DOI:10.3876/j.issn.1000-1980.2018.05.013

基于GPU的并行拟牛顿神经网络训练算法设计

刘 强[1,2],李佳峻[1,2]

(1.天津大学微电子学院,天津 300072;2.天津市成像与感知微电子技术重点实验室,天津 300072)

摘要:针对人工神经网络训练需要极强的计算能力和高效的最优解搜寻方法的问题,提出基于GPU的BFGS拟牛顿神经网络训练算法的并行实现。该并行实现将BFGS算法划分为不同的功能模块,针对不同模块特点采用混合的数据并行模式,充分利用GPU的处理和存储资源,取得较好的加速效果。试验结果显示:在复杂的神经网络结构下,基于GPU的并行神经网络的训练速度相比于基于CPU的实现方法最高提升了80倍;在微波器件的建模测试中,基于GPU的并行神经网络的速度相比于NeuroModeler软件提升了430倍,训练误差在1%左右。

关键词:神经网络;GPU;并行计算;拟牛顿算法;OpenCL;加速算法

中图分类号:TP311.1 **文献标志码**:A **文章编号**:1000-1980(2018)05-0458-06

Parallel BFGS quasi-Newton algorithm of neural network training based on GPU

LIU Qiang[1,2], **LI Jiajun**[1,2]

(1. *School of Microelectronics*, *Tianjin University*, *Tianjin* 300072, *China*;
2. *Tianjin Key Laboratory of Imaging and Sensing Microelectronic Technology*, *Tianjin* 300072, *China*)

Abstract: One application challenge of the neural networks is the training, the essence of which is an iterative optimization process involving numerous data. The training process requires significant computing power and efficient searching methods for optimal solutions. To meet the requirement, a parallel BFGS quasi-Newton training algorithm based on GPU is proposed in this paper. To maximize the parallelism, the BFGS quasi-Newton algorithm is divided into different function modules, and each module is designed with a specific parallelism regarding its characteristics. In addition, the processing and memory resources of GPU are fully utilized to achieve a better parallelization. Experimental results show that the GPU implementation accelerates the neural network training by up to 80 times compared to the CPU implementation for the complicated neural network structure, while the speed up ratio is up to 430 for the modelling test of microwave device, where the training error is about 1%.

Key words: neural network; GPU; parallel computing; quasi-Newton method; OpenCL; accelerating algorithm

人工神经网络(ANN)是一种由大量神经元相互连接构成的运算模型。在这个模型中,神经元接收来自其他神经元传递过来的信号,这些信号通过带有权重的连接进行传递,然后通过激活函数产生神经元的输出[1]。近年来,神经网络结构经历了巨大发展,许多复杂的神经网络结构如卷积神经网络、反馈神经网络以及深度学习等相继出现,在很多领域都有大量应用,例如:生物医学[2]、自然语言处理[3]、能效预测[4]以及微波电路设计[5]。在训练之前,神经网络不携带任何信息;经过训练,神经网络的权重得以确定,从而建立精确的模型。神经网络的训练过程实际上是一个优化过程,即神经网络通过一些优化算法,比如梯度下降法、拟牛顿算法、粒子群优化算法以及共轭梯度算法等,经过反复迭代找到满足预期的权重。这一过程不仅迭代次数多,而且参与运算的数据量大,因此十分耗时。

GPU的大规模数据处理能力为这个问题的解决提供了有效途径。陈风[6]使用GPU加速PSO算法并对

基金项目:国家自然科学基金(61574099)

作者简介:刘强(1978—),男,副教授,博士,主要从事集成电路设计与神经网络应用研究。E-mail:qiangliu@tju.edu.cn

引用本文:刘强,李佳峻.基于GPU的并行拟牛顿神经网络训练算法设计[J].河海大学学报(自然科学版),2018,46(5):458-463.
LIU Qiang, LI Jiajun. Parallel BFGS quasi-Newton algorithm of neural network training based on GPU[J]. Journal of Hohai University (Natural Sciences),2018,46(5):458-463.

神经网络的训练进行优化。文中根据 PSO 算法的粒子来划分并行度,并基于并行归约技术提出一种分阶段求解粒子全局最优的方法,运行速度相比于 CPU 实现提升了 68.3 倍。李熙铭等[7]使用 GPU 加速共轭梯度法,16 个线程为一组,计算结果向量中的一个元素,并针对不同线程块中的线程使用原子操作进行同步,相比于 CPU 串行实现获得了 70 倍的加速。董学辉[8]使用 GPU 加速随机梯度下降法并用于逻辑回归算法。文中根据随机梯度下降法的样本数划分并行度,并行度较低,与 CPU 实现相比仅获得 17 倍的加速。韦向远等[9]使用 GPU 加速布谷鸟搜索算法,根据布谷鸟个体数据与个体中每一维的数据设计线程块与线程,对算法并行度进行了精细划分,相比于 CPU 实现方法获得了 110 倍的加速。夏俊峰等[10]将牛顿算法的雅可比矩阵及其逆矩阵计算进行了 GPU 并行实现,GPU 与 CPU 间存在大量通信,相比于 CPU 实现取得近 5 倍的加速。由上可见,算法的加速效果与并行度的划分有很大关系。文献[6-7,9]针对加速算法的结构特点以及硬件特点进行了较精细的并行度划分,而文献[8,10]由于并行度划分粗糙或者与算法结构契合度低等因素使得加速效果不明显。

BFGS 拟牛顿算法具有全局收敛性和超线性收敛速度,适合求解神经网络的最优权重。但是,由于拟牛顿算法中的多个计算阶段存在着数据依赖性,并且各个计算阶段的数据结构存在差异,并行度划分难度较大。现有的 GPU 加速方法如文献[6-9]并不能直接应用获得优化的结果。

为了实现精细的并行度划分和优化,笔者将 BFGS 拟牛顿神经网络训练算法划分为 5 个内核函数;依据每个内核函数的特点,采用混合模式的数据并行处理,提升并行效率;同时,最大限度地减小 GPU 中全局内存的使用,将重复使用的数据存放于本地内存或私有内存中,减少了工作项之间的通信成本。

1　BFGS 拟牛顿神经网络训练算法

1.1　神经网络训练误差评估函数

多层感知神经网络由输入层、隐层和输出层组成。本文所用神经网络为单隐层神经网络。相邻 2 层的神经元之间相互连接,每一个连接都有一个权重,权重最终会决定神经网络的输出。神经网络的训练过程就是不断减小与理想输出差异的过程。一个典型的神经网络误差评估函数是输出神经元的值与理想值的均方差函数:

$$E_{\mathrm{T}}(\boldsymbol{w})=\sqrt{\frac{1}{SN_{\mathrm{y}}}\sum_{i=1}^{S}\sum_{m=1}^{N_{\mathrm{y}}}\left|\frac{y_m(\boldsymbol{x}_i,\boldsymbol{w})-d_{im}}{d_{\max,m}-d_{\min,m}}\right|^2} \tag{1}$$

式中:$E_{\mathrm{T}}(\boldsymbol{w})$——训练误差;$S$——训练数据集的规模;$N_{\mathrm{y}}$——输出神经元个数;$y_m$——第 m 个输出神经元的计算结果;d_{im}——与第 m 个输出神经元对应的第 i 组输入数据的理想的输出结果;$\boldsymbol{x}_i$——第 i 组输入数据;$\boldsymbol{w}$——神经网络权重;$d_{\max,m}$——与第 m 个输出神经元对应的最大的理想输出,$d_{\min,m}$——与第 m 个输出神经元对应的最小的理想输出。

神经网络训练过程就是找到一组 $\boldsymbol{w}(\boldsymbol{w}\in\mathbf{R}^n,n$ 表示实数的维度)使得 E_{T} 小于一个预设的阈值。

1.2　BFGS 拟牛顿算法

BFGS 拟牛顿算法是一种通过计算海森矩阵的近似矩阵 $\boldsymbol{H}$ 来求解目标函数极小值的优化算法,其相对于牛顿法而言不必计算目标函数二阶导数,从而降低了时间复杂度。神经网络训练误差评估函数则作为 BFGS 拟牛顿算法的目标函数。

BFGS 拟牛顿算法的具体步骤:(a)初始化阶段。随机生成初始权重 $\boldsymbol{w}_0\in\mathbf{R}^n$,$\boldsymbol{g}_0=\Delta E_{\mathrm{T}}(\boldsymbol{w}_0)$,$\boldsymbol{I}=\mathrm{diag}(1,1,\cdots,1)_n$,设定 $\boldsymbol{H}_0=\boldsymbol{I}$、$\varepsilon=1\times10^{-5}$(阈值变量)、$k=0$(迭代次数);(b)计算搜索方向 $\boldsymbol{d}_k$;(c)使用黄金分割法缩小步长的取值范围,通过比较不同步长的训练误差,缩小步长区间,再通过多次迭代,确定最优步长 λ_k;(d)更新权重$\boldsymbol{w}_k$;(e)如果梯度 $\boldsymbol{g}$ 的范数小于预设阈值,将当前权重 $\boldsymbol{w}_k$ 输出并视为最优,否则更新矩阵 $\boldsymbol{H}_k$ 并开始新一轮的迭代。

2　并行 BFGS 拟牛顿神经网络训练算法设计流程

为保证实现的可移植性,使用 OpenCL 设计了并行 BFGS 拟牛顿神经网络训练算法。作为一个完整的并行编程框架,OpenCL 主要包含平台模型、内存模型、编程模型和执行模型。OpenCL 平台模型定义了宿主机

(如CPU)和计算设备(如GPU),在计算设备上执行的函数称为内核函数。其中,宿主机负责管理计算设备和计算任务的发起,计算设备则承担并行计算任务。OpenCL的编程模型定义如何将并发模型映射到具体物理硬件上。OpenCL的执行模型定义了OpenCL环境配置及内核函数的执行方式[11]。OpenCL中执行内核函数的最小单位为工作项,计算设备中的工作项通常被划分为许多个工作组。OpenCL内存模型主要包括全局内存、本地内存和私有内存。全局内存可被同一个计算设备中的所有工作项读写,其存储容量较大但带宽较小,用于存储多个内核函数需要用到的数据以及与宿主机进行交互的数据。本地内存可被同一个工作组中的工作项读写,访问速度较快,用于存储在计算过程中反复用到的数据。私有内存为工作项专属内存,其他工作项不可访问,只能在内核函数内部分配使用,带宽较大但容量较小,通常只用于存放中间变量。常量内存可被所有工作项读写,由宿主机进行初始化,执行内核函数期间不可被更改。

将BFGS拟牛顿算法分为5个独立内核函数(kernel1 ~ kernel5),针对每个内核函数的特点分别进行不同程度的并行度划分,在提高并行度与减少工作项的通信之间取得平衡,使算法整体运行效率最大。

采用2种数据并行模式:(a)基于训练数据组数的并行数据模式;(b)基于神经网络权重维数的数据并行模式。如图1所示,计算训练误差的内核函数与计算梯度的内核函数均将工作项数目定为S;其他内核函数将工作项数目定为K(K为神经网络权重$\boldsymbol{w}$的维数)。

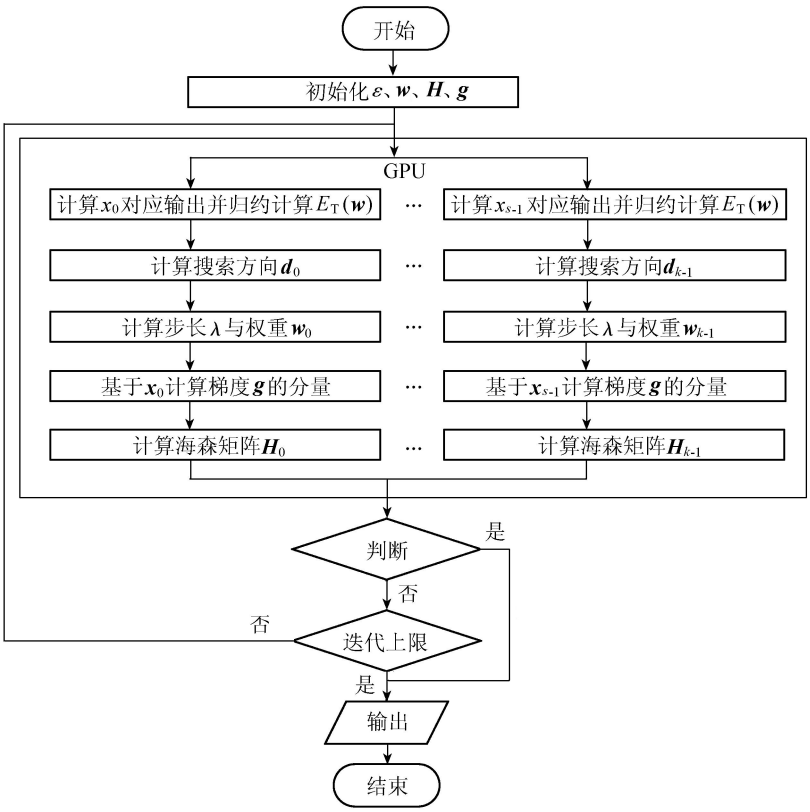

图1 BFGS拟牛顿神经网络训练并行算法流程

Fig. 1 Flow chart of the parallel BFGS quasi-Newton ANN training algorithm

将一个工作组中工作项的数量设置为K(当K不为2的指数幂时,将工作组内工作项的数量补至最小的大于K的2的指数幂),以便于使用并行归约计算梯度$\boldsymbol{g}$的范数。梯度$\boldsymbol{g}$是一个K维的向量,在进行归约时,可以将$\boldsymbol{g}$中的所有元素映射到一个工作组内的工作项中,并通过读写本地内存提高归约效率。

这种混合并行度的划分虽然增加了设计难度,但是带来了较高的运行效率。如果将所有内核函数按照维数K划分并行度,一个工作项中迭代计算次数会为S/K,这样会使单个工作项的计算量变大,计算时间变长;而如果将所有内核函数按照S划分并行度,GPU会产生大量闲置工作项,从而浪费计算资源。

2.1 神经网络训练误差评估函数$E_T(\boldsymbol{w})$

将$E_T(\boldsymbol{w})$设计为单独的内核函数包括两方面原因:(a)该部分并行度划分与其他部分的并行度划分存在一定差异;(b)该函数会被计算步长的函数反复调用。

计算 $E_T(\boldsymbol{w})$时,其过程往往需要逐层计算各个神经元的值,最后对所有输出神经元的值进行处理。为了降低神经网络训练误差,往往需要大量的训练数据,所以本文对训练数据进行分组计算,即每一个工作项进行一组训练数据由输入神经元到输出神经元的计算。第 s 个工作项的详细计算内容如式(2)所示,输入神经元的值即为训练数据,用 $x_{i,s}$表示,即与第 i 个输入神经元对应的第 s 组训练数据的值。

$$y_m = \sum_{j=1}^{N} f(h_j) w_{mj} \tag{2}$$

其中

$$f(h_j) = \frac{1}{e^{-h_j} + 1} \qquad h_j = \sum_{i=1}^{n} x_{i,s} w_{ij}$$

式中:$f(h_j)$——第 j 个隐层神经元的值;w_{ij}——第 i 个输入神经元到第 j 个隐层神经元的权重;n——输入神经元数量;N——隐层神经元的数量。

将所有工作项运算得到的输出神经元的值进行归约求和,然后根据式(1)计算得到 $E_T(\boldsymbol{w})$的训练误差。由于整体计算任务划分在各个工作项中并行完成,随着训练数据增加,只需相应地增加工作项。如此,即便训练数据增加,运算时间也不会有明显增加,相比于串行计算,计算速度得到明显提升。该函数产生的 $f(h_j)$、y_m 以及训练误差均存于全局内容,以便其他内核函数使用。其他中间变量存于私有内存以提高读写效率。

2.2　计算搜索方向 $\boldsymbol{d}$

该部分计算海森矩阵 $\boldsymbol{H}$ 与梯度 $\boldsymbol{g}$ 的乘积。由于被其他函数使用,乘积结果存于全局内存中。矩阵 $\boldsymbol{H}$ 是 K 阶方阵,梯度 $\boldsymbol{g}$ 的维度与权重维度相同即为 K 维。因此,在这个内核函数中每一个工作项读取矩阵 $\boldsymbol{H}$ 的一行数据与梯度 $\boldsymbol{g}$ 进行乘累加运算产生搜索方向 $\boldsymbol{d}$ 的一个元素,共有 K 个工作项参与搜索方向的计算。

2.3　黄金分割法计算步长 λ 并更新权重 $\boldsymbol{w}$

黄金分割法计算步长的流程为:首先初始化一个步长区间$[a,b]$并计算该段区间的 2 个黄金分割点 c_1 和 c_2,再以 c_1 和 c_2 到端点 a 的距离为步长根据 BFGS 拟牛顿算法更新权重 $\boldsymbol{w}$,然后调用 $E_T(\boldsymbol{w})$分别求 2 组权重对应的训练误差并比较大小,留下较小的训练误差对应的步长,并删除较大的误差所对应的点与距其最近的端点的一段区间。如此反复,直到剩下的步长区间小于一个预设阈值,停止迭代并确定步长 λ[12]。由于这个内核函数主要目的是更新权重 $\boldsymbol{w}$,函数内部并行度的划分基于 $\boldsymbol{w}$ 的维数进行。即每个工作项计算权重 $\boldsymbol{w}$ 的一个元素。

该部分要反复调用 $E_T(\boldsymbol{w})$,即需要在一个内核函数内部调用另一个内核函数。由于 2 个内核函数具有不同的并行度,直接调用 $E_T(\boldsymbol{w})$会导致工作项冲突。为了避免这一问题,在每次调用 $E_T(\boldsymbol{w})$之前,要对所有工作项强制进行同步操作。

2.4　计算梯度 $\boldsymbol{g}$

BFGS 拟牛顿算法是一个基于梯度下降的最优化方法,所以梯度的计算对于 BFGS 拟牛顿算法的实现至关重要。传统的梯度计算方法一般是基于求极限的方法。这种方法不仅需要反复调用目标函数,而且算法精度受限于自变量增量的选取。在并行算法中,每一次目标函数梯度的计算都需要大量工作项参与,若使用传统方法计算梯度会消耗大量的计算资源,而且无法保证高精度。所以,Domingos 等[13]提出梯度计算方法,该方法基于传统的数学求导方法,因此该函数的并行度应与目标函数保持一致,即根据训练数据组数划分并行度,每个工作项计算与一组训练数据对应的权重的偏导数,最后通过并行归约求和求得目标函数的偏导数。

2.5　计算海森矩阵 $\boldsymbol{H}$

计算海森矩阵 $\boldsymbol{H}$ 用到的变量 $\boldsymbol{s}_k$ 和 $\boldsymbol{z}_k$($\boldsymbol{s}_k = \boldsymbol{w}_{k+1} - \boldsymbol{w}_k$,$\boldsymbol{z}_k = \boldsymbol{g}_{k+1} - \boldsymbol{g}_k$)分别由 kernel3 与 kernel4 计算并存于全局内存中,随后在该部分中将这 2 个变量由全局内存传入本地内存,以便在计算过程中反复调用。一些中间变量由 $\boldsymbol{s}_k$ 和 $\boldsymbol{z}_k$ 计算得来且只在计算海森矩阵 $\boldsymbol{H}$ 的内核函数中使用,因此存于本地内存中。每个工作项输出海森矩阵 $\boldsymbol{H}$ 一列的元素。

3　实 验 分 析

通过 2 组实验来分别测试本文并行实现的性能和实际应用效果,并与 CPU 的实现方法进行对比。实验使用的计算设备相同,CPU 为 Intel Core i5-4590,主频为 3.3 GHz;GPU 为 NVIDIA GT630,其流处理器数量为

384,频率为902 MHz。

3.1 算法性能测试

第一组实验分别测试在不同训练数据规模和不同神经网络规模下基于CPU、GPU及FPGA[14]的拟牛顿实现方法的单次迭代的平均运行时间。

3.1.1 不同训练数据规模下算法性能测试

实验中神经网络结构包含3个输入神经元、8个隐神经元和1个输出神经元。训练数据规模由32组依次翻倍到512组。图2为不同实现方法在不同规模的训练数据下单次迭代的平均运行时间。由图2可见,CPU和FPGA实现方法的平均单次迭代运行时间随着训练数据增加趋向于线性增长,因此随着训练数据的增加,基于CPU和FPGA的拟牛顿工作量都会随之增加。基于GPU的拟牛顿算法中,训练数据的增加对应的是工作项的增加,在不超出GPU峰值运算量的前提下,运算时间趋向于保持不变。在训练数据规模较小时,GPU的实现方法平均单次迭代运行时间较长,当训练数据增加到将近100组时,CPU的实现方法的运行时间开始超过GPU。当训练数据为512组时,基于GPU的训练时间缩短为CPU的约1/6。可以预见,训练数据规模继续增大,FPGA的实现方法的运行时间也会超过GPU。

3.1.2 不同神经网络结构下算法性能测试

本实验中训练数据规模为512组,测试的神经网络结构如图3所示,神经网络权重数根据不同神经网络结构依次由32~512逐渐增大。权重数在32~512之间的神经网络结构适用于大多数实际应用。由图3可见,CPU和FPGA的实现方法随神经网络结构变化趋向于线性增长,并且增长趋势较明显,而GPU的实现方法随神经网络结构变化的趋势较小。随着神经元数量增加,GPU的实现方法的运行时间与CPU的实现方法的运行时间的差距在逐渐增大,而FPGA的实现方法运行时间变化趋势与CPU的实现方法的运行时间变化趋势趋近于平行。当神经网络权重数增加到128时,FPGA实现方法的运行时间开始超过GPU的运行时间。对于测试的规模最大的神经网络结构,GPU的实现方法与CPU的实现方法相比获得了约80倍的加速,而与FPGA实现方法相比获得了近5倍的加速。

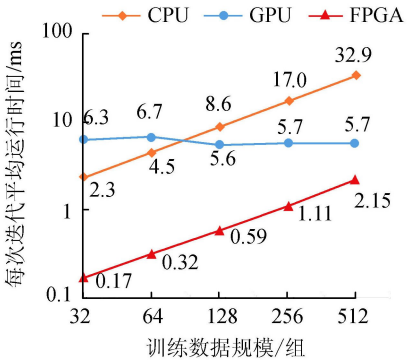

图2 不同训练数据规模下3种实现方法性能对比

Fig. 2 Comparison of three implementations under different data sizes

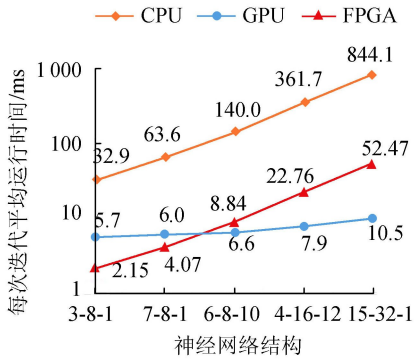

图3 不同神经网络结构下3种实现方法性能对比

Fig. 3 Comparison of three implementations under different Neural Network structures

3.2 微波器件建模实验

通过对文献[15]中2个微波结构的电磁行为进行建模并使用实际的训练数据进行训练来测试本文实现方法的性能和训练误差。为了与现有稳定的方法对比,使用了成熟的神经网络微波建模工具NeuroModeler[16]。该工具采用BFGS拟牛顿算法对神经网络进行训练。实验所用训练数据来自CTS Studio Suite 2014器件仿真。2种方法的训练误差差异在0.1%以下,产生的可能原因如下:(a)BFGS拟牛顿算法的初始权重随机产生,为了减少随机初始值的影响,实验结果是经过多次测试取平均值计算得到;(b)CPU和GPU浮点运算单元不同,在同样输入下,二者获得的结果可能会有一些差异。

3.2.1 超宽带天线模型的测试

超宽带天线电磁行为模型有6个输入变量、2个输出变量,训练数据规模为8 192组,神经网络模型中含有30个隐神经元,训练迭代次数为2 000次。实验结果表明,本文方法的运行速度(31.90 s)比NeuroModeler实现方法(6 240.00 s)快了200倍,同时获得了相似的训练误差,都在2.4%以下。

3.2.2　微波结模型的测试

微波结的电磁行为模型有 8 个输入变量和 8 个输出变量,训练数据规模同样是 8 192 组,神经网络中含有 15 个隐神经元。理想情况下,根据各个内核函数在算法整体运行时间所占比例及其并行度计算,相比于基于 CPU 的串行实现能获得的理想加速比约 1 000。但各个工作项、工作组和计算设备之间的通信,以及 GPU 中处理单元的流水线的执行模式,再加上 GPU 中单个处理单元的计算能力远低于 CPU 核的计算能力,会使实际加速比与理想加速比之间存在一定差距。结果表明,在经过 2 000 次迭代后,NeuroModeler 的运行时间为 10 140 s,本文方法的运行时间为 23.17 s,本文方法的速度是 NeuroModeler 的 430 倍,而训练误差都在 1% 左右。

4　结　　语

提出了一种基于 GPU 的拟牛顿算法并行实现方法,并将其应用于神经网络的训练。通过划分功能模块,并针对不同功能模块的特点采用混合数据并行模式,获得较好的并行效率,并且通过合理利用 GPU 内存资源,获得了较好的加速效果。实验结果显示,本文提出的基于 GPU 的算法在实际应用中最小能够获得近 1% 的训练误差,并且最大可实现近 430 倍的加速,可以有效缩短神经网络模型的开发周期,拓展新的应用。

参考文献:

[1] 周志华. 机器学习[M]. 北京:清华大学出版社, 2016.

[2] SAN P P, LING S H, NURYANI, et al. Evolvable rough-block-based neural network and its biomedical application to hypoglycemia detection system[J]. IEEE Transactions on Cybernetics,2014,44(8):1338-1349.

[3] SUNDERMEYER M,NEY H,SCHLÜTER R. From feedforward to recurrent LSTM neural networks for language modeling[J]. IEEE/ACM Transactions on Audio Speech & Language Processing,2015,23(3):517-529.

[4] LIN Chewei,YANG Yating,WANG Jeenshing,et al. A wearable sensor module with a neural-network-based activity classification algorithm for daily energy expenditure estimation[J]. IEEE Transactions on Information Technology in Biomedicine,2012,16(5):991-998.

[5] KABIR Humayun,WANG Ying,YU Ming,et al. High-dimensional neural-network technique and applications to microwave filter modeling[J]. IEEE Transactions on Microwave Theory & Techniques,2010,58(1):145-156.

[6] 陈风. 基于 GPU 的粒子群神经网络研究与应用[D]. 镇江:江苏科技大学,2015.

[7] 李熙铭,欧阳丹彤,白洪涛. 基于 GPU 的混合精度平方根共轭梯度算法[J]. 仪器仪表学报,2012,33(1):97-104. (LI Ximing,OUYANG Dantong,BAI Hongtao. Mixed precision CGS algorithm based on GPU[J]. Chinese Journal of Scientific Instrument,2012,33(1):97-104. (in Chinese))

[8] 董学辉. 逻辑回归算法及其 GPU 并行实现研究[D]. 哈尔滨:哈尔滨工业大学,2016.

[9] 韦向远,杨辉华,谢谱模. 基于 CUDA 的并行布谷鸟搜索算法设计与实现[J]. 计算机科学与探索,2014,8(6):665-673. (WEI Xiangyuan,YANG Huihua,XIE Pumo. Research and implementation of parallel cuckoo search based on CUDA[J]. Journal of Frontiers of Computer Science and Technology,2014,8(6):665-673. (in Chinese))

[10] 夏俊峰,杨帆,李静,等. 基于 GPU 的电力系统并行潮流计算的实现[J]. 电力系统保护与控制,2010,38(18):100-103. (XIA Junfeng,YANG Fan,LI Jing,et al. Implementation of parallel power flow calculation based on GPU[J]. Power System Protection and Control,2010,38(18):100-103. (in Chinese))

[11] MUNSHI A. OpenCL 编程指南[M]. 北京:科学出版社, 2012.

[12] PRESS W H,TEUKOLSKY S A,VETTERLING W T,et al. Numerical recipes 3rd edition: the art of scientific computing[M]. New York: Cambridge University Press,2007.

[13] DOMINGOS P O,SILVA F M,NETO H C. An efficient and scalable architecture for neural networks with backpropagation learning[C]//Anon. International Conference on Field Programmable Logic and Applications. New York:IEEE,2005:89-94.

[14] LIU Qiang, SANG Ruoyu, ZHANG Qijun. FPGA-based acceleration of davidon-fletcher-powell quasi-Newton optimization method[J]. Transactions of Tianjin University, 2016, 22(5):381-387.

[15] FENG Feng,ZHANG Chao,MA Jianguo,et al. Parametric modeling of EM behavior of microwave components using combined neural networks and pole-residue-based transfer functions[J]. IEEE Transactions on Microwave Theory & Techniques,2016,64(1):60-77.

[16] ZHANG Qijun. Neuromodeler[EB/OL]. (2010)[2017-11-07]. http://www.doe.carleton.ca/~qjz/.

(收稿日期:2017-11-07　编辑:刘晓艳)