计算机科学
COMPUTER SCIENCE

批量厄米矩阵特征值分解的GPU算法

黄荣锋, 刘世芳, 赵永华

引用本文

黄荣锋, 刘世芳, 赵永华.批量厄米矩阵特征值分解的GPU算法[J]. 计算机科学, 2023, 50(4): 397-403.

HUANG Rongfeng, LIU Shifang, ZHAO Yonghua. Batched Eigenvalue Decomposition Algorithms for Hermitian Matrices on GPU [J]. Computer Science, 2023, 50(4): 397-403.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.220100232

批量厄米矩阵特征值分解的 GPU 算法

黄荣锋[1,2]　刘世芳[1,2]　赵永华[1]
1 中国科学院计算机网络信息中心　北京 100080
2 中国科学院大学　北京 100080
(hrf@sccas.cn)

摘　要　批量矩阵计算问题广泛存在于科学计算与工程应用领域。随着性能的快速提升,GPU 已成为解决这类问题的重要工具之一。矩阵特征值分解属于双边分解,需要使用迭代算法进行求解,不同矩阵的迭代次数可能不同,因此,在 GPU 上设计批量矩阵的特征值分解算法比设计 LU 分解等单边分解算法更具挑战性。文中针对不同规模的矩阵,基于 Jacobi 算法设计了相应的批量厄米矩阵特征值分解 GPU 算法。对于共享内存无法存储的矩阵,采用矩阵"块"操作技术提升计算强度,从而提高 GPU 的资源利用率。所提算法完全在 GPU 上运行,避免了 CPU 与 GPU 之间的通信。在算法实现上,通过 kernel 融合减少了 kernel 启动负载和全局内存访问。在 V100 GPU 上的实验结果表明,所提算法优于已有工作。Roofline 性能分析模型表明,文中给出的实现已接近理论上限,达到了 4.11TFLOPS。

关键词:厄米矩阵;特征值分解;批量计算;Roofline 模型;性能分析
中图法分类号　TP301

Batched Eigenvalue Decomposition Algorithms for Hermitian Matrices on GPU

HUANG Rongfeng[1,2], LIU Shifang[1,2] and ZHAO Yonghua[1]
1 Computer Network Information Center, Chinese Academy of Sciences, Beijing 100080, China
2 University of Chinese Academy of Sciences, Beijing 100080, China

Abstract　Batched matrix computing problems are widely existed in scientific computing and engineering applications. With rapid performance improvements, GPU has become an important tool to solve such problems. The eigenvalue decomposition belongs to the two-sided decomposition and must be solved by the iterative algorithm. Iterative numbers for different matrices can be varied. Therefore, designing eigenvalue decomposition algorithms for batched matrices on the GPU is more challenging than designing batched algorithms for the one-sided decomposition, such as LU decomposition. This paper proposes batched algorithms based on the Jacobi algorithms for eigenvalue decomposition of Hermitian matrices. For matrices that cannot reside in shared memory wholly, the block technique is used to improve the arithmetic intensity, thus improving the use of GPU resources. Algorithms presented in this paper run completely on the GPU, avoiding the communication between the CPU and GPU. Kernel fusion is adopted to decrease the overhead of launching kernel and global memory access. Experimental results on V100 GPU show that our algorithms are better than existing works. Performance evaluation results of the Roofline model indicate that our implementations are close to the upper bound, approaching 4.11TFLOPS.

Keywords　Hermitian matrix, Eigenvalue decomposition, Batch computing, Roofline model, Performance evaluation

1　引言

批量矩阵计算问题广泛存在于机器学习、计算机视觉、天体物理学等领域[1-4]。对于多核 CPU,这类问题的求解相对容易。例如结合 OpenMP 和高度优化的 LAPACK/BLAS 库(如 MKL,openBLAS)即可获得较好的性能,因为许多计算可以在 CPU 的高速缓存中进行。然而,由于缓存较小,在 GPU 上求解批量矩阵计算问题十分困难。

BLAS 库是科学计算领域的基础函数库之一,库中的矩阵乘(GEMM)函数占据着核心地位。为此,许多 GPU 供应商提供批量 GEMM 的高效实现[1),2)],开源软件包 MAGMA[3)] 和 KBLAS[4)] 也提供批量 GEMM 在 GPU 上的实现。LAPA-

1) https://docs.nvidia.com/cuda/pdf/cublas_Library.pdf
2) https://github.com/ROCmSoftwarePlatform/rocBLAS
3) http://icl.cs.utk.edu/magma/
4) https://github.com/ecrc/kblas-gpu

到稿日期:2022-01-25　返修日期:2022-08-24
基金项目:国家重点研发计划(2017YFB0202202);中国科学院战略性先导科技专项(XDC05000000)
This work was supported by the National Key Research and Development Program of China(2017YFB0202202) and Strategic Priority Research Program of Chinese Academy of Sciences(XDC05000000).
通信作者:赵永华(yhzhao@sccas.cn)

CK 库中函数的批量算法获得了广泛关注。例如，Dong 等[5]研究了批量 Cholesky 分解；Abdelfattah 等[6]研究了部分选主元的批量 LU 分解；文献[7]对批量矩阵计算问题的研究进行了较为全面的总结。与 Cholesky 分解等单边分解不同，矩阵特征值分解属于双边分解，必须使用迭代算法进行求解。不同矩阵的迭代次数可能不同，因此进行批量计算的矩阵几乎不能同时收敛。当一些矩阵收敛后，计算量变小，剩余的矩阵难以充分利用 GPU 上的众多计算核心。因此，在 GPU 上设计批量矩阵的特征值分解算法更具挑战性。

特征值分解算法主要分为两类：三对角化方法和 Jacobi 方法。三对角化方法主要包括三步：第一步使用 Householder 正交变换将原始矩阵转化为三对角矩阵；第二步使用 QR 迭代算法[8-9]、D&C 算法[10-11]或 MRRR 算法[12-13]计算三对角矩阵的特征值分解；最后使用 Householder 正交变换将三对角矩阵的特征向量恢复成原始矩阵的特征向量。与之不同，Jacobi 方法[14-15]无须将原始矩阵转化为三对角形式，而是直接对原始矩阵进行 Jacobi 旋转。Jacobi 方法的计算量通常比三对角化方法更大，但并行性优于后者。

对称矩阵特征值分解算法的时间复杂度为 $O(n^3)$（n 为矩阵行数）[16]，因此大规模矩阵的特征值分解算法属于计算密集型算法，可用 GPU 加速计算过程。ELPA-GPU 库[16]与 MAGMA 库基于三对角化方法给出了 GPU 加速的大规模矩阵特征值分解算法。对于大规模矩阵，CPU 与 GPU 之间的通信与 GPU 的计算能实现较好的重叠，从而隐藏通信代价。批量矩阵计算问题由于矩阵规模较小（通常不超过 512），计算时间短，因此通信时间与计算时间难以重叠。在设计算法时，应尽量避免 CPU 与 GPU 之间的数据拷贝，甚至可以研究完全运行在 GPU 上的算法。当前对批量矩阵特征值分解的研究较少，cuSOLVER[1)]提供了批量矩阵特征值分解的函数，但当前只支持规模不超过 32×32 的矩阵。cuSOLVER 属于商业库，其实现细节并未对外发布。

本文针对不同的矩阵规模，基于 Jacobi 算法设计了相应的批量厄米矩阵特征值分解 GPU 算法。本文的主要贡献包括：

(1)针对不超过 32×32 的矩阵，设计了共享内存算法，该算法优于 cuSOLVER 库。

(2)针对超过 32×32 的矩阵，设计了全局内存算法，填补了该领域的空白。

(3)使用 Roofline 模型[17]分析了算法的实现性能，结果表明本文的实现已接近理论上限，达到了 4.11TFLOPS。

本文第 2 节介绍了 Jacobi 方法，该方法是本文设计算法的基础；第 3 节介绍了针对不同矩阵规模的算法设计；第 4 节使用数值算例对算法进行实验验证；第 5 节分析算法的实现性能；最后总结全文并展望未来。

2 Jacobi 方法

本节介绍求解厄米矩阵的 Jacobi 方法。厄米矩阵为复共轭对称矩阵，一个 $n\times n$ 厄米矩阵 $\boldsymbol{A}$ 的特征值分解如下：

$$\boldsymbol{A}=\boldsymbol{Q}\cdot\boldsymbol{\Lambda}\cdot\boldsymbol{Q}^{C} \tag{1}$$

其中，$\boldsymbol{Q}$ 为 $n\times n$ 的正交矩阵，$\boldsymbol{Q}$ 的列称为 $\boldsymbol{A}$ 的特征向量；$\boldsymbol{\Lambda}$ 为 $n\times n$ 的对角矩阵，其对角元素称为矩阵 $\boldsymbol{A}$ 的特征值，且按升序排列。

Jacobi 方法每次从 $\boldsymbol{A}$ 中选择两列（如第 p,q 列）构造 Jacobi 旋转矩阵 $\boldsymbol{J}_{pq}$，并对矩阵 $\boldsymbol{A}$ 进行 Jacobi 旋转，使其收敛于对角矩阵。因此，Jacobi 方法的收敛条件为：

$$off(\boldsymbol{A})=\sqrt{\sum_{i\neq j}|a_{ij}|^2}<\varepsilon \tag{2}$$

其中，ε 为给定的常数。Jacobi 旋转矩阵 $\boldsymbol{A}_{pq}$ 只有第 p,q 行与第 p,q 列相交的 4 个位置与单位矩阵 $\mathbf{I}$ 不同，即具有如下的结构：

$$\boldsymbol{J}_{pq}=\begin{pmatrix}\mathbf{I} & & & \\ & c & -s & \\ & & \mathbf{I} & \\ & \bar{s} & c & \\ & & & \mathbf{I}\end{pmatrix} \tag{3}$$

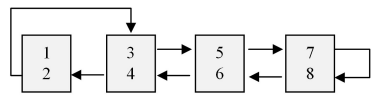
其中，c,s 可通过式(4)和式(5)进行解析求解。

$$t=2\cdot|a_{pq}|\cdot sign(a_{pp}-a_{qq})/(|a_{pp}-a_{qq}|+\sqrt{|a_{pp}-a_{qq}|^2+4\cdot|a_{pq}|^2}) \tag{4}$$

$$c=\frac{1}{1+\sqrt{t^2}},s=\frac{t\cdot a_{pq}}{|a_{pq}|\sqrt{1+t^2}} \tag{5}$$

Jacobi 方法将任意两列（也称为列对）都旋转一次的过程称为一个 sweep 迭代，因此一个 sweep 迭代包括 $n(n-1)/2$ 次 Jacobi 旋转。在实现 Jacobi 方法时，通常给出一个 sweep 迭代的旋转列对，周期性地进行 sweep 迭代，直至算法收敛。

Jacobi 方法给出旋转列对的常用方法包括行循环方法和列循环方法，但基于这两种方法得到的算法并行性较差。并行性较好的给定旋转列对方法为 Round-Robin 方法[18]。以 $n=8$ 为例来说明 Round-Robin 方法。如图 1 所示，将编号 1－8 依次放入方框内，一个框内为一个列对，共得到 $n/2=4$ 个列对。固定编号 1，其他编号沿着箭头所示方向依次移动一个位置，得到另外 $n/2=4$ 个列对。继续上述过程 $n-2$ 次，可得到一个 sweep 迭代所需的 $n(n-1)/2$ 个列对。该方法每次可同时对 $n/2$ 个列对进行 Jacobi 旋转，并行性较好。



(1 2)	(3 4)	(5 6)	(7 8)
(1 4)	(2 6)	(3 8)	(5 7)
(1 6)	(2 7)	(3 5)	(4 8)
(1 8)	(2 3)	(4 5)	(6 7)
(1 7)	(2 4)	(3 6)	(5 8)
(1 5)	(2 8)	(3 7)	(4 6)
(1 3)	(2 5)	(4 7)	(6 8)

图 1 Round-Robin 方法

Fig. 1 Round-Robin method

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

在求解厄米矩阵的 Jacobi 方法中，双边 Jacobi 特征值分解算法使用最广泛，其详细步骤如算法 1 所示。

算法 1 双边 Jacobi 特征值分解算法

输入：$\mathbf{A}$

输出：$\mathbf{Q},\mathbf{\Lambda}$

1. 初始化 $\mathbf{Q}=\mathbf{I}$ / * $\mathbf{I}$ 单位矩阵 * /
2. while off($\mathbf{A}$)$>\varepsilon$
3. 使用 Round-Robin 方法给出旋转列对(p,q)
4. 根据式(3)－式(5)计算 $\mathbf{J}_{pq}$
5. 更 $\mathbf{A}$ 的行，$\mathbf{A}=\mathbf{J}_{pq}^{C}\cdot\mathbf{A}$
6. 更新 $\mathbf{A}$ 的列，$\mathbf{A}=\mathbf{A}\cdot\mathbf{J}_{pq}$
7. 更新 $\mathbf{Q}$ 的列，$\mathbf{Q}=\mathbf{Q}\cdot\mathbf{J}_{pq}$
8. end while
9. 计算 $\mathbf{\Lambda}$：$\mathbf{\Lambda}=\mathbf{A}$
10. 对 $\mathbf{\Lambda}$ 的对角元素进行排序并交换 $\mathbf{Q}$ 对应的列

由于算法 1 的主要计算（第 5－7 行）均为访存受限（Memory Bound）的 BLAS 1 操作，因此，算法 1 属于访存受限型算法。

3 基于 GPU 架构的算法设计

为便于介绍 GPU 上的算法设计，本文采用 CUDA（Computer Unified Device Architecture）作为编程模型。本文的算法设计同样适用于其他 GPU 编程模型，如 OpenCL 和 HIP。首先简要介绍 CUDA 的一些基本概念。CUDA 是一个典型的 SIMT（Single Instruction Multiple Thread）架构，CUDA 编程模型中常用的内存主要包括全局内存、共享内存以及寄存器。从全局内存到寄存器，访问延迟逐渐减小，访存带宽逐渐增加。寄存器的访问速度最快，只能单线程独享。共享内存的访问延迟比寄存器内存大，但远小于全局内存的访问延迟。同一个 block 上的所有线程均可访问共享内存，因此，可通过共享内存实现线程间的数据交换。此外，同一个 block 上的线程可通过内置函数*__syncthreads*()进行同步。有效利用共享内存是在 GPU 上设计高效算法的关键，但共享内存属于稀缺资源，每个 block 可使用的共享内存至多为 96 kB（计算能力为 7. x 的设备）。所有线程均可以访问全局内存，但访问延迟较大，应尽量减少对全局内存的访问。在访问全局内存时，相邻线程的访问地址也是连续的，十分有利于提升访存带宽。

在 CUDA 中，一条 FMA（Fused Multiply Add）指令可以同时完成形如 $x=y\times z+w$ 的加和乘两种运算。因此，在算法实现时，浮点运算应尽量使用 FMA 指令。厄米矩阵的计算为复数运算，可使用内置函数 cuCfma 实现 FMA 指令的调用。

3.1 共享内存算法

对于规模较小的矩阵（如不超过 32×32），共享内存可以容纳待分解矩阵及其特征向量。为了减少数据访问，将整个矩阵加载到共享内存进行计算，待计算完成后再将结果写回全局内存。在 kernel 设计上，使用一个 block 计算一个矩阵。Grid，block 与矩阵的映射关系如图 2 所示。一维 grid 和二维 block 的大小分别为 *batch* 和 $(n,\lceil n/2\rceil)$，其中 *batch* 等于矩阵数量。在计算时，使用一列线程实施一个列对的 Jacobi 旋转。在开始 Jacobi 旋转前，所有线程都计算 $\boldsymbol{J}_{pq}$。虽然存在冗余计算（同一列线程的$\backslash_{pq}$相同），但所有线程都参与算法 1 中第 5－7 行的计算，可以提高线程利用率。

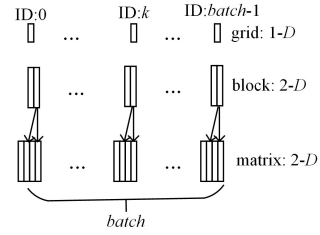


图 2 共享内存算法中 grid，block 与矩阵的映射

Fig. 2 Mapping between grid，block and matrix in shared memory algorithm

对于小于 8×8 的矩阵，单个 block 的线程数小于 32（CUDA 的线程调度单位），每个 block 的线程均存在浪费。为此，采用$(n,\lceil n/2\rceil,Dim.z)$的三维 block 增加单个 block 的线程数，且使用单个 block 计算 $Dim.z$ 个矩阵的特征值分解。但数值实验发现，这种方案并没有显著缩短批量矩阵特征值分解的时间，原因是这种方案导致计算单个矩阵的线程可能不属于同一个 warp，必须使用内置函数__syncthreads 避免数据竞争，这样就引入了额外的负载。因此，在数值实验部分没有给出这种方案的实验结果。

3.2 全局内存算法

对于规模较大的矩阵，共享内存无法容纳待分解矩阵及其特征向量，只能存储在全局内存中。最直接的设计是依据算法 1，每次从全局内存中读取两行或两列进行 Jacobi 旋转，但全局内存访问延迟较大，该设计性能较差。本文采用矩阵“块”操作技术来提升计算强度，从而提高 GPU 的资源利用率。

首先对矩阵进行剖分，例如将矩阵 $\boldsymbol{A}$ 进行如下剖分：

$$\boldsymbol{A}=\begin{pmatrix}\boldsymbol{A}_{11} & \boldsymbol{A}_{12} & \cdots & \boldsymbol{A}_{1K}\\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} & \cdots & \boldsymbol{A}_{2K}\\ \vdots & \vdots & \vdots & \vdots\\ \boldsymbol{A}_{K1} & \boldsymbol{A}_{K2} & \cdots & \boldsymbol{A}_{KK}\end{pmatrix} \tag{6}$$

不失一般性，假设块$\boldsymbol{A}_{ij}$的大小为 $nb\times nb$，满足 $nb\times K=n$，且 K 为偶数。

双边块 Jacobi 特征值分解算法将 Jacobi 旋转推广为块 Jacobi 旋转，详细步骤如算法 2 所示。

算法 2 双边块 Jacobi 特征值分解算法

输入：$\mathbf{A}$

输出：$\mathbf{Q},\mathbf{\Lambda}$

1. 初始化 $\mathbf{Q}=\mathbf{I}$ / * $\mathbf{I}$ 单位矩阵 * /
2. while off($\mathbf{A}$)$>\varepsilon$
3. 给定 Jacobi 旋转列对(p,q)
4. 计算块 Jacobi 旋转矩阵 $\mathbf{BJ}_{pq}$
5. 更新 $\mathbf{A}$ 的行块，$\mathbf{A}=\mathbf{BJ}_{pq}^{C}\cdot\mathbf{A}$
6. 更新 $\mathbf{A}$ 的列块，$\mathbf{A}=\mathbf{A}\cdot\mathbf{BJ}_{pq}$
7. 更新 $\mathbf{Q}$ 的列块，$\mathbf{Q}=\mathbf{Q}\cdot\mathbf{BJ}_{pq}$
8. end while
9. 计算 $\mathbf{\Lambda}$：$\mathbf{\Lambda}=\mathbf{A}$

10. 对 $\mathbf{\Lambda}$ 的对角元素进行排序并交换 $\mathbf{Q}$ 对应的列

算法 2 的收敛条件与算法 1 相同，但块 Jacobi 旋转矩阵 $\boldsymbol{BJ}_{pq}$ 的计算不存在显示计算公式，需要计算子矩阵 $\begin{pmatrix}\boldsymbol{A}_{pp} & \boldsymbol{A}_{pq}\\ \boldsymbol{A}_{qp} & \boldsymbol{A}_{qq}\end{pmatrix}$ 的特征分解。为利用 CUDA 共享内存的快速访存性质，可在共享内存上实现算法 1 来求解子矩阵 $\begin{pmatrix}\boldsymbol{A}_{pp} & \boldsymbol{A}_{pq}\\ \boldsymbol{A}_{qp} & \boldsymbol{A}_{qq}\end{pmatrix}$ 的特征分解。算法 1 需要的共享内存大小至少为 $2\times 2nb\times 2nb\times sizeof(cuDoubleComplex)$，这使得 nb 的取值不能超过 27，小于 CUDA 中的线程调度单位 32。无论矩阵在 GPU 中以行主序或列主序存储，算法 2 第 5 行或第 6 行必然存在相连线程访问地址不连续的情况。为此，本文采用算法 3 所示的单边块 Jacobi 特征值分解算法。

算法 3 单边块 Jacobi 特征值分解算法

输入：$\mathbf{A}$

输出：$\mathbf{Q}$，$\mathbf{\Lambda}$

1. 初始化 $\mathbf{Q}=\mathbf{I}$ / * $\mathbf{I}$ 单位矩阵 * /
2. while off($\mathbf{Q}^{\mathrm{C}}\cdot\mathbf{A}$)＞ε
3. 给定 Jacobi 旋转列对(p,q)
4. 计算矩阵 $\mathbf{G}_{\mathrm{pq}}$
5. 计算块 Jacobi 旋转矩阵 $\mathbf{BJ}_{\mathrm{pq}}$
6. 更新 $\mathbf{A}$ 的列块，$\mathbf{A}=\mathbf{A}\cdot\mathbf{BJ}_{\mathrm{pq}}$
7. 更新 $\mathbf{Q}$ 的列块，$\mathbf{Q}=\mathbf{Q}\cdot\mathbf{BJ}_{\mathrm{pq}}$
8. end while
9. 计算 $\mathbf{\Lambda}$：$\mathbf{\Lambda}=\mathbf{Q}^{\mathrm{C}}\cdot\mathbf{A}$
10. 对 $\mathbf{\Lambda}$ 的对角元素进行排序并交换 $\mathbf{Q}$ 对应的列

算法 3 第 4 行的矩阵 $\boldsymbol{G}_{pq}$ 使用式(7)计算得到：

$$\boldsymbol{G}_{pq}=[\boldsymbol{Q}_{:,p}\boldsymbol{Q}_{:,q}]^{\mathrm{C}}\cdot[\boldsymbol{A}_{:,p}\boldsymbol{A}_{:,q}] \tag{7}$$

与算法 2 不同，算法 3 虽然每次计算块 Jacobi 旋转 $\boldsymbol{BJ}_{pq}$ 时需计算 $\boldsymbol{G}_{pq}$，但在更新矩阵 $\boldsymbol{A}$ 时无须对行条块进行块 Jacobi 旋转。对于 GPU 上以列主序存储的矩阵，由于算法 3 的主要计算(第 4－7 行)只需访问矩阵的列条块，因此，相连线程访问地址连续，避免了双边块 Jacobi 特征值分解算法相连线程访问地址不连续的问题，进而增加了访存带宽，提升了算法性能。此外，只要两个列块之间的列互不相同，则算法 3 的第 4－7 行可以同时进行，因此算法 3 具有很好的并行性，且可以使用 Round-Robin 方法给定 Jacobi 旋转列对。

算法 3 的 4 个主要计算为计算 $\boldsymbol{G}_{pq}$、计算 $\boldsymbol{BJ}_{pq}$(求解矩阵 $\boldsymbol{G}_{pq}$ 的特征分解)、更新 $\boldsymbol{A}$ 的列块和更新 $\boldsymbol{Q}$ 的列块。下面介绍这 4 个主要计算在 CUDA 上的实现。除计算 $\boldsymbol{BJ}_{pq}$ 外，其他 3 个计算属于矩阵乘积(GEMM)。由于不同列对之间的计算可同时进行，因此可调用 cuBLAS 库中的批量 GEMM 实现，即 *cublasZgemmStridedBatched* 函数。对于求解矩阵 $\boldsymbol{G}_{pq}$ 特征分解的 kernel 设计，本文采用大小为($2nb$，nb)的二维 block，大小为($K/2$，*batch*)的二维 grid。为了求解矩阵 $\boldsymbol{G}_{pq}$ 特征分解的 kernel 在迭代过程中只需访问 GPU 共享内存，nb 的大小设置需使得 GPU 共享内存可以存储 $\boldsymbol{BJ}_{pq}$ 和 $\boldsymbol{G}_{pq}$。由于 $\boldsymbol{BJ}_{pq}$ 和 $\boldsymbol{G}_{pq}$ 的大小均为 $2nb\times 2nb$，因此使用一列线程实施一个列对的 Jacobi 旋转。Grid，block 与矩阵的映射关系如图 3 所示，一个矩阵需要多个 block 共同计算。

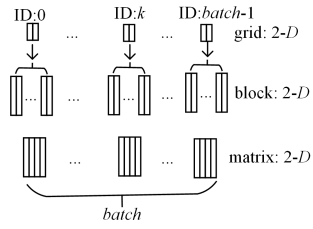


图 3 全局内存算法中 grid，block 与矩阵的映射

Fig. 3 Mapping between grid, block and matrix in global memory algorithm

由于算法 3 中需要进行块 Jacobi 旋转的两个列块在存储空间上不连续，为了调用 *cublasZgemmStridedBatched* 函数，需将第 p 列块和第 q 列块拷贝到连续的存储空间，这增加了额外的数据拷贝。此外，cuBLAS 库只提供调用接口，算法 4 的 4 个主要计算需要分别启动不同的 kernel，而不同的 kernel 之间无法共用相同的共享内存，只能通过访问延迟较大的全局内存进行数据传递。为解决上述问题，本文使用一个 kernel 完成 4 个主要计算，即扩展求解 $\boldsymbol{G}_{pq}$ 特征分解的 kernel，使其支持批量 GEMM 操作。

本文采用计算矩阵乘积的 tile 算法[19]。文中只给出计算 $\boldsymbol{G}_{pq}$ 的算法设计，类似可得矩阵 $\boldsymbol{A}$ 与矩阵 $\boldsymbol{Q}$ 的算法设计。Tile 算法依次将 $\boldsymbol{A}$ 与 $\boldsymbol{Q}$ 第 p 列块和第 q 列块的子块从全局内存读到共享内存再计算子块的乘积。我们注意到从全局内存中加载子块和计算子块乘积存在数据依赖，当从全局内存中加载子块到共享内存时，数据计算单元处于等待状态。由于全局内存访问延迟较大，数据计算单元将长时间处于空闲状态，降低了算法性能。为此，本文采用数据预取技术(也被称为双缓冲技术)重叠数据计算与数据访问[19]。如图 4 所示，在计算当前子块乘积时从全局内存中预取下一个子块，那么，数据计算单元只有在读取第一个子块时处于空闲状态。另一方面，在算法 3 第 5 行求解 $\boldsymbol{G}_{pq}$ 的特征分解时也可从全局内存中加载矩阵 $\boldsymbol{A}$ 的第一个子块，进一步隐藏第 6 行的数据访问延迟。

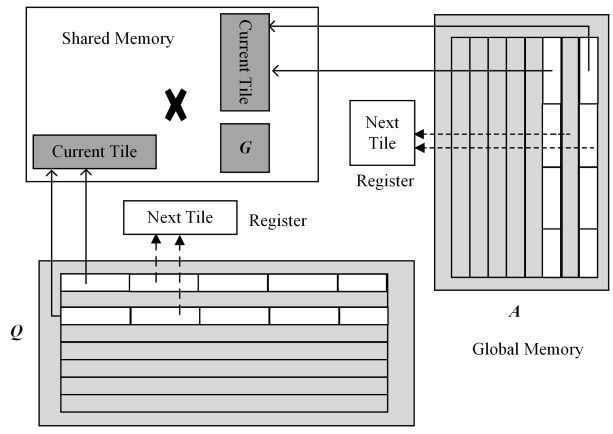


图 4 计算内积矩阵的示意图

Fig. 4 Schematic diagram of calculating inner product matrix

本节给出的共享内存算法和全局内存算法在设置 kernel 的 grid 时，每个矩阵具有唯一的 id，使得所有矩阵的计算可以同时进行。由于不同矩阵的迭代次数不相同，

本文采用 ETM[20-21] 机制，在计算过程中监测每个矩阵的收敛状态。当识别到矩阵收敛后，及时修改 kernel 的 grid 设置，将 GPU 计算资源用于未收敛矩阵的计算，避免无效计算。

4 数值实验

本文实验采用的 GPU 加速卡为 NVIDIA V100-PCIe GPU，该加速卡的双精度浮点性能达 7TFLOP/s，显存带宽达 900 GB/s，CUDA 版本号为 11.1。在实现算法时，本文只使用 CUDA C 进行编程，未使用更低级别的指令优化。在进行数值实验时，矩阵的特征值和特征向量均进行计算，且假设矩阵已经存储在 GPU 的全局内存中，因此本节的时间不包括从 CPU 拷贝矩阵到 GPU 的时间。测试矩阵为双精度复厄米矩阵，矩阵元素以列主序方式进行排列。测试矩阵随机产生，矩阵元素服从均匀分布 U(0,1)。

图 5 给出了本文设计的共享内存算法与 cuSOLVER 库的时间对比，批量特征值分解的矩阵数为 1000。可以看出，随着矩阵规模超过 16×16，cuSOLVER 库的时间出现跳跃式增长，而本文算法时间增长相对平缓。本文算法在所有测试规模中均优于 cuSOLVER 库，获得了 1.0～3.5 ×的加速，平均加速比为 1.9×。

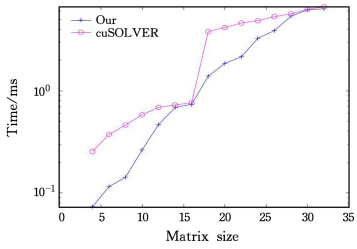


图 5 共享内存算法与 cuSOLVER 库的时间对比

Fig. 5 Time comparison between shared memory algorithm and cuSOLVER

由于 cuSOLVER 库的批量矩阵特征值分解函数不支持超过 32×32 的矩阵，作为对比，本文使用 cuSOLVER 库求解单个矩阵特征值分解的函数，但在调用函数时不同矩阵使用不同的流(Stream)，通过流重叠不同矩阵的计算从而提高性能，测试结果用 cuSOLVER+Stream 表示。图 6 给出了本文设计的全局内存算法与 cuSOLVER+Stream 的时间对比，批量特征值分解的矩阵数为 200。

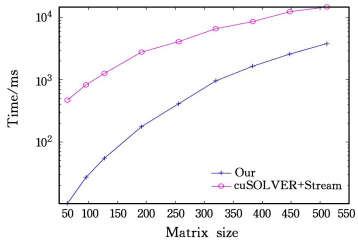


图 6 全局内存算法与 cuSOLVER 库的时间对比

Fig. 6 Time comparison between global memory algorithm and cuSOLVER

可以看出，相对于 cuSOLVER+Stream 计算批量矩阵特征值分解的方式，本文算法能有效减少矩阵特征值分解时间，且小矩阵的加速效果更好。对于 64×64 至 512×512 规模的矩阵，最大加速比为 46.0×，最小加速比为 3.9×，平均加速比为 9.8×。

最后给出批量矩阵特征值分解的误差对比结果。批量矩阵特征值分解的误差定义为所有测试矩阵误差的最大值。下面给出单个矩阵误差的定义，特征值误差的计算式如式(8)所示：

$$err_\lambda = \max_{1\leqslant i\leqslant n} \frac{|\lambda_i^{ex} - \hat{\lambda}_i|}{\max(1,\lambda_i^{ex})} \tag{8}$$

其中，λ_i^{ex} 为矩阵的精确特征值，$\hat{\lambda}_i$ 为计算的特征值。分解误差计算式如式(9)所示：

$$err_D = \frac{\| \boldsymbol{A} - \hat{\boldsymbol{Q}}^C \cdot \hat{\Lambda} \cdot \hat{\boldsymbol{Q}} \|_F}{\| \boldsymbol{A} \|_F \cdot n} \tag{9}$$

其中，$\boldsymbol{A}$ 为待分解的矩阵，$\hat{\boldsymbol{Q}}$ 为计算的特征向量，$\hat{\boldsymbol{\Lambda}}$ 为计算的特征值，范数为 Frobenius 范数。特征向量正交性误差计算式如式(10)所示：

$$err_Q = \frac{\| \mathbf{I} - \hat{\boldsymbol{Q}}^C \cdot \hat{\boldsymbol{Q}} \|_F}{n} \tag{10}$$

图 7－图 9 分别给出了特征值误差、分解误差以及特征向量正交性误差。可以看出，本文的算法在特征值误差、分解误差和特征向量正交性误差 3 方面均优于 cuSOLVER 库，进而验证了本文算法的有效性。共享内存算法针对的矩阵规模较小，本文的算法与 cuSOLVER 库的误差均较小，且比较接近，此处不再列出。

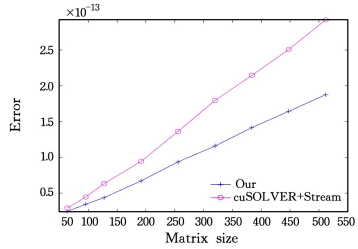


图 7 特征值误差

Fig. 7 Errors of eigenvalues

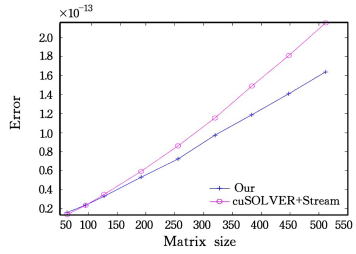


图 8 分解误差

Fig. 8 Errors of decompositions

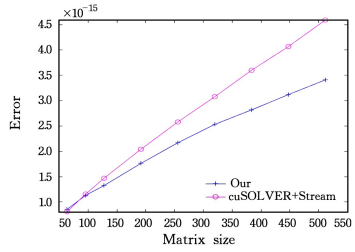


图 9 特征向量正交性误差

Fig. 9 Orthogonality errors of eigenvectors

5 性能分析

Roofline 模型提供了一种分析应用性能的方法。在

Roofline 模型中，应用性能(FLOPS)满足：

$$性能 \leqslant \begin{cases} 理论峰值 \\ 显存带宽 \times 计算强度 \end{cases} \tag{11}$$

其中，计算强度(Arithmetic Intensity)被定义为应用的浮点操作数除以数据访问量。计算强度刻画了应用特征，而理论峰值和显存带宽则描述了机器特征。假如应用的计算强度小于理论峰值除以显存带宽，那么该应用属于访存受限型，优化应用的数据访问更有助于提升应用性能。而如果应用的计算强度大于理论峰值除以显存带宽，那么该应用属于计算密集型，应主要优化应用的浮点运算。

使用 Roofline 模型分析性能分为 3 步，首先获取机器的理论峰值和显存带宽，其次获取应用的浮点操作数和数据访问量，最后使用式(11) 整合这些数据并画图。其中，机器的理论峰值和显存带宽可通过机器制造商的官网进行查询，应用的浮点操作数和数据访问量可通过理论分析或者工具包统计，例如 NVIDIA nvprof 和 Nsight Compute。

本文采用 NVIDIA Nsight Compute 统计 V100 加速卡上浮点操作数和数据访问量。图 10 给出了 $n=128$ 时的 Roofline 模型，可以看出：

(1)批量厄米矩阵的特征值分解属于带宽受限型。

(2)本文的实现已接近理论上限，说明本文算法设计适合 CUDA 编程模型，能有效利用 GPU 的计算资源和访存带宽。

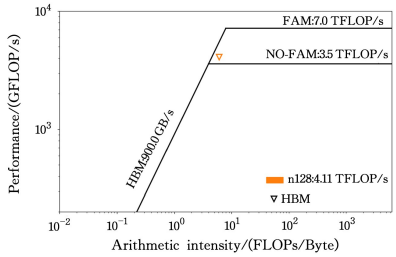


图 10 Roofline 模型

Fig. 10 Roofline model

由于共享内存算法是将整个矩阵加载到共享内存之后再进行计算，图 10 所示的分析模型不适合，因此采用 GPU 占有率(Occupancy)作为测量标准进行分析。GPU 占有率的定义是实际运行的 warp 数量与 GPU 支持的最大 warp 数量的比值。一般而言，对于访存受限的应用，该值越大，说明数据访问延迟更能通过 warp 切换进行隐藏，从而带来更高的性能。图 11 给出了共享内存算法的 GPU 占有率。对于 8×8 矩阵，单个 block 只有一个 warp，因此 GPU 占有率远小于另外两个规模，这也为进一步优化指明了方向。

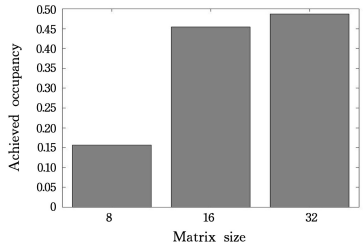


图 11 GPU 占有率

Fig. 11 Achieved occupancy of GPU

结束语 本文面向 GPU 架构，提出了两个适合不同矩阵规模的批量厄米矩阵特征值分解算法：共享内存算法和全局内存算法。共享内存算法将整个矩阵加载到共享内存进行计算，减少了全局内存的访问。全局内存算法使用矩阵“块”操作技术增加计算强度，提高了 GPU 资源利用率。数值算例验证了本文算法的有效性且表明了本文算法优于 cuSOLVER 库。未来的工作方向包括共享内存算法的深度优化以及本文算法在实际问题中的应用。

参 考 文 献

[1] ABDELFATTAH A, BABOULIN M, DOBREV V, et al. High-performance Tensor Contractions for GPUs [J]. Procedia Computer Science, 2016, 80(1): 108-118.

[2] MOLERO J M, GARZÓN E M, GARCÍA I, et al. Efficient Implementation of Hyperspectral Anomaly Detection Techniques on GPUs and Multicore Processors [J]. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 2014, 7(6): 2256-2266.

[3] VILLA O, GAWANDE N, TUMEO A. Accelerating Subsurface Transport Simulationon Heterogeneous Clusters [C]// International Conference on Cluster Computing. New York: IEEE Press, 2013: 1-8.

[4] ZHANG T, LIU X, WANG X, et al. cuTensor-Tubal: Efficient Primitives for Tubal-Rank Tensor Learning Operations on GPUs [J]. IEEE Transactions on Parallel and Distributed Systems, 2020, 31(3): 595-610.

[5] DONG T, HAIDAR A, TOMOV S, et al. A Fast Batched Cholesky Factorization on a GPU [C]// International Conference on Parallel Processing. New York: IEEE Press, 2014: 432-440.

[6] ABDELFATTAH A, HAIDAR A, TOMOV S, et al. Factorization and Inversion of a Million Matrices using GPUs: Challenges and Countermeasures [C]// International Conference on Computational Science. Berlin: Springer, 2017: 606-615.

[7] ABDELFATTAH A, COSTA T, DONGARRA J, et al. A Set of Batched Basic Linear Algebra Subprograms and LAPACK Routines [J]. ACM Transactions on Mathematical Software, 2021, 47(7): 1-23.

[8] FRANCIS J G F. The QR Transformation: A Unitary Analogue to The LR Transformation—Part 1 [J]. The Computer Journal, 1961, 4(3): 265-271.

[9] FRANCIS J G F. The QR Transformation—Part 2 [J]. The Computer Journal, 1964, 4(4): 332-345.

[10] CUPPEN J. A Divide and Conquer Method for The Symmetric Eigenproblem [J]. Numerical Mathematics, 1980, 36(2): 177-195.

[11] GU M, EISENSTAT S C. A Divide-and-Conquer Algorithm for the Symmetric Tridiagonal Eigenproblem [J]. SIAM Journal on Matrix Analysis and Applications, 1995, 16(1): 172-191.

[12] BIENTINESI P, DHILLON I S, GEIJN R A V D. A Parallel Eigensolver for Dense Symmetric Matrices Based on Multiple Relatively Robust Representations [J]. SIAM Journal on Scientific Computing, 2005, 27(1): 43-66.

[13] WILLEMS P R,LANG B. A Framework for The MR 3 Algorithm:Theory and Implementation [J]. SIAM Journal on Scientific Computing,2013,35(2):A740-A766.

[14] WANG T,GUO L,LI G,et al. Implementing the Jacobi Algorithm for Solving Eigenvalues of Symmetric Matrices with CUDA [C]// International Conference on Networking, Architecture,and Storage. New York:IEEE Press,2012:69-78.

[15] TORUN M U,YILMA O,AKANSU A N. FPGA,GPU,and CPU Implementations of Jacobi Algorithm for Eigenanalysis [J]. Journal of Parallel & Distributed Computing, 2016, 96(12):172-180.

[16] YU W Z,MOUSSA J,KUS P,et al. GPU-Acceleration of The ELPA2 Distributed Eigensolver for Dense Symmetric and Hermitian Eigenproblems [J]. Computer Physics Communications, 2021,262(5):107808.

[17] WILLIAMS S,WATERMAN A,PATTERSON D. Roofline:An Insightful Visual Performance Model for Multicore Architectures [J]. Communication of the ACM,2009,52(4):65-76.

[18] BRENT R P,LUK F T. The Solution of Singular Value and Symmetric Eigenvalue Problems on Multiprocessor Arrays [J]. SIAM Journal on Scientific and Statistical Computing, 1985, 6(1):69-84.

[19] RIVERA C,CHEN J,XIONG N,et al. TSM2X:High-Performance Tall-and-Skinny Matrix-Matrix Multiplication on GPUs [J]. Journal of Parallel and Distributed Computing, 2021, 151(3):70-85.

[20] ABDELFATTAH A,HAIDAR A,TOMOV S,et al. Performance,Design, and Autotuning of Batched GEMM For GPUs [C]// International Conference on High Performance Computing. Berlin:Springer,2016:21-38.

[21] ABDELFATTAH A,HAIDAR A,TOMOV S,et al. Performance tuning and optimization techniques of fixed and variable size batched Cholesky factorization on GPUs[J]. Procedia Computer Science,2016,80(C):119-130.

HUANG Rongfeng, born in 1990, Ph. D candidate. His main research interests include efficient implementation of math library on heterogeneous platforms and so on.

ZHAO Yonghua, born in 1966, Ph. D, professor, Ph. D supervisor. His main research interests include parallel algorithm and software, parallel programming model, spectral clustering data analysis and so on.

(责任编辑:何杨)