doi:10.11918/j.issn.0367-6234.2016.11.009

面向异构并行架构的大规模原型学习算法

苏统华[1], 李松泽[1], 邓胜春[1], 于 洋[2], 白 薇[3]

(1.哈尔滨工业大学 软件学院, 哈尔滨 150001;2.中建八局大连公司, 辽宁 大连 116021;
3.诺基亚通信系统技术(北京)有限公司浙江分公司, 杭州 310053)

摘 要: 为解决当前原型学习算法在大规模、大类别机器学习和模式识别领域的计算密集瓶颈问题,提出一种采用 GPU 和 CPU 异构并行计算架构的可扩展原型学习算法框架. 一是通过分解和重组算法的计算任务,将密集的计算负载转移到 GPU 上,而 CPU 只需进行少量的流程控制. 二是根据任务类型自适应地决定是采用分块策略还是并行归约策略来实现. 采用大规模手写汉字样本库验证本框架,在消费级显卡 GTX680 上使用小批量处理模式进行模型学习时,最高可得到 194 倍的加速比,升级到 GTX980 显卡,加速比可提升到 638 倍;算法甚至在更难以加速的随机梯度下降模式下,也至少能获得 30 倍的加速比. 该算法框架在保证识别精度的前提下具有很高的可扩展性,能够有效解决原有原型学习的计算瓶颈问题.

关键词: 原型学习;学习矢量量化;手写汉字识别;并行归约;异构并行计算
中图分类号: TP181 **文献标志码**: A **文章编号**: 0367-6234(2016)11-0053-08

Massively scalable prototype learning for heterogeneous parallel computing architecture

SU Tonghua[1], LI Songze[1], DENG Shengchun[1], YU Yang[2], BAI Wei[3]

(1. School of Software, Harbin Institute of Technology, Harbin 150001, China; 2. Dalian Branch China Construction Eighth Engineering Division Corp. Ltd, Dalian 116021, Liaoning, China; 3.Nokia Solutions and Networks, Hangzhou 310053, China)

Abstract: Current learning algorithms for prototype learning require intensive computation burden for large category machine learning and pattern recognition fields. To solve this bottleneck problem, a principled scalable prototype learning method is proposed based on heterogeneous parallel computing architecture of GPUs and CPUs. The method can transfer the intense workload to the GPU side instead of CPU side through splitting and rearranging the computing task, so that only a few control process is needed to be managed by the CPU. Meanwhile, the method has the ability to adaptively choose the strategies between tiling and reduction depending on its workload. Our evaluations on a large Chinese character database show that up to 194X speedup can be achieved in the case of mini-batch when evaluated on a consumer-level card of GTX 680. When a new GTX980 card is used, it can scale up to 638X. Even to the more difficult SGD occasion, a more than 30-fold speedup is observed. The proposed framework possess a high scalability while preserving its performance precision, and can effectively solve the bottleneck problems in prototype learning.

Keywords: prototype learning; learning vector quantization; Chinese character recognition; parallel reduction; heterogeneous parallel computing

学习矢量量化 LVQ (learning vector quantization)是一种适用于大规模、大类别分类任务的原型学习算法,具有低存储、高识别吞吐率的优点. 已有的研究表明,当采用判别学习准则时,原型学习能在数字识别、汉字识别(含日本文字识别)问题上获得较先进的识别结果[1]. LVQ 同样在选取有限候选类方面具有非常好的效果. 例如,LVQ 可以用来为复杂模型筛选小部分有潜力的类别集,从而有效缓解 PL-MQDF 模型[2]的高强度训练过程. 更重要的是,随着移动设备的普及,LVQ 以其模型精

收稿日期: 2015-05-11
基金项目: 国家自然科学基金(61203260);黑龙江省自然科学基金重点项目(ZD2015017);哈尔滨工业大学科研创新基金(HIT.NSRIF.2015083)
作者简介: 苏统华(1979—),男, 博士, 副教授;
邓胜春(1971—),男, 博士,教授,博士生导师
通信作者: 苏统华, thsu@hit.edu.cn

巧、速度快等特点能很好地适用于智能手机、平板电脑等嵌入式设备上的输入法应用[3-4].

学习一个鲁棒的大规模 LVQ 模型,其计算复杂性令人望而生畏. 若使用单核 CPU 的传统实现方法,将需要若干天的训练时间. 而对于判别学习准则来说,数据越多识别效果则越好,当然学习时间会相应的加长. 部分研究者尝试通过收集或人工合成的方式获取更多的训练样本. 最新的一些研究结果认为,当指数级增长样本数量时,识别效果可以得到稳步提升[5-6]. 然而,目前已进入了多核计算时代. 英特尔公司的 Pat Gelsinger 曾表示,若芯片仍按照传统方式设计,到 2015 年芯片将如同太阳表面一样热[7]. GPU 所使用的异构并行计算架构已开始逐渐补充甚至部分代替传统的串行计算架构. 因此,从单核处理器转向大规模并行处理器是未来算法设计的必然趋势.

GPU 异构计算架构在机器学习以及模式识别任务领域具有突出的加速效果. Raina 等人研究了深度信念网络 DBN (deep belief network)与稀疏编码在 GPU 上的实现[8],他们的 DBN 实现方案达到了 70 倍的加速比,而稀疏编码的并行算法则获得了 5 到 15 倍的加速比. 一些学者也研究了大型多层神经网络在 GPU 上的实现. Scherer 与 Behnke[9] 在 GPU 上实现了加速比达 100 倍的卷积神经网络 CNN (convolution neural network). Ciresan 等人则在 GPU 上实现了深度多层感知机,并且取得了当前最好的识别性能[10]. 周明可等人则针对改进二次判别函数 MQDF (modified quadratic discriminative function)实现了基于 GPU 的判别训练方法,并将其成功应用到了汉字识别上,其小批量处理实现方案获得了 15 倍的加速比[6].

本文提出一种适用于大规模、大类别分类任务的异构原型学习算法框架. 与已有的研究工作不同,本文提出的框架几乎将所有的计算负载都调度到 GPU 上,CPU 只负责协调部分计算逻辑. 为了充分利用 GPU 的计算资源,算法深入分析了计算负载的可并行度,大量使用了分块平铺以及并行归约等并行计算模式. 算法在大类别手写汉字识别任务下进行验证,得到比较高的可扩展性. 在小批量处理的模式下,使用消费级显卡 GTX680,该算法最高可达 194 倍的加速比. 当升级到新一代 GTX980 时,加速比提升到 638 倍. 最值得一提的是,该算法在随机梯度模式下也可以获得至少 30 倍的加速比.

1 算 法

1.1 LVQ 串行算法

假设有一个包含 C 个类别的分类任务,原型学习即产生一个原型向量集 $\boldsymbol{\Theta} = \{m_i,\ i = 1,\ 2,\cdots,C\}$. 为便于讨论,本文的形式化主要对每个类包含一个原型(类中心)的情况展开,所述框架同样适用于每个类包含多个原型的情况. 预测未知标号样本 x 的标记问题,可以转化为查找最小距离问题. 计算 x 与每个原型向量的欧氏距离,通过查找 C 个距离的最小值,就可以把 x 的标号设为拥有最小距离的原型向量所在类别,表示 x 与该类别的原型最相似. 该预测过程可形式化为把 x 赋予标号 j:

$$j = \arg\min_i \| x - m_i \|_2.$$

为了学习一个原型向量集,需要从大规模训练样本中进行有监督的训练. 用 $\{(x_n, y_n),\ n = 1,2,\cdots,N\}$ 表示训练样本集,其中 y_n 为样本 x_n 的真实标号,整个训练过程的目标就是针对训练集最小化经验损失[11]:

$$\hat{\boldsymbol{\Theta}} = \arg\min_{\Theta} \frac{1}{N}\sum_{n=1}^{N} \varphi(f(x_n, \boldsymbol{\Theta}))\ , \tag{1}$$

其中 $\varphi(\cdot)$ 为针对评分函数 $f(x, \boldsymbol{\Theta})$ 的损失函数.

为了求解式(1)中的最小化问题,通常采用随机梯度下降 SGD (stochastic gradient descent)的方法对参数进行更新:

$$\boldsymbol{\Theta}(t+1) = \boldsymbol{\Theta}(t) - \eta \nabla_{\Theta} \varphi(f(x))\mid_{x = x_n},$$

其中 η 表示学习步长.

为说明 LVQ 中的判别学习思想,此处以广义学习矢量量化 GLVQ (generalized learning vector quantization)为例. 首先需定义一个可以度量样本 x 误差的测量函数:

$$f(x) = \frac{d_c - d_r}{d_c + d_r}\ , \tag{2}$$

这里 d_c 与 d_r 分别表示样本 x 与真实类别 c 以及竞争类别 r 之间的距离. 分类损失函数可以通过 sigmoid 函数近似:

$$\varphi(f(x)) = \frac{1}{1 + \mathrm{e}^{-\xi f(x)}}\ , \tag{3}$$

其中 ξ 用来调节 sigmoid 函数的平滑度.

若使用欧氏平方距离, 则 m_c 与 m_r 的更新公式可以表示为 x 的下列函数:

$$\begin{cases} m_c \leftarrow m_c + \eta f(x)(1 - f(x))\dfrac{d_r}{(d_c + d_r)^2}(x - m_c)\ , \\ m_r \leftarrow m_r - \eta f(x)(1 - f(x))\dfrac{d_c}{(d_c + d_r)^2}(x - m_r)\ . \end{cases} \tag{4}$$

整个 GLVQ 学习算法可由算法 1 中的伪码表示. 算法 1 主要重复执行下列任务:采样一个样本,计算出该样本与所有原型之间的距离,获得真实类

别与竞争类别，计算损失函数以及梯度，最后更新原型向量.

算法 1　GLVQ 学习算法(串行版本)

Input: training set $\{x_n, y_n\}_{n=1,\cdots,N}$, initial prototypes $\{m_i\}_{i=1,\cdots,C}$

Output: $\{m_i\}_{i=1,\cdots,C}$

1: while not convergent do
2:　for each $\{x_n, y_n\}$
3:　　find out (m_c, d_c) and (m_r, d_r) through compare- then-exchange distances
4:　　compute error measure $f(x)$ using Eq.(2)
5:　　derive loss function $\phi(x)$ using Eq.(3)
6:　　update m_c and m_r using Eq.(4)
7:　end for
8: end while
9: return $\{m_i\}_{i=1,\cdots,C}$

1.2　并行原型学习框架

算法 1 的整体处理流程是一个串行过程. 为了将其扩展到异构并行计算框架，采用带小批量处理(数量为 mb) 的梯度下降算法. 改进的算法框架如算法 2 所示，其中的每个计算步骤(第 3 到 6 行)都可以并行执行. 本框架不是一个接着一个的逐一计算每个样本与每个原型的距离，而是一次性计算一个批次的样本与全部原型的距离，保存为距离矩阵(见算法 2 第 3 行)；与距离矩阵有关的计算涉及高密度的计算，具有较高的可扩展性. 对于查找真实类别与竞争类别可以与计算分类损失函数合并进行，需要考察不同的并行执行备选方案(见算法 2 第 4 和 5 行). 最后执行的参数更新操作，由于是针对一批样本的计算，再加上每个样本包含数百维特征，所以具有天然的并行性.

算法 2　GLVQ 学习算法(并行框架)

Input: training set $\{x_n, y_n\}_{n=1,\cdots,N}$, initial prototypes $\{m_i\}_{i=1,\cdots,C}$

Output: $\{m_i\}_{i=1,\cdots,C}$

1:　while not convergent do
2:　　for each mini-batch $T_i = \{(x_{i_1}, y_{i_1}), \cdots, (x_{i_M}, y_{i_M})\}$
3:　　compute all distances as a matrix in parallel
4:　　find out genuine/rival pair in parallel
5:　　derive loss functionin parallel
6:　　update prototypesin parallel
7:　end for
8:　end while
9:　return $\{m_i\}_{i=1,\cdots,C}$

本文中的 GPU 程序设计围绕英伟达公司的计算统一设备架构 CUDA(compute unified device architecture)编程模型展开. GPU 硬件从物理上提供了两个层面的并行模式：一块 GPU 上包含多个流多处理器 SM (streaming multiprocessor)，每个流多处理器上又包含若干个流处理器(或称为 CUDA 核心). 代码的最终物理执行在 SP 上，CUDA 核函数将计算封装然后分配到 GPU 上执行. 逻辑上，CUDA 也包含两个软件抽象层与两个物理层相对应，即线程块与线程，一个线程块由一组线程组成，线程块调度到 SM 上执行，线程块中的每个线程再具体调度到 SP 上执行.

同一线程块上的所有线程可以对一小块称为共享内存的存储区(SM3.0 的设备共享内存大小为 64KB)进行访问，并且在执行的任一时刻都可以进行同步. GPU 上还有一块叫做全局内存的存储区，其容量较大，GTX680 上的全局内存达到 2GB. GPU 上所有的线程均可访问全局内存，但全局内存的访问速度却比共享内存慢两个数量级. 因此，英伟达公司提供了一种叫存储合并的技术为特定连续数据的存储访问提供优化方案. 由于 GPU 的计算与存储都是并行执行的，因此，许多算法的主要瓶颈都出现在 CPU 与 GPU 全局内存的数据传输上以及全局内存的访问，另外合理利用共享内存也能对程序的加速做出很大贡献.

为了充分挖掘 GPU 的性能，有两点必须注意. 首先，CPU 与 GPU 全局内存的数据传输次数应尽量少. 对于机器学习和模式识别问题，可以通过将原型模型数据一直保存在 GPU 全局内存的方式来减少数据的传输次数. 然而，有时全局内存并不能保存所有的训练数据，此时可以只在使用时传输相应的数据，每次传输的数据量尽量多，以此保证总体的传输次数最少. 由于原型参数数据以及训练数据均保存在 GPU 的全局内存中，参数的更新操作也可以直接在 GPU 上完成.

另外需要注意的一点是学习算法的设计和实现需考虑线程块与线程这两个层面，这样才能高效使用共享内存，实现全局内存合并访问. 通常，线程块的选取控制着整体数据并行策略，而线程块内的线程通过使用共享内存和同步操作，控制着最底层的并行效果. 此外，已调度到 SM 上准备执行的线程块在等待全局内存访问时，图形硬件能很好地隐藏存储延迟. 为了充分利用这些延迟时间，线程块的数量可以尽量多，且相互之间独立执行.

根据以上两点并行设计原则，本文提出一种可以把密集型计算分发到 GPU 上的计算框架，其流程

图如图1所示. 图中GPU与CPU控制权的更迭用虚线箭头表示. 由图1可知:原型向量只在程序启动时传输到GPU上,并在程序结束时从GPU传输回CPU;若全局内存无法一次性容纳整个训练集,则通过小批量处理的方式分批将训练样本传输到GPU全局内存(CPU和GPU端的数据传输采用曲线箭头表示),否则,只需一次性将所有数据拷贝到GPU. 整个执行过程中,仅在流程控制和准备数据上,需要少量的CPU干预.

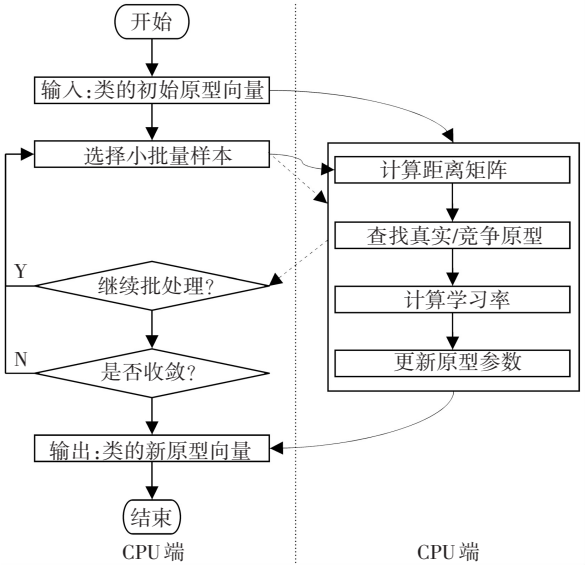

图1　原型学习的异构计算模型

Fig.1　Heterogeneous computing model for prototype learning

2　算法实现

本文的算法实现采用CUDA并行编程架构. 在每轮的训练过程中主要调用了3个CUDA核函数. 算法2中的第3到6步分别在3个独立的核函数中执行,其中第4步和第5步在同一核函数内执行. 在这3个核函数中,前2个最消耗时间,是所谓的"热点". 针对这2个核函数,本文分别提供了两种不同的并行计算算法,并对其效率进行分析.

2.1　基于并行归约的距离计算

归约操作可以在K个输入元素上执行操作,转化得到1个输出元素. 它可以用来并行执行可交换的二元操作. 标准的归约算法可以在文献[12]中找到. 此处采用归约算法来计算两个向量之间的欧氏平方距离. 在计算过程中,需定义操作符⊕:

$$a_i \oplus b_i \equiv + \ (a_i - b_i)^2.$$

图2显示了16个线程时的归约结构(每个线程处理一个元素). 第一轮计算时,前8个元素依序与后8个元素进行操作符为⊕的运算. 第二轮则是前8个元素中的前4个元素与后4个元素对应运算. 以此类推. 假设K是2的幂次倍,对于一个K维的向量则只需要计算$\log_2(K)+1$轮就能得到平方距离.

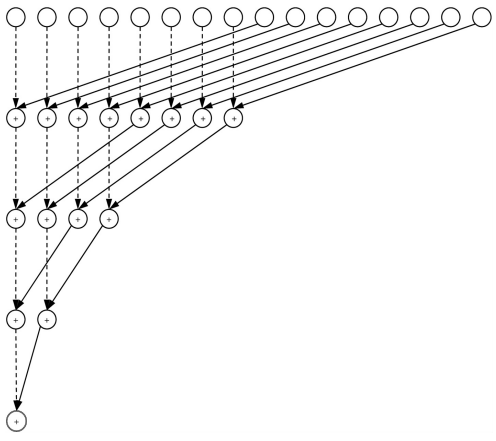

图2　归约过程示意(以16元素为例)

Fig.2　Reduction of 16 elements

初始时,每个线程块从全局内存加载一个样本向量和原型向量,负责一个平方距离的计算. 因此,一共需要(C,mb)个线程计算$C \times mb$个平方距离.

然而,在具体实现的时候,仍有许多细节问题需要注意. 例如有时样本特征向量的维度(用dim表示)并不是2的幂次倍,如160. 此时,线程块的线程数目仍可以开启为2的幂次倍,但其大小必须是小于特征向量的维度和物理硬件允许的线程块线程数目的最大2的幂次值. 对于超出线程块大小的数据,可以在进行对数步(log-step)归约之前,通过额外一次⊕运算累加到线程块前部线程的部分和上. 另外,程序应尽可能复用共享内存. 相较于一个线程块计算一个平方距离,也可以让每个线程块计算一个样本与TILE_LEN个原型之间的欧氏距离,以提高样本数据的利用率.

2.2　基于分块加和的距离计算

与归约算法不同,该算法的思想是一个线程独立计算一个欧氏距离,每个线程首先从全局内存中获取2个维度为dim的向量,然后执行一个序列化的归约操作. 但这样读取全局内存的效率并不高,容易造成阻塞. 为了解决矩阵乘法任务中全局内存的访问阻塞问题,文献[13]采用基于数据分块思想进行改善. 这种思想进行适当改进,也能很好地用来解决这里的平方距离的计算问题.

算法的整体目标是计算$mb \times C$个距离,因此可以将这些距离看做一个矩阵,并将这个矩阵划分为TILE_LEN×TILE_LEN个块,每个分块的维度大小与线程块的大小相同. TILE_LEN值的选择受到可用共享内存大小的限制和线程块大小的限制,必须保证每个线程块的共享内存能够容纳该线程块内的所有计算数据. 然后把样本矩阵与原型矩阵也照此划分成块. 因此,每个线程计算平方距离需要(dim+

TILE_LEN-1)/TILE_LEN 个阶段，每个向量距离均是通过迭代累加而得. 该算法的详细过程如图 3 所示. 利用这种方法，全局内存的访问次数可减少到原来的 1/TILE_LEN.

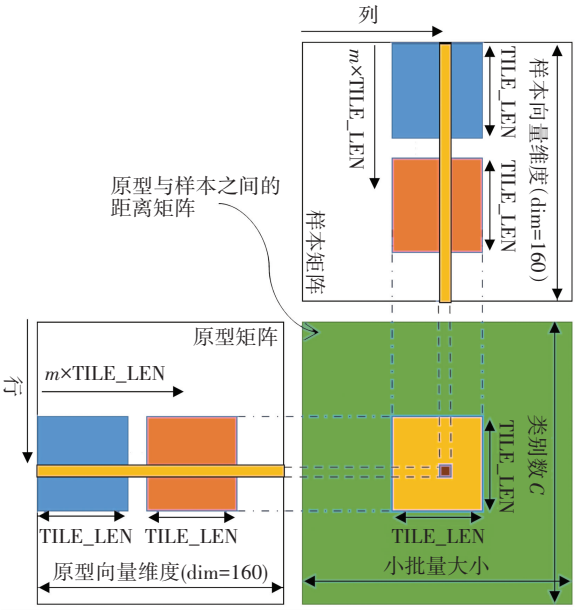

图 3　通过分块思想实现平方距离的计算

Fig.3　Caculation of squared distances via tiling idea

2.3　基于并行归约的最小距离搜索

当计算出样本 (x_n, y_n) 与 C 个原型的距离后，接下来的任务就是找出该类的真实类别(含额外的距离信息)(d_c, c) 与竞争类别(含额外的距离信息)(d_r, r). 从而可以通过梯度下降方法更新 m_c 与 m_r. 这里提出一种并行归约的方式来寻找与该样本最近的距离以及与其对应的原型索引，并通过一些技巧来避免程序的条件判断代码. 例如，既然 d_r 是除了 d_c 之外的最小距离，在程序执行时，可先通过 y_n 获取到 d_c，然后将其设为一个无限大的值，以防止寻找最小距离时需要对 d_c 进行特别的判断操作. 若每个类使用多个原型中心，则需要提前通过一遍归约操作来计算出最近的原型索引.

算法的内核函数启动了 mb 个线程块，每个线程块包含 1024(或 512) 个线程. 同一个线程块内的所有线程将在共享内存中做归约操作，找出与该样本距离最近的竞争类别. 同样，若类别数 C 不为 2 的幂次倍，也可采用 3.1 节中相同的技巧. 另外，C 可能比一个线程块包含的线程数还要大，此时需要在进行对数步归约之前先对那些相距线程块大小的数据进行迭代加和.

2.4　基于比较交换的最小距离搜索

该算法只需调用 mb 个线程，每个线程通过比较交换操作找出一个样本的真实类别(含额外的距离信息)(d_c, c) 与竞争类别(含额外的距离信息)(d_r, r). 若 mb 比线程块理论最大线程数还大，则需要分配多个线程块.

该算法是对串行搜索算法进行的最粗粒度的并行化. 一个线程将要执行时间复杂度为 $O(C)$ 的操作才能得到输出，因此这是一种线程密集型的处理策略. 若 mb 的值很小，则无法充分利用 GPU 的资源.

2.5　参数更新

若使用随机梯度下降(即 $mb = 1$)，则每次更新时只需对两个原型向量进行更新，即真实类别原型向量和竞争类别原型向量. 参数更新的内核函数只需调用 dim 个线程，每个线程按式(2) 对向量的一个元素进行更新. 当 $mb > 1$ 时，每个线程需要迭代 mb 次，每轮迭代都考虑来自一个样本的贡献，对两个原型向量进行更新.

3　验　证

将本文中的异构原型学习算法在大规模、大类别手写汉字识别任务上进行测试，验证过程中主要对每类单个原型和每类 8 个原型的情况进行测试，TILE_LEN 的大小设置为 16. 算法测试的硬件平台采用统一的 GPU 服务器. 计算机配置有 Xeon X3440 服务器级 CPU，主频为 2.53 GHZ. 消费级显卡 GTX680 插在服务器的 PCI-E 插槽上. 当考虑算法在未来硬件上的适应性时，使用了更新的显卡 GTX980.

3.1　汉字样本库描述

本文使用的汉字数据库为 CASIA-HWDB1.0 (DB1.0)与 CASIA-HWDB1.1 (DB1.1)[15]，这是文献[2]中使用的数据库的子集，其包含 3 755 个类别(来自国标 GB1 字符集)，每个类约有 570 个训练样本. 这样共有 2 144 749 个训练样本，533 675 个测试样本. 在收集样本时，每一套字符集按照 6 种不同的排列次序预先打印在版面上，为的是抵消每个书写者在书写过程中的书写质量变化.

为了描述每个样本，提取 512 维的梯度特征作为特征向量. 随后使用线性判别分析(LDA)将特征向量从 512 维降到 160 维. 因此，本文中使用的变量 dim 大小为 160.

3.2　算法正确性验证

同时运行 CPU 版本的 GLVQ 算法与异构计算架构版本的 GLVQ 算法进行背对背测试，且均在训练数据上运行 40 轮. 每轮测试都计算出训练样本上的识别错误率以及测试样本的识别错误率. 实验中也对不同大小的 mb 进行了测试，图 4 与图 5 分别显示了整个学习过程以及对应的性能. 鉴于基于

CPU 的串行算法的训练过程过于耗时,图 5 中仅训练了 24 轮. 由图可知, 串行版本的 GLVQ 算法与异构计算架构版本的 GLVQ 算法的运行结果并无明显差异,这说明本文提出的适用于异构计算架构的大规模 GLVQ 算法具备正确性.

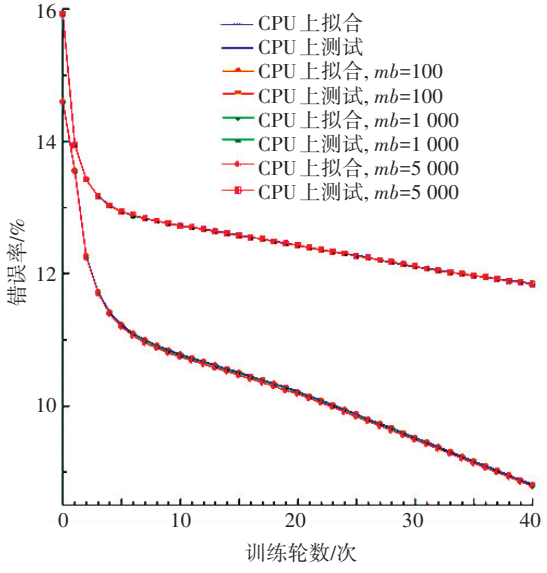

图 4 比较 CPU 版本和 GPU 异构版本的学习过程(每类单个原型)

Fig.4 Learning process comparison between CPU and GPU (single prototype/class)

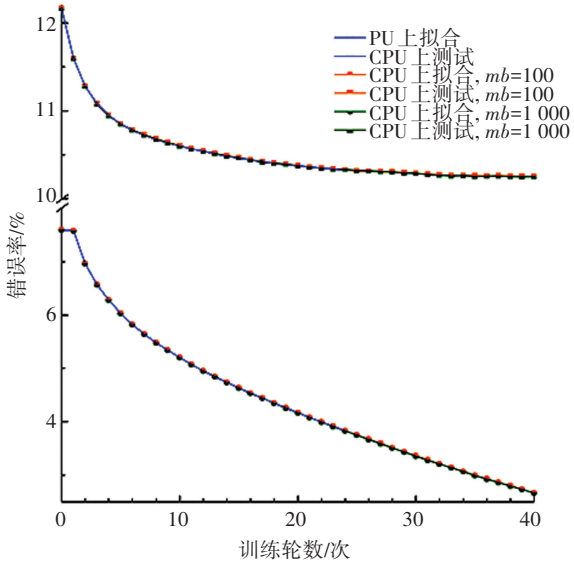

图 5 比较 CPU 和 GPU 的学习过程(每类八个原型)

Fig.5 Learning process comparison between CPU and GPU (eight prototypes/class)

3.3 算法中间过程评估

算法 1 中距离计算与最短距离搜索这两步的计算量最大. 首先对距离计算的效率进行评估. 实验中主要使用了归约和分块加和两种算法,并比较了不同 mb 大小时的运算效率. mb 的大小分别取 $\{2^0, 2^1, \cdots, 2^{13}\}$. 实验结果如图 6 与图 7 所示,实验只对训练数据进行了一轮测试. 由图可知,并行归约算法的执行时间比较稳定,在小批量的规模较小时,应优先使用并行归约算法. 而当 mb 的值大于或等于 TILE_LEN 时,分块加和算法则效率更高.

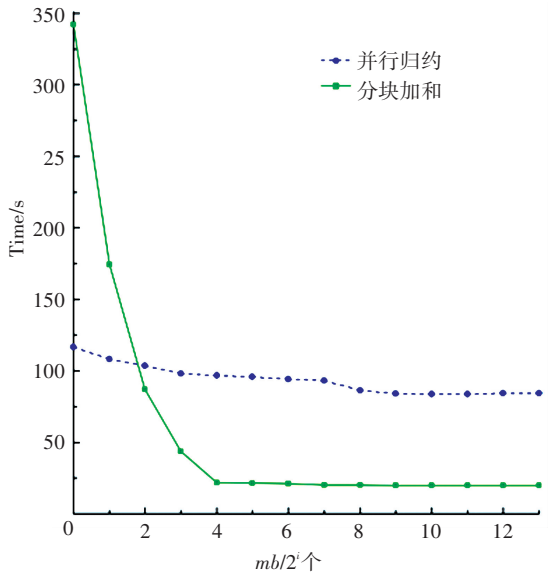

图 6 分别基于并行归约和分块加和计算距离的比较(每类单个原型)

Fig.6 Distance computation comparison between parallel reduction and tiling sum (single prototype/class)

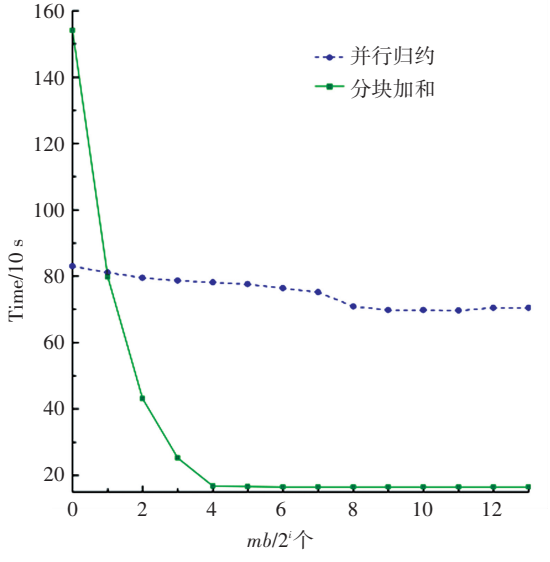

图 7 分别基于并行归约和分块加和计算距离的比较(每类八个原型)

Fig.7 Distance computation comparison between parallel reduction and tiling sum (eight prototypes/class)

对于最短距离搜索的实现,采用了并行归约和比较交换两种方案. 对这两种方案进行一轮学习测试,测试结果如图 8 与图 9 所示. 从图中可以看出,并行归约算法的可扩展性更强,针对不同的 mb 值算法的执行时间都比较稳定,即使当 mb 很大时,比较交换算法仍比并行归约慢很多. 另外,当一个类使用多个原型向量时,比较交换算法的效率就变得更差. 图中同样显示,当使用过大的 mb 时,会带来少量的额外消耗. 很重要的原因在于 mb 取值很大时,可能损害 GPU 系统缓存的效率.

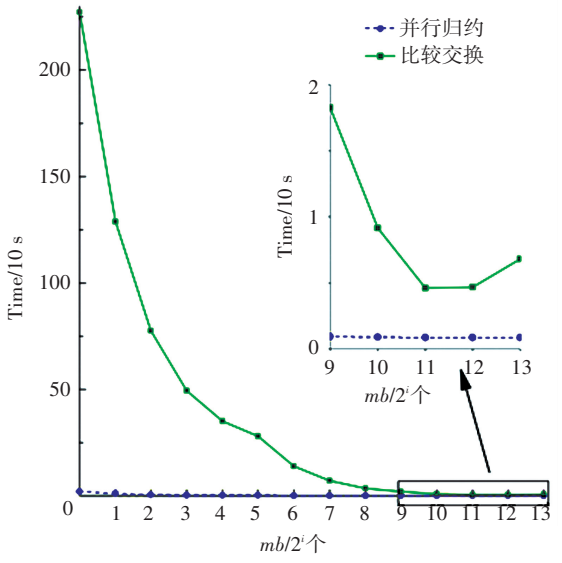

图 8　分别基于并行归约和比较交换策略求最小值的比较（每类单个原型）

Fig.8　Minima searching comparison between parallel reduction and compare-and-exchange (single prototype /class)

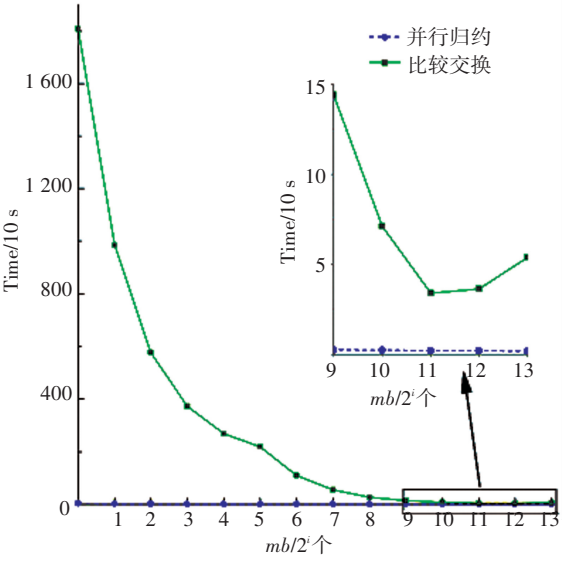

图 9　分别基于并行归约和比较交换策略求最小值的比较（每类八个原型）

Fig.9　Minima searching comparison between parallel reduction and compare-and-exchange (eight prototypes/class)

3.4　算法加速比评估

训练 GLVQ 模型的串行学习过程比较耗时。若使用单核 CPU 进行一轮学习，当每个类使用单个原型时需要花费 4 332 s，而每类使用 8 个原型则需要消耗 32 843 s. 若对 GLVQ 算法循环训练 40 轮，则需要花费数天时间.

基于本文提出的异构并行学习框架，开发了一个可根据计算负载规模自适应切换最优内核函数的 GLVQ 原型学习算法. 分别针对每类单个原型和每类 8 个原型的情况，收集数据并计算算法的加速比，如图 10 所示. 加速比的计算是采用串行执行时间除以并行执行时间. 可从图 10 看出，当使用本文所提出的异构原型学习算法时，每类使用单个原型时运行速度最高可加速 184 倍，而每类使用 8 原型则最高可加速 194 倍.

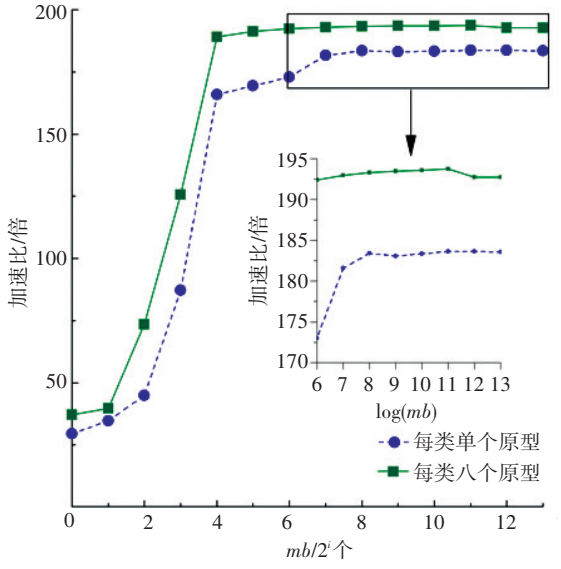

图 10　异构大规模原型学习相对 CPU 的加速比（分别考虑每类单个原型和每类八个原型）

Fig.10　Prototype learning speedup (with both single prototype/class and eight prototypes/class)

这里需要强调，并不是小批量规模越大，加速越高. 加速峰值均出现在小批量 mb = 2 048 个样本时，说明此时 GPU 的计算部件的利用率和访存效率之间经折衷后达到最优. 一旦继续加大小批量的规模，虽然可以加大 GPU 的占用率，但会对参加运算数据的局部性有所损害，从而导致 GPU 硬件二级缓存的命中率降低. 以每类单个原型为例，从 mb = 2 048增加小批量的规模到 mb = 8 196，GPU 的占用率从 98.08%增加为 98.13%，但却导致二级缓存的命中率从 91.0%恶化到 87.1%. 缓存命中率的下降，意味着访存效率的恶化，由于占用率的增加无法抵消访存恶化的负面效应，最终影响了每秒可以执行的浮点以及整型运算数量. 在本文的实验中，随着 mb 的进一步增大，加速比的下降幅度一般较小.

随机梯度下降模式（$mb = 2^0$）通常很难达到较高的加速比，这是由于受到可并行化的计算负载的局限. 当采用随机梯度下降时，本文中距离计算与最短距离搜索均采用并行归约的算法，以此实现细粒度并行. 最终，本文的方法在随机梯度下降模式下实现了最少 30 倍的加速比（每类单个原型时 30 倍，每类 8 个原型时 37 倍），这一成绩在文献上比较少见.

最后，讨论所提出框架在未来新 GPU 硬件上的扩展性和适应性. 当前，在消费级显卡行业出现了

计算能力高于 GTX680 的 GPU 硬件,GTX980 是典型代表. 本文的原型学习算法未做任何修改,重新运行在 GTX980 上并考察在新硬件上的性能表现. 当采用随机梯度下降模式时,两种原型分配方案下获得的加速比分别为 30 倍(每类单个原型)和 38 倍(每类 8 个原型),跟 GTX680 的加速比例相当. 但当采用小批量处理模式时,可以看到更优异的性能提升:每类分配单个原型时获得加速比为 493 倍,每类 8 个原型时的加速比为 638 倍. 这表明本文的异构并行学习框架对于未来的新硬件具有较强的自动扩展能力.

4 结 语

本文提出的适用于异构计算架构的大规模原型学习算法框架通过重组串行原型学习算法的计算任务,将串行学习算法转化为高度并行化的形式,可以将密集型计算负载转移到 GPU 上,而 CPU 只需进行少量的流程协调和数据传递. 为了充分利用 GPU 的资源,该算法框架可自动选择分块策略与并行归约策略. 算法的正确性和有效性均在大规模手写汉字识别任务上进行了验证. 在消费级显卡 GTX680 上使用小批量处理模式进行模型学习时,最高可得到 194 倍的加速比,即使是随机梯度下降模式下,也可实现 30 倍的加速比. 当升级显卡到 GTX980,在小批量下的加速比可提升到 638 倍,表明本文提出的框架和算法具有很好的可扩展性和适应性,能够有效解决原有原型学习的计算瓶颈问题.

参考文献

[1] LIU C, NAKAGAWA M. Evaluation of prototype learning algorithms for nearest-neighbor classifier in application to handwritten character recognition[J]. Pattern Recognition, 2001,34: 601-615.

[2] SU T, LIU C, ZHANG X. Perceptron learning of modified quadratic discriminant function[C]//International Conference on Document Analysis and Recognition. 2011:1007-1011.

[3] LV Y, HUANG L, et al. Learning-based candidate segmentation scoring for real-time recognition of online overlaid Chinese handwriting[C]//International Conference on Document Analysis and Recognition. 2013: 74-78.

[4] 苏统华,戴洪良,张健,等. 面向连续叠写的高精简中文手写识别方法研究[J],计算机科学, 2015, 42(7):300-304.
SU T, DAI H, et al. Study on High Compact Recognition Method for Continuously Overlaid Chinese Handwriting[J]. Computer Science, 2015, 42(7):300-304.

[5] SU T, MA P, et al. Exploring MPE/MWE training for Chinese handwriting recognition[C]//International Conference on Document Analysis and Recognition. 2013:1275-1279.

[6] ZHOU M, YIN F, LIU C. GPU-based fast training of discriminative learning quadratic discriminant function for handwritten Chinese character recognition[C]//International Conference on Document Analysis and Recognition, 2013:842-846.

[7] GELSINGER P. Microprocessors for the new millennium: Challenges, opportunities and new frontiers[C]//ISSCC Tech. Digest, 2001: 22-25.

[8] RAINA R, MADHAVAN A, NG A Y. Large-scale deep unsupervised learning using graphics processors[C]//International Conference on Machine Learning. 2009. 873-880.

[9] SCHERER D, SCHULZ H, BEHNKE S, Accelerating large-scale convolutional neural networks with parallel graphics multiprocessors [C]//International conference on Artificial neural networks. 2010: 82-91.

[10] CIRESAN DC, MEIER U, et al. Deep, big, simple neural nets for handwritten digit recognition[J]. Neural computation. 2010, 22: 3207-3220.

[11] JIN X, LIU C , HOU X. Regularized margin-based conditional log-likelihood loss for prototype learning[J]. Pattern Recognition, 2010, 43(7): 2428-2438.

[12] SATO A, YAMADA K. Generalized learning vector quantization [C]//Advances in Neural Information Processing Systems. 1996: 423-429.

[13] WILT N. CUDA 专家手册: GPU 编程权威指南[M]. 苏统华等,译. 北京: 机械工业出版社, 2014.
WILT N. The CUDA Handbook: A Comprehensive Guide to GPU Programming[M]. Addison-Wesley, 2013.

[14] KIRK DB, WENMEI WH.大规模并行处理器编程实战[M]. 赵开勇等,译. 第二版. 北京: 清华大学出版社, 2013.
KIRK DB, WENMEI WH. Programming massively parallel processors: a hands-on approach (Second Edition)[M]. Morgan Kaufmann, 2012.

[15] LIU C, YIN F, et al. Online and offline handwritten Chinese character recognition: Benchmarking on new databases[J]. Pattern Recognition, 2013,46(1): 155-162.

(编辑 王小唯 苗秀芝)