神经网络前向传播在 GPU 上的实现*

刘进锋,郭 雷
(西北工业大学 自动化学院,陕西 西安 710129)

摘 要:基于 CUDA 架构在 GPU 上实现了神经网络前向传播算法,该算法利用神经网络各层内神经元计算的并行性,每层使用一个 Kernel 函数来并行计算该层神经元的值,每个 Kernel 函数都根据神经网络的特性和 CUDA 架构的特点进行优化。实验表明,该算法比普通的 CPU 上的算法快了约 7 倍。研究结果对于提高神经网络的运算速度以及 CUDA 的适用场合都有参考价值。

关键词:神经网络;CUDA;GPU
中图分类号:TP183 **文献标识码**:A **文章编号**:1674-7720(2011)18-0069-03

A forward propagation implementation of neural network on GPU

Liu Jinfeng, Guo Lei
(School of Automation, Northwestern Polytechnical University, Xi'an 710129, China)

Abstract: A neural network forward propagation algorithm based on CUDA is implemented. This algorithm utilizes the parallelism of each layer, uses a kernel function to handle the parallelism computation of the neuron values of that level, and each kernel function is optimized according the characteristic of network and CUDA architecture. Experiment shows that this method can achieve up to 7 times of speedup over the ordinary method on CPU. This research has reference value for both increasing neural network computing speed and suitability of CUDA.

Key words: neural network; CUDA; GPU

GPU(图形处理器)是处理计算机图形的专用设备。近十年来,由于高清晰度复杂图形实时处理的需求,GPU 发展成为高并行度、多线程、多核的处理器。目前主流 GPU 的运算能力已超过主流通用 CPU,从发展趋势上看将来差距会越来越大。GPU 卓越的性能对开发 GPGPU(使用 GPU 进行通用计算)非常具有吸引力。最初 GPGPU 需要将非图形应用映射为图形应用的方式,这个处理过程与图形硬件紧密相关,程序实现非常困难。近年来,GPU 的主要供应商 NVIDIA 提出了新的 GPGPU 模型,称为 CUDA(统一计算设备架构)[1]。

CUDA 是一种在 NVIDIA 公司的 GPU 上进行计算的新型的硬件和软件架构,可以将 GPU 视为一个并行数据计算的设备,对所进行的计算给予分配和管理。在 CUDA 的架构中,这些计算不再像过去的 GPGPU 架构那样必须将计算映射到图形 API 中,开发者不需要去学习艰涩的显示芯片的指令或是特殊的结构,因此开发门槛大大降低。CUDA 的 GPU 编程语言基于标准的 C 语言,开发 CUDA 的应用程序较为方便。

在 CUDA 架构下,一个程序分为两个部分:host 端和 device 端。host 端是指在 CPU 上执行的部分,而 device 端则是在 GPU 上执行的部分。device 端的程序又称为 Kernel。通常 host 端程序会将数据准备好后,复制到显卡的内存中,再由 GPU 执行 device 端程序,完成后再由 host 端程序将结果从显卡的内存中取回。

在 CUDA 架构下,GPU 执行时的最小单位是线程(thread)。几个线程可以组成一个线程块(block),每个线程都有自己的一份寄存器(register)和本地内存(local memory)空间。同一个线程块中的每个线程则可以共享一份共享内存(shared memory)。此外,所有的线程都共享一份全局内存(global memory)、常量内存(constant memory)和纹理内存(texture memory)。

自从 CUDA 发布以来,大量应用问题使用 CUDA 技

* 基金项目:国家自然科学基金资助项目(61063020)

术取得了令人惊奇的效果，加速十几倍甚至上百倍的实例很多。参考文献[2]对这些应用有一个综合介绍。

最适合利用 CUDA 处理的问题，是可以大量并行化的问题，这样能有效利用显示芯片上的大量执行单元。使用 CUDA 时，上千个线程同时执行是很正常的。如果问题不能大量并行化，使用 CUDA 就不能达到最好的效率。参考文献[3]通过许多成功应用 CUDA 加速的实例探讨了 CUDA 的高效实现策略及应用范围问题。

1 本文使用的神经网络简介

本文实验选用的是一个较为复杂的用于手写汉字识别的神经网络[4-6]。该网络是一种五层卷积神经网络，结构如图 1 所示。选择这样一个神经网络基于两点考虑：其一，过于简单的神经网络结构不易体现 GPU 计算的优势；其二，此网络的结构比较全面，在它上面实现的算法有一定的代表性，做一些修改就可以用在其他不同的神经网络结构上。

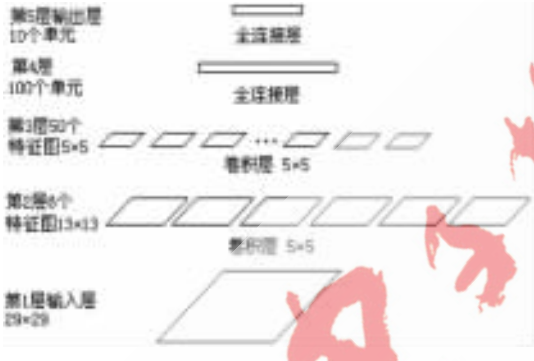

图 1 本文使用的卷积神经网络结构

该神经网络第一层输入层是手写数字的灰度图像，图像大小为 29×29 像素，这样输入层有 29×29=841 个神经元。第二层是卷积层，由 6 个特征图组成，每个特征图的尺寸是 13×13 个神经元，每个神经元的值是由输入图像的 13×13 个位置(输入图像水平和垂直方向的 29 个位置间隔选择 13 个)上作 5×5 的卷积得到。一个特征图的卷积核相同，各特征图的卷积核不同。这样第二层有 13×13×6=1014 个神经元，有(5×5+1)×6=156 个权值(每个 5×5 多加一个偏向权值，后面各层中权值数量计算公式中+1 都是加一个偏向权值)。另外，因为 1 014 个神经元每个都有 26 个与输入层的连接，所以从输入层到第二层共有 1014×26=26364 个连接。这里就能看出卷积神经网络共享权值的好处了：虽然有 26 364 个连接，但只需要 156 个权值来控制。第三层也是一个卷积层，它有 50 个特征图构成，每个特征图是 5×5 大小，是从第二层的 6 个 13×13 的特征图的相应区域作 5×5 卷积得到。这样第三层就有 5×5×50=1250 个神经元，(5×5+1)×6×50=7800 个权值，1250×26=32500 个与第二层的连接。第四层是有 100 个神经元的全连接层，因为是全连接，本层的每个神经元都与第三层的全部 1 250 个神经元连接，这样所有的连接就有 100×(1250+1)=125100 个，有 125 100 个权值。第五层是输出层，该层有 10 个全连接的单元，每个单元都与第四层的所有 100 个神经元连接，这样本层就有 10×(100+1)=1010 个连接，有 1010 个权值。这 10 个神经元对应 10 个数字，其中对应于识别结果的一个神经元的输出值为+1，其他 9 个神经元的输出值为-1。

整个网络总共有 3 215 个神经元，134 066 个权值，184 974 个连接。输入层神经元数据和各层神经网络的权值已知。

2 算法的实现

神经网络前向传播算法可以表示为：

$$x_n^i = F\left(\sum_{j=0}^{C_{n-1}} \omega_n^{ij} \times x_{n-1}^j\right)$$

其中，x_n^i 表示第 n 层的第 i 个神经元的值，ω_n^{ij} 表示第 n 层的第 i 个神经元与第 $n-1$ 层的第 j 个神经元的连接权值。$F()$函数是激活函数，本神经网络使用的激活函数是双曲正切函数。

GPU 上神经网络前向传播算法基本过程是逐层计算各层的所有神经元的值。

输入层神经元值已知，其余每层有一个 Kernel 函数来计算该层的所有神经元的值，上述的神经网络需要 4 个 Kernel 函数。并行计算只能体现在一层中，不同层之间没有并行性。

首先将输入层的神经元值和每层的权值保存在 5 个数组中，并从 host 内存传递到 device 内存。由于每层的权值是不变的，所以可以将这些权值传递到 device 的常量内存中，由于常量内存有 cache，这比放到全局内存的存取速度要快很多。在 device 中为第二到第五层的神经元值分配内存空间，第一个 Kernel 函数根据输入层的神经元值和权值计算第二层神经元值，第二个 Kernel 函数根据第二层的神经元值和权值计算第三层神经元值，如此往下，第四个 Kernel 函数计算出第五层即输出层的值，然后将该值从 device 内存传递到 host 内存。神经网络的连接体现在每个 Kernel 函数处理计算过程里。

计算第二层神经元值的基本 Kernel 函数的伪代码如下：

```
1：bid=blockIdx.x;
2：tx=threadIdx.x;  ty=threadIdx.y;
3：wt=ty*2*29+tx*2;  result=0;
4：templet[25]={  0,  1,  2,  3,  4,
                 29, 30, 31, 32, 33,
                 58, 59, 60, 61, 62,
                 87, 88, 89, 90, 91,
                 116,117,118,119,120};
5：for(i=0;i<25;++i)
6：  result+=Gn1[wt+templet[i]]*Gw1[bid*26+i+1];
7：result=(1.7159*tanhf(0.66666667*result));
```

```
8:Gn2[13*13*bid+ty*13+tx]=result;
```

该函数的输入为第一层的神经元值数组*Gn*1和权值数组*Gw*1，输出为第二层的神经元值数组*Gn*2。

因为第二层由6个特征图构成，每个特征图有13×13个神经元，所以这个函数在host端调用时，线程块参数设置为6，线程参数设置为13×13，这样一个线程块处理一个特征图，每个神经元值由一个线程处理，由第一层中取25个神经元乘以权值再加上一个偏向权值的累加得到。

这个函数的第1、2行得到线程块号和线程号，第4行的templet数组是为了在输入层中选择25个值的模版。第5、6行的循环计算出激活值，第7行对该值进行激活运算，第8行将结果送到第二层的神经元值数组*Gn*2。

上述函数还可以修改的效率更高些，第4行的templet数组是默认分配在全局内存中的，而访问全局内存需要相对较长的时间，因此有必要消除这种访存时间消耗。通过将第5、6行的for循环展开，templet[*i*]的值直接写在代码里，就既可以在程序中不要templet数组，又减少了循环判断的时间代价。

对函数的4~6行进行修改，其他行不变。改变了的部分代码如下：

```
4:__shared__ float s1[29*29];
5:                          //将Gn1数组元素送到s1数组;
6:result=s1[wt]*Gw1[bid*26+1]+
         s1[wt+1]*Gw1[bid*26+2]+
         s1[wt+2]*Gw1[bid*26+3]+…
                            //其他省略，一共有25个乘累加
```

第4行定义了一个在共享内存中的数组*s*1。将*Gn*1数组中的数据先传到*s*1中，第6行就不再需要引用*Gn*1数组，而是使用数组*s*1。由于存取共享内存的速度比存取全局内存快得多，这种改进进一步提高了速度。

计算第三层神经元值的Kernel函数，基本结构与上述的函数类似。也可以使用类似处理方式进一步提高速度。该函数在host端调用时，线程块数参数设置为50，线程参数设置为5×5。一个线程块处理一个特征图，每个神经元值由一个线程处理。

第三层到第四层是全连接层，计算第四层的Kernel函数可以有两种实现方案。方案1：由于第四层有100个神经元，可以设计成有100个线程块，每个线程块只有一个线程，一个线程计算一个神经元的值。这样一个线程计算量还是很大的，大约需要作1 250个乘法和加法。另外，每个线程块只有一个线程，无法充分利用CUDA的Warps切换能力[1]。这种方案没有完全发挥GPU的能力，因而效率不太高。方案2：还是有100个线程块，每个线程块有250个线程，250个线程计算一个神经元的值，每个线程作5次乘法和加法，最后由第一个线程将这些和累加，得到最终的值。这种方案效率较高。

计算第五层的神经元值需要约1 010次浮点乘法和加法，实验证明该层由于计算量相对较小，体现不出在Kernel端计算的优势，计算时间甚至比在host端更长。

3 实验结果

本试验使用机器的CPU是Intel Core 2 Duo E7400，时钟频率为2.8 GHz，内存为1 GB。GPU是NVIDIA GeForce GTX9800+，该GPU有128个频率为1.836 GHz的流处理器，分为16个SM，512 MB显存。

GPU上算法实现采用上文描述的最优化的方案，即每层的权值放入device端的常量内存，函数中的循环展开，利用共享内存，计算第四层神经元值时采用方案2。

试验结果如表1所示，分别给出了使用CPU和GPU计算本文神经网络前向传播时各层所用时间对比，这是经过三次试验求得的平均值。GPU实现采用最优化的方案，CPU上的实现方法就是通常的神经网络前向传播算法，这里不再赘述。CPU上计算整个神经网络前向传播需要1.851 ms，GPU计算需要0.272 ms，比CPU上计算快了约7倍。

表1 试验结果

	计算各层所用时间			
	第二层	第三层	第四层	第五层
CPU时间/ms	0.287	0.985	0.567	0.012
GPU时间/ms	0.034	0.065	0.160	0.013

本文对神经网络的前向传播在GPU上计算进行了研究，得到了比较满意的结果。使用本方法需要注意以下几个方面：

(1)GPU算法前期处理时要把一些数据从CPU装入GPU，还需要在GPU上给一些数据分配内存空间，这些前期处理都要花费时间，所以如果只做一次神经网络前向传播运算，GPU上的实现可能比CPU上的还要慢，但由于前期处理只做一次，所以多次神经网络前向传播运算能体现出GPU运算的好处。这种情况可能出现在多个测试数据的验证或计算神经网络后向传播时。

(2)当神经网络规模较小时，可能GPU方法会比CPU方法慢，比如本网络的最后一层的规模。

(3)有些应用问题在GPU上计算时速度能提高上百倍[3]，而神经网络在GPU上运算速度提高有限，这是因为神经网络的并行性主要体现在同一层，总体并行性不太高。要想充分发挥GPU的运算能力，那些应用问题应该具有计算复杂度高、数据传输量少的特点，或者能修改原来的算法，使得计算-内存访问比提高。

参考文献

[1] NVIDIA Corporation.NVIDIA CUDA programming Guide Version 2.2[EB/OL],2009-04.http://developer.nvidia.com/cuda.

[2] RYOO S,RODRIGUES C I,BAGHSORKHI S S,et al. Optimization principles and application performance evalua-

tion of a multithreaded GPU using CUDA[J].In Proceedings of ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 2008:73-82.

[3] HWU W M, RODRIGUES C, RYOO S, et al.Compute unified device architecture application suitability[J].Computing in Science and Engineering, 2009, 11(3):16-26.

[4] LECUN Y, BOTTOU L, BENGIO Y, et al.Gradient-based learning applied to document recognition[J].Proceedings of the IEEE, 1998, 86(11):2278-2324.

[5] SIMARD P Y, STEINKRAUS D, PLATT J.Best practices for convolutional neural networks applied to visual document analysis[C].International Conference on Document Analysis and Recognition(ICDAR), IEEE Computer Society, Los Alamitos, 2003:958-962.

[6] Mike O'Neill.Neural network for recognition of handwritten digits[EB/OL], 2006-12.http://www.codeproject.com/KB/library/NeuralNetRecognition.aspx.

(收稿日期:2011-06-24)

作者简介:

刘进锋,男,1971年生,博士研究生,副教授,主要研究方向:图像处理,GPU通用计算。

郭雷,男,1956年生,博士生导师,教授,主要研究方向:图像处理,模式识别。