doi: 103969/j. issn. 0490-6756. 2019. 02. 008

基于 GPU 加速的恶意代码字节码特征提取方法研究

周紫瞻, 王俊峰
(四川大学计算机学院, 成都 610065)

摘 要: 随着恶意代码的数量和种类增长,快速有效地检测恶意代码显得十分有必要,其中关键技术就是恶意代码特征提取. 针对现有恶意代码字节码序列特征提取速度的不足,提出了一种 GPU 加速提取恶意代码字节码序列特征的方法. 使用目前比较成熟的统一计算设备架构 CUDA,将传统恶意代码字节码序列特征提取方法中字节码 N-Gram 特征的提取、TFIDF 特征的计算等密集计算型任务移交给 GPU 进行并行计算. 实验表明,针对不同样本文件大小的数据集,该方法均有 2~4 倍以上的速度提升,大幅提高恶意代码字节码序列特征提取的速度.

关键词: 恶意代码; 特征提取; CUDA; 字节码序列; N-Gram
中图分类号: TP309. 7 **文献标识码**: A **文章编号**: 0490-6756(2019)02-0227-08

Research on feature extraction of malware bytecode based on GPU acceleration

ZHOU Zi-Zhan, WANG Jun-Feng
(College of Computer Science, Sichuan University, Chengdu 610065, China)

Abstract: With the increase of the number and type of malicious code, it is necessary to detect malicious code quickly and effectively. One of the key point is the feature extraction of malicious code. Aiming at the insufficiency of the feature extraction speed of malicious code bytecode sequences in the existing methods, a method of GPU acceleration to extract the features of malicious code bytecode sequences is proposed. Compute-intensive tasks such as the feature extraction of bytecode and the calculation of TFIDF features in traditional methods are transferred to the GPU for parallel computing by using the more mature CUDA architecture. The experimental results show that the method has a speed increase of 2 to 4 times for data sets with different sample file sizes, which greatly improves the speed of feature extraction of malicious code bytecode sequences.

Keywords: Malware; Feature extraction; CUDA; Bytecode sequence; N-Gram

1 引 言

恶意代码又称恶意软件,是指在未经用户授权的情况下,在计算机系统执行恶意任务,侵犯用户合法权益的软件,包括病毒、蠕虫、木马、后门等. 据《瑞星 2017 年中国信息安全报告》称,2017 年瑞星“云安全”系统共截获病毒样本总量 5,003 万个,病毒感染次数 29. 1 亿次,病毒总体数量比 2016 年同期上涨 15. 62%,网络安全态势不容乐观. 随着恶意代码的数量和种类日趋增多,检测恶意代码变得越来越困难,其中特征提取速度慢便是主要原因之一,实现快速有效地提取恶意软件特征成为现阶段

收稿日期: 2018-08-18
基金项目: 国家重点研发计划项目(2018YFB0804503, 2016QY06X1205); 装备预研教育部联合基金(6141A02011607, 6141A02033304); 四川省重点研发计划项目(18ZDYF3867, 2017GZDZX002)
作者简介: 周紫瞻(1994-), 男, 湖北武汉人, 硕士生, 研究方向为网络与信息安全. E-mail: zhouzizhan@foxmail. com
通讯作者: 王俊峰. E-mail: wangjf@scu. edu. cn

亟待解决的问题.

恶意代码检测过程,通常可以分为三个步骤:①特征提取与选择;②选取适当的分类模型;③获取分类结果.其中关键技术就是恶意代码特征提取,特征的质量直接影响恶意代码的检测效果.目前提出的基于机器学习的恶意软件分类方法在恶意软件特征提取方法上存在较大差异.

Schultz[1]等人提出使用DLL、API、字符串和字节序列作为恶意软件特征.Kolte[2]等人选择恶意软件的字节码N-Grams信息,并使用信息增益来构造恶意软件特征.Shafiq[3,4]等人提取文件结构属性中关键部分作为特征.在所提出的这些特征提取算法中,字节码N-Gram算法具有良好的实验性能,比其他算法具有更强的鲁棒性.然而在实际应用中,提取恶意软件字节码N-Gram特征成本过高,当数据集增大,时间开销和系统资源开销快速增加.

GPU的出现为算法设计和优化带来了新的契机,将原有算法中计算密集型任务使用GPU加速计算,大幅提高了算法的效率.在恶意代码检测过程中,训练分类模型大多采用机器学习、深度学习等技术,GPU高性能计算已经普遍的用于加速深度学习算法.但是,在恶意代码特征提取的过程中还很少使用GPU加速计算.

本文实现了一种基于GPU加速计算的恶意代码字节码序列特征提取方法,使用GPU高性能计算,将传统字节码N-Gram特征提取方法中,字节码N-Gram特征的提取、选择和特征矩阵的TFIDF计算部分转移到GPU上进行,提高字节码序列特征提取速度,具有十分重要的意义.

2 相关工作

2.1 字节码序列特征

在对恶意代码的分析过程中发现,特征提取的质量直接影响恶意代码检测的效果.根据检测过程执行方式的不同,恶意软件检测方法可分为动态检测方法和静态检测方法.动态检测方法指的是在虚拟机、模拟器、沙盒等受控环境中运行恶意代码样本,记录并分析其系统调用序列、系统调用参数、运行指令序列、信息流等,从而识别其恶意行为.动态方法能准确的识别其恶意行为的本质,对加壳、变形、多态、混淆的样本仍然有效.文献[5]提出了一种基于动态行为指纹的恶意代码同源性分析方法,通过提取字符串的命名特征、注册表的变化特征和API函数的调用顺序特征,使用指纹匹配算法有效地对不同恶意代码及其变种进行同源性分析.但动态检测方法通常需要消耗较多的时间和系统资源,受运行环境影响较大,不能完整地遍历软件的所有可执行路径.静态方法无需执行样本文件,直接分析样本的可执行文件格式信息、字节码序列、系统函数调用、操作码序列等.其中基于字节码序列的静态检测方法,使用N-Gram提取样本字节码特征,并使用机器学习分类器来进行检测,取得了较好的效果.对于PE格式二进制文件而言,Schultz[1]等人最早提出将DLL、字符串、字节序列N-Gram当作恶意软件的特征,使用不同的分类器进行分类.其中选择字节N-Gram特征,使用多重朴素贝叶斯算法进行训练,综合准确率达到96.9%.Kolter和Maloof[2,6]提取字节码N-Gram特征,对良性可执行文件和恶意可执行文件进行分类.在一组较小的1037个文件的数据集中,提取4-gram特征,使用AdaBoost[7]与C4.5决策树[8]实验结果最好.此外,他们还利用信息增益将4-gram特征降维,该方法的AUC达到0.984.对一组更大的3622个文件的数据集进行测试,AUC达到了0.996.实验表明,随着数据集增大,实验效果提升.

虽然上述基于字节码序列特征的恶意软件检测方法都取得了很好的实验结果,但是在实际应用中,提取字节N-Gram特征是十分消耗资源的,特别是在对N-Gram特征进行统计与选择的过程中.但是随着样本数据集的增大,时间开销和系统资源开销快速增加,不能满足需要.针对这些问题,本文使用GPU加速技术,大幅提高了提取字节码N-Gram特征的速度.

2.2 GPU加速计算

近些年来,CPU虽然有很大的发展,但还是显示出了其单线程处理性能在功耗墙(Power Wall)、存储墙(Memory Wall)、频率墙(Frequency Wall)和过低的指令级并行方面的限制.GPU的出现为算法设计和优化带来了新的契机,由CPU负责任务调度、逻辑处理、事物管理等控制密集型计算任务,GPU负责密集计算型的大规模数据并行计算,将原有算法中计算密集型任务使用GPU加速计算,大幅提高了算法的效率.对于那些计算密集且并行度较高的领域,GPU高性能计算能起到非常好的加速效果.目前GPU加速计算已经在图像信号处理、生物计算、流体力学、人工智能等领域取得了非常成功的应用.

在特征提取领域，文献[9]提出了一种基于 CUDA 的 SIFT 的提取算法 HartSift，充分利用 CPU 和 GPU 计算资源实现了高精度实时特征提取. 文献[10]提出了一种基于 GPU 加速的 SURF 算法局部特征提取，实现了 SURF 的检测与扫描实时性要求. 在机器学习领域，文献[11]提出了一种基于 CPU+GPU 异构 Spark 平台的机器学习算法，在许多机器学习应用上有着 18 倍的速度提升. 在数据挖掘领域，文献[12]提出了一种基于 CUDA 的并行数据挖掘算法，包括 CU-Apriori、CU-KNN 和 CU-K-means.

GPU 加速计算在恶意代码检测领域也有许多应用，文献[13]提出了一基于动态子字符串跟踪的恶意代码检测分析方法，使用 CUDA 加速计算有着 8 倍的性能提升. 文献[14]提出了一种基于流量分析的恶意代码分类方法，将流量数据当作图像，使用 GPU 加速神经网络来对样本进行分类. 目前，基于机器学习的恶意代码检测方法中，GPU 加速计算已经普遍的应用于模型的训练中，但是在恶意代码特征提取的过程中应用很少.

3　GPU 加速字节码序列特征提取

3.1　传统字节码特征提取过程

本文选择的字节码序列特征提取方法，将样本当作字节流，采用 N-Gram 滑动窗口方式对样本进行处理. N-Gram 是由一个长度为 N 的滑动窗口收集的一系列重叠的子字符串，这个窗口每次滑动一个单位长度. 比如，样本字节流十六进制表示为{4D,5A,90,00,03,...}，经过 3-gram 处理后，变成特征值为{[4D5A90]，[5A9000]，[900003]，[...]}. 根据特征值不同，对每个样本的出现的特征值数量进行统计并以{key,value}字典形式存储特征词频，其中 key 代表特征值，value 代表该特征值在当前样本中出现的次数. 当所有样本处理结束后，对样本集所有特征值进行统计，挑选出词频最高的 M 个特征作为有效特征. 然后将每个样本转换为 M 维向量，每个维度代表该特征在样本中出现的次数，样本集就转换为一个特征词频矩阵，每一行代表一个样本，使用 TFIDF 算法将词频矩阵转换为特征矩阵，并使用信息增益等方法对特征矩阵进行特征筛选，形成最终的特征矩阵.

3.2　GPU 加速计算字节码 N-Gram 特征

GPU 加速字节序列特征提取指的是在传统的提取方式中加入 GPU 作为加速部件，并配合 CPU 共同完成计算任务. 采用 CPU+GPU 异构并行计算的模式，由 CPU 负责任务调度并执行复杂的逻辑处理和事物管理等控制密集型计算任务，GPU 负责密集计算型的大规模数据并行计算. 在提取样本字节序列特征的过程中，首先要考虑哪些任务适合用于 GPU 并行算法实现，将不同的任务分别放在 CPU 和 GPU 上完成.

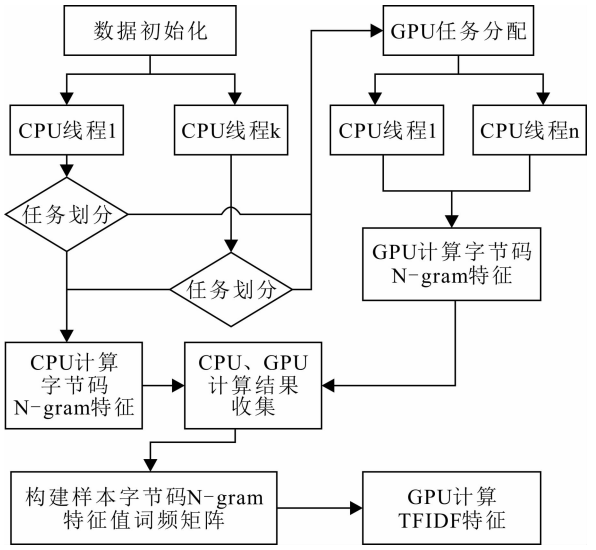

图 1　GPU 加速提取字节码特征

Fig. 1　Features extraction of bytecode using GPU acceleration

如图 1 所示，使用 CPU 多线程来读取样本文件，因为 GPU 计算存在数据传递、通信延时等开销，所以根据样本文件的大小进行任务划分，当文件大小超过某一阈值，采用 GPU 加速计算样本字节 N-Gram 特征，未超过阈值时采用 CPU 计算. 剔除出现次数极少的特征，构建样本{特征值，词频}字典. 为了降低特征维度，本方法挑选出词频较高的有效特征，将所有样本转换为有效特征频次矩阵，使用 GPU 计算 TFIDF(Term Frequency-Inverse Document Frequency)特征.

使用 GPU 计算样本字节码 N-Gram 特征的过程中，首先将样本数据转移到 GPU，根据样本文件大小，初始化 GPU 资源，然后将样本文件分配到多个不同的 Grid 中，每个 Grid 中包含多个不同的 Block，每个 block 中包含多个线程.

例如文件大小为 1 MB，此 Grid 中包含 1024 个 block，每个 block 中包含 1024 个线程，总共 1048576 个线程，线程并行执行过程如下图 2 所示，每个线程根据自身索引读取样本文件对应位置的 N 个字节.

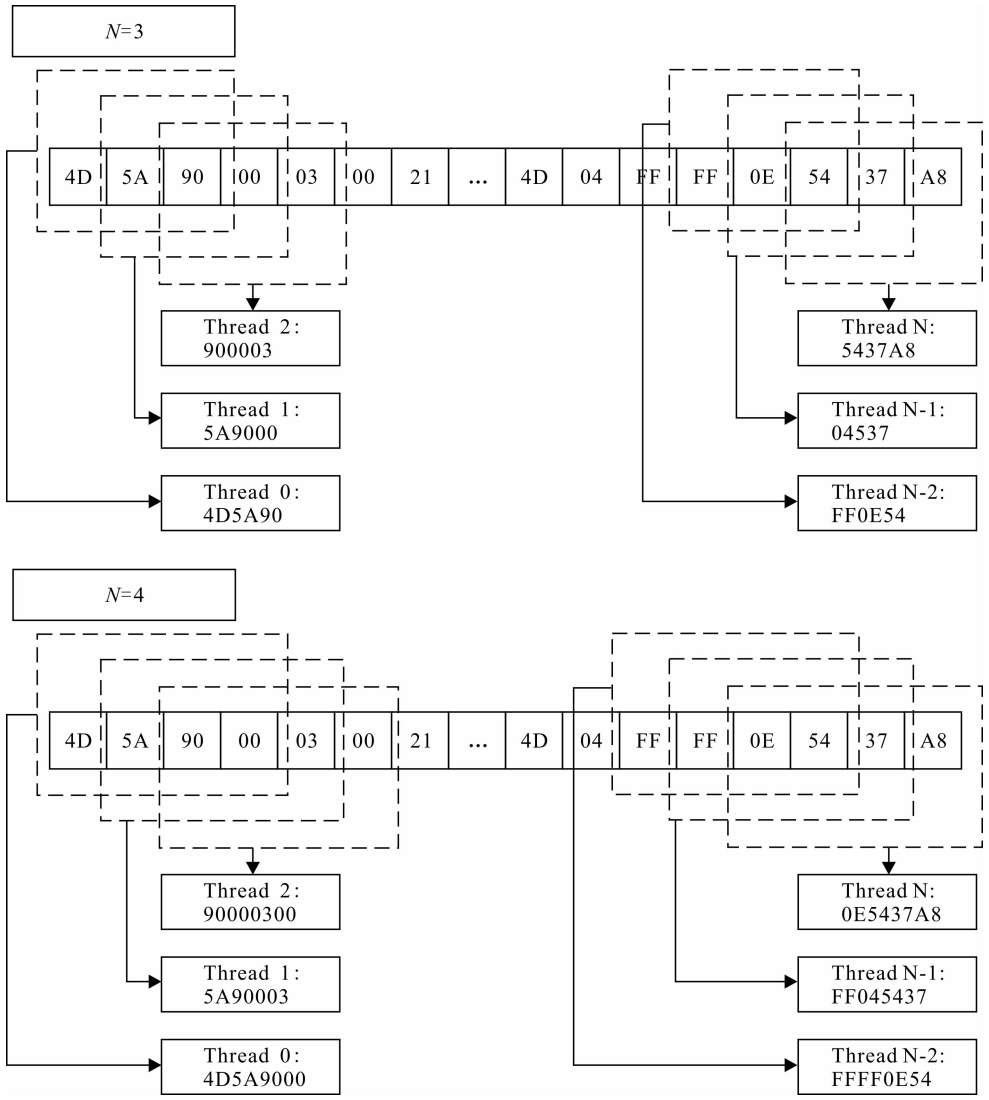

图 2 GPU 计算 N-Gram 特征

Fig. 2 Computing N-Gram features using GPU

样本文件字节 N-Gram 特征计算完成后,需要统计每种特征值出现的频次,实验中发现,N-Gram 方法提取出的特征种类数量巨大,当 N 选取为 3 时,特征种类可能有 16777216 种,直接统计每种特征值出现的次数非常消耗系统资源,当 N 继续增大时,资源消耗难以接受,为了节省存储空间以及查找时间,本文剔除出现次数较少的特征值. 实验中还发现使用 GPU 对特征值进行排序速度大幅领先使用 CPU 进行排序,所以在统计每种特征值出现的频次过程中,采用 GPU 对特征值进行排序,然后遍历特征值,存储词频大于阈值的特征值以及出现频次. 对于使用 N-Gram 方法提取的特征,需要统计每种特征在每个样本中出现的频率. 由于特征的维度比较大,需要进行降维,只选择那些在代码检测过程中比较有效的特征. 每一个样本对应自身的特征词频字典,将所有样本的字典做并集,特征值相同时,词频叠加,构建出整个样本集的特征词频字典. 然后使用 GPU 对此字典进行排序,筛选出词频较高的特征. 特征选择后的所有样本可以转换为一个有效特征出现频次的矩阵,矩阵每一行代表一个样本,每一列代表一种有效特征在该样本中出现的频次.

3.3 GPU 加速计算 TFIDF 特征

每种特征对实体的贡献度不同,需要对这些特征赋予不同的权重,TFIDF 是一种常用的方法. 词频指的是某一个给定的词语在该文件中出现的次数. 这个数字通常会被归一化, 以防止它偏向长的文件. 逆向文件频率的主要思想是:如果包含词条 t 的文档越少, IDF 越大,则说明词条具有很好的类别区分能力. 某一特定词语的 IDF,可以由总文

件数目除以包含该词语之文件的数目，再将得到的商取对数得到. 某一特定文件内的高词语频率，以及该词语在整个文件集合中的低文件频率，可以产生出高权重的 TFDF. 因此，TFIDF 倾向于过滤掉常见的词语，保留重要的词语.

使用 GPU 计算 TFIDF 特征，根据特征频次矩阵大小初始化 GPU 资源，假设矩阵大小为 M 行 N 列，创建 M 个线程并行处理矩阵每一行，计算当前样本词频，得到词频矩阵，创建 N 个线程并行处理每一列，统计当前特征值频次不为零的样本出现的个数，得到逆文档频率矩阵.

实验中发现，特征频次矩阵中出现零的概率很大，为稀疏矩阵，为了减少存储量和计算量，最终计算 TF * IDF 时，创建线程数等于特征频次矩阵中不为零的元素个数.

为了进一步提高 GPU 计算性能、降低主机内存与 GPU 显存之间数据传递开销，本文采用 CUDA stream 流和 CUDA Unified Memory 统一内存技术.

CUDA 中使用多个流并行执行数据复制和核函数运算可以进一步提高计算性能. CUDA 流表示一个 GPU 操作队列，并且该队列中的操作将以指定的顺序执行. 我们可以在流中添加一些操作，如核函数启动，内存复制以及事件的启动和结束等. 这些操作的添加到流的顺序也是它们的执行顺序. 可以将每个流视为 GPU 上的一个任务，并且这些任务可以并行执行. 同一个 stream 里的操作有严格的执行顺序，不同的 stream 则没有此限制. 由于不同 stream 的操作是异步执行的，就可以利用相互之间的协调来充分发挥资源的利用率.

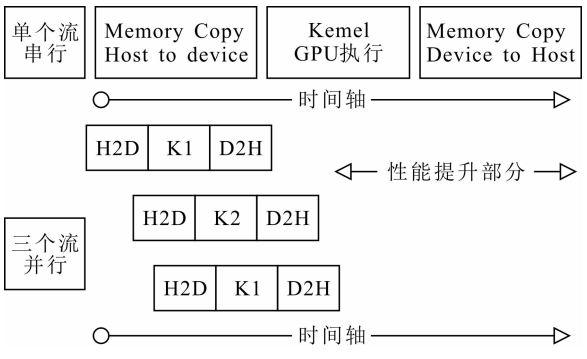

图 3　GPU 流执行过程

Fig. 3　The process of GPU stream

如图 3 所示，单个流串行分为 H2D(Memory Copy Host to Device)、Kernel(GPU 执行核函数)、D2H(Memory Copy Device to Host)三个任务. 将单个流中任务切分成三个流，不同流间可异步执行，大幅提高了 GPU 使用率.

在当今典型的 PC 或群集节点中，CPU 和 GPU 的存储器在物理上是独立的，使用 PCI-Express 总线通信. 在 CPU 和 GPU 之间共享的数据必须分配到两个存储器中，并且由程序在它们之间复制. Unified Memory 创建一个由 CPU 和 GPU 共享的托管内存池，用于桥接 CPU 与 GPU 之间的鸿沟. CPU 和 GPU 都可以使用单个指针访问托管内存，关键在于系统会自动迁移主机和设备之间的统一内存中分配的数据. Unified Memory 统一内存结合了显式拷贝和零拷贝访问的优势：GPU 可以访问整个系统内存的任何页面，同时将数据按需迁移到其自己的内存以实现高带宽访问.

4　实验结果

为了验证本文提出的基于 GPU 加速的恶意代码字节码序列特征提取方法的有效性，根据字节码序列特征提取过程中使用了 GPU 加速的部分，从以下四个方面进行了实验. (1) CPU 与 GPU 计算字节码 N-Gram 特征实验；(2) CPU 与 GPU 统计字节码 N-Gram 特征词频实验；(3) CPU 与 GPU 计算 TFIDF 实验；(4) CPU 与使用 GPU 加速提取样本字节码序列特征整体对比实验.

实验环境：操作系统为 CentOS 7.0；处理器：Intel(R) Xeon(R) E5-2620 v4；加速计算卡：NVIDIA Tesla P100；内存：32 GB；CUDA 版本为 9.0.

4.1　计算 N-Gram 特征实验

计算 1 G 字节文件的 N-Gram 特征速度对比结果如表 1 所示，使用 CPU 计算结果随着 N-Gram 窗口扩大，时间从 3.06 s 增长至 3.49 s 和 4.37 s，分别增加 14% 和 42.8%；而使用 GPU 计算 N-Gram 特征，时间均稳定在 0.48 s 左右，且不会随着 N 值扩大而增加，速度提升 7 倍. 随着恶意代码样本的增多和恶意样本文件大小增加，提升效果会更加明显.

表 1　特征计算速度对比图

Tab. 1　Comparison of feature extraction speed between CPU and GPU

处理时间(s)	N-Gram 窗口大小		
	2	3	4
CPU	3.06	3.49	4.37
GPU	0.47	0.48	0.48

4.2 统计 N-Gram 特征词频实验

使用 CPU 和 GPU 对 N-Gram 特征值进行排序,排序结果如表 2 所示,本次实验窗口大小 N 选择为 3.

表 2 特征排序速度对比图

Tab. 2 Comparison of feature sorting speed between CPU and GPU

处理时间(s)	特征值个数				
	1 万	10 万	1 百万	1 千万	1 亿
CPU	0.003	0.02	0.27	2.63	26.8
GPU	0.31	0.3	0.29	0.32	0.44

使用 CPU 排序时,随着元素增多,时间开销大幅增加,增长趋势十分明显,当元素数量从一千万增长到一亿时,时间开销从 2.63 s 增长到 26.8 s,增长了 11 倍;使用 GPU 进行排序时,因存在通信开销,元素数量较少时,排序时间开销略大于 CPU,但元素个数大于一百万时,GPU 排序取得明显优势,且增长趋势不明显,当元素数量为一亿时,速度提升 60 倍.

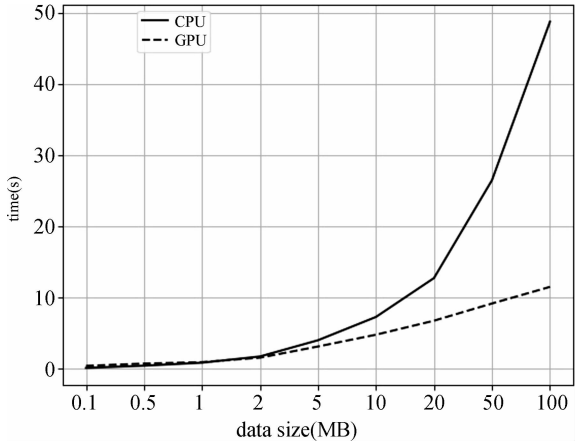

图 4 CPU 与 GPU 排序速度对比图

Fig. 4 Comparison of sorting speed between CPU and GPU

使用 CPU 和 GPU 分别统计样本每种 3-Gram 特征值词频,速度如图 4 所示,当文件大于 1 MB 时,GPU 统计特征值词频时间开销小于 CPU 开销,且文件越大,GPU 加速效果越明显. 所以在提取恶意代码字节 N-Gram 时,根据文件大小划分 CPU、GPU 任务的阈值为 1 MB. 当恶意代码文件小于 1 MB,使用 CPU 计算 N-Gram 特征值并进行频次统计;反之使用 GPU 加速计算.

4.3 计算 TFIDF 速度对比实验

CPU 与 GPU 计算 TFIDF 速度对比实验中,采用 Scikit-Learn 做对比项,Scikit-Learn 是一款基于 Python 的科学计算工具包,其中计算 TFIDF 功能不支持 GPU 加速计算. 实验中特征维度是 1 万维,即特征频次矩阵是 1 万列,横轴表示样本数量. 实验结果如表 3 所示,使用 GPU 计算 TFIDF 速度有 10 倍以上提升.

表 3 计算 TFIDF 速度对比图

Tab. 3 Comparison of computing speed on TFIDF

处理时间(s)	样本数量(万)			
	1	5	10	20
scikit	3.46	16.6	33.4	83.6
GPU	0.3	1.4	2.5	7.4

4.4 提取字节码特征速度对比实验

经过对 203237 个实验样本进行分析,样本文件大小大多处于 0~50 MB 区间. 4.2 小节中提出 CPU、GPU 任务划分的阈值为 1 MB. 为了保证实验公平性,将 0~50 MB 区间划分为四个区间(0~1 MB、1~10 MB、10~30 MB、30~50 MB),同时构建了三类数据集,每一类数据集样本数量均为 4000 个,处于不同大小区间的样本数量所占比例不同:第一类数据集,样本文件分为两种,大小处于 0~1 MB、1~10 MB 区间;第二类数据集,样本文件分为三种,大小处于 0~1 MB、1~10 MB、10~30 MB区间;第三类数据集,样本文件分为三种,大小处于 0~1 MB、1~10 MB、10~50 MB 区间. 为了方便实验,第三类数据集中 10~30 MB 区间与 30~50 MB 区间合并. 每类数据集中包含多个数据集,且小文件比例逐步减少,大文件比例逐步增加,具体所占比例如表 4 所示.

经过多次实验后,整体对比实验结果如图 5 所示,其中 N-Gram 窗口大小为 3,特征词频的删除阈值为 50,TFIDF 特征矩阵为 1 万维. 在第一类数据集 1~8 中,随着样本文件中较大文件比例提升,CPU 计算与 GPU 加速计算整体时间开销均成上升趋势,但 GPU 加速计算速度明显优于 CPU 计算;在第二类数据集 9~14 中,增加了 10~30 MB 样本文件,随着较大文件所占比例提高,CPU 计算整体时间开销快速增长,而 GPU 加速整体时间开销缓慢增长,加速比(CPU 计算时间开销/GPU 加速时间开销)明显提高至 3 倍以上,加速效果更佳;在第三类数据集 15~20 中,增加了更大的样本,随着大文件所占比例提高,CPU 计算整体时间开销依旧快速增长,而 GPU 加速整体时间开销缓慢增

长，加速比进一步提高至 4 倍以上，加速效果提升　　明显.

表 4　数据集各区间样本所占比例表

Tab. 4　The proportion of different interval files in the dataset

	数据集编号	0−1MB 样本所占比例	1−10MB 样本所占比例	10−30MB 样本所占比例	30−50MB 样本所占比例
第一类数据集	1	100	0		
	2	95	5		
	3	90	10		
	4	85	15		
	5	80	20		
	6	75	25		
	7	70	30		
	8	65	35		
第二类数据集	9	95	3	2	
	10	90	6	4	
	11	85	9	6	
	12	80	12	8	
	13	75	15	10	
	14	70	18	12	
第三类数据集	15	95	3		2
	16	90	6		4
	17	85	9		6
	18	80	12		8
	19	75	15		10
	20	70	18		12

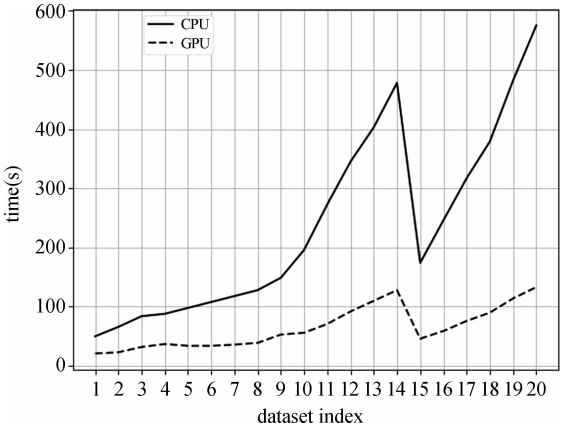

图 5　CPU 与使用 GPU 加速速度对比图

Fig. 5　Comparison of speed between CPU and using GPU acceleration

5　结　论

本文深入研究了 CUDA 并行计算架构和恶意代码特征提取方法，针对目前恶意代码字节码序列特征提取速度慢等问题，提出了 GPU 加速恶意代码字节码序列特征提取的方法，使用 CPU+GPU 异构平台，将计算字节码 N-Gram、统计特征值频次、特征筛选和计算 TFIDF 等计算密集型任务移交到 GPU 进行加速处理，大幅提高了特征提取的速度，提高恶意代码检测效率. 但是本文还存在一些不足，在特征统计完成后，特征频次检索还未能使用 GPU 加速，如何提高特征检索的速度也是今后研究的方向.

参考文献：

[1]　Schultz M G, Eskin E, Zadok E, *et al*. Data mining methods for detection of new malicious executables [C]// Proceedings of the IEEE Symposium on Security and Privacy. Washington: IEEE, 2001.

[2]　Kolter J Z, Maloof M A. Learning to detect and classify malicious executables in the wild [J]. J Mach Learn Res, 2006, 7: 2721.

[3]　Shafiq M Z, Tabish S M, Mirza F, *et al*. A framework for efficient mining of structural information to detect zero-day malicious portable executables [R]. Islamabad, Pakistan: Next Generation Intelligent Networks Research Cente, Citeseer, 2009.

[4] Shafiq M Z, Tabish S, Mirza F, *et al*. PE-Miner: mining structural information to detect malicious executables in realtime [C]//Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection. Berlin Heidelberg: Springer-Verlag, 2009.

[5] 郑荣锋, 方勇, 刘亮. 基于动态行为指纹的恶意代码同源性分析 [J]. 四川大学学报: 自然科学版, 2016, 53: 793.

[6] Kolter J Z, Maloof M A. Learning to detect malicious executables in the wild [C]// Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining. New York: ACM, 2004.

[7] Freund Y, Schapire R E. Experiments with a new boosting algorithm [C]// Proceedings of the thirteenth international conference on machine learning. New York: ACM, 1996.

[8] Quinlan J R. C4. 5: programs for machine learning [M]. San Francisco: Morgan Kaufmann, 1993.

[9] Li Z H, Jia H P, Zhang Y Q. HartSift: a high-accuracy and real-time SIFT based on GPU [C]// Proceedings of the IEEE International Conference on Parallel and Distributed Systems. Shenzhen, China: IEEE, 2018.

[10] 王志国. 局部特征算法 SURF 的 GPU 加速研究与实现 [D]. 北京: 清华大学, 2011.

[11] Li P, Luo Y, Zhang N, *et al*. HeteroSpark: a heterogeneous CPU/GPU spark platform for machine learning algorithms [C]// Proceedings of the IEEE International Conference on Networking, Architecture and Storage. Boston, MA, USA: IEEE, 2015.

[12] Liu Y, Jian L, Liang S, *et al*. Parallel data mining on CUDA-enabled graphics processing unit (GPU) [J]. e-Sci Technol Appl, 2010, 73: 2608: 1.

[13] Acosta J C, Mendoza H, Medina B G. An efficient common substrings algorithm for on-the-fly behavior-based malware detection and analysis [J]. IEEE, 2012, 2012: 1.

[14] Wang W, Zhu M, Zeng X, *et al*. Malware traffic classification using convolutional neural network for representation learning [C]// International Conference on Information Networking. Da Nang, Vietnam: IEEE, 2017.

引用本文格式:

中 文: 周紫瞻, 王俊峰. 基于 GPU 加速的恶意代码字节码特征提取方法研究 [J]. 四川大学学报: 自然科学版, 2019, 56: 227.

英 文: Zhou Z Z, Wang J F. Research on feature extraction of malware bytecode based on GPU acceleration [J]. J Sichuan Univ: Nat Sci Ed, 2019, 56: 227.