计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.190900202

面向语音分离的深层转导式非负矩阵分解并行算法

李雨蓉[1] 刘 杰[1,2] 刘亚林[1] 龚春叶[1] 王 勇[1]
1 国防科技大学并行与分布处理重点实验室 长沙 410073
2 国防科技大学复杂系统软件工程湖南省重点实验室 长沙 410073
(liyvrong17@nudt.edu.cn)

摘 要 非负矩阵分解(Non-negative Matrix Factorization,NMF)能保存语音信号的非负特征,是用于语音分离的重要方法,但该方法存在数据运算复杂、计算量太大的问题,需要研究能减少计算时间的并行计算方法。针对语音分离预训练及分离过程的计算问题,文中提出深层转导式非负矩阵分解并行算法,综合考虑迭代更新过程的数据关联性,设计了一种任务间和任务内多级并行算法。该并行算法在任务级将分解训练语音得到对应基矩阵的过程作为两个独立的任务进行并行计算;在任务内部进程级把矩阵按行列划分,主进程把矩阵块分发到从进程,从进程接收当前矩阵块并计算结果矩阵子块,然后将当前进程矩阵块发送到下一进程,实现第二个矩阵中每一个矩阵块在所有进程的遍历,并计算结果矩阵对应子块的乘积,最后由主进程收集从进程数据块;在线程级子矩阵乘法运算的过程中,采取生成多线程,通过共享内存交换数据计算子矩阵块的加速策略。该算法为首个实现深层转导式非负矩阵分解的并行算法。在天河二号平台上的测试结果表明,在分离多说话人混合语音信号时,相比串行程序,所提出的并行算法能在不改变分离效果的前提下,使得预训练过程中使用 64 个进程的加速比为 18,分离过程使用 64 个进程的对应加速比为 24。相较于串行及 MPI 模型分离,混合模型分离时间大大缩短,从而证明了设计的并行算法可有效提高语音分离的效率。

关键词:深层转导式非负矩阵分解并行算法;乘性迭代更新规则加速算法;消息传递接口;共享存储并行编程;语音分离
中图法分类号 TP391

Parallel Algorithm of Deep Transductive Non-negative Matrix Factorization for Speech Separation

LI Yu-rong[1],LIU Jie[1,2],LIU Ya-lin[1],GONG Chun-ye[1] and WANG Yong[1]
1 Science and Technology on Parallel and Distributed Processing Laboratory,National University of Defense Technology,Changsha 410073,China
2 Laboratory of Software Engineering for Complex Systems,National University of Defense Technology,Changsha 410073,China

Abstract Non-negative matrix factorization can preserve the non-negative features of the speech signal. It is an important method for speech separation. However,this method has the problems of complicated data operation and computation,it is necessary to propose a parallel method to reduce computation time. Aiming at the calculation problem of speech separation pre-training and separation process,this paper proposes a deep transductive non-negative matrix factorization multi-level parallel algorithm,which considers the data correlation of the iterative update process,and designs a multi-level parallel algorithm between tasks and within tasks. The parallel algorithm at the task level decomposes the training speech to obtain the corresponding base matrix as two independent tasks in parallel calculation. The matrix is divided into rows and columns at the internal process level of the task. The master process distributes the matrix blocks to the slave process,and the slave process receives the current sub-matrix,then the matrix block calculates the result matrix sub-block,and then sends the current process matrix block to the next process to achieve the traversal of each matrix block of the second matrix in all processes,calculating the product of the corresponding sub-block of the result matrix,and finally the sub-block is collected by the main process from the slave process. During the thread-level sub-matrix multiplication operation,an acceleration strategy of generating multiple threads and exchanging data through shared memory for sub-matrix block calculation is adopted. This algorithm is the first one to implement deep transduction non-negative matrix factorization algorithm. The experiment is performed on the Tianhe II platform. The test results show that when separating multi-speaker mixed speech signals without changing the separation effect,in the pre-training process,the proposed parallel algorithm performs well using 64 processes at a speed ratio of 18,and in the separation process,the corresponding speedup is 24.

到稿日期:2019-09-30 返修日期:2020-01-13 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:重点研发计划(2018YFB0204301)
This work was supported by the National Key Research and Development Program of China(2018YFB0204301).
通信作者:刘杰(liujie@nudt.edu.cn)

Compared to serial and MPI model separation, hybrid model separation time is greatly shortened, which proves that for the speech separation process, the parallel algorithm proposed in this paper can effectively improve the separation efficiency.

Keywords Parallel algorithm of deep-transductive non-negative matrix factorization, Acceleration algorithm based on multiplicative update rules, MPI, OpenMP, Speech separation

1 引言

非负矩阵分解[1]方法是指对于给定的一个矩阵 $\boldsymbol{V}\in R^{m*n}$，以最小化一个目标函数为目的，将其分解为两个低阶的非负子矩阵 $\boldsymbol{W}\in R_{+}^{m*r}$ 和 $\boldsymbol{H}\in R_{+}^{r*n}$ 的乘积的过程。这种方法由于自身具有非负约束特性，能保存由现实生活建模形成的数据特征，并对分解结果具有天然的可解释性，如图像数据的像素点不可能为负值，在文档统计中统计数据出现的次数不可能为负等。该方法被广泛应用于人脸检测、计算机视觉、文本聚类、数据挖掘和语音分析等领域。

大量学者对非负矩阵分解方法及其收敛性的证明做出了详细描述，且在其基础上进行算法改进，以提高算法的执行效率。常用的非负矩阵分解算法有乘性更新(Multiplicative Update, MU)[2]、层级替代最小二乘(Hierarchical Alternative Least Squares, HALS)[3]、随机梯度下降更新(Stochastic Gradient Descent Updates, SGD)[4]和块主体旋转(Block Principal Pivoting, BPP)[5]等。这些方法虽然都能保证分解结果非负，但它们对收敛速度或者误差精确度的倾向性不同，因此在面对具体问题时，须选取相适应的分解算法。虽然乘性更新分解的下降速率较其他分解算法低，但学者们已证明使用乘性更新迭代算法可以有效减小弗罗贝尼乌斯范数的误差值。由于本课题的目标函数采用弗罗贝尼乌斯范数，且语音分离的结果需要尽可能接近纯语音信号，即误差尽可能小，因此本文选择乘性更新分解算法进行运算。

近年来，研究者们针对非负矩阵分解方法的并行化处理提出了很多方案。文献[6]在证明了基于 MU 算法和共享存储并行算法的实现存在收敛速度慢等不足后，设计了基于 MU 算法的 MPI 实现算法。该算法将两个因子矩阵分成较小的数据块，再将每个块分配到不同的线程，每个块同时更新因子矩阵的相应子矩阵，最后通过 MPI 的收集操作将子矩阵规约到主进程。文献[7]也提出了一种矩阵划分方法，该方法沿着较短的维度而不是较长的维度对因子矩阵进行划分，这种情况下，矩阵可以被划分为更多分区，从而在执行块矩阵乘法运算时增加数据局部性并降低通信成本。该方法在分解过程中使用了乘性更新、K-L 散度、对误差求平方以及对损失函数求指数等方法。文献[7]还证明了在 Hadoop 框架中使用所提并行方案进行稀疏矩阵操作对性能提升是有效的。文献[8]提出了使用 OpenMP 对 MU 迭代更新过程进行加速的并行算法，实验表明使用 OpenMP 可以加速计算过程，但 OpenMP 使用共享内存，不适用于跨节点的矩阵运算。文献[9]提出了几种基于 GPU 的并行 NMF 实现算法，算法使用欧几里得距离和 KL 散度作为目标函数，在分解过程使用 MU 分解和 AU 分解算法。文献[10]提出一种分布式内存的并行算法，该算法使用 BPP 分解方法，通过把数据和向量矩阵分布地存储在处理器或内存中，并使用 MPI 进行进程间通信，来解决 $\boldsymbol{W}$ 和 $\boldsymbol{H}$ 的交替非负最小二乘问题。即使在稠密矩阵的情况下，这种方法也可以通过最小化通信消耗达到加速的效果。文献[11]先使用 HALS 算法的计算组件对 NMF 进行分析，识别数据移动开销，开发了一个计算移动开销的函数，进而产生可以选择有效图块大小的模型；然后读取需要移动的图块的数据，将其分配到不同的线程上独立执行更新过程；最后由主进程进行聚合操作。该方法通过多线程共享数据和代码，但也可以独立执行分配到单个线程的计算，从而实现并行处理。文献[12]提出一种并行算法来解决 NMF 的数据局部优化问题，该方法对使用 Fast HALS 算法更新分解矩阵的元素进行重新排序，针对 GPU 和多核 CPU 提供新的位置优化的并行算法实现。文献[13]提出基于 CUDA 架构，使用图形处理单元(Graphics-Processing Units, GPU)把输入矩阵从系统主存以块状方式传输到 GPU 的内存，矩阵子块乘法由 CUBLAS 库计算，使用 MPI 进行数据传输等操作则由 CUDA 内核执行。该方案在更新之前需要 CPU 进行一次通信，以将两个矩阵的副本传输到 GPU 内存，在每次更新结束后需要进行集体通信，以把计算结果传回 CPU，从而保持本地、副本之间的一致性，此过程会耗费大量时间。这些并行方案极大地促进了非负矩阵分解算法的并行化进程。

深层转导式非负矩阵分解[14](Deep Transductive NMF, DTNMF)应用于语音信号分离的过程，是在传统非负矩阵分解的基础上引入深层结构，在分离阶段将混合语音信号分解得到的特征表达加入字典优化特征矩阵，再进行语音信号还原操作。这些改进会导致分离过程矩阵的维度较大[15]、参数非常多、计算的复杂度较高、分解过程耗时过长。针对这一新的语音分离方法，国内外没有加速算法进行并行处理，因此本文将设计一种多级并行加速算法。

本文将主要解决深层转导式非负矩阵分解应用于语音分离场景时分离时间过长的问题，根据分解过程的特点，提出一种多级并行处理算法。本文第 2 节简述深层转导式非负矩阵分解语音信号的基本原理；第 3 节详细描述多级并行算法；第 4 节进行实验验证，并对实验结果加以分析；最后对本文进行总结，并指出进一步的研究方向。

2 面向语音分离的深层转导式非负矩阵分解的基本原理

2.1 非负矩阵分解语音信号的基本原理

非负矩阵分解方法是指对于一个给定的输入矩阵 $\boldsymbol{V}\in R^{m*n}$，以最小化一个目标函数为目的，将其分解为两个低阶的非负子矩阵 $\boldsymbol{W}\in R_{+}^{m*r}$ 和 $\boldsymbol{H}\in R_{+}^{r*n}$ 的乘积：

$$\boldsymbol{V}\approx\boldsymbol{WH} \tag{1}$$

该目标函数的数学表达如下：

$$\min_{\boldsymbol{W}\geqslant 0,\boldsymbol{H}\geqslant 0} \| \boldsymbol{V}-\boldsymbol{W}\boldsymbol{H} \|_F^2 \tag{2}$$

其中，$\| \cdot \|_F$ 表示弗罗贝尼乌斯范数(Frobenius Norm)。弗罗贝尼乌斯范数的平方用来衡量 $\boldsymbol{V}$ 和 $\boldsymbol{WH}$ 乘积的距离，即求矩阵中每项元素平方和的最小开方值。

大多数非负矩阵分解方法意在求解一个交替非负最小二乘问题(Alternating Non-negative Least Squares，ANLS)，如式(3)和式(4)所示：

$$\boldsymbol{W}\leftarrow \arg\min_{\widetilde{\boldsymbol{W}}\geqslant 0} \| \boldsymbol{V}-\widetilde{\boldsymbol{W}}\boldsymbol{H} \|_F^2 \tag{3}$$

$$\boldsymbol{H}\leftarrow \arg\min_{\widetilde{\boldsymbol{H}}\geqslant 0} \| \boldsymbol{V}-\boldsymbol{W}\widetilde{\boldsymbol{H}} \|_F^2 \tag{4}$$

迭代求解 $\boldsymbol{W}$ 和 $\boldsymbol{H}$ 的过程如算法 1 所示。

算法 1 基于交替非负最小二乘的非负矩阵分解算法

输入：m * n 维矩阵 **V**，分解阶数 r

输出：m * r 维基矩阵 **W**，r * n 维特征矩阵 **H**

1. 初始化 $\mathbf{W}_0$，$\mathbf{H}_0$

2. While(不满足循环次数)

3. 使用非负分解算法用 $\mathbf{W}_0$ 和 $\mathbf{H}_0$ 求出 $\mathbf{W}_1$

4. 使用非负分解算法用 $\mathbf{W}_1$ 和 $\mathbf{H}_0$ 求出 $\mathbf{H}_1$

5. End while

在预训练阶段，给定两个源信号($P\geqslant 2$)，用 $\boldsymbol{V}_p\in R^{m*n}$ 标记说话人的纯语音信号的矩阵表示，用 $\boldsymbol{V}_n\in R^{m*n}$ 标记噪声语音信号的矩阵表示。在 $\boldsymbol{V}_p$ 和 $\boldsymbol{V}_n$ 上分别使用 NMF 进行分离，将两个语音信号分为 $\boldsymbol{W}$ 和 $\boldsymbol{H}$ 两个子矩阵，结果形如：

$$\boldsymbol{V}_p\approx \boldsymbol{W}_p\boldsymbol{H}_p \tag{5}$$

其中，$\boldsymbol{W}_p\in R_+^{m*r}$ 表示学习到的说话人的基矩阵，基矩阵即表示包含音调变化和辅音等声学特征在内的矩阵；$\boldsymbol{H}_p\in R_+^{r*n}$ 表示学习到说话人对应的特征矩阵。同样地，噪声信号也表示为对应基矩阵与特征矩阵的乘积。

在式(5)的分离过程中，需要一个目标函数衡量分离的程度。由于 IS 散度对重构的语音信号的准度具有良好的感知特性，且对语音信号频谱图的频率域具有一定的不变性，使用 IS 散度作为目标函数进行非负矩阵分解也有可信的统计解释，因此本文选择 IS 散度作为预训练过程中说话人和噪声纯语音信号分解的目标函数，如式(6)所示：

$$D_{\mathrm{IS}}(\boldsymbol{V}\| \boldsymbol{WH})=\sum_{ij}\left(\frac{V_{ij}}{\sum_{\mu}\boldsymbol{W}_{i\mu}\boldsymbol{H}_{\mu j}}-\log\frac{V_{ij}}{\sum_{\mu}\boldsymbol{W}_{i\mu}\boldsymbol{H}_{\mu j}}-1\right) \tag{6}$$

对于给定的目标函数式(6)，使用优化最小化方法(Majorization Minimization，MM)计算该目标函数的乘性更新法则：

$$\boldsymbol{W}_{i\mu}\leftarrow \boldsymbol{W}_{i\mu}\frac{(\boldsymbol{V}_{i\mu}\cdot(\boldsymbol{WH})_{i\mu}^{-2})\boldsymbol{H}_{i\mu}}{((\boldsymbol{WH})_{i\mu}^{-1}\boldsymbol{H}_{i\mu})} \tag{7}$$

$$\boldsymbol{H}_{i\mu}\leftarrow \boldsymbol{H}_{i\mu}\frac{(\boldsymbol{V}_{i\mu}\cdot(\boldsymbol{WH})_{i\mu}^{-2})}{(\boldsymbol{W}(\boldsymbol{WH})_{i\mu}^{-1})} \tag{8}$$

在分离阶段，$\boldsymbol{V}^m\in R_+^{m*n}$ 表示混合语音信号的相应幅度谱图。以式(7)计算结果矩阵拼合而成的基矩阵作为 NMF 的输入，可分解得到混合语音信号的特征矩阵。$\boldsymbol{V}^m$ 分解结果如式(9)所示：

$$\boldsymbol{V}^m\approx \boldsymbol{W}^m\boldsymbol{H}^m \tag{9}$$

其中，$\boldsymbol{W}^m\in R_+^{m*rp}$ 表示所有说话人的基矩阵拼合形成的字典，$\boldsymbol{H}^m\in R_+^{rp*n}$ 表示学习到的混合语音对应的特征矩阵，则特征矩阵 $\boldsymbol{H}^m$ 即为下一还原阶段的一个输入。

根据文献[16]，说话人的频域语音信号可以利用式(10)进行还原：

$$\boldsymbol{Y}_p^m=\frac{\boldsymbol{V}_p^m}{\sum_{p=1}^{P}\boldsymbol{V}_p^m}\circ\boldsymbol{Y}^m \tag{10}$$

其中，$\circ$ 表示点积操作，$\boldsymbol{V}_p^m$ 表示由学习到的混合语音信号对应的特征矩阵与纯语音信号基矩阵运算形成的矩阵。

2.2 面向语音分离的深层转导式非负矩阵分解算法

虽然 NMF 在语音分离方面表现出了良好的效果，但在分离阶段，每个说话人的基矩阵并没有混合语音信号的特征信息。因此，这些由基矩阵拼接而成的字典并不能捕获到混合语音信号中的声学信息，从而导致我们不能通过学到的字典信息更精确地还原每个说话人的语音信号。

为了弥补这种不足，Guan 等[17]提出了一种转导式非负矩阵分解模型(Transductive NMF，TNMF)。该方法通过将每个说话人的训练语音信号和混合语音信号结合在一起进行预训练，来联合学习字典。TNMF 的目标函数定义为：

$$\min_{\substack{\forall 1\leqslant p\leqslant P,\boldsymbol{W}_p\geqslant 0\\ \boldsymbol{H}_p\geqslant 0,\boldsymbol{H}^m\geqslant 0}}\{\| \sum_{p=1}^{P}\boldsymbol{V}_p-\boldsymbol{W}_p\boldsymbol{H}_p \|_F^2+\lambda\| \boldsymbol{V}^m-\boldsymbol{W}^m\boldsymbol{H}^m \|_F^2\} \tag{11}$$

其中，$\boldsymbol{W}^m=[\boldsymbol{W}_1,\cdots,\boldsymbol{W}_P]$，$\lambda$ 是一个控制混合语音权重的正常数值。

实验表明，由于 TNMF 在学习字典时加入了混合语音信号的声学信息，其在语音分离过程中的效果超过了传统的非负矩阵分解方法。

深度学习由于能更好地学习到特征信息，近年来被广泛应用于各个领域，并取得了良好的效果。Liu 等[18]在 TNMF 算法的基础上，将深层结构引入矩阵计算过程，通过学习到的层次结构特征构建更精确的字典，使得从混合语音信号中还原出的信号更接近于各说话人的纯语音信号。

该方法在优化过程中将纯语音信号的基矩阵 $\boldsymbol{W}_p^k$ 和混合语音信号的基矩阵 $\boldsymbol{W}_k^m$ 均分解为 K 层，对于第 k 层，有：

$$\boldsymbol{W}_p^k=[\boldsymbol{Z}_p^1\boldsymbol{Z}_p^2\cdots\boldsymbol{Z}_p^k] \tag{12}$$

$$\boldsymbol{W}_k^m=[\boldsymbol{W}_1^k,\boldsymbol{W}_2^k,\cdots,\boldsymbol{W}_P^k] \tag{13}$$

式(13)中混合语音信号第 k 层的基矩阵由说话人和噪声的第 k 层基矩阵拼接而成。

对于 DTNMF 算法的目标函数式(11)，我们使用优化最小化方法计算 DTNMF 的乘性更新法则。对应的乘性更新法则为：

$$\boldsymbol{W}_k\leftarrow \boldsymbol{W}_k\circ\frac{\boldsymbol{V}_k\boldsymbol{H}_k^{\mathrm{T}}+\lambda\boldsymbol{V}^m\boldsymbol{H}_k^{m\mathrm{T}}}{\boldsymbol{W}_k\boldsymbol{H}_k\boldsymbol{H}_k^{\mathrm{T}}+\lambda\boldsymbol{W}^m\boldsymbol{H}^m\boldsymbol{H}_k^{m\mathrm{T}}} \tag{14}$$

$$\boldsymbol{H}^k\leftarrow \boldsymbol{H}^k\circ\frac{\boldsymbol{W}^{k\mathrm{T}}((\boldsymbol{W}^k\boldsymbol{W}^k)^{-2}\circ\boldsymbol{V})}{\boldsymbol{W}^{k\mathrm{T}}(\boldsymbol{W}^k\boldsymbol{H}^k)^{-1}} \tag{15}$$

其中，除 $\boldsymbol{V}_p$ 和 $\boldsymbol{V}^m$ 外，其余矩阵都要在每次迭代中进行更新。可以证明，式(11)、式(14)和式(15)可以逐渐降低目标函数值。

在分离混合语音时，说话人的频谱信号由说话人基矩阵以及经过 DTNMF 算法学习到的混合语音信号对应的特征矩阵进行运算来还原。由于深层结构对基矩阵和特征矩阵进行了迭代优化计算，最终可以得到最优的基矩阵和特征矩阵。

实验表明，深层结构的引入提高了语音分离的效果。

3 面向语音分离的深层转导式非负矩阵分解的并行算法

深层转导式非负矩阵分解方法进行语音分离的过程如图1所示。整个分离过程可分为3个阶段，计算需求主要表现在训练阶段的NMF以及分离阶段的DTNMF过程中。针对分离信号的预训练和分离过程的计算问题，综合考虑任务间和任务内的多级并行，采用MPI+OpenMP混合模型[19]，本文提出了一种多层级并行算法。算法设计过程中所涉及的符号及其定义如表1所列。

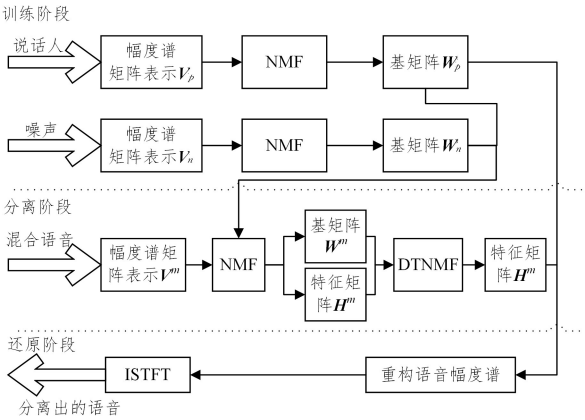


图1 面向语音分离的DTNMF分解过程

Fig. 1 Decomposition process of DTNMF for speech separation

表1 算法符号的定义

Table 1 Definition of algorithmic symbols

符号	定义
$\boldsymbol{A}$	初始矩阵
$\boldsymbol{W}$	基矩阵
$\boldsymbol{H}$	特征矩阵
$\boldsymbol{M}_i$	按行划分的第 i 行子矩阵块
$\boldsymbol{M}^i$	按列划分的第 i 列子矩阵块
m	基矩阵行数
n	特征矩阵列数
r	基矩阵的列数或特征矩阵的行数

3.1 任务级并行算法

任务级并行的实现要求所分配进程的计算量相当，若进程完成时间相差较大，则进程间的等待可能浪费资源。

在语音分离的预训练过程中，由于说话人和噪声的预训练过程是针对不同的语音信号使用NMF进行分解，各个功能的计算为独立的两部分，两个训练过程中的数据不具有依赖性且矩阵计算规模相同，因此本算法在外层将这两部分作为独立的任务执行。

任务级并行算法先将进程按照偶数号和奇数号划分成组，每个进程组创建对应的通信域，当通信域不为空时，两个进程组在各自的通信域内分别执行说话人和噪声的预训练过程，得到说话人和噪声各自的基矩阵和特征矩阵。任务级并行算法的设计如算法2所示。

算法2 预训练过程的任务级并行算法

输入：说话人的纯语音信号 $\mathbf{V}_p$，噪声语音信号 $\mathbf{V}_n$

输出：分离说话人和噪声语音信号得到的基矩阵 $\mathbf{W}_p$ 和 $\mathbf{W}_n$，对应的特征矩阵 $\mathbf{H}_p$ 和 $\mathbf{H}_n$

```
1. if(numprocs%2 ==0)
2.    rankeven[numprocs/2]={even process num}
3. else
4.    rankodd[numprocs/2]={odd process num}
      //创建两个进程组对应的通信域
5. create(rankeven[],&comm_even);
6. create(rankodd[],&comm_odd);
7. if(comm_odd !=NULL)
8.    以 Vp 为输入计算式(7)、式(8)，得到 W 和 H
9. else if(comm_even !=NULL)
10.   以 Vn 为输入计算式(7)、式(8)，得到 W 和 H
```

3.2 任务内的分布式存储并行算法

在预训练和分离过程中，使用NMF分离说话人、噪声纯信号以及混合信号，得到相应的基矩阵与特征矩阵时，MU分解需要不断迭代更新矩阵，即主进程初始化 $\boldsymbol{W}$ 以及 $\boldsymbol{H}$，使用 $\boldsymbol{W}_0$ 和 $\boldsymbol{H}_0$ 计算 $\boldsymbol{W}_1$，然后使用 $\boldsymbol{W}_1$ 和 $\boldsymbol{H}_0$ 计算 $\boldsymbol{H}_1$，不断迭代更新，直到满足迭代次数。如图2所示，分布式存储并行算法将矩阵按行列划分，主进程使用MPI[20]进行进程间通信，把矩阵块分发到从进程进行子矩阵计算，并把第二个子矩阵块在所有处理器之间轮转，计算两个矩阵所有子块的乘积，最后由主进程收集从进程数据块的分布式存储并行策略[21]。

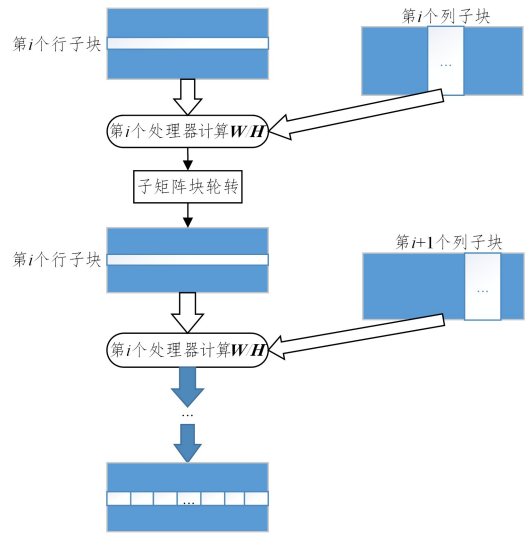


图2 矩阵分块乘法流程图

Fig. 2 Flowchart of matrix block multiplication

假设在每个通信组的分解过程中使用了 n 个进程，进程编号分别为0，1，…，$n-1$。以输入为说话人语音信号、更新 $\boldsymbol{W}$ 为例，式(12)的迭代更新过程为：把说话人语音信号矩阵 $\boldsymbol{V}$ 以及由初始值 $\boldsymbol{WH}$ 乘积得到的 $\boldsymbol{R}$ 按行划分，把基矩阵 $\boldsymbol{W}$、特征矩阵 $\boldsymbol{H}$ 按列划分为子矩阵；由主进程使用MPI_Scatter进行节点间通信，将子矩阵块发送到从进程，$\boldsymbol{W}^i$，$\boldsymbol{H}^i$，$\boldsymbol{V}_i$ 和 $\boldsymbol{R}_i$ 表示第 i 个进程上接收到的进程块，在第 i 个进程上，调用MKL库函数进行子矩阵块的计算，并将计算结果发送到第 i 号进程的结果矩阵子块进行保存，然后偶数号进程以下一个进程号为目的发送当前 $\boldsymbol{H}^i$ 矩阵块，奇数号进程接收上一个进程发送的数据，完成第 i 行第 $i-1$ 列的结果计算；再将计算结果发送至第 i 号进程保存，矩阵不能整除的剩余部分在主进程

中做相同运算；计算完成后，第 i 号进程的结果矩阵子块中保存了按行划分的第 i 个基矩阵子块和特征矩阵所有子矩阵的计算结果，从进程将结果子矩阵块发送给主进程，主进程按序接收，并将子矩阵按内存位置赋值，即更新 $\boldsymbol{W}$。$\boldsymbol{H}$ 的更新过程类似。

预训练过程中的 MU 并行算法如算法 3 所示。

算法 3 MU 并行算法

输入：维度为 m * n 的矩阵 **V**，**WH** 矩阵乘积维度为 m * n 的矩阵 **R**，分解维度 r，进程数目 p，第 i 号进程上的运算 i

输出：m * r 的基矩阵 **H**，r * n 的特征矩阵 **H**

```
1. Initialize V,W,H,R
2. while(not satisfied with circulation number)
3.    for m=1 to p
4.       recv m-th block V,W,H,R to process 0
5.       calculate formula(7) using cblas
6.       send m-th block to nextrank
7.       recv( m-1)-th block from previousrank
8.       calculate the sub-resultblock
9.       send sub-resultblock to rank 0
10.   rank 0 recv sub-resultblock from rank m
11.   rank 0 calculate the rest matrix on formula(7)
12.   for m=1 to p
13.      recv m-th block V,W,H,R to process 0
14.      calculate formula(8) using cblas
15.      send m-th block to nextrank
16.      recv (m-1)-th block from previousrank
17.      calculate the sub-resultblock
18.      send sub-resultblock to rank 0
19.   rank 0 recv sub-resultblock from rank m
20.   rank 0 calculate the rest matrix on formula(8)
21. end while
```

算法 3 中第 3－11 行表示更新 $\boldsymbol{W}$ 的过程，第 12－20 行表示更新 $\boldsymbol{H}$ 的过程。

3.3 任务内共享存储并行算法

OpenMP 采用 fork-join 的执行模式。程序开始，单节点内只有主线程运行，当遇到指导语句 # pragma 时，主线程会派生出多个分支线程来执行指导语句内的程序，程序结束后，分支线程又把控制权交回给主线程。

在子矩阵乘法运算过程（如算法 3 中第 8 行）中，当第 i 个进程分配到基矩阵 $\boldsymbol{W}$ 和特征矩阵 $\boldsymbol{H}$ 的第 i 个子矩阵块时，两个子矩阵块保存在同一个节点内，共享内存变量，所以通过共享内存、交换数据的加速策略，在每个进程内部子矩阵块计算前，加入指导语句，由主进程生成多线程并行计算矩阵块乘法（见算法 4），在矩阵块计算结束后各线程又合并回主线程。

算法 4 子矩阵乘的线程级并行算法

输入：维度大小为 row * r 的基矩阵 $\mathbf{W}_i$ 和 r * col 的特征矩阵 $\mathbf{H}_i$

输出：维度大小为 row * col 的矩阵 $\mathbf{R}_i^i$

```
1. # pragma omp parallel for shared(R,W,H) private(i,j,k)
2.    for(i=0;i<row;i++)
3.       for(j=0;j<col;j++)
4.          R_i^i[i][j]=0;
5.          for(k=0;k<r;k++)
6.             R_i^i[i][j]+=W_i[i][k] * H_i[k][j];
7. # pragma omp parallel for
8. for(i=0;i < row * col;i++)
9.    Rbuf[i]=pow(Rbuf[i],(beta-2)) * Vbuf[i];
10.   Rbuf[i]=pow(Rbuf[i],(beta-1));
11. # pragma omp sections
12.      # pragma omp section
13. cblas(Rbuf,Hbuf,RHbuf);
14.    W # pragma omp section
15. cblas(Wbuf,Hbuf,Rbuf).
```

4 实验结果与分析

NMF 的广泛使用，使其不仅适用于理论研究，也适用于密集的现实数据（如语音、视频、图像等）的处理过程。本文选择 MPI＋OpenMP 的编程模型对并行算法进行实现；此外，我们还使用了其他技术优化算法，如线性代数计算中通用的函数集合库函数 BLAS[22]（Basic Linear Algebra Subprograms）中的相关函数。这些接口的调用与编程模型的混合使用具有以下优势：1）可以利用最新的硬件进行改进，最大程度地发挥硬件的优势；2）数值稳定的 BLAS 的可用性较高。

4.1 测试环境

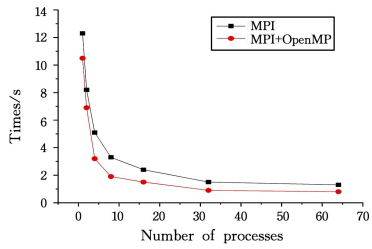
测试的硬件平台为天河二号超级计算机，配置麒麟 Linux 操作系统，支持 Fortran，C，C＋＋和 Java 等语言，以及 OpenMP 和 MPI3.0 标准的并行编程。每个计算节点有 2 个至强 E5-2692 v2 2.20 GHz 中央处理器，每个 CPU 有 12 个核，每个节点的 CPU 内存为 64 GB。代码通过 C，MPI 和 OpenMP 实现，通过 mpiicc 编译。

数据集 LibriSpeech 是由 Vassil Panayotov 发布的时长约为 1 000 h、频率为 16 kHz 的英语演讲语料库。这些数据来自 LibriVox 项目的阅读有声读物，并经过细分和对齐，是语音领域常用的数据集。

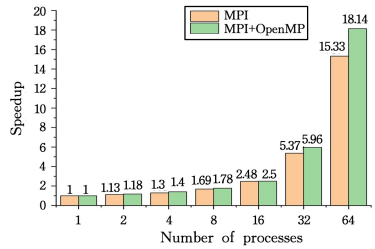
4.2 实验结果分析

本文共在 4 种条件下做了相应的实验测试：1）相同矩阵规模下使用 MPI 和 MPI＋OpenMP 混合模型进行语音分离的时间和加速比实验；2）相同矩阵规模下使用 MPI＋OpenMP 混合模型且取不同进程和线程数的时间实验；3）不同规模矩阵下使用 MPI＋OpenMP 混合模型进行语音分离的时间和加速比实验；4）待分解矩阵规模相同但分解阶数不同时使用 MPI＋OpenMP 混合模型的时间实验。

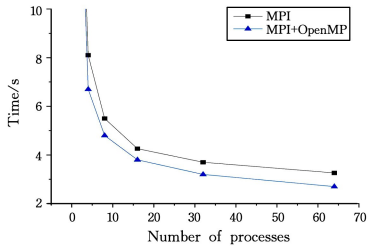
图 3 描述了预训练过程和分离过程中同一规模矩阵分别使用 MPI 和 MPI＋OpenMP 混合模型时，对应运行时间随核数增加的变化情况。从图 3(a)和图 3(c)可以看出，预训练过程中随着进程数的增加，两种编程模式的运行时间都在减少，当 CPU 核数达到 64 时，两种编程模式的效率都达到最大。图 3(b)和图 3(d)计算了对应的加速比。可以看出：预训练阶段使用纯 MPI 编程模式的最大加速比达到 15，使用 MPI＋OpenMP 编程模式的加速比达到 18；分离阶段使用 MPI 模型的最大加速比为 15，使用 MPI＋OpenMP 混合模型的最大加速比可达 24。由此可得，使用混合编程模型可以极大地提高分离的效率。



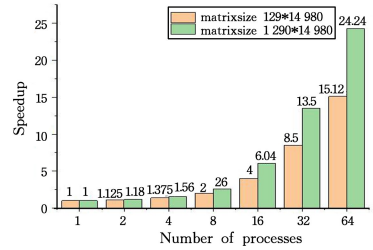
(a)预训练阶段 MPI 和混合模型的运行时间



(b)预训练阶段 MPI 和混合模型的加速比



(c)分离阶段 MPI 和混合模型的运行时间

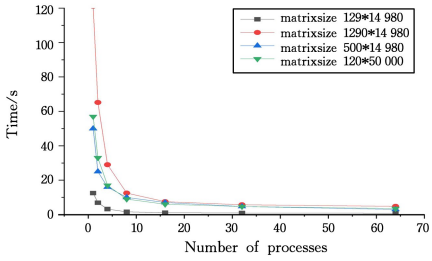


(d)分离阶段 MPI 和混合模型的加速比

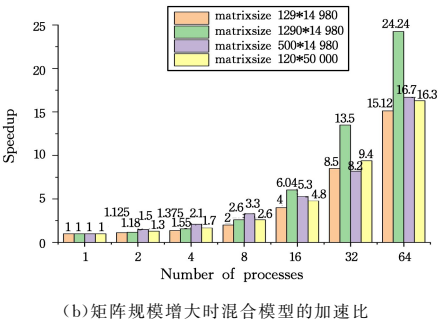
图 3 纯 MPI 与混合模型在不同阶段的运行时间与加速比

Fig. 3 Runtime and speedup of pure MPI and hybrid models at different stages

图 4 给出了不同规模矩阵使用 MPI+OpenMP 混合模型在多节点上的运行时间。由图 4(a)可知，随着矩阵规模的增大，分离时间增加，混合模型在核数相同的情况下，矩阵越大时分离时间越接近线性增加；随着核数的增加，运行时间都在逐渐减少。可见，本文设计的并行算法适用于变化的矩阵规模，具有一定的可扩展性。图 4(b)给出了对应的加速比，可以看出，当矩阵规模扩大时，使用混合模型可以获得更大的加速比。



(a)矩阵规模增大时混合模型的运行时间



(b)矩阵规模增大时混合模型的加速比

图 4 当矩阵规模变化时混合模型的运行时间与加速比

Fig. 4 Running time and speedup of hybrid model when matrix size changes

图 5 给出了同一规模矩阵从单节点到多节点上使用不同进程数和线程数的运行时间。可以看出，使用 MPI+OpenMP 的混合编程模型相比纯 MPI 编程模型缩短了运行时间，当进程数达到 64，线程数为 16 时，其分离时间最短，加速比最高达到 15.125。可知，使用 MPI+OpenMP 的混合模型可以有效缩短分离时间。

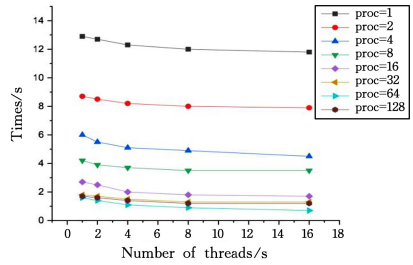


图 5 混合模型在不同进程与线程数情况下的运行时间

Fig. 5 Running time of hybrid model with different processes and threads

图 6 给出了待分解矩阵规模不变的情况下，改变阶数大小时使用混合模型进行分离的运行时间。可以看出，阶数的增加，同时增大了矩阵块在内存中占用的空间，使得 MPI 通信过程花费更长的时间，同时矩阵运算时间增加。当进程数为 32 时，阶数为 500 的分离过程取得最好效果；进程数为 64 时，阶数分别为 750 和 1 000 的分离过程取得最好效果。

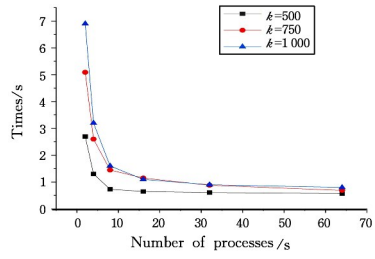


图 6 分解阶数不同时的运行时间

Fig. 6 Running time at different decomposition layers

以上实验结果分析表明，使用 MPI+OpenMP 混合模型的效果优于纯 MPI 编程模型的效果，这是由于混合模型能最大限度地适应计算机硬件体系结构，高效利用计算资源。在粗粒度上使用 MPI 进行通信，虽然需要消耗一部分资源，但

当计算时间远远超过通信时间时，通信开销可以忽略；在细粒度上使用 OpenMP，可利用共享内存体系结构派生出多个线程共享单一节点资源，提高了并行化程度。

综上所述，本文提出的并行算法对于面向语音分离的深层转导式非负矩阵分解算法具有较好的并行效果，且可扩展性较强。

结束语 本文以 DTNMF 的预训练过程和分离过程计算需求问题为研究对象，针对计算量过大的问题，首次提出深层转导式非负矩阵分解的并行算法。通过对并行算法的实验结果进行分析得出：使用混合模型，对加速分离过程、缩短分离时间有明显效果。

该并行算法为将混合语音分离为纯语音后进行性能优化提供了新的方向。如何在众多参数情况下对算法进行进一步优化，实现计算与通信的重叠，以及提高数据传输效率，或者将算法应用于异构体系以获取更优的加速效果，是下一步研究的重点。

参 考 文 献

[1] LEE D D, SEUNG H S. Algorithms for non-negative matrix factorization[C]// NIPS. 2001: 556-562.

[2] LEE D D, SEUNG H S. Learning the parts of objects by non-negative matrix factorization[C]// Nature. 1999, 401: 788-791.

[3] ANDRZEJ C, HUY P A, RAFAL Z, et al. Nonnegative matrix and tensor factorizations applications to exploratory multi-way data analysis and blind source separation[M]. Wiley Publishing, 2009.

[4] GEMULLA R, NIJKAMP E, HAAS P J, et al. Large-scale matrix factorization with distributed stochastic gradient descent [C]// Proceedings of the KDD. ACM, 2011: 69-77.

[5] KIM J, PARK H. Fast Nonnegative Matrix Factorization: An Active-Set-Like Method and Comparisons[J]. SIAM Journal on Scientific Computing, 2011, 33(6): 3261-3281.

[6] DONG C, ZHAO H, WANG W. Parallel Nonnegative Matrix Factorization Algorithm on the Distributed Memory Platform [J]. International Journal of Parallel Programming, 2010, 38(2): 117-137.

[7] LIU C, YANG H C, FAN J, et al. Distributed nonnegative matrix factorization for web-scale dyadic data analysis on map-reduce[C]// Proceedings of the 19th International Conference on World Wide Web. ACM, 2010: 681-690.

[8] KANJANI K. Parallel Non Negative Matrix Factorization for document clustering[Z]. CPSC-659 Spring 2007 Course Project: Texas A&M University, 2007.

[9] LOPES N, RIBEIRO B. Non-negative Matrix Factorization. Implementation using Graphics Processing Units[C]// International Conference on Intelligent Data Engineering & Automated Learning. Springer-Verlag, 2010.

[10] KANNAN R, BALLARD G, PARK H. HPC-NMF: A High-Performance Parallel Algorithm for Nonnegative Matrix Factorization [J]. arXiv: 1509.09313.

[11] ROBILA S A, MACIAK L G. A parallel unmixing algorithm for hyperspectral images[C]// Optics East. International Society for Optics and Photonics, 2006.

[12] MOON G E, SUKUMARAN-RAJAM A, PARTHASARATHY S, et al. PL-NMF: Parallel Locality-Optimized Non-negative Matrix Factorization[J]. arXiv: 1904.07935, 2109.

[13] MEJÍA-ROA E, TABAS-MADRID D, SETOAIN J, et al. NMF-mGPU: non-negativematrix factorization nonmulti-GPU systems [J]. BMC Bioinfor-matics, 2015, 16(1): 43.

[14] LIU Y L. Research on Key Technologies of Speech Separation and Speech Recognition [D]. Changsha: National University of Defense Technology, 2018.

[15] CHEN X H, XIE P Z, CHI LH, et al. An efficient SIMD compression format for sparse matrix-vector multiplication[J]. Concurrency Computat Pract Exper, 2018, 30: e4800.

[16] MOHAMMADIHA N, SMARAGDIS P, LEIJON A. Supervised and Unsupervised Speech Enhancement Using Nonnegative Matrix Factorization[J]. IEEE Transactions on Audio Speech & Language Processing, 2013, 21(10): 2140-2151.

[17] GUAN N, LAN L, TAO D, et al. Transductive nonnegative matrix factorization for semi-supervised high-performance speech separation[C]// IEEE International Conference on Acoustics, Speech and Signal Processing. IEEE, 2014: 2534-2538.

[18] LIU Y, GUAN N, LIU J. Deep Transductive Nonnegative Matrix Factorization for Speech Separation[C]// 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA). IEEE, 2017.

[19] ZHAO Y H, CHI X B. MPI+OpenMP hybrid programming model based on SMP cluster and effective implementation [J]. Microelectronics and Computer, 2005, 22(10): 7-11.

[20] GU K H, HUANG M, HE J Y. Research on MPI+OpenMP hybrid programming model based on multi-core cluster[J]. Gansu Techenology, 2018, 34(19): 10-14.

[21] XU C, DENG X, ZHANG L, et al. Collaborating CPU and GPU for large-scale high-order CFD simulations with complex grids on the TianHe-1A supercomputer[J]. Journal of Computational Physics, 2014, 278: 275-297.

[22] Intel Math Kernel Library[EB/OL]https://software.intel.com/en-us/mkl-developer-reference-c-blas-level-3-routines.

LI Yu-rong, born in 1994, postgraduate. Her main research interests include DTNMF parallel methods and high performance computing.

LIU Jie, born in 1969, professor, Ph. D supervisor. His main research interests include parallel algorithm and high performance computing.