一段代码临界区之前
parallel	用在一个代码段之前，表示这段代码将被多个线程并行执行

OpenMP 常用的库函数如表 2 所列。

表 2 OpenMP 库函数说明

Table 2 OpenMP library function description

函数名	含义
omp_get_num_procs	返回运行本线程的多处理机的处理器个数
omp_get_num_threads	返回当前并行区域中的活动个数
omp_get_thread_num	返回线程号
omp_set_num_threads	设置并行执行的线程个数
omp_set_lock	上锁操作
omp_unset_lock	解锁操作
omp_get_num_procs	返回运行本线程的多处理机的处理器个数

4 模拟实现

4.1 串行执行

本文的目的是在经典多核计算机上进行 HHL 算法的模拟并进行多线程并行处理，因此我们采用 QPanda2.0 量子软件开发工具包通过 C++高级编程语言来模拟算法的具体实现，并通过 OpenMP 对算法进行改进，实现并行加速，缩短 HHL 算法的模拟时间，提高模拟效率，确定量子算法在经典计算机上模拟的可行性。

我们首先对 4×4 线性方程组进行模拟实验。对于 4×4 线性方程组，我们选取系数矩阵 $\boldsymbol{A}$ 为：

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

$\boldsymbol{b}$ 向量为：

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

又可以通过 $\sum_{j=1}^{j=4}\beta_j|\boldsymbol{\mu}_j\rangle$ 来表示，其中 $\beta_j=\frac{1}{2}$。对于该线性方程我们可以计算出准确的经典解为：

$$\frac{1}{32}(-1 \quad 7 \quad 11 \quad 13)^{\mathrm{T}}$$

Qpanda2.0 量子编程框架支持量子计算中绝大多数的量子逻辑门操作、量子线路构建操作等。我们将在此基础上进行 HHL 算法的模拟并对算法进行热点分析，寻找可以并行的代码块，实现算法的并行操作。HHL 算法伪代码如算法 1 所示。

算法 1 HHL 算法

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

我们通过代码构建了 HHL 算法的 3 个模块，分别为量子相位估计、控制旋转、逆量子相位估计。在构建好 HHL 算法电路后，我们对该算法的 3 个模块进行了分析，确定了其中可以进行加速的部分。

其中相位估计部分如算法 2 所示。

算法 2 相位估计

```
1. QCircuit QPE( Qubit * control,Qubit * target )
2. {
3. QCircuit qpe_cir;
4. qpe_cir≫apply_QGate(control,H)<<
5. control_unitary(Qubit * Control)<<
6. QFT_Inverse();
7. }
```

该部分主要由 Hadamard 变换、受控相位变换和量子傅里叶逆变换 3 部分组成，经过分析，发现其中耗费时间较多的为受控相位变换和量子傅里叶逆变换两个阶段，于是我们尝试对这两个阶段的代码进行改进。

首先我们针对量子相位估计中的受控相位部分进行操作，代码中构建受控相位门的操作采用的是线程池 ThreadPool 函数，该操作的目的是为了后续实现多线程并发，线程池通过几个固定线程为大量的操作服务，减少了创建和销毁线程所需的时间从而提高了效率。我们将构造相位门的任务通过循环逐一添加到线程池中，当线程池里的每个线程任务执行完毕后，在保存构建好的线路时采用加锁操作，此时，我们构造的相位门线路就可以在之后的操作中按照顺序将每个部分的相位门添加到线路中。测得结果如下：

$(-0.0542 \quad 0.3801 \quad 0.5974 \quad 0.7051)^{\mathrm{T}}$

由理论值和测得的测量值计算得到其保真度为96.73%，整个程序运行是在经典 8 核计算机上，时间消耗为 928.16ms，因为该过程模拟的是 4×4 矩阵，矩阵的规模不大，在量子云台中采用 7 个量子比特即可构建，所以时间的消耗不算太大。

4.2 并行改进

接下来本文针对算法实现过程中线程池处理任务阶段

进行改进，在线程池中的任务执行结束前，采用加锁的方法来同步保存线路操作，进而保证了所构造的相位门线路按照正确的顺序保存。实验结果证明，即使通过并行操作使所有的任务通过多线程添加到线程池中运行也不会造成线路构建顺序的紊乱。

我们根据控制比特的数量构建了针对线程池添加任务的 for 循环，用 pragma omp parallel for 来标识并行程序块，通过多线程使线程池中的任务进行构建受控相位门操作，且不对代码添加额外的同步操作。之后我们测试代码，发现与串行代码执行相比，任务进入线程池的顺序发生了改变，执行顺序也变得无序，但是由于每个任务在进入线程池时，都给该任务传递了按照控制比特位置制定的索引参数，并对保存线路操作进行了加锁，最后对所有保存的线路进行排序，使得这些构造还能够按照正确顺序进行线路组建，因此我们得出的结果还是与之前得到的近似解一样，时间为 263.34 ms。

之后我们又尝试对逆量子傅里叶设计并行操作，逆量子傅里叶由 Hadamard 变换和相位变换两部分组成，我们通过两层 for 循环，分别把 H 门和相位门添加到线路上，当循环执行完毕后线路构建也完成了。在构建线路时我们首先尝试对两层 for 循环同时设计并行操作，并且我们不在并行过程中添加同步操作，在这种情况下获得的结果为：

$(2.962 \quad 3.492 \quad 6.195 \quad 8.097)^{T}$

可以看出，每次结果都不一样且都与我们所期望的结果有着很大的出入，经过分析之后发现，代码中内层循环的条件值依赖于外层循环条件值，经过并行后使内层循环发生错误，使得受控相位门的构建产生问题，这就导致计算结果发生错误。于是我们改进代码，因为内层循环执行的任务量大，所以我们对内层循环进行并行操作，在内层循环中通过 pragma omp parallel for schedule 任务调度，其工作原理如图 3 所示。

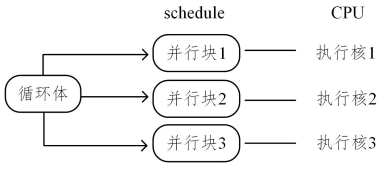

图 3 任务调度

Fig. 3 Task scheduling

动态地将迭代分配到各个线程，最后得到正确的近似解。经过并行后的时间为 236.57 ms，结果证明我们的并行操作是有效且正确的。

4.3 结果分析

我们首次在 8 核上测得的串行程序运行时间为 928.16 ms，加速后的时间为 236.57 ms，接下来我们分别测试了在 2 核、4 核、6 核上经过并行操作后的时间，结果如表 3 所列。

表 3 4×4 矩阵运行时间

Table 3 4×4 matrix running time

（单位：ms）

核数	2 核	4 核	6 核
并行时间	653.58	522.15	254.75

经过多次测试后，我们发现构建了 OpenMP 框架的程序在 6 核下执行和在 8 核下执行，程序的时间对比相差不大，这是因为我们的程序在 6 核操作下已经可以执行所有分配的线程。

本节对 4×4 矩阵在经典计算机上进行了模拟，测得了程序通过 OpenMP 实现多线程的实验结果，实现在经典计算机上对量子算法进行正确的模拟和并行加速。相对于量子云平台上实现的 4×4 矩阵，我们在经典计算机上构建 HHL 算法的线路深度较大，基于 4×4 矩阵的线路在经典计算机上模拟需要 976 个门构成，而在量子云平台上实现 4×4 矩阵的线路的深度为 353[8]，是经典计算上线路模拟的 1/3，可见在经典计算机上进行线路模拟，在内存方面的消耗较大。接下来我们又按照以上方法进行了 8×8 矩阵和 16×16 矩阵的实验，8×8 其正常运行的结果为 1781.39 ms，并行操作后的结果如表 4 所列。

表 4 8×8 矩阵运行时间

Table 4 8×8 matrix running time

（单位：ms）

核数	2 核	4 核	6 核	8 核
并行时间	1065.51	869.56	760.06	584.72

同时我们也记录了该 8×8 矩阵分解所需要消耗的门为 7250。对 16×16 矩阵进行模拟的结果为 3232.84 ms，并行操作后的结果如表 5 所列。

表 5 16×16 矩阵运行时间

Table 5 16×16 matrix running time

（单位：ms）

核数	2 核	4 核	6 核	8 核
并行时间	2616.59	2502.03	2455.81	2408.72

经过测试我们发现对于 16×16 的矩阵，在开启多线程之后，随着核数的增加，加速效果并不明显，经过分析我们得出结论，单台 8 核计算机已经无法负载该矩阵分解的任务，因为对于 16×16 矩阵的分解，搭建线路所需要分解矩阵形成的门个数达到了 35039，而且对于构建受控相位门的任务也达到了 12 个，当我们的线程数大于计算机核数时，因为同时运行的线程数只能等于核数，此时其他的线程变为并发运行，这样在线程切换上会浪费更多的时间。我们对比了所做的 3 组矩阵模拟，得到的运行时间比如图 4 所示。我们发现随着矩阵模拟规模的增大，其加速效果将会逐渐降低，这是因为在经典计算机上进行量子计算模拟时，所需要的时间和空间的开销随模拟比特位数的增长呈指数级增长，随着模拟的矩阵规模越来越大，模拟的量子比特位数也越来越大，所要处理的任务量越来越大，单台计算机的内存和处理器将无法满足模拟需求。

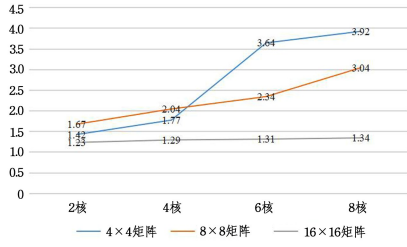

图 4 不同规模矩阵在不同核数下的加速比

Fig. 4 Acceleration ratio of different scale matrix with different kernel numbers

我们可以通过计算机集群来处理基于更大规模的量子算法经典模拟，文献[9]采用 MPI 和加速线程库在超级计算机上对量子傅里叶变换进行了大规模模拟，充分利用多核 CPU

和分布式架构的潜力，实现了 46 位量子比特量子傅里叶算法的模拟和优化。文献[10]使用 CPU-GPU 异构集群技术实现多比特位的量子计算模拟，并且通过经典模拟 QFT 算法[11]证明了模拟系统的有效性和可扩展性。采用集群分布式存储分摊指数级存储开销和计算需求，将进一步解决量子计算模拟的内存和时间瓶颈问题，由此可见，未来我们可以通过搭建分布式计算机体系来进行更大规模的算法模拟实现。

结束语 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，有望成为满足未来计算需求、加速科技创新的新引擎。虽然量子计算机的研究已经取得了很大的进步，但是由于条件技术的限制，目前量子计算机的使用还远不能普及，因此对于量子算法的研究，我们可以在经典计算机上对算法进行模拟。建立基于经典计算机的量子计算模拟平台，具有重要意义，它不仅可以为量子算法和量子线路[12-14]提供一个可靠的验证平台，而且可以帮助我们了解经典计算和量子计算的边界[15]，促进量子计算的发展。

参 考 文 献

[1] HARROW A W, HASSIDIM A, LlOYD S. Quantum Algorithm for Linear Systems of Equations[J]. arXiv: 103, 150502, 2009.

[2] CAI X D, WEEDBROOK C, SU Z E, et al. Experimental Quantum Computing to Solve Systems of Linear Equations[J]. Physical Review Letters, 2013, 110(23): 230501.

[3] DUTTAS, SUAU A, DUTTA S, et al. Quantum circuit design methodology for multiple linear regression[J]. IET Quantum Communication, 2020, 1(2): 55-61.

[4] CAO Y, DASKIN A, FRANKEL S, et al. Quantum circuit design for solving linear systems of equations[J]. Molecular Physics, 2012, 110(15/16): 1675-1680.

[5] MIAO X. Universal construction of unitary transformation of quantum computation with one- and two-body interactions[J/OL]. https://arxiv.org/pdf/quant-ph/0003068.pdf.

[6] LIU X X. HUANG P F. OpenMP program for heterogeneous systems is automatically generated[J]. Journal of Information Engineering University, 2012, 13(4): 489-495.

[7] ZHAO H, XU J G. Research on parallel ant colony algorithm based on OpenMP multi-core architecture[J]. Microcomputers and Applications, 2011, 30(16): 6-8, 11.

[8] SUAU A. Working 4×4 HHL implementation Zenodo[EB/OL]. https://doi.org/10.5281/zenodo.1480660.

[9] LIU X N, JING L N, WANG L X, et al. Large-scale quantum Fourier transform simulation based on Shenwei 26010 processor [J]. Computer Science, 2020, 47(8): 93-97.

[10] ZHANG P. Research on quantum computingSimulation method based on CPU+GPU heterogeneous cluster[D]. Nanjing: Nanjing University of Aeronautics and Astronautics, 2016.

[11] YORAN N, SHORT A J. Efficient classical simulation of the approximate quantum Fourier transform[J]. Physical Review A, 2007, 76(4): 538-538.

[12] OULAND A, FEFFERMAN B, NIRKHE C, et al. On the complexity and verification of quantum random circuit sampling[J]. Nature Physics, 2019, 15(2): 159-163.

[13] CHEN Z Y, ZHOU Q, XUE C, et al. 64-qubit quantum circuit simulation[J]. Science Bulletin, 2018.

[14] CHEN J, ZHANG F, HUANG C, et al. Classical Simulation of Intermediate-Size Quantum Circuits[J]. arXiv: 1805.01450, 2018.

[15] WUJ, LIU Y, ZHANG B, et al. A benchmark test of boson sampling on Tianhe-2 supercomputer[J]. National Science Review, 2018, 5(5): 715-720.

XIE Hao-shan, born in 1998, postgraduate. His main research interest is quantum computer.

LIU Xiao-nan, 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.