计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.201200023

基于“嵩山”超级计算机系统的量子傅里叶变换模拟

谢景明[1]　胡伟方[1]　韩　林[2]　赵荣彩[2]　荆丽娜[3]
1 郑州大学信息工程学院　郑州 450000
2 郑州大学河南省超算中心　郑州 450000
3 郑州大学中原网络安全研究院　郑州 450001
(624797981@qq.com)

摘　要　“嵩山”超级计算机系统是中国自主研发的新一代异构超级计算机集群,其搭载的CPU和DCU加速器均为我国自主研发。为扩充该平台的科学计算生态,验证量子计算研究在该平台上开展的可行性,文中使用异构编程模型实现了量子傅里叶变换模拟在“嵩山”超级计算机系统上的异构版本,将程序的计算热点部分分配至DCU上运行;然后使用MPI在单计算节点上开启多进程,实现DCU加速器数据传输和计算的并发;最后,通过计算与通信的隐藏避免了DCU在数据传输时处于较长时间的空闲状态。实验首次在超算系统上实现了44 Qubits规模的量子傅里叶变换模拟,结果显示,异构版本的量子傅里叶变换模拟充分利用了DCU加速器计算资源,相较于传统CPU版本,其取得了11.594的加速比,且在集群上具有良好的可拓展性,该方法为其他量子算法在“嵩山”超级计算机系统上的模拟实现以及优化提供了参考。

关键词:异构计算;量子傅里叶变换;DCU加速器;HIP-C;MPI;通信隐藏
中图法分类号　TP387

Quantum Fourier Transform Simulation Based on “Songshan” Supercomputer System

XIE Jing-ming[1],HU Wei-fang[1],HAN Lin[2],ZHAO Rong-cai[2] and JING Li-na[3]
1 Information Engineering Institute,Zhengzhou University,Zhengzhou 450000,China
2 Henan Supercomputing Center,Zhengzhou University,Zhengzhou 450000,China
3 Zhong Yuan Network Security Research Institute,Zhengzhou University,Zhengzhou 450001,China

Abstract　The “Songshan” supercomputer system is a new generation of heterogeneous supercomputer cluster independently developed by China. The CPU and DCU accelerators it carries are all independently developed by my country. In order to expand the scientific computing ecology of the platform and verify the feasibility of quantum computing research on this platform,the paper uses a heterogeneous programming model to implement a heterogeneous version of the quantum Fourier transform simulation on the “Songshan” supercomputer system. The computing hotspots of the program are allocated to run on the DCU;then MPI is used to enable multiple processes on a single computing node to realize the concurrent data transmission and calculation of the DCU accelerator;finally,the hiding of computing and communication prevents the DCU from being in the middle of data transmission. The experiment implements a 44 Qubits-scale quantum Fourier transform simulation on a supercomputing system for the first time. The results show that the heterogeneous version of the quantum Fourier transform module makes full use of the computing resources of the DCU accelerator and achieves 11.594 compared to the traditional CPU version. The speedup ratio is high,and it has good scalability on the cluster. This implementation method provides a reference for the simulation implementation and optimization of other quantum algorithms on the “Songshan” supercomputer system.

Keywords　Heterogeneous computing,Quantum Fourier transform,DCU accelerator,HIP-C,MPI,Communication hiding

1　引言

自20世纪80年代以来,量子计算机的发展在全球范围内都获得了重视并取得了长足的进步,在2019年,IBM公司展示了首台量子计算机原理样机IBM Q System One[1]。但由于量子计算机的设计、构建、编程等一系列客观因素的制约,现阶段量子计算机上可实施的实验规模较小,可供进行各项量子计算研究的物理平台依然十分稀缺[2],各类量子实验项目的实施苦难重重。因此,在当前环境下,利用经典架构计算机对量子算法的模拟实现是验证和研究量子算法非常高效

到稿日期:2020-12-02　返修日期:2021-03-19　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家重点研发计划(2018YFB0505000)
This work was supported by the National Key R&D Program of China(2018YFB0505000).
通信作者:韩林(strollerlin@163.com)

可行的方式[3]。

经典的傅里叶变换实现了时域到频域的线性转换，它的出现无论是对工程应用还是对数据科学都产生了深远的影响。而量子傅里叶变换是经典傅里叶变换的量子对应，是Shor算法等众多量子算法中关键的一环[4]，深入研究量子傅里叶变换对于完善量子算法和推动量子计算的发展有着至关重要的作用。

在经典架构计算机上，量子傅里叶变换模拟的高效实现受限于程序中存在的大规模内存占用与稠密数据的密集计算。为了解决这一问题，现有量子傅里叶变换模拟往往基于超级计算机平台实现[5]。然而，传统的超级计算机平台虽然较好地解决了模拟实现过程中庞大的内存需求问题，但对于高度密集的计算过程仍存在着并行度较低、访存时延较大等问题[6]，难以取得较优的性能。

“嵩山”超级计算机系统是我国自主研发的新一代超级计算机，主海光一号CPU＋海光一号DCU加速器的体系架构符合当前国际主流异构超算的建设趋势，所配备的DCU加速器件使得该平台更适合部署高度密集型计算应用。因此，基于该平台实现量子傅里叶变换模拟能有效提升程序性能，有助于在量子计算机实验环境不完善的条件下深入研究量子傅里叶变换模拟应用的特性，完善一系列以量子傅里叶变换为核心的量子算法，并推动量子计算领域的发展。同时，作为我国自主可控的新一代超级计算机，量子傅里叶变换模拟在“嵩山”超级计算机系统上的实现与部署扩充了该平台的科学计算生态，验证了量子计算的研究在该平台上开展的可行性，为深入开展其他量子计算领域的研究奠定了基础。

本文在“嵩山”超级计算机系统上实现了量子傅里叶变换模拟的优化以及验证。根据CPU＋DCU异构编程模型模拟量子傅里叶变换的逻辑操作，本文通过对算法的静态分析与测试，较好地进行了CPU与DCU加速器的任务划分，使得密集计算任务部分能够充分利用DCU加速器件的计算资源。另外，本文通过在单个节点上开启MPI多进程，实现了多DCU数据传输和计算的并发优化，利用通信和计算的隐藏减少了DCU计算单元的闲置时间。实验结果表明：相较于在CPU上的串行实现，量子傅里叶变换模拟在CPU＋DCU架构上的异构并行实现运行效率显著提升，并且在集群上拥有良好的可拓展性。

本文第2节介绍了“嵩山”超级计算机平台的体系架构、DCU加速器硬件特性以及该平台异构编程模型；第3节介绍了量子傅里叶变换的理论基础；第4节详细叙述了量子傅里叶变换模拟在“嵩山”超级计算机平台上的实现过程以及优化方法；第5节根据实验数据对模拟实现的效果进行了分析；最后总结全文并展望未来。

2 “嵩山”超级计算机系统

部署于国家超级计算郑州中心的“嵩山”超级计算机系统是我国自主研发的新一代E级高性能计算机，它采用处理器＋加速器架构、多层次超高速网络技术，集成多层次、多协议、多种数据模式的高性能分布式并行存储系统，理论峰值算力可达到100 PFlops，整机实测峰值达到65 PFlops。其配备了我国自主研发的海光一号X86 CPU和海光一号DCU加速器。海光一号CPU采用14 nm工艺，单片CPU拥有32核心，稳定运行频率可达3.0 GHz以上；DCU加速器采用7 nm工艺，内部配备HBM2片上高速内存，其运算能力FP64的峰值可达6.5 TFlops。该平台单个计算节点上配备1颗CPU和4片DCU加速器，CPU和DCU间由PCIe3.0高速总线连接，能够实现节点内设备间的数据高速互通，并且通过CPU-Direct和DCU-Direct技术提升了CPU和DCU跨节点通信的均衡性。

2.1 海光一号DCU加速器架构

“嵩山”超级计算机系统配置的DCU加速器采用类GPU架构，其全局内存为16 GB，每片DCU加速器拥有60个计算单元(Compute Unit，CU)，CU上集成了大量用于数据处理的晶体管，指令处理器(command processor)用来进行全局的指令调度分配。

CU主要由4个SIMD计算组和1个线程束调度器(scheduler)构成。在每个SIMD计算组内有16个支持整型和浮点型计算的DCU核心以及16 384个32 bit寄存器(VGPR)。由64个线程(thread)组成的线程束(wavefront)在1个SIMD计算组上执行，每个SIMD计算组最多可以实现10个线程束的并发，单个线程最多可使用的VGPR资源为256个。CU内拥有64 kB的共享内存(LDS)，运行在当前CU上的线程对共享内存的访问速度相比全局内存更快，且不同CU间的共享内存相互独立。

DCU加速器上部署了大面积的计算单元，精简了控制单元，使得它在面向具有一定并行性、数据规模大、计算密集的程序时，拥有相比CPU更强的计算能力[7]。DCU加速器及CU的架构如图1所示。

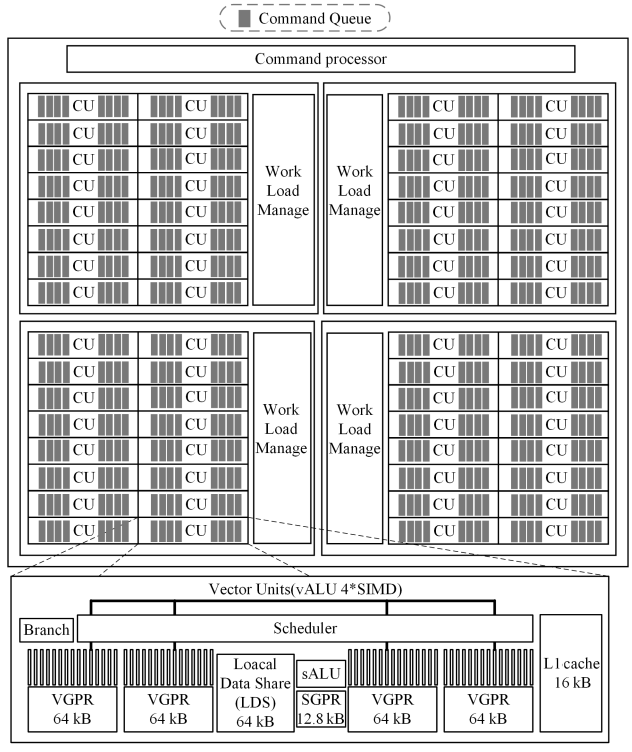

图1 DCU加速器及CU架构

Fig. 1 DCU accelerator and CU architecture

2.2 CPU+DCU 异构编程

CPU+DCU 异构编程模型将系统分为主机端(CPU)和设备端(DCU),主机端代码由 C/C++构成并运行在 CPU 上,设备端代码由 HIP-C 构成并运行在 DCU 加速器上,被称为核函数(Kernel)。典型 CPU+DCU 异构程序实现的流程如下:1)将数据从主机端内存传输至设备端内存;2)启动核函数对已传输至 DCU 上的数据进行运算等操作;3)将运算结果从 DCU 内存传输回 CPU 内存[8]。主机端通过调用运行时(hip_runtim)API 控制 DCU 上的内存分配、主机端和设备端之间的数据传输、核函数的启动以及 DCU 上内存的释放。

核函数作为在 DCU 上执行的代码,控制着 DCU 加速器上每一个线程按要求进行数据访问和计算操作。由单个核函数启动所产生的所有线程被称为网格(grid),一个网格由一定数量的线程块(block)构成,而一个线程块由一组线程(thread)构成。线程块和网格可以被组织成一维、二维和三维的形式,当核函数被主机端启动时,每一个线程、线程块、网格都会被自动分配不同的坐标变量。在编写核函数时,根据算法特性可以选用分治设计、平衡树设计、流水线设计、倍增设计等手段[9],并结合"线程-线程块-网格"的线程层次结构进行 DCU 加速器上的任务划分,根据参与计算的数据规模来确定开启的线程数,根据处理数据的类型来确定开启线程块和网格的维度,线程根据自身的坐标变量对数据进行读写和计算,从而实现 DCU 上多线程的并发计算。CPU+DCU 异构编程模型架构如图 2 所示。

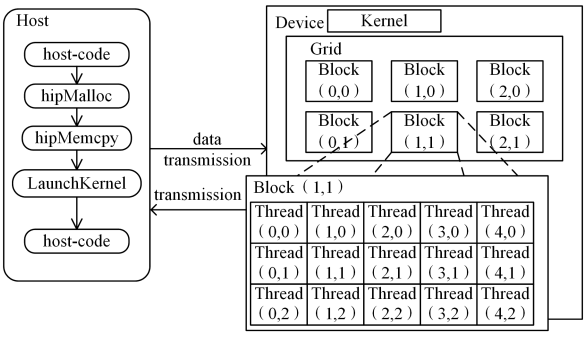

图 2 CPU+DCU 异构编程模型

Fig. 2 CPU+DCU heterogeneous programming model

根据 CPU+DCU 异构编程模型,通过合理的任务划分将程序中逻辑控制部分集中在主机端执行,令设备端承载密集的计算任务,以此充分利用计算资源,使得程序能够在异构平台上取得优秀的性能[10]。

3 量子傅里叶变换

量子傅里叶变换是传统的离散傅里叶变换的量子解决办法[11]。标准的离散傅里叶变换的输入是一个长度为 N 的复向量 $\boldsymbol{x}_0,\cdots,\boldsymbol{x}_{N-1}$,以复向量 $\boldsymbol{y}_0,\cdots,\boldsymbol{y}_{N-1}$ 为输出,其完整定义如下:

$$\boldsymbol{y}_k \rightarrow \frac{1}{\sqrt{N}}\sum_{j=0}^{N-1}\boldsymbol{x}_j e^{2\pi ijk/N}$$

量子傅里叶变换是一个定义在 N 维 Hilbert 空间上的离散傅里叶变换,它的输入为 $|0\rangle,\cdots,|N-1\rangle$,是 n^2 个可能的量子态的线性组合,其完整定义如下:

$$\sum_{j=0}^{2^n-1}x_j|j\rangle^n \rightarrow \frac{1}{\sqrt{2^n}}\sum_{k=0}^{2^n-1}\sum_{j=0}^{2^n-1}x_j e^{2\pi ijk/N}|k\rangle^n$$

当 n 个量子比特进行变换时,输入可表示为 $|j_1,\cdots,j_{n-1},j_n\rangle$,则输出应为:

$$|j_1,\cdots,j_{n-1},j_n\rangle \rightarrow (|0\rangle+e^{2\pi i_0\cdot j_n}|1\rangle)(|0\rangle+e^{2\pi i_0\cdot j_{n-1}j_n}|1\rangle)\cdots(|0\rangle+e^{2\pi i_0\cdot j_1\cdots j_{n-1}j_n}|1\rangle)/\sqrt{2^n}$$

量子傅里叶变换的线路主要由两种量子逻辑门构成,一种是 Hadamard 门,另一种是受控相位旋转的 R 门[12]。

$$H=\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix},R_k=\begin{pmatrix}1 & 0\\ 0 & 2^{2\pi i/2^k}\end{pmatrix}$$

输入为 $|j_1,\cdots,j_{n-1},j_n\rangle$ 时,量子傅里叶变换的线路如图 3 所示。

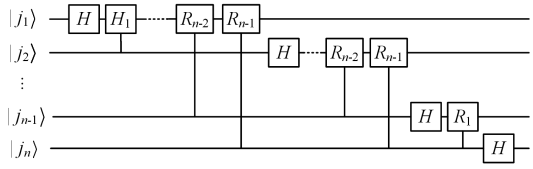

图 3 量子傅里叶变换线路图

Fig. 3 QFT circuit diagram

4 模拟实现

为了充分发挥"嵩山"超级计算机系统的异构计算性能,本文根据 CPU+DCU 异构编程模型对量子傅里叶变换模拟程序进行实现,通过任务划分将程序中的密集计算任务映射至 DCU 上,利用由 HIP-C 语言编写的核函数实现了部分量子门的逻辑操作,使用 MPI 多进程实现了计算节点上多 DCU 数据传输和计算的并发,利用计算和通信隐藏缩短了 DCU 加速器的闲置时间。

4.1 量子傅里叶变换模拟程序分析

传统量子傅里叶变换模拟程序由 C/C++语言编写并运行在 CPU 上,其伪代码如算法 1 所示。

算法 1

输入:N

输出:quantum_register

```
1. /* initialize quantum register */
2. Struct qr(int width=N, int id, float _Complex * amplitude[0~2^N-1])
3. Init_amplitude(qr.amplitude)
4. Init_id(qr.id)
5. /* Do R gate trasform */
6. If qr.id=qu_position
7. for i=0→2^N-1 do
8. i=i+1
9.     if (qr.amplitude[i] > qu_limit)
10.        R_update(qr.amplitude[i])
11. else
```

```
12. continue
13. end for
14. / * Do Hadamard transform * /
15. quantun_H_gate_wq(qr)
16. quantum_prob_inline( )
17. quantum_reg_print( )
18. check the result
```

根据代码，程序首先由输入参数确定模拟的量子比特位数，在设备上开辟足够的存储空间对每一个量子寄存器（qu_register）进行初态的制备，当输入的量子比特位数为 N 时，每个量子寄存器中需要储存 2^N 个概率幅信息，Init_id 函数和 Init_amplitude 函数分别赋予其量子基态信息和概率幅初始值。然后依据量子线路对量子傅里叶变换中的 Hadamard 变换（H 门）和相位移动变换（R 门）进行模拟：相移变换操作会对不同位量子寄存器的概率幅度进行更新，每个量子寄存器的相移变换仅与自身相位信息有关，与其他量子寄存器所包含的信息无关；量子寄存器中概率幅的更新操作相同，但由于进行更新操作的概率幅数据量庞大，因此其串行效率不佳。当概率幅全部完成更新并存回量子寄存器时，全局量子信息开始进行 Hadamard 变换，直至计算完毕并输出[13]，完成整个模拟过程。

对传统量子傅里叶变换模拟程序进行热点分析后发现，相移变换部分的操作占用了程序执行总时间的 90%，结合对程序的静态分析，量子傅里叶变换模拟具有如下特点：

（1）参与运算的 qr. amplitude 数据规模庞大，并且随着输入量子比特数的增加，量子寄存器中的数据规模呈指数级上升，数据的访存与传输对模拟性能影响较大。

（2）量子寄存器的相移变换操作仅与自身相位信息有关，与其他位的量子寄存器不存在先后数据的依赖性，具有较大粒度的并行性。

（3）相移变换操作中的概率幅更新过程存在数据密集且具有并行性的计算。

4.2 CPU＋DCU 版本量子傅里叶变换模拟的实现

“嵩山”超级计算机系统搭载的 DCU 加速器相较于 CPU 更擅长处理数据规模大、计算密集并具有并行性的过程。根据 DCU 加速器的硬件特性并结合对热点代码的分析，根据 CPU＋DCU 异构编程模型，本文通过 HIP-C 编写的核函数对 R 门的相位移动操作进行了模拟实现，从而将程序中的密集计算部分分配至 DCU 加速器上运行，逻辑控制密集部分、量子寄存器的初态制备以及 H 门的操作则由主机端代码在 CPU 端实现。

R 门中的概率幅更新过程是独立过程，“嵩山”超级计算机系统在单个计算节点上配备 4 片 DCU 加速器，因此本文选择将相位移动部分任务均匀划分成 4 份并传输至 4 片 DCU 加速器上，充分利用节点上的计算资源，从而进一步提升模拟实现的效率。异构版本的量子傅里叶变换模拟流程如图 4 所示。

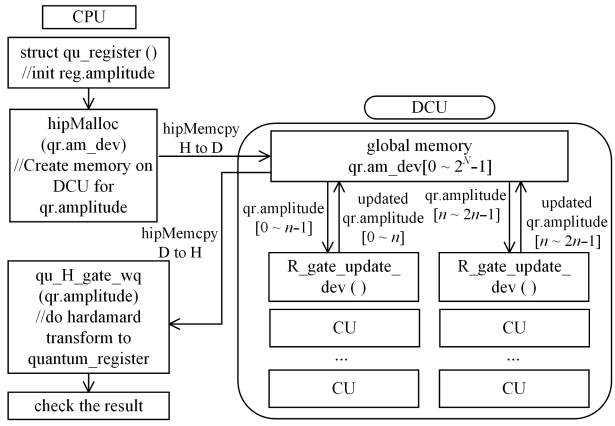

图 4 异构量子傅里叶变换模拟流程

Fig. 4 Heterogeneous QFT simulation process

量子寄存器的初始化由节点上的 CPU 完成后，CPU 调用 hipMalloc 函数在每一个 DCU 上开辟用于存储概率幅信息的空间，对概率幅信息进行连续的划分，然后调用 hipMemcpy 函数将量子寄存器中存储的概率幅信息的 1/4 通过 PCIe 总线分别传输至每一个 DCU 的全局内存内。

接下来根据输入的量子比特数目确定 DCU 上网格和线程块的配置情况。在每片 DCU 加速器上开启的线程数目等同于 CPU 端初始化的量子寄存器总数目的 1/4，根据概率幅信息的排列形式，确定线程组织的形式为一维。DCU 加速器的基本运算单元线程束内的线程个数为 64，考虑到当模拟量子比特位数为 N 时，概率幅信息的数量为 2^N，本文设定线程块内的线程数目为 256，当 $N \geqslant 8$ 时，设备端可以产生整数个活跃线程块，提高了 R 门操作在 DCU 上的设备利用率，同时，256 是线程束内线程个数的整数倍，保证了线程束调度器的切换效率。

完成网格和线程块的配置后，调用 hipLanchKernel 启动核函数，设备代码开始在 DCU 加速器上运行。在 Kernel 对 R 门操作的实现中，各线程首先根据自身的坐标变量（hipThreadIdx）和线程块的坐标变量（hipBlcokIdx）计算出其在全局中的偏移量，然后根据其全局偏移量从 DCU 的全局内存中读取对应位的概率幅信息（reg. amplitude）。由于概率幅信息的数据类型为复数，本文将线程执行的设备函数 R_gate_update_dev( )进行改写，将概率幅的更新操作分解为两部分，首先针对实数部分进行更新操作，然后再针对复数部分进行更新操作，避免了设备端计算过程中因出现格式转换而带来的效率问题。

DCU 上运算结束后，将更新过的概率幅信息从设备端传输回主机端，进行 Hadamard 变换，直至完成模拟过程。

4.3 CPU＋DCU 版本量子傅里叶变换模拟的优化

4.3.1 多进程控制 DCU 并发

CPU 和 DCU 间的数据传输过程由 hipMemcpy 函数控制，该函数以同步方式执行，调用该函数会阻塞主机端的运行。因此在异构版本的量子傅里叶变换中，若使用单个进程向 4 片 DCU 传输概率幅数据，当主机端调用 hipMemcpy 函

数开始向 1 片 DCU 传输数据时，CPU 端会被强制空闲，直至本次传输完成，单个进程无法控制主机端同时向 4 片 DCU 传输概率幅数据，并且数据传输的串行模式会严重损害模拟实现的运行效率。

为实现数据传输的并行，传统的优化方法是把需要传输的数据初始化为锁页内存，创建 hip 流，将数据向 4 片 DCU 的传输和传回加载到不同的流上[14]，以实现并发。但是由于创建量子寄存器数目巨大，初始化过大数据规模的锁页内存会严重降低主机性能[15]，因此采用锁页内存的方式不适用于“嵩山”超级计算机平台上量子傅里叶变换模拟的实现。

通过对平台计算节点上的硬件架构分析得知，海光一号 CPU 将 32 核心(core)划分为 4 个分区(Die)，每个分区通过硬件连接直接控制节点上的 1 个 DCU 加速器。根据该硬件特性，本文使用 MPI 开启 4 个进程，提交作业时使用参数--sockets-per-node=4，确保每个进程由 4 个不同分区上的 core 启动[16]。

首先在主机端使用 MPI_Comm_rank[17] 获取当前进程的 rankID，调用 hipGetDevice 函数获取 DCU 加速器的设备编号 DeviceID，然后确定进程编号和设备编号的对应关系。在当前进程中使用 hipSetdevice 函数完成其与指定编号 DCU 的绑定，此时，在主机端对设备端的操作均由 4 个独立进程对其绑定的 DCU 并发进行。

该优化方法下，根据硬件设计的特性并利用多进程的并行特性，对来自 CPU 上 4 个不同分区上的 4 个进程独立控制开辟 DCU 的显存空间，进行概率幅信息的数据传输、核函数的启动以及设备端 R 门操作的运算，解决了原先的概率幅信息在主机和设备间传输时出现的串行情况，避免了跨分区控制 DCU 加速器的额外开销，提高了程序的模拟效率，缩短了程序的运行时间。多进程对 DCU 加速器的并发控制如图 5 所示。

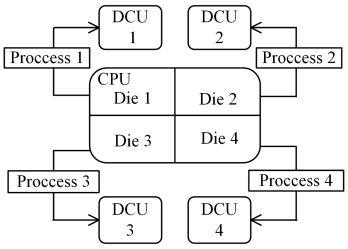

图 5 多进程控制 DCU 加速器

Fig. 5 Multi-process control DCU accelerator

4.3.2 计算通信隐藏

尽管 CPU 和 DCU 加速器间的连接采用 PCIe 3.0 总线，具有非常快的传输速度[18]，但当模拟的量子比特数增大时，概率幅信息的数据规模呈指数级上升，通过 dcuprof 工具分析，当输入量子比特位数 $N \geqslant 30$ 时，概率幅信息在主机和设备端的传输时间急剧增加，设备端进行 R 门转换操作的时间占比出现下降，DCU 在较长时间内处于闲置状态，损害了模拟实现的效率。

为了减少 DCU 加速器在数据传输时的等待时间，进一步提高模拟实现的效率，本文对数据传输和 DCU 上相移变换运算的时间进行了统计，根据参与运算的概率幅的数目与运算时间的关系，将控制相移变换的核函数 Kernel 按照数据规模分割为 Kernel_0，Kernel_1 和 Kernel_2，以实现时间上的均匀划分，从而达到数据传输和 DCU 上计算的时延隐藏，避免了 DCU 加速器在数据传输时处于较长时间的闲置状态，其优化效果如图 6 所示。

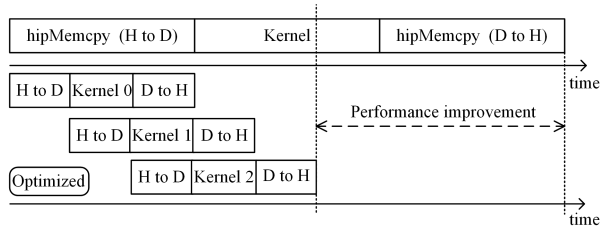

图 6 数据传输和计算的时延隐藏

Fig. 6 Data transmission and calculation delay hiding

5 实验验证

5.1 实验环境

“嵩山”超级计算机系统的操作系统为 Centos-6.7，集群作业管理系统为 Gridview，CPU 编译器为 gcc/g++-7.3.1，DCU 加速器编译器为 hipcc-2.9.6，MPI 版本为 hpcx-2.4.1。海光一号 CPU 单片上拥有 32 个核心，最多可同时开启 64 个线程，稳定运行可达 3.0 GHz 以上；DCU 加速器采用类 GPU 架构，具有 16 GB 的全局内存，FP64 峰值可达 6.5 TFlops。平台的单个计算节点上 CPU 和 DCU 的配比为 1∶4，拥有 128 GB 内存空间，计算节点间配备 200 Gbps 的高速互通网络。

5.2 功能性测试

在“嵩山”超级计算机系统的单个计算节点上，首先对并行异构的量子傅里叶变换模拟进行 20 量子比特输入的实现并记录其输出，然后进行量子傅里叶变换模拟的串行实现并记录其输出。经过对比，输出中对应位上的概率信息一致，且通过正确性验证。运行耗时对比如表 1 所列。

表 1 20 Qubits 耗时对比

Table 1 20 Qubits time-consuming comparison

版本	时间/s	加速比
串行实现	8.171	5.68
并行异构实现	1.438	

由于并行异构的模拟中存在 CPU 和 DCU 间的数据传输，在输入量子比特规模较小的情况下，数据传输所用时间在模拟实现过程中占比较大，计算耗时占比较小，因此，随着输入量子比特数目的增大，加速比应取得进一步的提升。接下来逐步增大量子比特的输入，在单个计算节点上依次对 24 量子比特、28 量子比特、30 量子比特、32 量子比特进行两个版本的模拟实现，运行耗时对比如表 2 所列。

表 2 24～32 Qubits 耗时对比

Table 2 24 Qubits to 32 Qubits time-consuming comparison

Qubits	24	28	30	32
串行实现耗时/s	42.337	1021.043	4019.502	20332.016
并行异构实现耗时/s	6.399	115.478	412.875	1753.621
加速比	6.616	8.841	9.735	11.594

由表 2 数据可知，加速比情况符合分析预期，随着输入量子比特数的增加，加速效果逐步提升，当输入量子比特数为 32 时，加速比达到了 11.594，实验数据证明了并行异构实现的量子傅里叶变换模拟在“嵩山”超级计算机平台的单个节点上取得了优秀的性能，大幅提高了模拟实现的效率。

由于单个节点的内存限制，32 量子比特为单个计算节点模拟实现的规模上限，本文选择对并行异构实现的量子傅里叶变换模拟在集群上的实现进行实验，输入量子比特数为 32，以单个节点上程序运行时间为基准，测试该模拟程序跨节点运行的 MPI 并行效率，测试结果如表 3 所列。

表 3 32 Qubits 可扩展性测试

Table 3 32 Qubits scalability test

nodes	Time/s	MPI parallel efficiency/%
1	1753.621	100
2	897.032	97.74
4	474.172	92.45
8	251.218	87.33
16	131.104	83.59
32	69.781	78.53
64	37.332	73.39

测试结果显示，当输入量子比特数为 32 时，程序在 32 个节点上的 MPI 并行效率接近 80%，并且取得了优秀的性能。实验证明了并行异构实现的量子傅里叶变换模拟在“嵩山”超级计算机系统的集群上具有良好的可拓展性。由于 32 量子比特的数据规模有限，随着并行节点数目的进一步增大，MPI 消息传递的开销持续增加，MPI 并行的效率不佳，输入为 32 量子比特的量子傅里叶变换模拟已经不适合在更大节点规模下实现。为了进一步证明并行异构量子傅里叶变换模拟在更大节点规模下针对更大规模数据输入的性能表现优异，本文对输入量子比特数从 34 增加至 44 的情况，分别在 32 至增加 2048 个计算节点上进行了大规模测试，结果如表 4 所列。

表 4 大规模测试

Table 4 Large-scale testing

Qubits	nodes	time/s
34	32	285.387
36	64	681.218
38	128	1485.948
40	256	3779.029
42	1024	3862.838
44	2048	8542.651

实验首次在该超算系统上实现了 44 量子比特输入的量子傅里叶变换模拟。实验数据表明，并行异构实现的量子傅里叶变换模拟在“嵩山”超级计算机系统平台上针对大规模数据输入取得了优秀的性能。

结束语 本文根据“嵩山”超级计算机系统的异构架构，利用 C/C++语言和 HIP-C 语言实现了量子傅里叶变换模拟的异构版本，并利用 MPI 多进程控制 DCU 的并发、通信与计算隐藏，提高了模拟实现的效率。实验结果验证了本文提出的方案的有效性。同时，本文验证了量子计算模拟在“嵩山”超级计算机系统上的可行性，拓展了该平台的科学研究生态领域，为其他量子算法模拟在该系统上的部署奠定了基础。本文对实现量子门逻辑操作的核函数部分未做进一步的优化，对其访存及线程束分化的处理会进一步提升模拟实现的效率。

参 考 文 献

[1] GIBNEY E. Quantum computer race intensifies as alternative technology gains steam[J]. Nature, 2020, 587(7834): 342-343.

[2] CHO A. Google claims quantum computing milestone [J]. Science, 2019, 365(6460): 1364.

[3] LLOYD S, GARNERONE S, ZANARDI P. Quantum algorithms for topological and geometric analysis of data[J]. Nature Communications, 2016, 7: 10138.

[4] ZHOU S S, LOKE T, IZAAC J A, et al. Quantum Fourier transform in computational basis[J]. Quantum Information Processing, 2017, 16(3): 1-19.

[5] LIU X N, JING L N. Large scale Quantum Fourier Transform Simulation Based on SW26010[J]. Computer Science, 2020, 47(8): 93-97.

[6] LIU X, YANG Z, YANG Y. A nested split load balancing algorithm for Tianhe No. 2[J]. Computer Research and Development, 2018, 55(2): 418-425.

[7] BAKHODA A, YUAN G L, FUNG W W L, et al. Analyzing CUDA workloads using a detailed gpu simulator[C]// the 2009 IEEE International Symposium on Performance Analysis of Systems and Software. 2009.

[8] JOHN C. Professional CUDAC Programming [M]. Wiley Inter Science, 2014.

[9] GUPTA S, BABU M R. Generating Performance Analysis of GPU Compared to Single-core and Multi-core CPU for Natural Language Applications[J]. International Journal of Advanced Computer Science and Applications, 2011, 2(5): 50-53.

[10] CHENG S Y. Research on performance evaluation and optimization technology of heterogeneous(CPU-GPU) computer systems [D]. National University of Defense Technology, 2011.

[11] HASANIJAFARI S, PARSAMEHR S. Solving the Fourier Transform Issue Using Quantum Coherent States[J]. International Journal of Theoretical Physics, 2019, 58(8): 2407-2413.

[12] LIU X, GUO H, SUN R J, et al. The Characteristic Analysis and Exascale Scalability Research of Large Scale Parallel Applications on Sunway Taihulight Supercomputer[J]. Journal of Computer, 2018, 14(10): 2209-2220.

[13] YE C, ZHENG S G, LONG C, et al. Quantum Fourier Transform and Phase Estimation in Qudit System[J]. Communica-

tions in Theoretical Physics, 2011, 55(5): 790-794.
[14] COURTNEY D. The guide to CUDA[M]. Create Space Independent Publishing Platform, 2015.
[15] YU Q, CHILDERS B, HUANG L, et al. A quantitative evaluation of unified memory in GPUs[J]. The Journal of Supercomputing, 2020, 76(2): 2958-2985.
[16] SEREN S, CAN Ö. Integer programming based heterogeneous CPU-GPU cluster schedulers for SLURM resource manager [J]. Journal of Computer and System Sciences, 2015, 81(1): 38-56.
[17] FORUM M P. MPI: A Message-Passing Interface Standard[J]. Intl J of Supercomputing Applications, 1994, 8(2): 179.
[18] WANG Y H, QIAO J Z, LIN S K, et al. An Optimization Strategy for Improving Throughput of GPU Global Memory[J]. Journal of Grey System, 2018, 30(2): 42-56.

XIE Jing-ming, born in 1995, postgraduate. His main research interests include high-performance parallel computation and heterogeneous computation.

HAN Lin, born in 1978, Ph.D, associate professor, master instructor, is a member of China Computer Federation. His main research interests include high-performance computing and compilation optimization.