计算机科学
COMPUTER SCIENCE

基于混合精度的分布式GMRES算法优化

郭帅哲，高建花，计卫星

引用本文

郭帅哲，高建花，计卫星. 基于混合精度的分布式GMRES算法优化[J]. 计算机科学，2024，51(9)：15-22.

GUO Shuaizhe, GAO Jianhua, JI Weixing. Optimizing Distributed GMRES Algorithm with Mixed Precision [J]. Computer Science, 2024, 51(9): 15-22.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.231000204

基于混合精度的分布式 GMRES 算法优化

郭帅哲　高建花　计卫星
北京理工大学计算机学院　北京 100081
(chakra_guo@bit.edu.cn)

摘　要　广义最小残差法(Generalized Minimum Residual,GMRES)是一种求解稀疏线性系统的迭代方法,被广泛应用于科学与工程计算等领域。数据量的爆炸式增长,使得 GMRES 算法求解的问题规模快速膨胀。为了支持大规模问题的求解,研究人员提出了面向集群的分布式 GMRES 算法。然而在现有的大多数集群中,节点间的网络性能仍与节点内的 GPU 高速互联网络存在较大差距,限制了分布式 GMRES 算法的性能。针对 GPU 集群上的分布式 GMRES 算法,提出了一种基于混合精度的加速求解方法,使用低精度浮点表示,显著降低了通信过程的时间开销。此外,提出了一种数据传输的精度调控算法,动态自适应调整传输数据的精度,以保证迭代算法最佳的求解效果。实验结果表明,所提基于混合精度的优化方法可实现平均 2.4 倍的加速比,结合其他优化方法后可实现平均 7.6 倍的加速比。

关键词:广义最小残差法;混合精度;GPU 集群;分布式系统
中图分类号　TP311;O246

Optimizing Distributed GMRES Algorithm with Mixed Precision

GUO Shuaizhe,GAO Jianhua and JI Weixing
School of Computer Science and Technology,Beijing Institute of Technology,Beijing 100081,China

Abstract　The generalized minimum residual(GMRES) method is an iterative method for solving sparse linear systems. It is broadly used in many areas like scientific and engineering computing. The exponential data growth makes the scale of problems solved by the GMRES algorithm expand rapidly. To support the solving of large-scale problems,researchers have implemented distributed GMRES algorithm on clusters. However,the current inter-node network still significantly lags behind intra-node fabrics in terms of both bandwidth and latency,which greatly limits the performance of the distributed GMRES algorithm. This paper proposes a mixed-precision approach for optimizing the GMRES algorithm on GPU clusters,where the data transferred is represented in a low-precision format,the network traffic during inter-GPU communication is significantly reduced. In addition,this paper proposes a balancing algorithm that dynamically adjusts the precision of the data transferred to achieve the satisfied residual. Experimental results show that the proposed method achieves an average speedup of 2.4×,and a further average speedup of 7.6× when combined with other optimizations.

Keywords　Generalized minimum residual,Mixed precision,GPU cluster,Distributed system

1　引言

GMRES[1]即广义最小残差法(Generalized Minimum Residuals),是一种求解形如 $\boldsymbol{Ax}=\boldsymbol{b}$ 的大型稀疏线性系统的迭代方法,被广泛应用于计算电磁学、计算流体力学等多个领域。近年来信息社会的数据量发生了爆炸式的增长,出现了不少超大规模的稀疏矩阵。在 SuiteSparse[2]这个公开的稀疏矩阵数据集上记录的最大稀疏矩阵是 2020 年的 AGATHA_2015,该矩阵拥有超过 1 亿行和超过 115 亿的非零元,若以双精度浮点格式和 64 位整型索引的 CSR 格式进行存储,仅稀疏矩阵本身就将占用超过 186 GB 的存储空间。目前,绝大多数的 GPU 都无法容纳如此规模的稀疏矩阵,即使是在 GPU 集群上,单个节点内的 GPU 数量也是有限的。因此,为了求解超大规模的稀疏线性系统问题,设计适用于 GPU 集群的稀疏矩阵相关算法是非常必要的。

随着 GPU 向通用 GPU(GPGPU)的演化,近几年出现了大量以 GPU 为核心的高性能集群系统[3]。常见的 CPU+GPU 集群系统主要包含了以下几类部件:CPU,GPU,NIC 和 PCIe Switch。由于客户要求和制造商的不同,各类部件的配比和连接方式存在一定的差异,这会显著影响集群上各种算法的性能。图 1 展示了一种典型的 GPU 集群的节点拓扑结构,其拥有双路 CPU 和 8 个 GPU,每个 GPU 拥有独立的 NIC 与其他节点相连。节点内的 8 个 GPU 两两之间均可通过私有的高速总线(如 NVLink)进行通信;而跨节点的 GPU 通信需要经过 PCIe Switch、NIC、外部的交换机等部件。

到稿日期:2023-10-30　返修日期:2024-03-29
通信作者:高建花(gjh@bit.edu.cn)

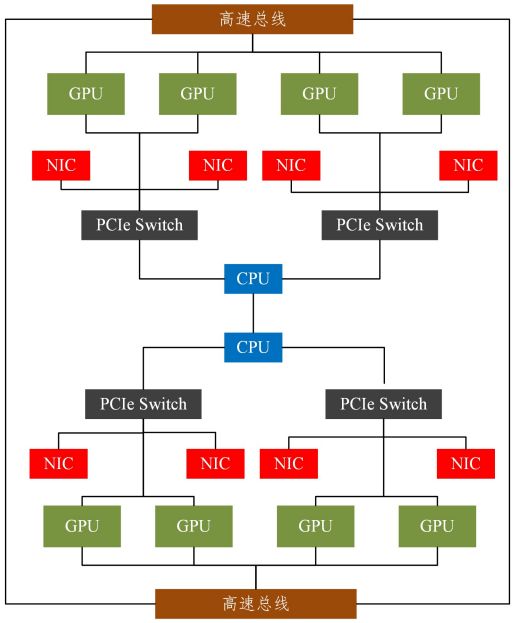

图 1 一种典型的 GPU 集群的节点拓扑结构

Fig. 1 Typical node topology of GPU cluster

单个节点内能够容纳的 GPU 数量是有限的，通常仅为 4～8 个。为了解决更大规模的问题，就需要使用分布在多个节点上的 GPU。前期的测试表明，在本文采用的集群平台上，使用相同数量的 GPU，GMRES 算法运行在 2 个节点上的时间约为其在单节点上运行时间的 10 倍。这是因为节点间的通信开销极大地限制了分布式 GMRES 算法的性能。图 2 展示了 GPU 间几种通信路径的带宽情况，其中图 2(a)代表了常见的 GPU 集群上的节点内的 GPU 通信情况，图 2(b)代表了常见的 GPU 集群上跨节点的 GPU 通信情况，图 2(c)则反映了近两年由 NVIDIA 发布的采用了 NVLink Switch 系统[4]的 GPU 集群上跨节点的 GPU 通信情况。不难看出，对于绝大多数现有的 GPU 集群系统来说，跨节点的 GPU 通信性能主要受 PCIe 和 NIC 带宽的限制。即使是 400 Gbps 网络逐渐成熟的今天，其带宽（双向共 100 GB/s）以及 PCIe 5.0×16 的带宽（双向共 128 GB/s）相比最新的 NVLink4（双向共 900 GB/s）[4]也有 8～9 倍的差距。

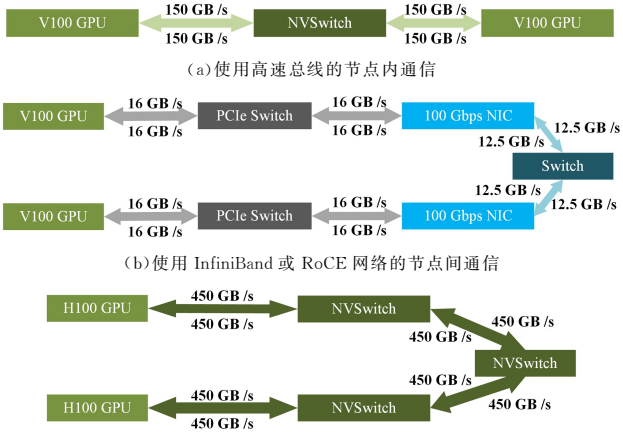

图 2 GPU 间几种通信路径的带宽情况

Fig. 2 Bandwidth of several types of inter-GPU communication routes

针对多节点的 GMRES 算法的加速，主要的思想就是尽可能减少通信（特别是跨节点通信）的开销。例如，Khodja 等通过对传输的数据进行压缩，有选择地传输对应节点需要的数据，实现了对分布式 GMRES 算法的加速[5]。本文提出了一种基于混合精度的分布式 GMRES 算法的优化方法，使用低精度的浮点表示，大幅降低了 GPU 间传输数据的网络流量。考虑到低精度表示对迭代收敛性的影响，本文进一步提出了一种调控算法，以平衡低精度传输带来的性能收益和迭代收敛性问题。在实验评估中，本文将基于混合精度的优化方法与 Khodja 等提出的选择性传输方法[5]相结合，结果表明整合之后的方法实现了更高的加速效果。

本文第 2 章详细介绍了本文使用的分布式 GMRES 算法，分析了其性能瓶颈，介绍了本文提出的混合精度优化算法和调控算法；第 3 章对本文提出的优化策略进行了性能评估；第 4 章介绍了针对多 GPU 上的 GMRES 算法优化的相关工作；最后总结全文。

2 基于混合精度的分布式 GMRES 优化

2.1 支持多节点多 GPU 的 GMRES 算法

目前已有多个开源库实现了 GPU 上的 GMRES 算法，但能够支持多 GPU 和多节点的版本较少。本文选择了由 ROCm 软件平台开源的 rocALUTION[6]，这个稀疏求解库实现了一种灵活的 GMRES 算法（FGMRES），并面向实验使用的 NVIDIA GPU 集群平台进行了移植。在不使用预条件器时，该算法的主要流程如算法 1 所示。

该算法首先使用稀疏矩阵向量乘（SpMV）和 BLAS 操作计算初始的 Krylov 子空间向量 $\boldsymbol{v}[0]$，然后通过求 2-范数计算初始残差 $\boldsymbol{r}[0]$。之后开始外迭代过程，先将 $\boldsymbol{v}[0]$进行标准化后，进入内迭代过程。内迭代中使用 SpMV，BLAS 和 2-范数操作构建海森堡矩阵 $\boldsymbol{H}$，接着使用标量运算生成并应用 Givens变换求 $\boldsymbol{H}$ 的 QR 分解。然后使用 2-范数计算残差，当残差满足条件或达到 Krylov 子空间维度时，结束内迭代过程。之后通过标量运算求解关于 $\boldsymbol{r}$，$\boldsymbol{H}$ 的上三角系统，并使用 BLAS 操作更新解向量 $\boldsymbol{x}$，最后使用 SpMV，BLAS 和 2-范数操作得到一轮外迭代后的残差情况，若满足收敛条件，则结束整个迭代过程。

算法 1 FGMRES 算法

输入：稀疏矩阵 **A**，向量 **b**，初始解 **x**

输出：近似解 **x**

1. 令 Krylov 子空间的维度 m=30
2. 创建存储标准正交基的数组 **v**[m][nRow]
3. **v**[0]=**b**−**Ax**
4. **r**[0]=‖**v**[0]‖$_2$
5. 检查初始残差 r[0]
6. 外迭代过程：
7. 　将 **v**[0]标准化：**v**[0]*=1/**r**[0]
8. 　内迭代（Arnoldi 过程）：
9. 　　构建海森堡矩阵 **H**
10. 　　使用 Givens 变换求 **H** 的 QR 分解
11. 　　计算残差 **r**[i]，检查收敛性
12. 　求解关于 **r**，**H** 的上三角系统

13. 更新解向量 $\boldsymbol{x}$

14. 计算残差 $\boldsymbol{r}[0]$，检查收敛性

由于 Krylov 子空间的维度相比稀疏矩阵的维度较小，因此本文实现的 GMRES 算法中关于海森堡矩阵 $\boldsymbol{H}$ 的操作在 CPU 上串行实现，而其余部分在 GPU 上实现。在 GPU 上实现的操作可以归纳为以下几类：SpMV、稠密向量的逐元素乘加、稠密向量点乘、稠密向量的 2-范数。其中，SpMV 的部分采用了 cuSPARSE[1)] 的实现，其余的稠密向量操作采用 cuBLAS[2)] 的实现。

然而，要将面向单 GPU 实现的算法扩展到多个节点的多个 GPU，还需要对算法进行一定的改造。比如，对稀疏矩阵和相关的向量进行划分，使其存放在相应的计算设备上。

前面对 GMRES 算法涉及的基本操作进行了归纳，其中在 CPU 上执行的部分由于数据量很小，因此多 GPU 版本的算法中仍由 GPU 对应的 CPU 进程负责，且所有进程均完成同样的操作，通过冗余计算来避免这部分的通信。而在 GPU 上实现的 4 类操作需要进行一定的改造。首先是 SpMV 操作，输入为稀疏矩阵 $\boldsymbol{A}$ 和向量 $\boldsymbol{x}$，计算 $\boldsymbol{y}=\boldsymbol{Ax}$。在多 GPU 的算法中，每个 GPU 仅持有原稀疏矩阵的部分行和向量 $\boldsymbol{x}$，$\boldsymbol{y}$ 的对应部分。为了进行 SpMV 操作，需要先从其他节点拿到相应的 $\boldsymbol{x}$ 向量，将其合成完整的向量才能进行 SpMV 的计算。而 SpMV 的结果正好是该节点持有的部分向量 $\boldsymbol{y}$，因此 SpMV 计算结束后不需要进行通信。然后是稠密向量的逐元素乘加操作，该操作属于逐元素操作，而乘加的系数也是各节点均持有的，因此只需要对当前设备上的部分向量进行操作即可，无需任何的通信。接着是稠密向量的点乘操作，点乘是将两个向量的对应元素相乘，然后将结果求和。各设备上部分向量计算出的结果是部分结果，需要额外进行 Allreduce 操作才能让所有设备都拿到正确的结果。最后是稠密向量求 2-范数，也就是对向量元素的平方求和并计算平方根。在单 GPU 上可以直接调用 cuBLAS 里的 cublas〈t〉nrm2 函数进行计算，而在多 GPU 上，需要拆分为 3 步：各 GPU 计算部分 $\boldsymbol{x}\cdot\boldsymbol{x}$ 的结果；使用 Allreduce 操作得到 $\boldsymbol{x}\cdot\boldsymbol{x}$ 的完整结果 $nrm2$；计算 $\sqrt{nrm2}$，得到向量 $\boldsymbol{x}$ 的 2-范数。

至此，GMRES 算法涉及的底层操作均实现了对多个计算设备的支持，之后只需要在迭代结束时添加从各 GPU 收集解向量 $\boldsymbol{x}$ 的通信过程，即完成了整个 GMRES 算法对多节点多 GPU 的支持。关于通信部分的实现，本文对 CPU 间的通信部分使用 MPI 的相关 API，GPU 间的通信部分则使用 NCCL 的相关 API。下一节将详细介绍使用混合精度来加速分布式 GMRES 算法的优化方法。

2.2 基于混合精度的优化方法

通信是分布式 GMRES 算法的性能瓶颈，在对 2.1 节介绍的分布式 GMRES 算法进行分析后发现，该算法中循环迭代部分的通信主要由两部分组成：SpMV 计算前对相乘向量 $\boldsymbol{x}$ 的通信，以及点乘和 2-范数计算时 Allreduce 操作产生的通信。由于 Allreduce 操作涉及的数据对每个 GPU 来说只有一个浮点数，因此绝大部分的通信来自于 SpMV 计算前的向量传输。

文献[5]提出了针对稀疏矩阵非零元分布选择性地传输一部分数据从而减少通信量的方法，但这种方法的效果与稀疏矩阵的非零元分布高度相关，且需要提前计算每个节点所需的具体数据的情况，并预留相应的通信缓冲空间，这会额外增加预处理的开销和运行时占用的显存空间。近年来，随着人工神经网络的流行，诞生了许多新型的低精度浮点数格式，如 TF32[7]，BF16[8] 和 FP8[9] 等。相较于传统高性能计算使用的双精度浮点数(FP64)，这些新型浮点数占用的空间要小很多，因此本文尝试了使用低精度浮点格式来表示待传输的数据，借此降低分布式 GMRES 算法的通信开销。由于待传输的数据本身就存储在 GPU 的显存中，因此本文设计了在 GPU 上进行数据转换的内核，实现低精度和高精度表示的相互转换。

算法 2 展示了在发送端将 FP64 数据转换为 BF16 数据的内核。该内核的基本思想是，通过调用 CUDA 内置的转换函数，每个 GPU 线程负责一个数据的精度转换。关于内核的启动参数，本文设置线程块大小为 128，grid 大小则为 $\lceil len/128 \rceil$，以保证覆盖到所有数据。

算法 2 FP64 转换为 BF16 的内核

输入：FP64 数组 **vh**，数组长度 len

输出：BF16 数组 **vl**

```
1. 计算当前线程的 id；
2. id=blockIdx.x * blockDim.x+threadIdx.x;
3. if id<len;
4.     vl[id]=__double2bfloat16(vh[id]).
```

实现了精度转换的功能后，还需要解决一个问题：使用何种浮点数格式来存储待传输的数据。可供选择的低精度浮点数主要有，FP32，FP16 和 BF16 这 3 种，它们在表示范围和有效数字位数方面均存在一定的差异。在已有实现中，参与计算的数值通常使用 FP64 或 FP32 进行表示，因此，本文主要考虑 FP16 和 BF16 这两种浮点格式。这两者的位宽均为 16 位，仅占用 2 字节的存储空间，相比 FP64 格式可以减少约 75%的网络流量。BF16 拥有与 FP32 相同的表示范围，而 FP16 的表示范围远小于 BF16 和 FP32，最大值仅为 65 504，但 FP16 比 BF16 拥有更多的有效位数，因此本文对两者都进行了测试。值得一提的是，BF16 的出现晚于 FP16，因此仅有最近几代的 GPU 对其提供运算的支持。本文提出的混合精度优化算法仅涉及 BF16 的类型转换，因此在诸如 NVIDIA V100 这种不支持 BF16 的 GPU 上也可以正常运行，这使得本文提出的优化算法具有广泛的适用性。用户只需要使用较新的 CUDA 库(CUDA 11 或更高版本)的 cuda_bf16.h 头文件中提供的__double2bfloat16 函数，即可实现 FP64 到 BF16 的类型转换。而低精度到高精度的转换并非是简单的逆过程，CUDA 库中不提供 BF16 直接转换为 FP64 的相关接口，需要先将其转换为 FP32 再转换为 FP64。因此，在数据接收端，BF16 向 FP64 转换的内核如算法 3 所示。

1) https://docs.nvidia.com/cuda/cusparse/index.html

2) https://docs.nvidia.com/cuda/cublas/index.html

算法 3 BF16 转换为 FP64 的内核

输入：BF16 数组 **vl**，数组长度 len

输出：FP64 数组 **vh**

```
1. 计算当前线程的 id：
2. id=blockIdx.x * blockDim.x+threadIdx.x；
3. if id<len：
4.    vh[id]=(double)__bfloat162float(vl[id]).
```

从算法 3 中可以观察到，与高精度到低精度的转换算法类似，低精度到高精度的转换过程依然是一个 GPU 线程处理一个数据，因此本文为两个内核设置相同的启动参数。调用 SpMV 前，在数据传输前后分别添加相应的高低精度转换的函数调用，并调整收发的缓冲区为低精度的缓冲，即可实现基于混合精度的传输优化。

2.3 性能与残差表现的调控算法

基于混合精度传输的优化可以减少约 75%的网络流量，但传输前后的精度转换使得传输数据的表示范围和有效位数受到了一定的影响，进而可能对迭代的收敛性产生一定的影响：经过相同的外迭代次数后，优化后的残差表现可能略差于未优化的版本；在极少数情况下，也可能出现残差无法下降的情况（具体见 3.3 节）。因此，本文提出了一种调控算法，用于平衡性能收益和残差表现。

该调控算法会记录过去 t 轮内/外迭代的残差情况，并在迭代算法每次检查残差时介入，当发现残差表现不及预期时，禁用基于混合精度的传输优化，以便在后续迭代中取得最佳的残差表现，并保证迭代算法的收敛性。该算法的流程大致如算法 4 所示。

算法 4 平衡调控算法

输入：当前外迭代轮数 iter，当前残差 res，历史残差数组 **resi**[t]，最大外迭代轮数 maxiter

输出：无

```
1. if iter≤initIters：
2.     禁用混合精度优化
3.     return
4. for i= 0 to t−2：
5.     resi[i]=resi[i+1]
6. resi[t−1]=res
7. if iter≤nxtIter：
8.     return
9. avg=desc=0
10. for i= 0 to t−2：
11.    if resi[i]−resi[i+1]>0：
12.        desc+=1
13. for i= 0 to t−1：
14.    avg+=resi[i]
15. avg/=t
16. evp=0
17. for i=0 to t−1：
18.    evp+=(resi[i]−avg)²
19. evp=√(evp/t)/avg
20. descR=(resi[0]−resi[t−1])/resi[0]
21. if iter>initIters+p&&
22.    ((evp>0.1&&desc<t/2)
23.    ||(desc≥t/2&&descR<R)
24.    ||desc==0)：
25.    nxtIter=maxIter
26.    禁用混合精度优化
27. else：
28.    继续启用混合精度优化进行迭代
```

该算法的具体思想是，对于初始的 $initIters$ 个外迭代周期，暂时不启用混合精度优化，以取得一个较好的初始残差表现，随后的 p 轮启用混合精度优化，在每次检查残差时更新过去 t 轮的残差记录，并计算以下 3 类指标.

1）过去 t 轮的残差的相对标准差 evp（即标准差除以平均值）；

2）过去 t 轮的残差的下降次数 $desc$；

3）过去 t 轮的残差的下降率 $descR$。

基于这 3 个指标，本文设计了 3 种判断是否终止混合精度优化的条件.

1）当 $evp>0.1\&\&desc<t/2$ 时，即认为残差已经难以下降并且残差出现了大幅的波动；

2）当 $desc>=t/2\&\&descR<R$ 时，即认为残差仍在下降，但下降速度已经非常缓慢；

3）当 $desc==0$ 时，即认为迭代过程出现了残差完全无法下降的情况。

当上述 3 种条件满足至少任意一种时，调控算法即认为当前的迭代过程已经不适用基于混合精度的优化，在之后的迭代中会返回到使用原始精度进行数据传输的状态，以保证最佳的残差表现。此外，其中的 $nxtIter$ 变量用于标记禁用混合精度优化条件已触发，在此后的迭代过程中不再执行算法第 8 行以后的部分。

3 加速效果与残差表现评估

3.1 测试环境与测试数据

本文的所有测试均在某集群平台上进行，其节点配置和相关软件信息如表 1 所列。该集群采用了常见的 CPU+GPU+NIC 的配置，所有 4 个 GPU 共享一个 Intel Omni-Path NIC 实现跨节点的通信，而存储系统则由存储节点提供，计算节点通过网络进行远程访问。

表 1 某集群平台的计算节点配置与软件环境

Table1 Compute node configuration and software environment of a cluster platform

CPU	2 * Intel Xeon E5-2640v4, 2.4 GHz
内存	128 GB DDR4-2133
GPU	4 * NVIDIA Tesla V100 16 GB
显存带宽	HBM2 900 GB/s
GPU 互联	NVLink2 x6 双向共 300 GB/s
NIC	Intel Omni-Path HFI Silicon 100 Series 双向共 25 GB/s 2 * Intel X540-T2 10 Gbps Ethernet Adapter
OS	CentOS 7.5
软件环境	Intel oneAPI 2023 NVIDIA HPC SDK 2022 CUDA 11.8 NCCL 2.13.4 GCC 7.5

在稀疏矩阵测试集方面，结合测试平台的 GPU 配置

情况，本文从 SuiteSparse Matrix Collection[2] 中选择了 12 个来自不同领域的大型稀疏矩阵，这些稀疏矩阵的基本信息如表 2 所列。它们的非零元规模均在千万级别，行列数从几十万到几千万不等。非零元数相近，行列数越大，通信在分布式 GMRES 算法中的开销占比就越高。

表 2　测试的稀疏矩阵的基本信息

Table 2　Basic information of tested sparse matrices

稀疏矩阵名称	行/列数	非零元数量
adaptive	6 815 744	27 248 640
AS365	3 799 275	22 736 152
cage14	1 505 785	27 130 349
F1	343 791	26 837 113
nlpkkt80	1 062 400	28 704 672
GAP-road	23 947 347	57 708 624
hugebubbles-00020	21 198 119	63 580 358
delaunay_n23	8 388 608	50 331 568
ljournal-2008	5 363 260	79 023 142
asia_osm	11 950 757	25 423 206
hugebubbles-00010	19 458 087	58 359 528
road_central	14 081 816	33 866 826

3.2　基于混合精度加速的性能表现

在本节的测试中，Krylov 子空间的维度设置为 30，对所有参与测试的稀疏矩阵统一执行 100 轮分布式 GMRES 算法的外迭代过程。根据集群平台的实际情况，所有的测试均在 2 节点共 4 个 GPU 的配置下进行。

为了尽可能降低 GPU 间通信的开销和预处理的开销，在稀疏矩阵的划分方面，这里统一采用按行数均分的方法而不是按非零元数量均分的方法，以平衡各 GPU 间的通信负载；同时，与稀疏矩阵相关的向量也按照此方法进行划分。此外，本文还针对测试平台实现了文献[5]提出的基于压缩向量传输的优化方法，并与其进行了对比。对于本文提出的基于混合精度的优化方法，实验选择了 FP16 和 BF16 两种浮点格式。性能测试的结果见图 3，对比的基准为不使用任何优化的分布式 GMRES 算法。

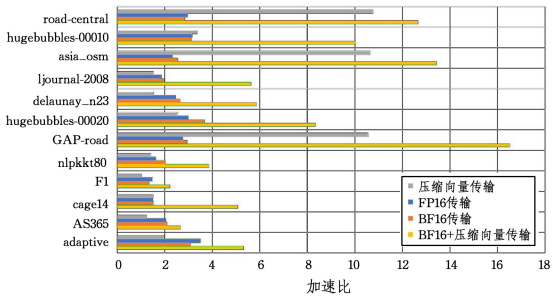

图 3　基于混合精度优化和压缩向量传输优化的分布式 GMRES 加速效果对比

Fig. 3　Performance comparison between mixed-precision optimization and compressed-vector optimization

测试结果表明，基于 FP16 版本的混合精度优化方法取得了平均 2.38 倍、最高 3.50 倍的加速比；基于 BF16 版本的方法取得了平均 2.48 倍、最高 3.67 倍的加速比；作为对比的基于压缩向量传输的优化方法取得了平均 4.45 倍、最高 10.76 倍的加速比。从这两项指标来看，基于压缩向量传输的方法似乎更好。然而，这种方法的加速效果高度依赖稀疏矩阵的非零元分布以及划分情况，对于非零元集中分布在每个子矩阵对角块内的稀疏矩阵可以取得非常好的压缩效果，但对于非零元分布较为分散的稀疏矩阵来说，压缩效果可能较差。

压缩效果可以用平均压缩率来衡量。本文定义压缩率 CR 为每个 GPU 上 SpMV 计算前从所有其他 GPU 设备上实际接收的总数据量与未优化时接收的总数据量的比值，如式(1)所示。

$$CR=\frac{\sum_{i=0}^{p}\boldsymbol{traffic\_opt}[i]}{\sum_{i=0}^{p}\boldsymbol{traffic\_org}[i]}\times 100\% \tag{1}$$

其中，GPU 总数为 p，$\boldsymbol{traffic_opt}[i]$为优化后每个 GPU 设备从设备 i 接收的数据量，$\boldsymbol{traffic_org}[i]$为优化前每个 GPU 设备从设备 i 接收的数据量。

而平均压缩率是所有 GPU 上的压缩率的平均值。平均压缩率越低，则通信部分的开销越小。对于本文提出的基于混合精度的压缩优化方法来说，在采用 16 位浮点格式传输时，其平均压缩率恒定为 25%，可以减少 75% 的网络流量。而对于基于压缩向量传输的优化方法来说，压缩效果直接由每个设备上的子矩阵的非零元分布决定，因此平均压缩率与稀疏矩阵高度相关，同时也受实际运行时的 GPU 数量影响。在本文的测试集和运行配置上，该方法取得了平均 29.05% 的压缩率，略高于本文提出的方法。图 4 展示了基于压缩向量传输方法的平均压缩率表现情况，可以看到，对于不同来源的稀疏矩阵，该方法的压缩效果差别极大，因此加速效果的差距也极大，最低的加速比仅为 1.02 倍，而最高的加速比却有 10.76 倍。事实上，从每个单独的测试矩阵的加速表现来看，在所有 12 个测试矩阵中，本文的方法在 8 个测试矩阵上的加速效果与基于压缩向量的方法相当或表现更好。简单来说，本文提出的优化方法对任何稀疏矩阵都可以提供相对稳定的加速效果。

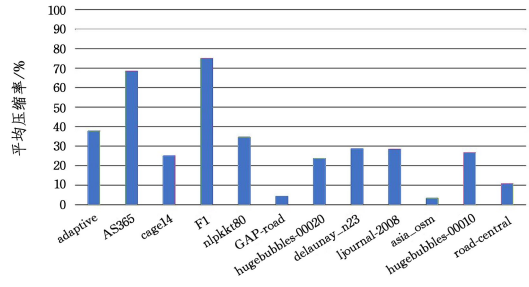

图 4　基于压缩向量传输的优化方法的平均压缩率

Fig. 4　Average compress ratio of compressed-vector optimization method

本文提出的方法主要是在传输前后添加了类型转换的操作，因此该方法可以与基于压缩向量传输的方法结合使用，以达到更好的加速效果。图 3 所示的测试结果表明，将两种方法结合起来可以取得平均 7.64 倍、最高 16.52 倍的加速比。

值得一提的是，压缩向量传输法会改变向量内数据的排列顺序，这可能会对点乘、2-范数等操作的结果产生影响，进而影响迭代的残差表现。因此，下一节的残差表现分析加入了基于 BF16 传输与压缩向量传输相结合的方法的残差表现。

3.3 基于混合精度加速的残差表现

本节将分析未优化的分布式 GMRES 算法、基于 FP16/BF16 优化后的算法以及基于 BF16 和压缩向量优化后的算法的残差表现。衡量迭代过程的残差表现的方法有很多，本文采用的是最小相对残差而非最终相对残差来衡量，即迭代过程中的最小残差与初始残差的比值。这是因为在测试设置的固定 100 轮外迭代的情况下，部分稀疏矩阵的迭代过程会出现残差下降到一定程度后开始振荡甚至是回升的情况。而在初值的选择方面，本文设定初始解为全 0，右端向量 $\boldsymbol{b}$ 为固定数值。

表 3 列出了这几种方法在各测试矩阵上经过 100 个外迭代周期后的残差表现。这里没有给出单独使用压缩向量方法的残差表现，因为本文对该方法的实现仅改变了 SpMV 时向量 $\boldsymbol{x}$ 元素的排列顺序，在单独使用时几乎不会对残差造成影响。然而，对于 BF16 和压缩向量相结合的情况来说，这种顺序的改变还是对残差表现造成了一定的影响，在部分情况下改善了残差表现，也有部分情况下恶化了残差表现，但在数值的数量级上没有出现明显的差别，本文认为这是低精度传输和改变向量元素顺序共同作用导致的。在分析了表 3 中各种优化方法的残差表现后可以发现，绝大部分情况下优化后的方法可以取得和未优化版本相近或者几乎一致的残差表现。尽管使用 FP16 的版本在这里取得了相对不错的结果，但其数值表示范围的限制仍然存在，因此对于测试矩阵和向量之外的情况，不能保证一定能够正常地迭代而不出现数值溢出的情况。相比之下，使用 BF16 的版本尽管有时残差表现不如 FP16 的版本，但其拥有与 FP32 相同的数值表示范围，因此在实际使用中通常无需担心数值溢出的问题。

表 3 各种版本的分布式 GMRES 算法在 100 轮迭代后的残差表现

Table 3 Residual of various distributed GMRES algorithms after 100 iterations

序号	稀疏矩阵	未优化版本	FP16	BF16	BF16+压缩向量
1	adaptive	$4.61*10^{-4}$	$4.83*10^{-4}$	$6.45*10^{-4}$	$8.82*10^{-4}$
2	AS365	$1.76*10^{-2}$	$1.80*10^{-2}$	$1.78*10^{-2}$	$1.98*10^{-2}$
3	cage14	$8.15*10^{-4}$	$8.42*10^{-4}$	$1.50*10^{-3}$	$7.62*10^{-4}$
4	F1	$6.63*10^{-1}$	未收敛	未收敛	未收敛
5	nlpkkt80	$4.00*10^{-1}$	$4.35*10^{-1}$	$5.35*10^{-1}$	$5.34*10^{-1}$
6	GAP-road	$3.26*10^{-1}$	$3.23*10^{-1}$	$3.25*10^{-1}$	$3.26*10^{-1}$
7	hugebubbles-00020	$1.19*10^{-3}$	$1.61*10^{-3}$	$3.04*10^{-3}$	$1.68*10^{-3}$
8	delaunay_n23	$2.80*10^{-2}$	$3.75*10^{-2}$	$4.08*10^{-2}$	$3.09*10^{-2}$
9	ljournal-2008	$8.07*10^{-1}$	$8.07*10^{-1}$	$8.07*10^{-1}$	$8.07*10^{-1}$
10	asia_osm	$3.72*10^{-2}$	$3.72*10^{-2}$	$3.72*10^{-2}$	$3.72*10^{-2}$
11	hugebubbles-00010	$1.31*10^{-3}$	$1.68*10^{-3}$	$2.70*10^{-3}$	$1.82*10^{-3}$
12	road_central	$1.04*10^{-1}$	$1.04*10^{-1}$	$1.04*10^{-1}$	$1.04*10^{-1}$

严格来说，BF16 的版本和 BF16 与压缩向量相结合的版本在大约一半的测试矩阵上的残差表现与未优化版本存在较大的差距；且在测试矩阵 4 上，任何使用 16 位浮点数进行传输的版本均出现了残差无法下降的情况。针对这类问题，本文在上一章提出了一种平衡调控算法，下一节将检验调控算法的实际效果。

3.4 平衡调控算法的效果评估

在实际使用中，若需取得最佳的残差表现，或者防止少数启用混合精度优化后可能出现的残差无法下降的情况，在采用混合精度传输和压缩向量传输相结合的基础上，可以启用本文提出的平衡调控算法。

本节测试了平衡调控算法在同时使用 BF16 和压缩向量优化方法上的残差控制情况，这里仅测试启用优化方法后残差表现与未优化版本存在显著差距的稀疏矩阵。对于其余的稀疏矩阵来说，调控算法对残差表现不会有明显的影响。

本节的测试中，平衡调控算法的参数 $initIters$，p，t 均取 10，R 取 0.025。从表 4 中可以看到，在启用了平衡调控算法后，残差表现明显更接近未优化的版本，同时对于此前残差无法下降的情况(测试矩阵 4)，调控算法能够检测到并及时纠正迭代过程。

值得一提的是，在启用调控算法后，基于混合精度传输的优化可能仅发生在迭代过程中的某一段，此时的加速效果还会受迭代收敛条件的影响。在不同领域中，收敛判断的方式和条件不同，导致基于混合精度优化传输的占比也不一致，无法统一衡量加速效果。因此，性能测试的部分将对不同的矩阵采用不同的迭代轮数进行测试。

表 4 启用平衡调控算法前后的 300 轮迭代的残差表现对比

Table 4 Residual comparison of 300 iterations before and after balancing control algorithmis enabled

序号	稀疏矩阵	未优化版本	BF16+压缩向量	BF16+压缩向量+平衡调控
1	adaptive	$4.58*10^{-4}$	$8.82*10^{-4}$	$5.65*10^{-4}$
4	F1	$3.79*10^{-1}$	未收敛	$4.40*10^{-1}$
5	nlpkkt80	$3.25*10^{-1}$	$5.34*10^{-1}$	$3.20*10^{-1}$
7	hugebubbles-00020	$8.98*10^{-4}$	$1.53*10^{-3}$	$9.00*10^{-4}$
11	hugebubbles-00010	$9.86*10^{-4}$	$1.66*10^{-3}$	$9.91*10^{-4}$

本文对这些测试矩阵的迭代过程进行了分析，对矩阵 1 进行 30 轮迭代，对矩阵 5 进行 50 轮迭代，对矩阵 4，7，11 进行 100 轮迭代，测试启用平衡调控算法的加速效果，结果如图 5 所示。启用平衡调控算法在这 5 个测试矩阵上取得了平均 1.22 倍、最高 1.34 倍的加速比。此时的加速比主要与迭代过程中使用混合精度传输优化阶段的占比相关，占比越高则加速效果越明显。

上述测试结果表明，基于混合精度传输的优化可以为分布式 GMRES 算法带来显著的性能提升。受稀疏矩阵非零元分布模式和分布式算法运行的 GPU 数量的影响，对于部分稀疏矩阵，优化后的算法的残差表现可能出现一定幅度的

退化。但在加入平衡调控算法后，可以在取得一定的性能提升的同时维持与双精度算法相近的残差表现。

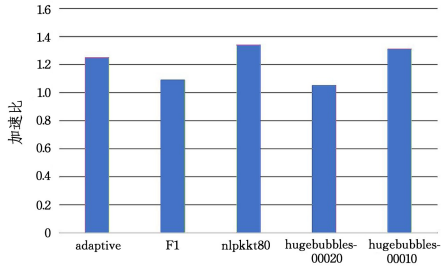

图 5 平衡调控算法的参考性能

Fig. 5 Reference performance of balancing control algorithm

4 相关工作

目前已有不少关于多 GPU 或者多节点多 GPU 上的 GMRES 算法的研究。除了基于压缩向量传输的优化方法[5]，Khodja 等还提出了基于超图划分的优化方法，其可以进一步减少通信的开销，实现更大的性能提升。

在迭代算法的实现方面，Ioannidis 分析了 PGMRES-classic，PGMRES-GPU 和 PGMRES-mGPU 等几种并行 GMRES 算法在 CPU 集群和 GPU 集群上的性能表现，并指出了各种实现的优势和局限[10]。Yamazaki 等实现了一种避免通信的 GMRES 算法(CA-GMRES)，并提出了基于域分解的预条件器，在 120 个 GPU 上相比标准 GMRES 算法有了 2.5 倍的性能提升[11]。

在集群上的迭代算法优化方面，Bahi 等在 GPU 集群上实现了一种高效的并行 GMRES 算法，使用相同数量的 GPU 相比使用 CPU 可实现 8 倍的性能提升[12]。Matsumoto 等在 GPU 集群上实现了一种用于 GT5D 的 CA-GMRES 算法，实现了最高 1.5 倍的加速效果[13]。He 等提出了一种名为 GPU-GMRES 的 GMRES 求解器[14]，其采用了不完全 LU 分解的预条件器，同时对 GMRES 算法中的 SpMV 部分提出了名为 segSpMV 的高效实现，使该迭代算法可以扩展至多个 GPU 上，对比纯 CPU 实现可提供 3～12 倍的加速效果。Lacoste 等分析了 PaStiX 这个并行稀疏求解器的内部调度器的优势和限制，并对 CPU/GPU 集群上稀疏求解器的调度和存储资源的占用进行了优化[15]。

在混合精度优化方面，Lindquist 等面向带重启动的 GMRES 算法，提出了基于混合精度的加速方法，通过对 GMRES 算法中的非关键操作使用低精度计算，而其余关键操作仍使用高精度计算，在 GPU 集群上实现了 8%～61%的性能提升[16]。

在应用相关方面，Bouchard 等讨论了一种主要用于 CFD 领域的在 CHAMPS 中实现的多 GPU 上的 GMRES 求解器，并在 1～8 个 GPU 的配置下评估了该实现的性能和扩展性，该算法中有大约 2/3 的时间用于通信和同步[17]。Ma 等在 PETSc 的框架上实现了一种近似的分块三角系统求解方法，并开发了分布式的分块的 SpMV 算法，以此优化了采用分块 Jacobi 预条件的多 GPU 上的 GMRES 算法，在使用 8 个 GPU 时相比纯 CPU 实现了约 4.4 倍的性能提升[18]。Devries 等提出了面向多核 CPU，GPU 以及多 GPU 系统的几种并行的 FGMRES 实现，并在真实的三维对流扩散问题的相关线性系统上进行了测试[19]。Zhang 等实现了一种性能可移植的用于求解可压缩的非结构化网格上的 Navier-Stokes 方程组的并行 GMRES 算法，相比常用的隐式算法(如 LU-SGS 等)拥有更好的收敛性和可移植性[20]。

总的来说，目前已有不少对分布式 GMRES 算法的通信优化工作，但基于混合精度的优化还处于起步阶段。另外有些面向通信优化的稀疏矩阵划分算法，比如基于超图划分的优化方法，存在预处理开销过大的问题。本文提出的基于混合精度的优化方法不仅显著减少了通信开销，还能和其他算法相结合实现进一步的通信优化。

结束语 本文提出了基于混合精度的优化方法，通过低精度的数据表示大幅降低了 GPU 间通信的网络流量，大幅提升了分布式 GMRES 算法的性能。同时，针对少部分情况下使用混合精度加速导致残差表现变差的情况，本文也设计了一种平衡调控算法，在发现启用混合精度优化不能达到理想的残差表现时，禁用混合精度的优化以保证最佳的迭代效果。此外，本文还对比了其他针对分布式 GMRES 算法的优化方法，相比之下，本文提出的方法拥有更稳定的加速效果，而在结合其他优化方法的情况下，也可以取得更进一步的性能提升。

参 考 文 献

[1] SAAD Y, SCHULTZ M H. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems[J]. Society for Industrial and Applied Mathematics, 1986, 7(3): 856-869.

[2] DAVIS T A, HU Y F. The University of Florida Sparse Matrix Collection[J]. ACM Transactions on Mathematical Software, 2011, 38(1): 1-25.

[3] Top500. June 2023 List [EB/OL]. [2023-10-01]. https://top500.org/lists/top500/2023/06/.

[4] NVIDIA. NVLink&NVSwitch[EB/OL]. [2023-10-01]. https://www.nvidia.com/en-us/data-center/nvlink/.

[5] KHODJA L Z, COUTURIER R, GIERSCH A, et al. Parallel sparse linear solver with GMRES method using minimization techniques of communications for GPU clusters[J]. The Journal of Supercomputing, 2014, 69: 200-224.

[6] ROCm Software Platform. rocALUTION [EB/OL]. [2023-10-01]. https://github.com/ROCmSoftwarePlatform/rocALUTION.

[7] NVIDIA Blog. TensorFloat-32 in the A100 GPU Accelerates AI Training, HPC up to 20x[EB/OL]. (2020-05-14)[2023-10-01]. https://blogs.nvidia.com/blog/2020/05/14/tensorfloat-32-precision-format/.

[8] Intel. BFLOAT16- hardware numerics definition [EB/OL]. (2018-11)[2023-10-01]. https://software.intel.com/sites/default/files/managed/40/8b/bf16-hardware-numerics-definition-white-paper.pdf.

[9] MICIKEVICIUS P, STOSIC D, BURGESS Net al. FP8 Formats

for Deep Learning[J]. arXiv:2209.05433v1,2022.
[10] IOANNIDIS E I,CHEIMARIOS N,SPYROPOULOS A N,et al. On the performance of various parallel GMRES implementations on CPU and GPU clusters[J]. arXiv:1906.04051,2019.
[11] YAMAZAKI I,RAJAMANICKAM S,BOMAN E G,et al. Domain Decomposition Preconditioners for Communication-Avoiding Krylov Methods on a Hybrid CPU/GPU Cluster[C]// Proceedings of the International Conference for High Performance Computing,Networking,Storage and Analysis. New Orleans(SC 14). LA,USA,2014:933-944.
[12] BAHI J M,COUTURIER R,KHODJA L Z,et al. Parallel GMRES implementation for solving sparse linear systems on GPU clusters[C]// Proceedings of the 19th High Performance Computing Symposia(HPC '11). 2011.
[13] MATSUMOTO K,IDOMURA Y,INAT,et al. Implementation and performance evaluation of a communication-avoiding GMRES method for stencil-based code on GPU cluster[J]. The Journal of Supercomputing,2019,75:8115-8146.
[14] HE K,TAN S X,ZHAO H Y,et al. Parallel GMRES solver for fast analysis of large linear dynamic systems on GPU platforms [J]. Integration,2016,52:10-22.
[15] LACOSTE X. Scheduling and memory optimizations for sparse direct solver on multi-core/multi-gpu duster systems[C]// Distributed,Parallel,and Cluster Computing. 2015.
[16] LINDQUIST N,LUSZCZEK P,DONGARRA J,et al. Accelerating Restarted GMRES With Mixed Precision Arithmetic[J]. IEEE Transactions on Parallel and Distributed Systems,2022,33(4):1027-1037.
[17] BOUCHARD A,PARENTEAU M,LAURENDEAU É. Toward a Multi-GPU Implementation of a GMRES Solver in CHAMPS [C]// The 8th Annual Chapel Implementers and Users Workshop. 2021.
[18] MA W P,HU Y W,YUANW,et al. Developing a Multi-GPU-Enabled Preconditioned GMRES with Inexact Triangular Solves for Block Sparse Matrices[J]. Mathematical Problems in Engineering:Theory,Methods and Applications,2021,2021(Pt. 9):6804723. 1-6804723. 17.
[19] DEVRIES B,IANNELLI J,TREFFTZ C,et al. Parallel Implementations of FGMRES for Solving Large,Sparse Non-symmetric Linear Systems[J]. Procedia Computer Science,2013,18:491-500.
[20] ZHANG J,DENG L,LI RT,et al. Achieving high performance and portable parallel GMRES algorithm for compressible flow simulations on unstructured grids[J]. The Journal of Supercomputing,2023,79:20116-20140.

GUO Shuaizhe, born in 2000, postgraduate, is a student member of CCF(No. P4392G). His main research interests include heterogeneous computing and performance optimization.

GAO Jianhua, born in 1995, Ph.D, postdoctoral researcher, is a professional member of CCF (No. 79759M). Her main research interests include parallel and high performance computing.

(责任编辑:柯颖)