基于CUDA的SVM算法并行化研究

张 巍[1,2] 张功萱[1] 王永利[1] 张永平[1] 朱昭萌[1]
(南京理工大学计算机科学与技术学院 南京 210094)[1]
(淮阴师范学院计算机科学与技术学院 淮安 223300)[2]

摘 要 SVM算法在统计分类以及回归分析中得到了广泛的应用。而随着物联网的迅速发展,SVM算法在各种应用中往往需要解决大量数据的快速处理问题。在SVM算法并行化研究中,首先对SVM算法进行分析研究,提出了基于CUDA的SVM算法并行化方案;其次,进一步研究海量数据的处理,提出海量数据处理的并行化方案;最后,通过实验分析对比了并行化算法的性能。

关键词 CUDA,GPU,支持向量机,并行计算
中图法分类号 TP391.4 **文献标识码** A

Research of Parallel SVM Algorithm Based on CUDA

ZHANG Wei[1,2] ZHANG Gong-xuan[1] WANG Yong-li[1] ZHANG Yong-ping[1] ZHU Zhao-meng[1]
(School of Computer Science and Technology, Nanjing University of Science and Technology, Nanjing 210094, China)[1]
(School of Computer Science and Technology, Huaiyin Normal University, Huaian 223300, China)[2]

Abstract SVM has been widely used in statistical classification and regression analysis. With the rapid development of Internet of Things, SVM algorithms in various applications often need to address the challenges in the rapid processing of large amounts of data. Firstly, this paper studied SVM algorithm parallelization and proposed a parallel CUDA-based SVM algorithm scheme, and then further researched the massive data processing, raised a massive parallel data processing program. Finally, the performance of the parallel algorithm was compared via the experimental analysis comparing.

Keywords CUDA, GPU, Support vector machine, Parallel computing

1 引言

越来越多的传感器、移动终端和计算机通过网络联系起来,这也就是常说的物联网[1]。物联网把传感器装备到电网、铁路、桥梁、隧道、公路、建筑、供水系统、大坝、油气管道以及家用电器等各种真实物体上,通过互联网联接起来,进而运行特定的程序,以达到远程控制或者实现物与物的直接通信。

物联网融合各种信息技术,将物体接入信息网络,实现人与物、物与物之间的互联互通。未来信息产业的发展由信息网络向全面感知和智能应用两个方向拓展、延伸和突破。

支持向量机(Support Vector Machine,SVM)[2]是一种监督式学习的方法,它广泛地应用于统计分类以及回归分析中。SVM在许多物联网的数据应用领域如数据质量[3]、异常数据检测[4]、数据挖掘[5]等都得到了广泛的应用。SVM是这些数据处理中一个重要的工具。

CUDA(Compute Unified Device Architecture)[6]是NVIDIA公司推出的GPU通用计算产品,能够有效地利用GPU强劲的计算能力和巨大的存储带宽进行图形渲染以外的计算。与传统的并行计算[7]相比,CUDA编程更简单,功能更强大,应用利用更广。随着物联网的快速发展,存储系统中信息数据变得越来越庞大,通过CUDA来对数据进行快速处理,已经成为一个重要的研究领域。

SVM在解决小样本、非线性以及高维模式识别问题中具有特有的优势。这里的小样本指的是SVM算法和问题的复杂度相对而言所需样本数比较少,而不是指样本的绝对数量。在Banko的研究中[8]显示,在各种分类算法中更多的样本总能具有更好的分类效果。随着物联网数据量处理的快速增大,对SVM算法并行化的研究具有很重要的意义。在参考文献[9-11]中对SVM算法的加速、并行化都进行了相关研究。

本文首先介绍CUDA的基本概念、SVM算法的基本原理等相关背景;其次分析SVM算法,提出了基于CUDA的SVM并行化算法;最后通过实验数据对算法进行了分析,验证了SVM并行算法具有很好的加速性能。

2 研究背景

2.1 CUDA架构

2.1.1 GPU和CPU

GPU和CPU的浮点计算能力差异的原因是:GPU是特

到稿日期:2012-07-21 返修日期:2012-10-09 本文受国家自然科学基金(61170035),江苏省"973"自然科学基金(BK2011022)资助。
张 巍(1975-),男,硕士,副教授,主要研究领域为物联网技术、计算机网络、并行与分布式计算,E-mail:zw@hytc.edu.cn;张功萱(1961-),男,博士,教授,主要研究领域为Web Service与信息安全、下一代Internet技术、分布式计算体系、嵌入式与多核体系结构;王永利(1974-),男,博士,副教授,主要研究领域为数据库技术、普适计算、数据挖掘、物联网数据处理、生物信息挖掘、云存储。

别为计算密集、高并行度的计算(如同图像渲染)而设计的,因此将更多的晶体管用于数据处理而不是数据缓存和流控,如图1所示。

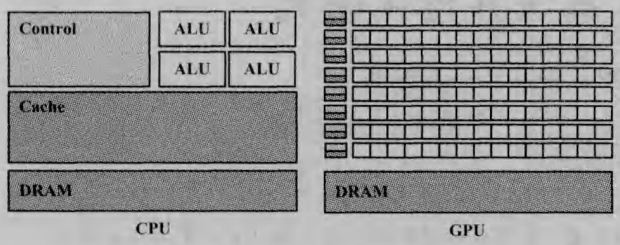

图1 GPU和CPU的区别

特别地,GPU非常适合处理那些能够表示为数据并行计算(同一程序在多个数据上并行执行)的问题,数据并行计算的算术计算密度(算术操作和存储器操作的比例)非常高。由于同一程序在每个元素上执行,因此对复杂流控的要求非常少,更因为在多个元素上执行和高计算密度,访存延迟可以被计算隐藏,因此用不着大的数据缓存。

数据并行处理将数据元素映射到并行处理的线程上。很多处理大的数据集的应用可以使用数据并行处理模型加速。三维图像渲染处理中,大量的像素和顶点被映射到并行线程。类似地,图像和多媒体处理应用、图像渲染的后处理、视频编解码、图像 缩放、立体视觉和模式识别能够将图像块和像素映射到并行处理的线程。实际上,除了图像渲染和处理领域,还有很多算法已被数据并行处理加速,从普通信号处理或物理模拟到计算金融或计算生物学。

2.1.2 SIMT架构

CUDA架构是围绕一个可扩展的多线程流多处理器(SMs)阵列构建的。当主机上的CUDA程序调用内核网格时,网格内块枚举并分发到有可用执行资源的处理器上。线程块内线程在一个多处理器上并发执行,并且多个块可在一个流多处理器上并发执行。

流多处理器设计为能同时并发执行上百线程。为了管理如此多的线程,多处理器采用了一种称为SIMT(单指令,多线程)的独一无二的架构。与CPU核心不同,其指令顺序发射,而且没有分支预测和猜测执行。

在使用单指令控制多处理元素上,SIMT架构类似SIMD(单指令,多数据)向量组织方法。重要的不同在于,SIMD组织方法会向应用暴露SIMD宽度,而SIMT指定单线程的执行和分支行为。与SIMD向量机相反,SIMT允许程序员为独立标量线程编写线程级并行代码,也为协作线程编写数据并行代码。为了正确性,程序员可忽略SIMT行为;而只要维护束内线程很少分支的代码就可显著提升性能。实践中,这类似于传统代码中缓存线的角色:以正确性为目标进行设计时,可忽略缓存线尺寸,但如果以峰值性能为目标进行设计,在代码结构中就必须考虑。另外,向量架构要求软件将负载合并成向量,并手动管理分支。

2.1.3 CUDA内存管理

并行程序在执行期间,CUDA线程可能访问来自多个存储器空间的数据,如图2所示。每个线程有私有的本地存储器。每个块有对块内所有线程可见的共享存储器,共享存储器的生命期和块相同。所有的线程可访问同一全局存储器。

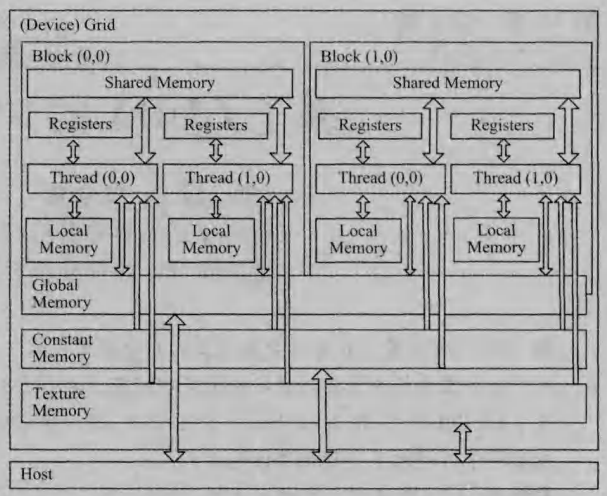

图2 CUDA内存结构

另外还有两种可被所有线程访问的只读存储器:常量和纹理存储器空间。全局、常量和纹理存储器空间为不同的存储器用途作了优化。

2.1.4 CUDA的处理流程

如图3所示,CUDA编程模型假设CUDA线程在物理上独立的设备上执行,设备作为主机的协处理器,主机运行C程序。例如,内核在GPU上执行,而C程序的其它部分在CPU上执行就是这种模式。

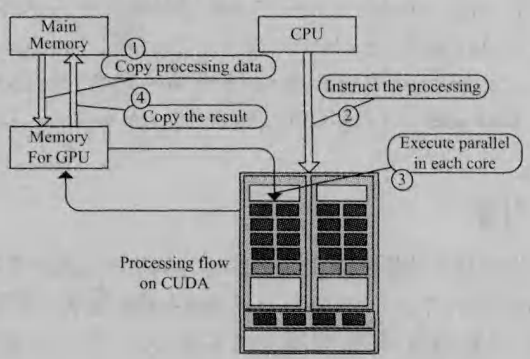

图3 CUDA的处理流程

CUDA编程模型同时假设主机和设备各自都维护着自己独立的DRAM存储器空间,各自被称为主机存储器空间和设备存储器空间。因此,程序通过调用CUDA运行,来管理对内核可见的全局、常量和纹理存储器空间。这包括设备存储器分配和释放,也包括在主机和设备间的数据传输。

2.2 SVM算法思想

SVM作为一种数据分类技术,已经在机器学习领域得到了广泛应用。SVM的基本思想是将向量映射到高维的空间里,然后在这个高维空间里建立使向量间隔距离最大的超平面以将数据分开。检测间隔距离的方法是在建立的超平面的两侧,根据已经分类的数据,再建立两个与此超平面平行的超平面,分隔超平面使两个平行超平面的距离最大化。这样通过构建使两个超平面间距离最大的方法来构建最大间隔分类器,是支持向量机的核心思想,如图4所示。

SVM构建最大间隔分类器的过程是一个机器学习的过程。过程中首先需要将已经分类的样本集划分为训练集和测试集。样本集中每一个样本实例都具有已知的分类目标值和

多个属性值，称其为样本向量。

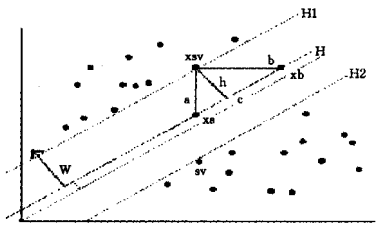

图 4　最大间隔分类

给定训练样本向量 $x_i \in R_n (i=1,\cdots,l)$，和对应的划分为两类的分类指示向量 $y \in R_l$，$y_i \in \{1,-1\}$，在 C-SVC(C-Support Vector Classification)[2] 将解决如下优化问题。

$$\min_{\theta,b,\zeta} \frac{1}{2}\theta^T\theta + C\sum_{i=1}^{l}\zeta_i$$

$$\text{s.t. } \frac{1}{2}\theta y_i(\theta^T\phi(x_i)+b) \geqslant 1-\zeta_i \quad (1)$$

$$\zeta_i \geqslant 0$$

此处，$\phi(x_i)$ 将向量 x_i 映射到高维空间，C 是正规化参数，$C>0$。向量变量 θ 是高维的向量，需要解决下面的问题。

$$\min_{\alpha} \frac{1}{2}\alpha^T Q\alpha - e^T\alpha$$

$$\text{s.t. } y^T\alpha = 0$$

$$0 \leqslant \alpha_i \leqslant C \quad (2)$$

$$i=1,\cdots,l$$

$e=[1,\cdots,1]^T$ 是值全为 1 的向量，$l \times l$ 半正定矩阵 Q，$Q_{ij} \equiv y_i y_j K(x_i,x_j)$，并且 $K(x_i,x_j) \equiv \phi(x_i)^T\phi(x_j)$ 是核函数。

当问题(2)解决时，θ 将满足

$$\theta = \sum_{i=1}^{l} y_i\alpha_i\phi(x_i) \quad (3)$$

这样决策函数可以表示为

$$\mathrm{sgn}(\theta^T\phi(x)+b) = \mathrm{sgn}\left(\sum_{i=1}^{l} y_i\alpha_i K(x_i,x)+b\right) \quad (4)$$

此处参与预测分类的数据有 $y_i\alpha_i \ \forall i, b$。其他的数据还有支持向量 x_i、核心函数的参数等。

2.3　分类准确度

在 Banko 的研究中[8]，分类的准确度和训练的样本数有很大的相关性。在图 5 中显示了不同学习算法的学习曲线：memory-based，winnow，naïve Bayes，perceptron learner of machine learning algorithms。每一个算法都使用上亿的单词量进行训练。从图 5 可以看出，算法的准确度随着训练样本数的上升呈现对数上升。

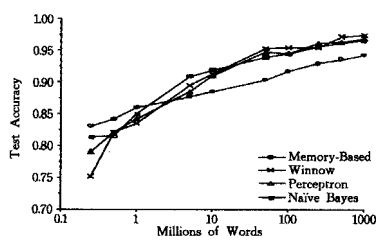

图 5　学习曲线和训练样本

从图 5 可以看出，只有对足够多的训练样本进行学习训练后，分类算法才可以达到一个比较理想的分类准确度。所以通过并行算法，提高 SVM 算法的执行效率，对数据量巨大的物联网应用具有重要的意义。

3　SVM 算法并行化

LibSVM[2] 是一个得到广泛使用的 SVM 计算工具，具有程序执行速度快、分类效果好的特点。在 SVM 算法中，主要经过 3 个阶段：学习阶段、交叉验证阶段和分类预测。而 SVM 算法在学习阶段是最消耗计算资源的，随着数据量的增加，如何提高数据处理效率是本文主要研究的内容。

CUDA(Compute Unified Device Architecture)[6] 是 Nvidia 公司开发的并行计算框架。开发者通过 CUDA 提供的虚拟指令可以使用并行计算元素和 GPU 内存并使用。有了 CUDA 的支持，开发者可以像使用 CPU 一样方便地使用 Nvidia 公司最新版本的 GPU[12]。编程人员通过写简单的 GPU 内核程序，实现并行计算的功能。一个内核程序可以被多个线程并行执行。

多个线程可以组织成一个线程块，线程块内的线程共享内存和寄存器等资源。多个线程块组成一个内核网格，内核网格内的线程块间相互独立地并行执行。我们的 SVM 算法并行化工作基于 LibSVM 和 CUDA。

在对 SVM 算法的分析中，根据矩阵 x 和向量 y 来计算矩阵 Q，$Q_{ij} \equiv y_i y_j K(x_i,x_j)$。

$$Q = \sum_{i=1}^{l}\sum_{j=1}^{l} y_i y_j K(x_i,x_j) \quad (5)$$

式中，K 为核函数，矩阵 Q 的计算在 CPU 中占用大量的计算时间。将算法并行化，在 GPU 中每一个线程只对矩阵 Q 的行向量 Q_i 进行计算：

$$Q_i = \sum_{i=1}^{l} y_i y_j K(x_i,x_j) \quad (6)$$

通过编写 GPU 内核程序获得较高的并行计算能力。CUDA 采用 SIMT 架构，此架构中指令顺序执行，而且没有分支预测，如果有大量的逻辑判断，则会导致并行性能下降。因此在进行内核程序设计时，每个内核程序只做简单的循环，以提高程序的并行执行能力。

算法 1　SVM 的并行算法 P-SVM(Parallel Support Vector Machine)

1. 在主存中初始化训练数据集 $x_i \in R_n (i=1,\cdots,l)$ 和分类指示向量 $y \in R_l, y_i \in \{1,-1\}$；
2. 初始化 Q，x_i，y 在 GPU 中的内存空间；
3. 将主存中 x_i，y 复制到 GPU 内存；
4. 在 GPU 调用内核程序并行计算矩阵 Q，实现式(6)的功能；
5. 将计算结果从 GPU 内存复制到主存中；
6. CPU 计算 α_i 和 b 等参数；
7. 根据参数对测试集进行分类。

为了提高内核程序的运行效率，在数据前期处理时，对训练数据集的样本属性值进行了两种不同的处理：在 P-SVM 中是普通表示方式，对每个样本向 x_i 进行所有属性值的填充处理，使 X 为 $l \times m$ 的方阵，l 为测试样本数，m 为样本的属性数；另一种为数据稀疏表示，只填入样本中不为 0 的属性值，样本训练集不再是一方阵。

算法 2　SVM 的稀疏并行算法 SP-SVM(Sparse Parallel Support Vector Machine)

1. 在主存中初始化训练数据集 $x_i \in R_n (i=1,\cdots,l)$，和分类指示向量 $y \in R_l, y_i \in \{1,-1\}$，$x_i$ 为稀疏表示；
2. 初始化 Q，x_i，y 在 GPU 中的内存空间；
3. 将主存中 x_i，y 复制到 GPU 内存；
4. 在 GPU 调用内核程序并行计算矩阵 Q，实现式(6)的功能；

5. 将计算结果从 GPU 内存复制到主存中；

6. CPU 计算 α_i 和 b 等参数；

7. 根据参数对测试集进行分类。

显然，P-SVM 适合样本属性值密集的情况，但在样本数据属性值稀疏情况下会导致空间浪费。而 SP-SVM 适合样本属性值比较稀疏的情况，但在样本数据属性值密集的情况下，由于要增加属性的标识，会导致内存空间的浪费。

对于 P-SVM 和 SP-SVM 这两种算法，在实验中采用稀疏数据，对性能进行了分析与比较。

4 实验与分析

4.1 实验环境

在实验中分别对训练样本数据量对于分类的准确度、并行算法对小数据量的运行效能和并行算法对大数据量的运行效能进行了对比与分析。实验数据采用文献[13]提供的数据。

我们实验的硬件平台是 PC 机，2.3GHz Intel i5 CPU，16GB 内存和 NVIDIA Tesla C2050 GPU。软件环境是 Windows 7，Visual studio 2010，CUDA4.0。

4.2 训练样本数对准确度的影响

为了分析训练集大小对分类准确度的影响，在实验中，采用不同数据量的训练样本进行测试。可以看出，随着训练样本数据量的增加，对测试样本分类的准确定度也所有提高，如图 6 所示。

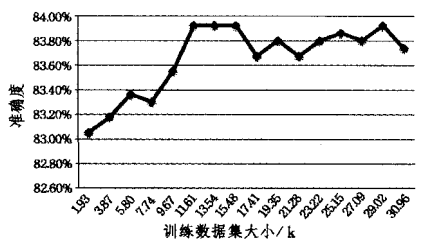

图 6 训练样本数和准确度

实验采用在数据集上进行分片，模拟不同数据集大小的方式，准确度值的绝对变化范围并不明显。但根据文献[8]，随着训练样本数据量增大，对准确定度会有较强的相关性。

由于是在 Lib-SVM 基础上进行的并行化，因此 P-SVM 和 SP-SVM 的分类准确度与 Lib-SVM 没有本质区别，故实验主要针对训练样本数据和分类准确度之间的关联性进行分析比较。

4.3 并行算法性能分析

在 CUDA 中，计算机主存和 GPU 内存间的数据传输带宽为 4GB/s，其远小于 GPU 内存 144GB/s 的带宽。当处理数据量比较小时，由于内核程序运行时间较短，而内存间数据传递占用了大量时间，因此显得并行算法的执行速度明显低于普通的串形算法。

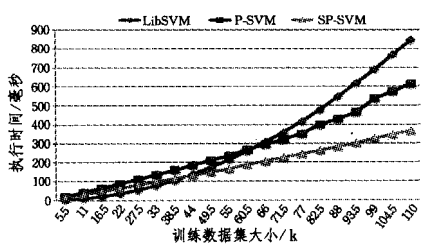

图 7 小数据量算法性能比较

在图 7 中，当训练集小于 38.5k 时，SP-SVM 算法执行时间长于 Lib-SVM；当训练集小于 60.5k 时，P-SVM 算法执行时间长于 Lib-SVM。但当训练集增大时，并行算法的运行效率明显提高。

由于在实验所采用的是稀疏的数据，因此 SP-SVM 在小数据量时，也明显比 P-SVM 有更高的执行效率。

从图 8 可以看出，当训练样本数据集大于 110k 时，计算机主存和 GPU 内存间数据传输的时间在计算中所占比例下降，并行算法总的计算时间明显优于串行算法，并且随着数据量的增加，并行算法执行时间保持较小增加。

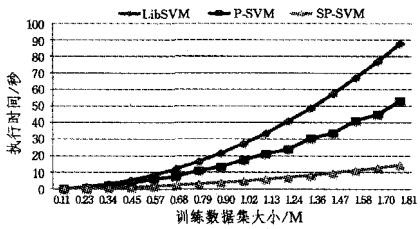

图 8 大数据量算法性能比较

结束语 随着信息技术的爆炸式发展，大量数据的处理对实时性的要求越来越高。基于 GPU 的数据并行处理作为一种有效提高数据处理速度、提高应用系统响应实时性的重要技术手段得到了广泛应用。

SVM 是物联网、数据挖掘、机器学习等领域非常重要的数据分类技术。本文基于 CUDA 对 SVM 算法进行并行化，取得了较高的运行效率，为 SVM 算法在大规模数据处理中的应用做出了基础性的研究工作。

通过实验，我们对比了 Lib-SVM、P-SVM 和 SP-SVM 的运行效率，并对实验数据进行了实验对比与分析，其表明了 SVM 并行化算法的有效性。

当前关于 SVM 算法的并行化研究文献[10,11]也做了相关工作。

Thanh 的研究[10]对于 SVM 的样本训练过程也进行了并行化，取得了较高的运行效率。但在文献[10]中直接使用了 CUDA1.1 提供的 CUBLAS 函数进行矩阵运算，而在本文的 SVM 并行化中直接使用 CUDA4.0 内核函数编程，可更好地对 GPU 中的线程进行调度与控制。

Qi Li 的研究[11]对 SVM 算法的交叉验证选择参数的算法进行了并行化处理。交叉验证的可并行化程度比较高，取得了很高的执行速度。而样本训练效率较低是 SVM 占用计算资源最大的一个问题，本文所做研究解决了样本训练并行计算的问题，可以结合文献[11]中的研究成果，进一步提高 SVM 算法的并行化效果。

参 考 文 献

[1] Atzori L, Iera A, Morabito G. The Internet of Things: A survey [J]. Computer Networks, 2010, 54: 2787-2805

[2] Chang C-C, Lin C-J. LIBSVM: a library for support vector machines[J]. ACM Transactions on Intelligent Systems and Technology (TIST) archive, 2011, 2(3)

[3] Redman T. The impact of poor data quality on the typical enterprise[J]. Commun. ACM, 1998, 2: 79-82

[4] Chandola V, Banerjee A, Kumar V. Anomaly Detection: A Survey[J]. ACM Computing Surveys, 2009, 41(3): 15

(下转第 106 页)

均只对标签激活一次，不能用于动态标签环境，一旦标签以较快速度运动，新进入的标签均会被漏读。但即使是在静态标签环境下，M-TMAC 的 T_{tag} 也是最小的，仅为 ALOHA 算法的 1/2、为树形算法的 1/3，这是由于最优分组和帧长的选取降低了碰撞时隙和空时隙的发生概率，碰撞时隙内的标签平均数量也达到最小，这保证了查询树算法较小的识别时间；发现一个碰撞时隙即采用查询树算法解决一个，降低了将不同碰撞时隙内标签全放在一起再次识别时原先不同时隙内的标签间发生碰撞的可能性，减少了总识别时间。

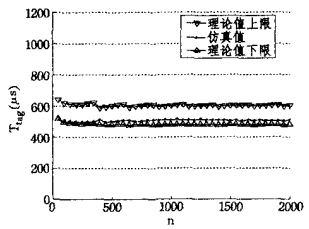

图 6 T_{tag} 理论值与仿真值的比较

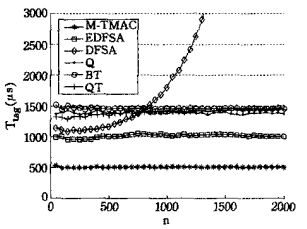

图 7 传统静态 MAC 与 M-MAC 的 T_{tag} 比较

图 8 为单读写器部署，$P=99.9\%$，$P'=1\%$，M-TMAC 协议 n、θ_{max} 和 v_{max} 的关系。可见，n 相同时，θ_{max} 与 v_{max} 成反比，且 n 越小、θ_{max} 相同时允许的 v_{max} 越大。当 n 较小时($n=100$)，θ_{max} 为 0 和 60°时允许的 v_{max} 分别高达 110m/s 和 60m/s 以上；即使在 n 很大的极端情况($n=2000$)，θ_{max} 为 0 和 60°时 v_{max} 也分别达到 8m/s 和 4m/s 左右，满足工程要求。

图 9 反映了占用读写器资源的情况，SDepth 数据是将仿真平均值向右取整得到。易见，即使标签数量为 2000 时，实际查询栈栈深也不大于 8，因此 M-TMAC 占用的额外读写器资源是很少的。

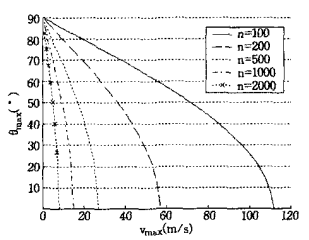

图 8 M-TMAC v_{max} 和 θ_{max} 的关系

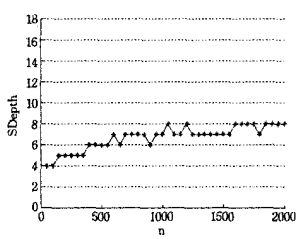

图 9 查询栈深度变化情况

结束语 由此可见，无论是动态或是静态密集标签环境，M-TMAC 协议的标签识别时间均较短，即使在静态密集标签环境下，与传统静态 MAC 协议相比，M-TMAC 协议标签识别时间也仅为 ALOHA 算法的 1/2、树形算法的 1/3；在指定总识别率、信道干扰率的条件下，所支持的标签入射角及最大运行速度能满足绝大多数的应用场合。如 $P=99.9\%$、$P'=1\%$、标签数量为 500 时，0°和 60°入射角支持的标签最大速度分别达到 60m/s 和 30m/s 左右。将来的工作可在分析标签到达率与主要性能指标的关系、标签非直线运动情况及多读写器部署策略等方面展开。

参 考 文 献

[1] 杨健，王永华，蔡庆玲，等. EHiQ：一种基于增强型 HiQ 的 RFID 读写器 MAC 协议[J]. 计算机科学，2011，38(7)：85-87，112

[2] 杨健，詹宜巨，王永华，等. 基于分组动态帧和查询栈的射频识别反碰撞算法[J]. 系统仿真学报，2010，22(12)：2920-2924

[3] 孙文胜，金陈敏. 新型的 RFID 动态帧时隙 ALOHA 防碰撞算法[J]. 信息与控制，2012，41(2)：233-237

[4] Cha J-R，Kim J-H. Dynamic Framed slotted ALOHA algorithms using fast tag estimation method for RFID system[C]// Proceedings of the 3rd Consumer Communications and Networking Conference，2006. USA：IEEE，2006：768-772

[5] Lee S-R，Joo S-D，Lee C-W. An enhanced dynamic framed slotted ALOHA algorithm for RFID tag identification[C]// Proceeding of the 2nd Annual International Conference on Mobile and Ubiquitous Systems：Networks and Services，2005. USA：IEEE，2005：166-174

[6] Myung J，Lee W，Srivastava J，et al. Tag-splitting：adaptive collision arbitration protocols for RFID tag identification [J]. IEEE Transactions on Parallel and Distributed Systems，2007，18(6)：763-775

[7] Auto-ID Center. Draft protocol specification for a 900MHz Class 0 Radio Frequency Identification Tag[OL]. http://www.autoidlabs.org，2009-07-02

[8] Venkatesh S，Mallareddy D，Sridhar R-S. A framework for fast RFID tag reading in static and mobile environments[J]. Computer Networks，2008，52(5)：1058-1073

(上接第 72 页)

[5] Xiao H. Towards parallel and distributed computing in large-scale data mining：A survey[R]. Technical University of Munich，2010：1-30

[6] http://developer.download.nvidia.com/compute/cuda/1_0/NVIDIA_CUDA_Programming_Guide_1.0.pdf

[7] Almasi G S，Gottlieb A. Highly Parallel Computing[M]. Benjamin-Cummings publishers Co.，Inc. Redwood City，CA，USA，1989

[8] Banko M，Brill E. Scaling to very very large corpora for natural language disambiguation[C]// ACL '01 Proceedings of the 39th Annual Meeting on Association for Computational Linguistics. Stroudsburg，PA，USA，2001：26-33

[9] Hu W J，Song Q. An accelerated decomposition algorithm for robust support vector Machines[J]. IEEE Transactions on Circuits and Systems II：Express Briefs，2004，51(5)：234-240

[10] Do T-N，Nguyen V-H. A novel speed-up SVM algorithm for massive classification tasks[C]// Research，Innovation and Vision for the Future，2008. RIVF 2008. IEEE International Conference. July 2008：215-220

[11] Li Qi，Salman R，Test E，et al. Parallel multitask cross validation for Support Vector Machine using GPU[J]. Journal of Parallel and Distributed Computing，2013，73(3)：293-302

[12] Nickolls J，Buck I，Garland M，et al. Scalable Parallel Programming with CUDA[J]. Queue-GPU Computing，2008，6(2)

[13] http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/