Overview

A survey on graphic processing unit computing for large-scale data mining

Alberto Cano*

General purpose computation using Graphic Processing Units (GPUs) is a well-established research area focusing on high-performance computing solutions for massively parallelizable and time-consuming problems. Classical methodologies in machine learning and data mining cannot handle processing of massive and high-speed volumes of information in the context of the big data era. GPUs have successfully improved the scalability of data mining algorithms to address significantly larger dataset sizes in many application areas. The popularization of distributed computing frameworks for big data mining opens up new opportunities for transformative solutions combining GPUs and distributed frameworks. This survey analyzes current trends in the use of GPU computing for large-scale data mining, discusses GPU architecture advantages for handling volume and velocity of data, identifies limitation factors hampering the scalability of the problems, and discusses open issues and future directions. © 2017 Wiley Periodicals, Inc.

How to cite this article:
WIREs Data Mining Knowl Discov 2017, e1232. doi: 10.1002/widm.1232

INTRODUCTION

The surge of large volumes of information to be processed by machine learning and data mining algorithms in the context of the big data era demand new transformative parallel and distributed computing solutions capable to scale computation effectively and efficiently.[1,2] Graphic processing units (GPUs) have become widespread tools for speeding up general purpose computation in the last decade.[3] They offer massive parallelism to extend algorithms to large-scale data for a fraction of the cost of a traditional high-performance CPU cluster.[4] The programming model provides data-level parallelism for the efficient processing of millions of threads which allows algorithms to scale to large-scale datasets not computable through traditional parallel approaches. Proof of the growing interest of GPUs for machine learning and data mining is the increasing number of research works in this area.[5,6] However, one cannot ignore the intrinsic high complexity of coding for GPUs and the careful design required to maximize the efficiency and occupancy considering the underlying architecture. Moreover, the relatively limited size of a single-GPU memory prevents its application to resolve big data problems comprising gigabytes or terabytes of data. Therefore, multi-GPU and distributed-GPU solutions are required to scale to even bigger datasets, which in turn introduces additional efficiency, latency, and synchronization challenges.[7]

Volume is seen as the most commonly discussed big data characteristic, as memory requirements and the computational complexity of algorithms depending on the data size are critical performance limitators. However, one cannot forget the importance of velocity, the speed at which data is being generated, demanding for real-time processing. Velocity becomes a predominant characteristic in many real-world data mining problems, as new instances are being continuously generated, forcing algorithms to provide very fast processing and adapt on-the-fly under real-time constraints. This has led to the

*Correspondence to: acano@vcu.edu

Department of Computer Science, Virginia Commonwealth University, Richmond, VA, USA

Conflict of interest: The author has declared no conflicts of interest for this article.

notion of data streams,[8] high-speed continuous data flows demanding very fast decision models. Not only GPUs allow for the processing of large-scale volumes, but they also offer fast model decisions by exploiting the asynchronous model of concurrent execution and data transferring. This is a clear advantage of GPUs for real-time systems where a timely prediction is critical, such as image processing for autonomous driving[9] or high-frequency trading and analytics.[10]

In recent years, newly distributed frameworks have emerged to address the scalability of algorithms to big data analysis using the MapReduce programming model,[11] being Apache Hadoop[12] and Spark[13] the two most popular implementations. The main advantages of these distributed systems are their elasticity, reliability, and transparent scalability in a user-friendly way. They are intended to provide users with easy and automatic fault-tolerant workload distribution without the inconveniences of taking into account the specific details of the underlying hardware architecture of a cluster. However, their simplicity and higher level of abstraction comes at the cost of less efficient implementations and no explicit control of scheduling in distributed environments. Importantly, these popular distributed computing frameworks and GPUs are not mutually exclusive technologies, as they can complement each other and target complementary computing scopes. Indeed, recent studies focus on hybrid solutions combining both technologies in order to take advantage of the automatic and dynamic workload distribution along with the exploitation of the GPUs horsepower to efficiently compute local data subsets.[14–18] This way, these systems benefit from the joint advantages and alleviate the individual limitations, especially in terms of memory spatial locality and capacity, and performance throughput.

There are two surveys on the use of GPU computing for general-purpose computation and data-parallel problems.[3,19] These surveys are focused on the architecture of the GPU and its programming model, the implementation of parallel computing models, strategies for designing efficient algorithms and operations (e.g., map, reduce, sort, filter, scatter, gather, search, segmentation), strategies for efficient data access patterns and structures, and general-purpose applications in data-parallel intensive simulations. Furthermore, there are two other surveys on the use of GPU computing for data mining.[5,6] These surveys are focused on the implementation of a reduced number of classical data mining algorithms (K-means, KNN, Apriori, FP-Growth) on a GPU. However, these latest works limit their scope to the implementation of the particular algorithms in a single-GPU, which are no longer state-of-the-art methods in data mining, especially when considering their scalability to big data, which is a crucial issue today. Nevertheless, all these works are very useful to illustrate the architecture of a GPU, the programming model, and the implementation of efficient data-parallel strategies.

This survey provides an overview and insight on the use of GPU computing for large-scale data mining, focusing on the advantages and limitations of GPUs for big data on state-of-the-art data mining algorithms, complementing and updating previous surveys on the status of this research area. First, it discusses how the GPU architecture is organized and its evolution, the programming model, and the advantages to match the computational requirements of large-scale data mining algorithms. Second, it analyzes current trends and research works on the implementation of different popular families of data mining techniques using GPUs, discussing specific optimizations and noteworthy results. Third, it reviews particular application areas where GPUs have significantly assisted to speedup real-world problems requiring fast processing of large data volumes in data mining. Fourth, it discusses novel research works on the integration of GPU computing with popular MapReduce frameworks for distributed computing from two perspectives: (1) implementations of the MapReduce programming model within a GPU and (2) exploitation of GPUs for computing the jobs in the nodes of a distributed cluster using existing MapReduce frameworks. Finally, it summarizes current GPU limitations, future architectures advantages, open challenges, and future directions to pursue in order to keep increasing the scalability of parallel data mining algorithms to the ever-increasing size of data.

GPU ARCHITECTURE FOR DATA MINING

This section presents the fundamentals of the NVIDIA GPU architecture and programming model, and discusses the potential advantages for speeding up big data mining algorithms.

GPU Architecture and Programming Model

GPUs are many-core architectures highly suitable for massively data parallel general-purpose computation. They consist of a number of multiprocessors, each of which contains a set of cores operating in a single instruction multiple data (SIMD) fashion, i.e., capable of synchronously processing an

TABLE 1 | NVIDIA graphic processing unit (GPU) and Intel CPU Architectures and Performance Evolution

Year	GPU Name	Cores	GFLOPS	Memory	Bandwidth	CPU Name	Cores	GFLOPS	Bandwidth
2006	Tesla G80	128	346	768 MB	86 GB/second	Core	6	60	8 GB/second
2008	Tesla GT200	240	708	1024 MB	159 GB/second	Nehalem	8	90	10 GB/second
2010	Fermi	512	1581	1536 MB	192 GB/second	Westmere	10	112	24 GB/second
2012	Kepler	2304	3976	3072 MB	288 GB/second	Sandy Bridge	12	480	50 GB/second
2014	Maxwell	3072	6144	12,288 MB	336 GB/second	Ivy Bridge	15	510	58 GB/second
2016	Pascal	3584	11,366	16,384 MB	547 GB/second	Broadwell	24	1340	64 GB/second
2018	Volta	5120	15,154	16,384 MB	900 GB/second	Coffee Lake	32	2010	96 GB/second

instruction on multiple data. Today's GPUs comprise thousands of relatively simple cores for parallel computing of very fast arithmetic and logical control, in contrast to the small number of complex cores of a high-end multicore CPU, which are optimized for fast sequential serial processing, out of order execution, and branch prediction.

The NVIDIA Volta GPU architecture comprises 64 cores per multiprocessor and sums 80 multiprocessors, adding a total of up to 5120 cores, 6144 KB of L2 cache, and up to 16 GB HBM2 high-bandwidth memory. One of the new capabilities in Volta is the ability to process 16-bit precision instructions and data, providing twice the throughput and capacity of 32-bit precision then achieving up to 30 TFLOPS FP16 performances. Moreover, it incorporates 640 Tensor cores designed specifically for deep learning, which deliver up to 120 TFLOPS. These numbers illustrate the impressive progress in the performance of GPU architectures since the introduction of the Tesla architecture in 2006, which only counted with 128 cores, 768 MB memory, and 346 GFLOPS FP32 performance. Table 1 compares the performance evolution of NVIDIA GPU and Intel CPU architectures in recent years. Not only has the number of cores in a GPU increased but also the memory bandwidth has significantly experienced great improvements, which allows to speedup data-intensive applications. On the other hand, CPUs have slowly progressed in performance numbers, particularly in memory bandwidth.

GPUs comprise less FP64 compute units than FP32 because they focus on single-precision performance, ratios depend on the particular architecture, e.g., 1:2 in Volta. Unlike other technical computing applications such as numeric simulations that require high-precision floating-point computation, data mining seldom requires FP64. For instance, deep neural networks have a natural resilience to errors due to the backpropagation algorithm used in training. Simpler precision also favors better generalization, avoiding the overfitting of a network to the data. Storing half-precision FP16 data compared to higher precision FP32 or FP64 reduces memory usage of the neural network and thus allows training and deploying larger networks. Using FP16 computation doubles performance compared to FP32.

GPUs' architecture is shown in Figure 1 and it comprises a global memory accessible by all threads, a small low-latency shared memory accessible by all threads in a block, and local memory per core in the form of registers for a given thread. Global memory is a high-bandwidth large device memory that permits high-efficient memory access patterns by using data coalescing. Coalesced memory reflects the optimal scenario where consecutive threads access to consecutive data memory positions, minimizing the number of memory transactions. On the other hand, irregular access patterns will penalize the memory performance with crossed relations between data and threads. Coalescing data access patterns and minimizing threads divergence are the most influential optimizations to maximize performance. Shared memory (multiprocessor level) allows for synchronization and combination of intermediate data, fitting iterative algorithms where contribution of individual threads is reintroduced into the model. However, shared memory is limited to 96 KB per multiprocessor in Volta. Finally, each thread has access to 255 32-bit registers to store local variables.

There are two main computing platforms for GPUs: Open Computing Language (OpenCL) developed by Khronos, and compute unified device architecture (CUDA) developed by NVIDIA. OpenCL provides compatibility across heterogeneous hardware from any vendor, whereas CUDA is specifically designed for NVIDIA GPUs. CUDA is more frequently used than OpenCL in the data mining research community. It provides a programming interface for C, C++, and Fortran in which the GPU code is defined in *kernel functions*, while the CPU functions and logic is managed by the *host*. CUDA offers a number of GPU-accelerated libraries for

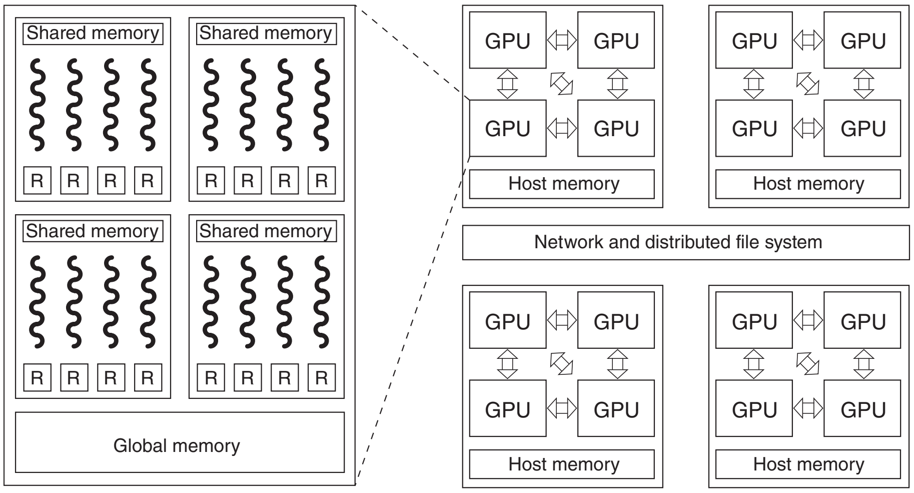


FIGURE 1 | Graphic processing unit (GPU) architecture, multi-GPU, and distributed-GPU scalability.

highly-optimized algorithms such as cuDNN for deep neural networks, nvGRAPH for graph analytics, NVBIO for high-throughput sequence analysis, and many other for numeric simulation.

The CUDA programming model provides a flexible hierarchical organization of threads and memory. Threads are grouped into user-defined three-dimensional (3D) *thread blocks* of up to 1024 threads. Thread blocks are also grouped into a 3D *grid* of thread blocks. This configuration allows the user to define the parallel computation of multidimensional problems with millions or billions of threads. Threads are mapped into the GPU multiprocessors through *warps* which comprise a group of 32 threads (warp size). The GPU scheduler maximizes the occupancy and performance by switching idle warps waiting for data accesses or function results with other warps ready for computation. Figure 2 illustrates the threads and blocks hierarchy of the multidimensional space of computation in CUDA.

Figure 3 shows an example in CUDA of the computation of the pairwise Euclidean distances in an $n \times m$ matrix, a common task in data mining for measuring similarity.[20] Threads are organized into 16 × 16 two-dimensional blocks, and the blocks are organized into an $n/16 \times n/16$ two-dimensional grid. Threads load submatrices in shared memory for low latency access, compute partial distance, and synchronize results. Once all partial results have been computed, the square root of the squared sum of the differences is calculated.

GPU's Suitability for Data Mining

The flexible architecture and programming model of a GPU facilitates the adaptability of massively parallel data mining algorithms. Granularity in the form of data-level parallelism schedules a thread to compute a single data instance, speeding up any data mining problem where the computational complexity lies on the data size. For instance, many algorithms involve distances computation among data instances to measure similarity metrics such as in K-means clustering and nearest neighbor classifiers. This approach provides excellent scalability to ever-increasing volumes in large-scale data mining, addressing data sizes not computable using traditional CPU-parallel programming. However, limitations surge for massive big data processing. NVIDIA Volta GPUs comprise a maximum of 16 GB memory per device, preventing them from handling terabyte scale data. Once the data size is bigger than the capacity of the GPU memory, the performance decreases significantly as the data transfers to the device become the primary bottleneck, limited by the bandwidth and latency of the PCIe bus.

The Volta architecture has introduced mixed-precision Tensor cores purpose-built for deep learning matrix arithmetic providing up to 6x higher peak TFLOPS compared to the previous Pascal architecture. Tensor cores help to speedup Matrix–Matrix multiplication operations, which are essential in neural network training, and are used to multiply large matrices of input data and weights in the connected layers of the network. Tensor cores operate on FP16 input data with FP32 accumulation, multiplying and adding 4 x 4 x 4 matrices. Not only does deep learning benefit from these *ad-hoc* Tensor cores, but also many other data mining techniques employing similar computation strategies will take advantage of the dedicated hardware.

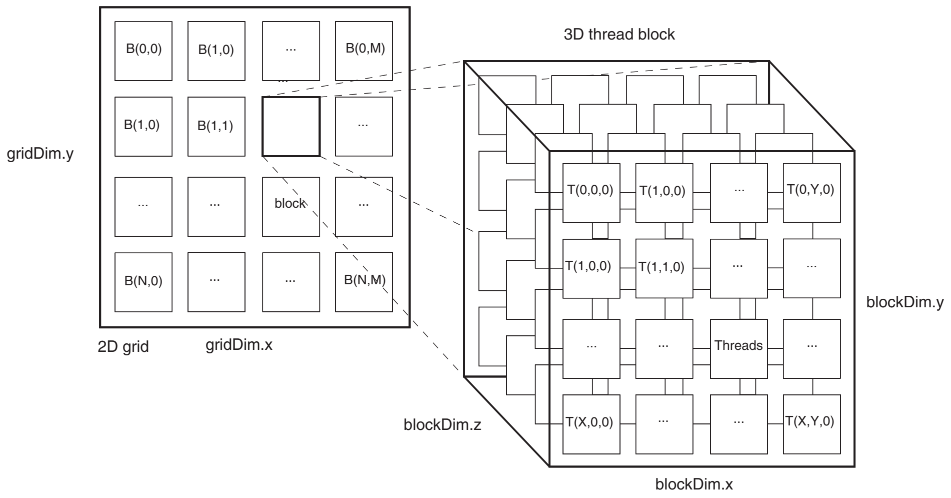


FIGURE 2 | Compute unified device architecture (CUDA) threads and blocks multidimensional programming model.

Researchers have proposed a number of general frameworks for implementing data mining algorithms on GPUs. Fang et al.[21] introduced GPUminer, a parallel data mining system that implemented clustering (*K*-means) and pattern mining (Apriori) algorithms. Ma and Agrawal[22] analyzed the common code structures in data mining algorithms using program analysis and code generation to extract programming patterns to facilitate the mapping of algorithms to GPUs, they compared CPU and GPU implementations of *K*-means and EM clustering, achieving a speedup of up to 50x as compared with the sequential single-threaded CPU implementation. Gainaru and Slusanschi[23] presented a framework offering a general methodology for parallelizing all types of data mining applications on hybrid architectures, improving the results of GPUMiner. They focused on K-means, KNN, Apriori, FP-Growth algorithms measuring speedups, latency, and utilization.

Other studies focused on proposing extensions of existing popular data mining software such as Weka and RapidMiner to GPUs.[24,25] Specifically, Engel et al.[24] profiled the Weka code to identify sets of time-consuming operations that could be easily adapted to the GPU, while *Kovacs* integrated a GPU plugin into RapidMiner through JCUDA and JNI, implementing a nearest neighbor classifier in a GPU, achieving a speedup as high as 171x as compared to a quad-core CPU. However, general frameworks have not succeeded because it is very difficult to have a unique general-purpose adaptation of every data mining algorithm to GPUs, but in practice, *ad-hoc* implementations are required per algorithm to optimize code to the architecture. Coding of the individual algorithms is a time-consuming task and GPU programming implies a relatively difficult learning curve, but allows for the implementation of specific algorithm's optimizations to the GPU architecture that will maximize the occupancy of the hardware and improve the algorithm's speedup.

DATA MINING TASKS AND TECHNIQUES

This section presents a literature review of the most relevant works on the implementation of data mining algorithms on GPUs. It is structured according to the data mining tasks and techniques. Due to the vast number of research works in this area, we will focus on the most significant, recent, and highly influential contributions of GPUs to large-scale data mining.

Clustering

Clustering groups data into several clusters attempting to aggregate together instances with similar characteristics.[26] They are based on similarity joins, a basic operation which computes similarities of feature vectors using a similarity (distance) function. Similarity join is a function for data examples comparisons widely used in instance-based learners, easily parallelizable in a GPU as it was illustrated in Figure 3. The problem of computing pairwise distances among data instances is $\mathcal{O}(n^2)$ complex. Therefore, it is not compute in reasonable time on a CPU, whereas GPUs may speedup this problem by calculating in parallel the distances among every pair of instances. Figure 4 illustrates the pairwise distance computation among every data instance and the

```
// in: matrix size nxm, out: distances array
__global__ void distance(float *out, float *in, int n, int m)
{
  __shared__ float Ys[16][16]; // shared memory submatrix
  __shared__ float Xs[16][16]; // shared memory submatrix

  int yBegin = blockIdx.y * 16 * m;  // y block index
  int xBegin = blockIdx.x * 16 * m;  // x block index
  int yEnd, y, x, k, o;
  float tmp, s = 0;

  for(yEnd = yBegin+m-1, y=yBegin, x=xBegin; y<=yEnd; y+=16, x+=16) {
    // load submatrices (Xs transpose)
    Ys[threadIdx.y][threadIdx.x] = in[y + threadIdx.y*m + threadIdx.x];
    Xs[threadIdx.x][threadIdx.y] = in[x + threadIdx.y*m + threadIdx.x];

    __syncthreads(); // wait threads data load

    for(k=0;k<16;k++){
      tmp = Ys[threadIdx.y][k] - Xs[k][threadIdx.x];
      s += tmp*tmp;
    }

    __syncthreads(); // wait threads partial distance computation
  }

  o = blockIdx.y*16*n + threadIdx.y*n + blockIdx.x*16 + threadIdx.x;
  out[o] = sqrtf(s); // store final result in given index
}
```

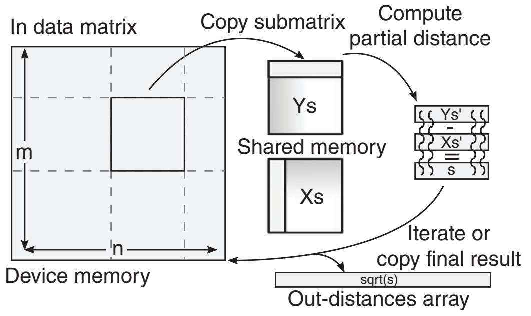


FIGURE 3 | Compute unified device architecture (CUDA) example for pairwise Euclidean distances computation on graphic processing units.

subsequent distance calculation to the data clusters centroids (three clusters).

Density-based clustering creates clusters using areas of high density separated by other areas of lower density.[27] For instance, Andrade et al.[28] presented a GPU implementation of DBSCAN, a density-based algorithm which innovates on the use of an efficient data indexing using graphs, achieving speedups over 100 times than its sequential CPU version. *K*-means clustering partitions data into a predefined number of *k* clusters. There are a number of GPU implementations for *K*-means. Huang et al.[29] transformed the operation of weighting *K*-means to the combination of multiplication, addition, and element-wise operations among vectors and matrices. Serapiao et al.[30] evaluated in parallel each individual of a swarm-based extension of *K*-means to avoid local optima, achieving a speedup of up to 46x when comparing a NVIDIA 460 GTX with an Intel i7 CPU. Li et al.[31] proposed different compute strategies depending on the dimensionality of the data. Specifically, for low-dimensional data, they exploit on-chip registers to decrease data latency, whereas for high-dimensional data not fitting in registers, they conduct matrix multiplications on shared memory to improve the compute-to-memory-access ratio. Experimental results show their approach is up to eight times faster than other *K*-means implementations on GPUs. Al-Ayyoub et al.[32] extended the GPU implementation to fuzzy C-means, achieving speedups 30 times faster than the serial implementation. Takizawa and Kobayashi[33] presented a divide-and-conquer hierarchical parallel clustering combining GPUs in multiple machines with message passing interface. However,

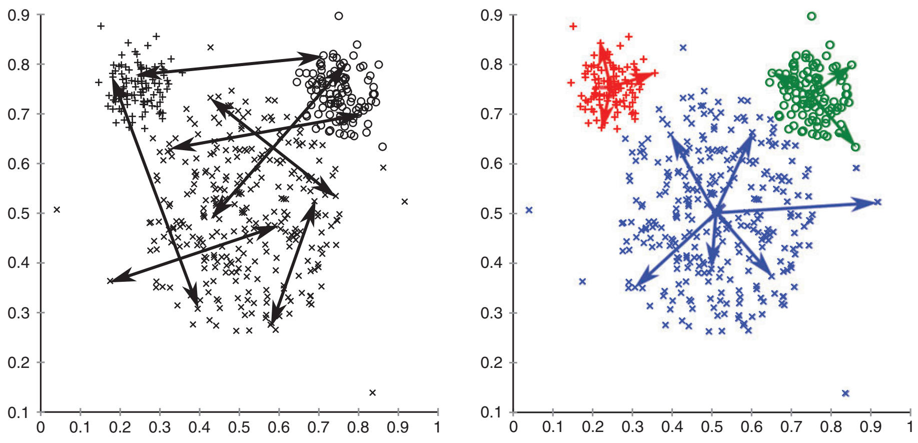


FIGURE 4 | Parallel pairwise distance computation and centroid-based clustering.

this approach requires explicit control on distribution and synchronization which in turn increases the difficulty of the code.

Pattern Mining

Pattern mining focuses on finding frequent itemsets and relationships in collections of data. There is a large number of implementations concerning the popular Apriori and FP-growth algorithms on GPUs. Teodoro et al.[34] proposed a multilevel tree projection frequent itemset mining algorithm on GPUs and several strategies to mitigate concurrent race conditions. Results indicate an impressive speedup up to 173x when using an NVIDIA GTX 470 compared with the multithread implementation in a quad-core processor. Analysis of the scalability reported better performance as the number of transactions in the database increased. Li et al.[35] presented a new multilayer vertical data structure and algorithm for mining frequent itemsets on GPUs. They use a unfixed-length bit vectors representation that allows to obtain high speedups, especially on large-scale sparse data.

Wang and Yuan[36] introduced an efficient methodology to build FP-Growth on GPUs without candidate generation. Rather than following the inherently sequential process of building a traditional FP-Tree, they propose to build an equivalent parallel binary radix tree as illustrated in Figure 5 (1). Similarly, counting the support (number of occurrences) of the itemsets (set of items that occur together) would be $\mathcal{O}(n)$ complex using a sequential approach, whereas the parallel reduction approach on GPUs provide $\mathcal{O}(\log n)$ complexity as in shown Figure 5 (2). The problem is the increasing number of idle threads in each reduction step.

Zhou et al.[37] innovated by introducing a compact data structure to store the entire dataset in the limited size of a GPU memory. Indeed, efficient data structure for pattern mining is a recurrent research area.[38] Zhang et al.[39] proposed GPApriori, which utilizes trie-based candidate set, vertical data layout, and bit set representation of vertical transaction lists. Moreover, Apriori is easily extensible to multiple GPUs to overcome memory and performance limitations.[40] Association rules are popular representations of the findings of pattern mining algorithms.[41] Cano et al.[42] presented a generalized methodology for speeding up association rules evaluation on multiple GPUs. Djenouri et al.[43] proposed a bee swarm optimization for association rules on GPUs, obtaining speedups 100 times faster using a Tesla C2075 GPU than the sequential mono-core CPU implementation. However, they point out the highly cost of the data communications between the CPU and GPU, and the significant performance impact of branch divergence of threads within the warp.

Nearest Neighbor Classifiers

The k nearest neighbor (KNN) classifier is one of the most popular data mining techniques[44] and has been implemented in many GPU works.[45,46] Most of the GPU implementations of the KNN comprise two main components. First, compute the similarity (distance matrix) containing the distances between test and train data. Second, sort the distance matrix. The computational and memory complexity to calculate and store all pairwise distances is $\mathcal{O}(n^2)$. Therefore, GPU memory capacity imposes limitations on the size of data, decreasing the performance of the KNN implementations when data is constantly transferred from the system's memory to the GPU's memory. Given the 16 GB memory of the Volta architecture, it is not possible to allocate at once all pairwise distances for datasets having more than 100 k instances. Thus, preventing KNN methods to scale to big data.

Researchers have proposed solutions to overcome the memory limitations of KNN on GPUs. Arefin et al.[47] proposed to reduce the usage of memory by dividing the computations in square-shaped portions, but the required data structures also limit the performance. Gutierrez et al.[48] introduced an incremental neighborhood computation scheme that eliminates the dependencies between dataset size and memory. It takes advantage of asynchronous memory transfers, making the data structures fit into the available memory while delivering high run-time performance independently of the data size. Figure 6 illustrates the behavior of the local and incremental neighborhood selection for finding the k closest neighbors. The methodology consists of an iterative process where train data (very large) is split into multiple subsets. All test data (small size) is copied to the device in the beginning. The iterative process copies several train data subsets into the GPU memory (as many as they fit). Then, the algorithm runs the local neighborhood search to find the k closest neighbors from the test data to the train data subsets. A tentative candidate pool of k neighbors is identified as potential solutions. The process continues merging existing candidates with new train data subsets. Finally, the true k global closest neighbors remain as the correct solution.

Other alternatives have emerged to reduce the memory complexity. Rocha et al.[49] proposed a compact data structure to represent sparse datasets for efficient GPU KNN data representation and distances

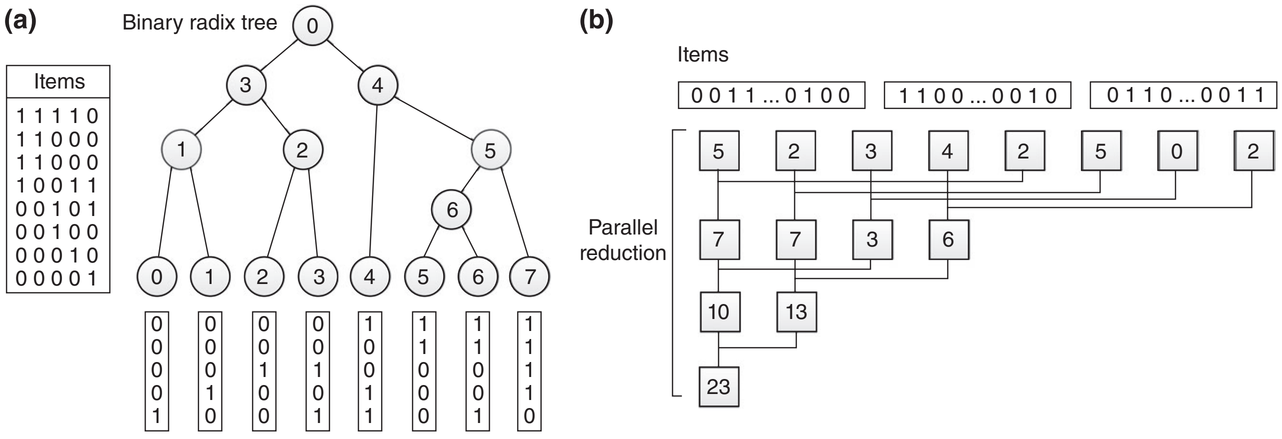


FIGURE 5 | (a) Binary radix tree and (b) parallel support reduction.

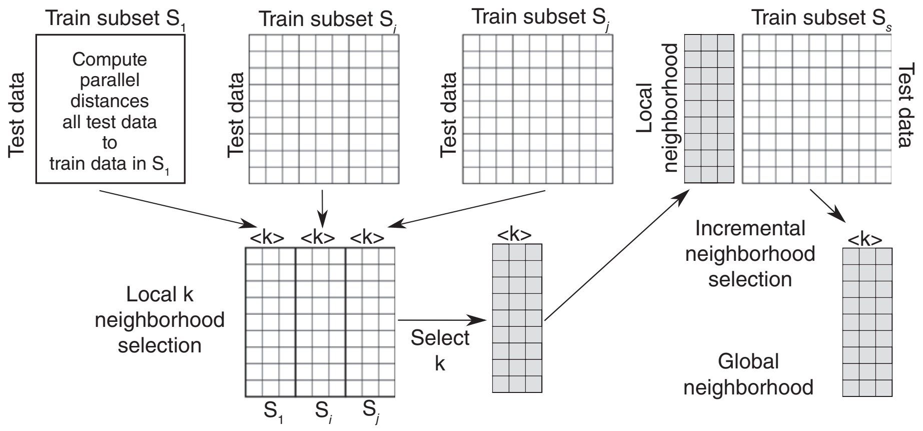


FIGURE 6 | Distance matrices for *k* nearest neighbor (KNN) local and incremental neighborhood selection.

computation. Masek et al.[50] presented a multi-GPU implementation scalable to four devices which splits train data uniformly into the many devices. The main advantage is that no intercommunication among GPUs is required and therefore using multiple GPUs does not introduce any additional overhead, which is a clear advantage. This is true as long as the runtime per device is constant by distributing uniform workloads among homogeneous GPUs, or by efficiently balancing the jobs on heterogenous devices.[51] GPU implementations of KNN do not only take advantage of computing distances per instance in parallel, but also may also parallelize the distance calculation for a single instance, which is especially interesting in high dimensional spaces with very large number of attributes.[52] These advances on efficient KNN parallel implementations also improve velocity of the response, a significant aspect for big data processing, allowing to provide fast decision for high-speed data streams.

Classification Rules

Classification rules are popular data mining techniques because they provide interpretability of the prediction model, which is vital in many decision support systems where reasons and motivation for the prediction must be given. Evolutionary algorithms, and specifically genetic programming, are nature-inspired heuristics commonly employed in literature for learning classification rules.[53–56] The main drawback of evolutionary algorithms in data mining is the computational complexity and runtime because they evolve a population of solutions for a number of generations.[57] In every iteration, the tentative solutions must be evaluated according to a fitness function and the train data. Thus, their computational complexity is $\mathcal{O}(n \times P \times G)$ where P is the population size and G is the number of iterations, which makes sequential approaches unsuitable for large-scale data. The main advantage is that the fitness function is massively parallelizable. On one hand, the evaluation of every individual in the population is an independent task, i.e., the fitness of the individuals can be computed in parallel. On the other hand, the evaluation of a given individual on each data instance is also an independent parallel operation, i.e., it scales as for the data size. Therefore, many research works propose both individual and data level parallelism because it perfectly matches the programming model of GPUs, maximizing the speedups of these algorithms for learning classification rules on large-scale data.

Franco and Barcardit[58] employed GPU computing to speedup the fitness computation of evolutionary algorithms for large-scale data mining using different coarse-grained and fine-grained strategies, but their approach is limited to a single device. Langdon[59] focused on the implementation of the genetic programming interpreter on a GPU. The interpreter is the function which executes the genetic programming tree representing the classification rule. The problem is that the postfix notation commonly employed for representing the rules requires push and pop operations on a stack, which is not the most efficient approach for a GPU. This was the motivation for Chitty[60] to conduct interpreter optimizations through exploitation of on-chip memory. Cano et al.[61–63] proposed a number of implementations of evolutionary rule-based classifiers on GPUs. In Ref 64 they presented an implementation using Ant Programming. In Ref 65, they proposed an implementation for Pittsburgh classifiers, which increase the computational complexity by representing an individual as a full classifier (set of rules) rather than individual rules. Extensions of these rule-based classifiers were proposed for multi-instance learning.[66] The main advantages of these proposals are their transparent scalability to multiple GPUs, since populations subsets may be assigned easily to different devices without any kind of additional overhead.

Once resolved both the parallelization of multiple rules on multiple data, a significant contribution was proposed by Cano and Ventura[67] on the intrarule parallelization of a rule evaluation. Figure 7 illustrates the idea that the evaluation of branches can also be computed in parallel in a bottom-up fashion. This way, the computational complexity of evaluating a rule is reduced from $\mathcal{O}(n)$ where n is the number of nodes to $\mathcal{O}(d)$ where d is the depth of the tree. However, performance increase depends on how well balanced is the tree. This way, in the case of a perfect full binary tree the computational complexity becomes $\mathcal{O}(log\ n)$. On the other hand, the worst-case scenario happens when evaluating an unbalanced tree because there are no possible parallelization opportunities and all operations must be conducted sequentially, i.e., $\mathcal{O}(n)$ complex. Another significant advantage is that it does not require a stack to maintain temporal results for the internal operators, since the intermediate results for a depth level are immediately used in the next iteration higher level.

Decision Trees

Decision trees build classification models in the form of a tree structure containing decision nodes and leaf nodes. They are intrinsically related with classification rules, as they can be converted into each other, and both provide interpretability to the inferred models. Thus, similar GPU approaches can be applied for parallelizing decision trees on GPUs. Chiu et al.[68] presented a decision tree model in which the key is finding the split points for the attributes in parallel, following a top-down fashion. Nasridinov et al.[69] proposed a ubiquitous parallel approach for decision tree construction on two levels. They apply parallelism at the outer level of building the tree node-by-node, and at the inner level of sorting data records within a single node. Decision trees are also commonly employed in Random Forest ensemble

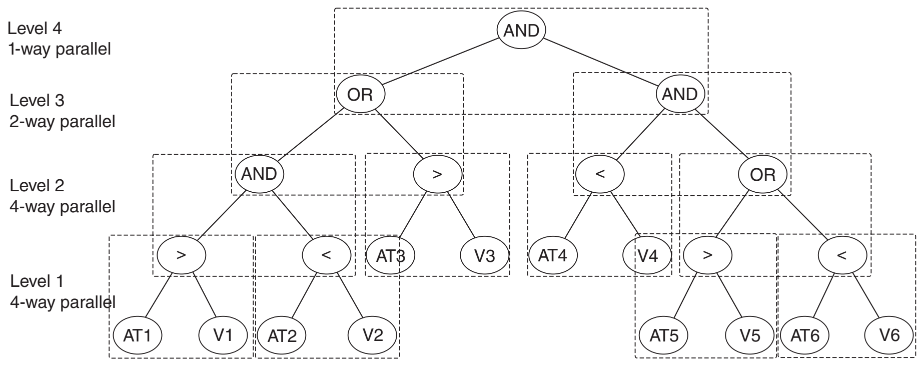


FIGURE 7 | Intrarule parallelization in genetic programming.

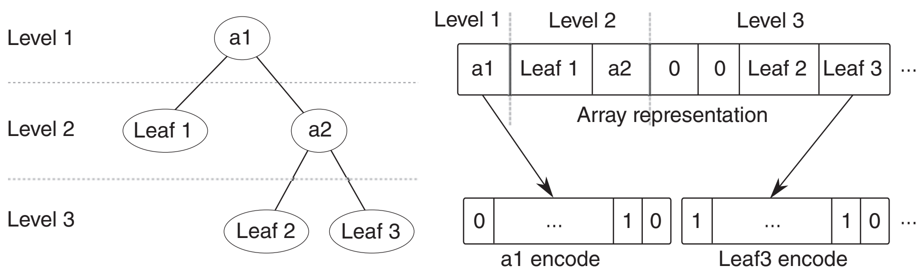
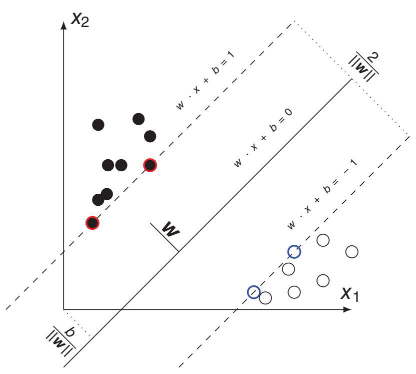


FIGURE 8 | Decision tree encoding in graphic processing unit linear memory.

learning. They operate by constructing a multitude of decision trees and outputting a prediction by combining the outputs of the individual trees. Grahn et al.[70] introduced one of the first implementations of Random Forest on GPU. Results showed to outperform 30x faster than the Weka CPU sequential implementation. Janssone et al.[71] presented two GPU implementations of the ensemble learning methods Random Forests and Extremely Randomized Trees. They also address multi-GPU scalability to add both memory throughput and additional cores for increased computational power. Marron et al.[72] adapted the GPU implementation of Random Forest for mining evolving high-speed streams, where having a timely prediction is critical. Experimental results showed to be able to provide class prediction in less than 1 second, achieving a speedup of up to 25x as compared with the traditional single-thread implementation in the MOA software. They employ a breadth-first traversal to encode the binary tree in a single array, level by level, as illustrated in Figure 8. This representation is again best for perfect full binary trees, whereas unbalanced trees will waste memory within the array. Each node is encoded through 32-bit unsigned integers containing the node ID and a flag. Using this layout, it is easy to obtain the child offset of the child nodes of any given node. Given a node at offset i, its left child will be at offset $2i + 1$ and its right child will be at offset $2i + 2$.

Support Vector Machines

Support vector machines (SVMs) construct hyperplanes that represent the best separation between data classes. Lu et al.[73] and Drozda et al.[74] published the most recent surveys on the use of GPUs for accelerating SVMs. SVMs benefit from parallel GPU computing since their computational complexity lies on the necessity of calculating a large number of matrix by vector multiplications[75] as illustrated in Figure 9. Therefore, they will also take advantage of the increased performance provided by Volta Tensor cores. GPU-LIBSVM[76] is a modification of the popular LIBSVM library that exploits GPU processing while producing identical results. Rgtsvm[77] is a SVM package for GPU architecture based on the GTSVM software, which takes full advantage of the GPU architecture and efficiently handles sparse datasets through clustering techniques. The larger the number of support vectors, the better speedup it can be obtained as compared with the sequential CPU implementations. Catanzaro et al.[78] presented the first implementation of a SVM in a GPU based on the Sequential Minimal Optimization algorithm, achieving speedups up to 35 times faster than



$$\underset{w}{\text{Minimize}} \quad \frac{1}{2}\|\mathbf{w}\|^2$$

$$\text{subject to} \quad y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1$$

$$\mathbf{Y} \cdot (\mathbf{X}\mathbf{w} + b) \geq \mathbf{1}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \cdot \left(\begin{bmatrix} x_{11} & x_{12} & \dots & x_{1d} \\ x_{21} & x_{22} & \dots & x_{2d} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nd} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} + b \right) \geq \mathbf{1}^n$$

FIGURE 9 | Support vector machine (SVM) formulation and matrix vector multiplication.

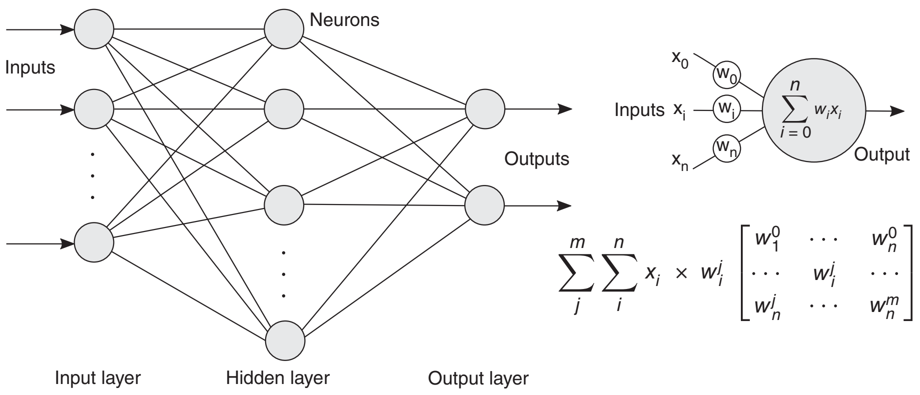

Inputs
Neurons
Outputs
Input layer
Hidden layer
Output layer
Inputs x_0 x_i x_n
w_0 w_i w_n
$\sum_{i=0}^{n} w_i x_i$
Output
$\sum_j^m \sum_i^n x_i \times w_i^j \begin{bmatrix} w_1^0 & \cdots & w_n^0 \\ \cdots & w_i^j & \cdots \\ w_n^j & \cdots & w_n^m \end{bmatrix}$

FIGURE 10 | Neural network architecture, activation function, and neuron weight updates.

LIBSVM. Herrero et al.[79] implemented a multiclass SVM by conducting one versus all decomposition and integrated GPUs into a MapReduce cluster. The main advantage was the ability to run all decomposed classification models at the same time, which in turn, increments the parallelization granularity and speedup. Yan et al.[80] accelerated both the kernel matrix calculation and cross validation on the GPU, but speedups were limited especially in small datasets for which it is not worthy to transfer computation to GPUs. Benatia et al.[81] proposed a sparse matrix format for multiclass SVM. The sparse representation allows to increase the dimensionality of the large-scale data to compute. This is a recent trend in GPU SVM implementations which focus on the use of sparse matrix formats that allow for processing large-scale sets overcoming the limitations of memory size.

Neural Networks

Neural networks are interconnected large groups of nodes (neurons) organized in multiple layers as illustrated in Figure 10. The parallel structure and behavior of a neural network can be easily modeled in a GPU. Weights and inputs are computed in the form of parallel matrix multiplications. Sierra et al.[82] introduced an implementation of the back-propagation algorithm on a GPU using the CUDA Basic Linear Algebra Subroutines (cuBLAS) library. Brito et al.[83] improved the back-propagation implementation by parallelizing all the weights updates. Each thread is responsible for a single operation between an input and weight. Li et al.[84] focused on the scalability to large-scale data on recurrent neural networks with a fine-grained two-stage pipeline architecture. Strigl et al.[85] implemented the convolutional neural networks on a GPU. The convolution operation can be efficiently addressed in the architecture of a GPU, especially in big matrix sizes where speedup is maximized. Juang et al.[86] extended the GPU implementation to a fuzzy neural network. They map large sets of input data into thread blocks, and divide the datasets into smaller chunks transferred to the multiprocessor shared memory, minimizing memory latency. Their model scales better on a large number of input features. However, limitations of shared memory size impose a performance bottleneck. Li et al.[87] presented two scheduling light-memory-cost algorithms to improve the running speed and use a larger convolutional model overcoming the shortage of GPU memory. Experimental results provide speedup of 7x compared with the NVIDIA cuDNN GPU software.

Deep Learning

Deep learning algorithms are becoming a focus of attention for many research studies and industrial applications due to their superior performance. The complexity of training deep learning models is related to handling large size models and large datasets. Hence, the programming patterns used on GPUs for computing deep learning are basically two: (1) divide the dataset and train the same model on different data (data parallelism) and (2) divide the model and run different sections of the network in different devices using a pipeline-like approach (model parallelism). Figure 11 illustrates and compares these two approaches. Chen et al.[88] implemented a deep belief network on a GPU using the cuBLAS library. However, the training procedures of deep belief networks are highly serial and dependent, which makes it difficult to convert into parallel form. Chen et al.[89] developed a pipelining system for image deep learning on a GPU cluster to leverage the heavy workload of training procedure. They organized the training of multiple deep learning models in parallel, where each

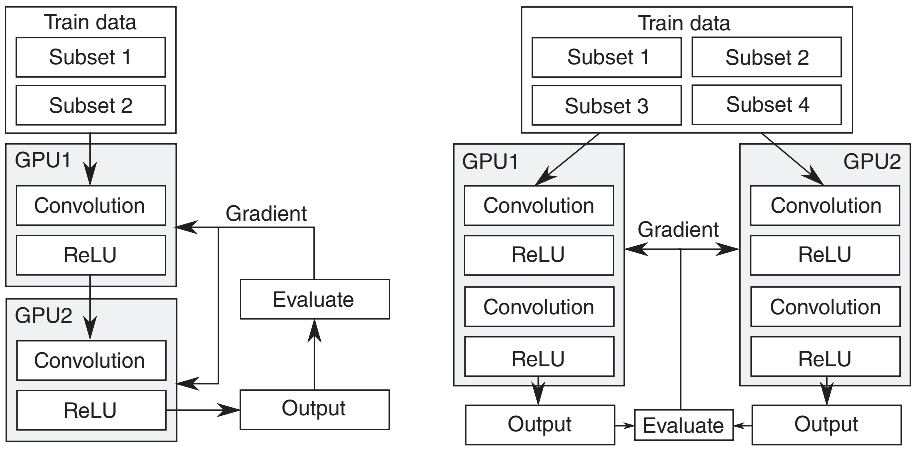


FIGURE 11 | Model and data parallelism in deep learning.

stage of the pipeline is managed by a particular GPU with a partition of the train data. First, this is an effective approach to scale to bigger data by splitting partitions into multiple devices. Second, it is more efficient since rather than migrating data, their proposal moves partially trained models to reduce bandwidth consumption. Krizhevsky[90] developed a deep convolutional neural network to classify high-resolution images in the ImageNet contest by implementing the convolution operation in the GPU. Fonseca and Cabral[91] presented a prototyping system and tutorial for deep neural networks in big data analytics, demonstrating the advantages of the GPU as compared with the sequential versions.

Cui et al.[92] introduced GeePs, a scalable deep learning system across distributed GPUs to overcome the still slow performance of single-GPU solutions on large-scale data. GeePS manages the synchronization and communication associated with sharing the model learned. GeePS comprises a number of optimizations, including prebuilt indexes for gathering the parameter being updated, along with GPU-friendly caching, data staging, and memory management. Experimental results show good scalability using eight distributed machines and GPUs. Del Monte and Prodan[93] presented a scalable framework for training deep neural networks using heterogeneous computing resources in a grid or cloud infrastructure implemented via Apache Flink, an open-source stream processing framework for distributed high-performing data streaming applications. However, sublinear speedups are sometimes achieved due to overhead of the underlying synchronization and the shared access to the PCI-e bus. Dong and Kaeli[94] developed DNNMark, a GPU benchmark suite that consists of a collection of deep neural network primitives. Moreover, there are a number of popular frameworks such as Deeplearning4j, PaddlePaddle, MXNet, Torch, and Google's TensorFlow available as open source software libraries implementing deep learning primitives accelerated on GPUs.

Ensemble Learning

Ensemble learning combines the prediction of multiple base classifiers or regressors in order to reduce the model prediction error.[95] However, it comes at the cost of even longer runtimes required to build many base models. The computational complexity is incremented even further in ensembles for multiview learning where models are inferred from complementary data in multiple views.[96] van Heeswijk et al.[97] presented an approach to train the learners in an extreme learning machine ensemble for large-scale regression on a GPU. Experimental results reported runtimes 20x faster than in a single CPU. Tran and Cambria[98] also focused on GPU implementations of extreme learning machine ensembles for sentiment analysis, achieving improvements of up to 42x for feature extraction compared to CPU-based counterparts. Arnaldo et al.[99] presented a genetic programming ensemble system specifically designed for scaling to large datasets, focusing on individual-parallel and data-parallel approaches. Riemenschneider et al.[100] introduced a GPU implementation of the multilabel ensemble of classifier chains, achieving a speed of up to 70x when comparing a K20c GPU with a quad-core Intel Xeon. The software throughput is capable of classifying more than 25 k instances per second. Moreover, ensembles of decision trees are very popular due to their very fast building and decision time, and several approaches were referenced in the decision trees section before. However, there are not many more research works on ensemble models for GPUs. This is motivated because the GPU architecture and programming model is more oriented toward speeding up specific low-level algorithm implementations, while ensembles (meta classifiers) focus on high-level combinations of classifier predictions. Therefore, the GPU approach would be similar than the one illustrated in Figure 11 for deep learning, where model and data parallelism may be applied and combined to simultaneously improve the accuracy results and speedup learning by splitting train data.

APPLICATIONS

A growing number of researchers and industry partners are using GPUs to speedup time-consuming algorithms for real-world applications in large-scale data mining. This section presents some of the most relevant application cases of the GPU-based data mining techniques considering their importance with regards of the data volume or processing velocity.

Image Analysis

Image annotation and recognition is one of the main GPU application areas due to the intrisic parallel nature of image processing algorithms and the large number of real-world uses. Specifically, there is an increasing number of works on image analysis in health care summarized in *Pratx*'s review.[101] Fast image reconstruction of PET, CT, and MR scans allow clinicians real-time analysis, then optimizing explorations. GPUs implement fast FFT, non-Cartesian k-space sampling, and sensitive-encoded parallel imaging to speedup image rendering. Boubela et al.[102] presented software tools for the application of Apache Spark and GPUs to neuroimaging datasets. The computation times for reading the fMRI data, computing the connectivity graph, and writing the thresholded connectivity matrix to RData files involves more than 2 h per subject in a single GPU. In order to scale to the full dataset with more than 500 subjects, they propose a four-way multi-GPU approach combined with Spark to distribute the computation into nodes and decrease the total runtime from 36 days down to 9 days. Cuomo et al.[103] developed a GPU distributed implementation of a denoising algorithm for magnetic resonance images. Denoising requires to compute the neighborhood of each voxel of the 3D image (similarity window), which make its runtime too expensive in large high-definition images. However, when using GPUs they propose to compute each voxel using an independent thread, thus achieving a speedup of up to 112x in largest datasets. Kim et al.[104] proposed a programmable medical ultrasound imaging system using GPUs to reduce image processing time. Their objective was to develop a real-time system whose processing time was lower than the ultrasound acquisition time (20 ms). Thanks to an efficient implementation using concurrent copy and kernel execution, multiple frames are copied and processed at the same time. Adeshina and Hasim[105] offered a GPU-accelerated advanced encryption standard (AES) implementation for securing radiology-diagnostic images in health networks. However, performance results were not impressive, only about 2x faster on a GPU, due to the native implementation of AES encryption instruction sets in CPUs, i.e., the CPU is already really efficient encrypting/decrypting AES.

Another important application area for GPU image analysis is autonomous car navigation because these systems handle large volumes of images and require real-time processing.[106] Lim et al.[9] and Ciresan et al.[107] proposed real-time traffic sign recognition based on SVM and deep-learning in GPUs. The real-time objective is critical to prevent life-threatening situations in driving. By using a NVIDIA Titan X GPU, they are able to process full HD images in less than 5.9 ms, whereas the CPU single-threaded takes 487 ms, which means the GPU's speedup is 82x and achieves the aim of real-time processing. However, power consumption of a Titan X is not realistic to be implemented in a car and studies should be focusing on low-power NVIDIA Jetson devices.[108] Analyzing surrounding cars' direction in very complex environment has a significant role for autonomous driving. You and Kwon[109] presented a convolution neural network for classifying the orientation of a vehicle. They compare different convolution neural network architectures to achieve a trade-off between accuracy (higher than 95%) and runtime (less than 100 ms). Vasquez et al.[110] proposed an open framework for human-like autonomous driving using GPU-based implementations of inverse reinforcement learning algorithms. However, no results are reported for specific experiments. Wang and Yeung[111] analyzed the problem of tracking the trajectory of a moving object in a video with possibly very complex background using deep learning and particle filters implemented on GPUs. The tracker is capable to achieve an average frame rate of 15 FPS, which is sufficient for real-time tracking applications.[112]

Classification and Decision Support Systems in Medicine

Analysis of health records and classification systems to support medical diagnosis is a hot topic for GPU computing due to the large volume of historic information gathered from patients in hospitals. Li et al.[113] presented a disease classification system based on a SVM implementation on multi-GPU cluster MapReduce platform. Large datasets having more than 100 k patient records are split into chunks and distributed across all nodes in the cluster. Experiments were run in a four-node cluster each comprising two NVIDIA 680 GPUs. Compared to other GPU MapReduce systems, they achieve a speedup of up to 13x while keeping an accuracy higher than 85%. Martinez-Angeles et al.[114] introduced a relational learning system for accelerating the rule coverage on GPUs for mining rules on health record data. Relational learning is devoted to model relationships among data items or attributes of datasets. Therefore, GPU kernels include relational algebra operations such as selection, join, and projection of the data. Results report a speedup of up to 8x when comparing the GPU Titan to the multithreaded CPU implementation on datasets comprising more than 5 M records, managing tables larger than the total amount of GPU memory. Li et al.[40] proposed a multi-GPU implementation of the Apriori algorithm for mining association rules in medical data. Distribution of the data into multiple devices allows for both increasing the dataset dimensionality and combine horsepower to compute faster the support of the itemsets. Experiments on two NVIDIA 660 GPUs showed 109x as compared with the sequential implementation on an Intel i5 processor, handling datasets with more than 4 M records. Speedup increased as the datasets became bigger. Scalability from one to two devices proved to double the speedup effectively (1.98x). Galvao et al.[115] presented a parallel simulation of epidemiological models based on individuals, which are of great value in epidemiology to help understand the dynamics of the various infectious disease. The individual-parallel approach allowed to increase the number of individuals evaluated to more than 8 k while reducing runtime and achieving a speedup of up to 20x. Shamonin et al.[116] developed a fast parallel image registration on GPU for diagnostic classification of Alzheimer's disease. Based on OpenCL computation to combine CPU and GPU resources, they achieve a speedup up to 60x in image resampling while keeping the relative error lower than 10^{-6}. GPUs have also been employed to support physicians on the dose calculation and treatment plan optimization for radiation therapy. de Greef et al.[117] accelerated the dose calculation in intensity-modulated radiation therapy to reduce the clinical workload and the risk of damaging the organs. Algorithms implemented ray tracing on the GPU shows to perform 10x faster than the multithreaded CPU versions, a major reduction in the workload of radiotherapy treatment planning.

Biometrics Authentication

Biometric authentication systems comprise a number of techniques to identify individuals based on physical characterization, including fingerprint, palm veins, face recognition, iris and retina recognition, and so on. Specifically, there are a number of research works employing GPU computing for speeding up fingerprint biometrics. The scalability of fingerprint matching algorithms is determined not only by the number of fingerprints but also by the number of minutiae per fingerprint. Therefore, obtaining a fast fingerprint matching system requires processing millions of minutiae per second. Gutierrez et al.[118] introduced a GPU implementation of the minutia cylinder-code algorithm, the best performing algorithm in terms of accuracy, but achieving a throughput of only 55,700 fingerprints per second on a single GPU. Nevertheless, the speedup obtained was up to 100x faster than the single-thread CPU implementation. Lastra et al.[119] presented a multi-GPU implementation increasing the processing throughput to up to 1.5 million fingerprints per second when using four GPUs. Speedups achieved 15x faster than the multithread version when using one GPU, and up to 54x when using four GPUs. The main advantage is the linear scalability to any number of devices without introducing any data dependencies. Le et al.[120,121] proposed a minuta cylinder-code 3D representation to provide both good accuracy and speedup, increasing the performance to 1.8 million matches per second on a single GPU. Cappelli et al.[122] implemented a multi-GPU automated fingerprint identification system scaling the throughput to up to 35 million matches per second while keeping similar accuracy as compared with related state-of-the-art systems. They achieve a speedup of up to 207x over a CPU implementation with SIMD instruction optimization. However, the best throughput is achieved by Peralta et al.[123,124] when decomposing algorithms to adapt the implementation to any minutae-based method in Spark, increasing the performance to up to 55 million minuate per second.

Business Intelligence

Business intelligence for financial and stock market analysis is a challenging application demanding the efficient processing of large volumes of data in real time. Zhang[125] presented a genetic deep neural network on GPUs for analyzing Dow Jones industrial market index. He proposed using genetic algorithms to optimize the parameters and select the best function combination for the neural network. Increasing the number of activation functions reduced the prediction error but increased the computational complexity. Therefore, the GPU evaluates the behavior of the network regarding to the simultaneous evaluation of the different activation functions. Zhang et al.[126] compared performance of GPU and Xeon Phi on accelerating option pricing algorithms. In order to exploit the compute power of the combination of multicore CPUs, GPUs, and Xeon Phi, they proposed a hybrid computing model which consists of two types of data parallelism: worker level and device level. The worker level data parallelism uses a distributed computing infrastructure for task distribution, while the device level data parallelism uses both the multicore CPUs and many-core accelerators for fast option pricing calculation. They obtain a speedup of up to 9x when comparing the single-GPU configuration to the multithreaded CPU code. Singh et al.[127] presented a research study on accelerating the open source critical line algorithm for portfolio optimization by using GPUs. Increasing speedups were reported when increasing the number of assets evaluated, up to 8x faster than the sequential version. Ha and Moon[128] developed genetic programming on a GPU for financial data time series to predict the stock market evolution. Similar to the results obtained in other genetic programming studies for data mining on GPUs, they obtain impressive speedups up to 56x faster than the sequential CPU implementation when using a single GPU, and up to 277x when scaling to eight NVIDIA GTX 690 GPUs.

Data Streams Processing

All these applications and many more take advantage of the GPU to speedup algorithms in order to scale to bigger data and to provide faster outcome, especially in the context of real-time systems. In recent years, research studies have focused on the online implementations of algorithms for data stream processing, in which velocity and processing time is critical.[129] Krawczyk[130] presented an online version of extreme learning machine for high-speed data streams on GPUs to provide very fast learning and decision. Extreme learning machines are stochastic single-layer feedforward neural networks randomly trained in order to downsize their computational complexity. GPU speeds up computationally costly operations such as matrix calculations and the Moore-Penrose pseudoinverse. Speedups achieved up to 10x faster than sequential CPU version, reducing the update times per data chunk to less than 2 seconds. Hewa-Nadungodage et al.[131] proposed GStreamMiner, a general-purpose GPU-accelerated data stream mining framework. They presented a case study demonstrating its application using outlier detection and on-the-fly collaborative filtering over continuous streaming data.[132] Efficient implementation using concurrent kernel execution and data transfer allowed to compute the data for a current data chunk while the next chunk is being transferred to the device memory, then increasing the speedup to up to 12x as compared with the multithreaded CPU implementation. Chen et al.[133] developed G-Storm, a GPU system which harnesses the massively parallel computing power of GPUs for high-throughput online stream data processing. Implementation on the GPU showed to perform 7x faster than the single-thread CPU. All these works illustrate the potential of GPUs for high-speed streams because they provide very fast response with minimum latency.

Figure 12 summarizes the speedups collected from relevant referenced works for data mining techniques and applications on single-GPU and multi-GPU computing. Speedups depend on a number of factors including the power of the GPU and CPU architectures being compared, the parallelizability of the computational task, the computational and memory workflow and dependencies, the size of the data problem, and so on. Generally speaking there is trend in all research works on the improvement of the speedup as the data dimensionality increases.

GPU COMPUTING AND MAPREDUCE

GPUs and MapReduce distributed computing frameworks aim at different scaling purposes. Scalability approaches include vertical and horizontal scaling. Vertical scaling focuses on increasing the processing power, memory, and resources of a single node in a system, whereas horizontal scaling adds nodes to a system and distributes the workload across them. High-performance computing mainframes and GPUs are vertical scaling systems, while Hadoop and Spark MapReduce frameworks are horizontal systems. True big data problems address terabytes and petabytes of information, which is way too much for a vertical scaling system. This is why horizontal scaling is preferred for big data. Moreover, horizontal scaling offers easier and more economic expansion of the nodes in the system. However, it requires a fault-tolerant distributed file system, such as the Hadoop distributed file system (HDFS).

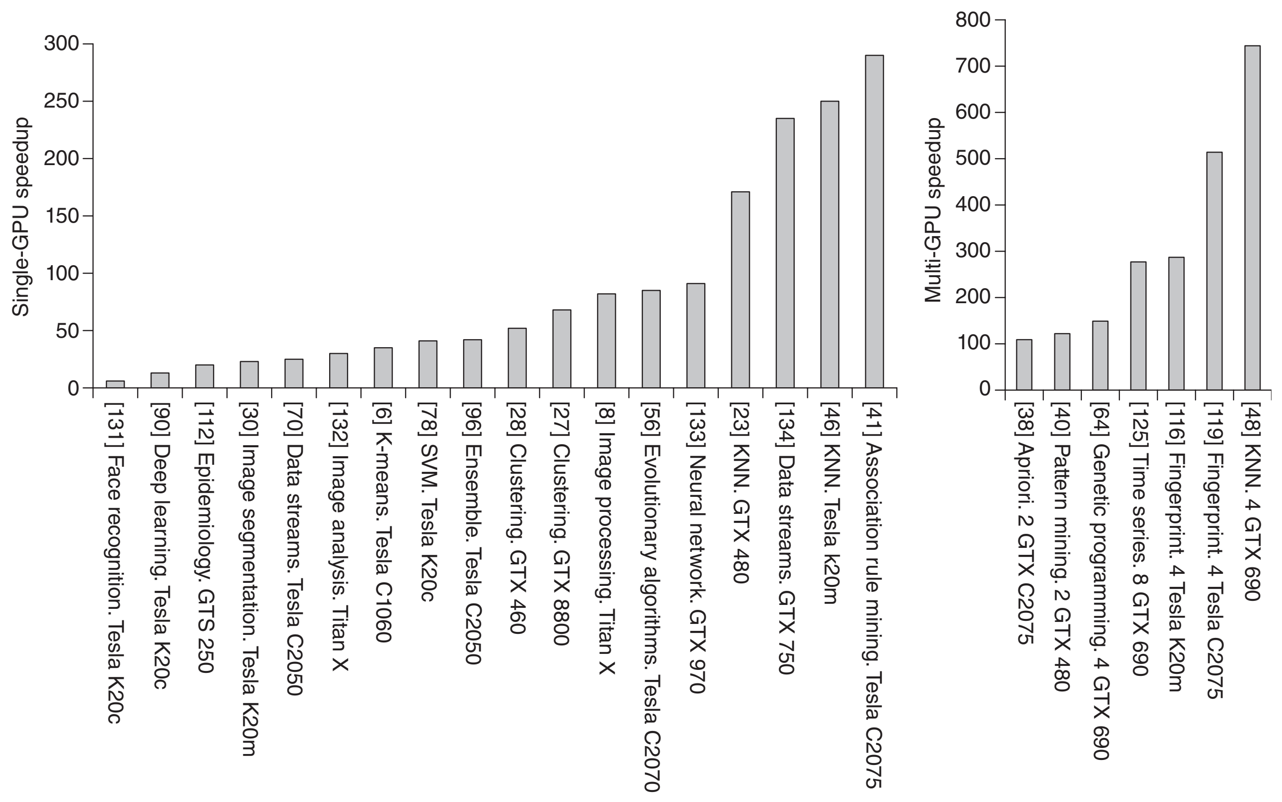


FIGURE 12 | Summary performance chart for techniques and applications.

One of the main drawbacks of MapReduce is that is not well suited for iterative algorithms due to performance impact of the launch overhead. The creation of the jobs, data transfers, and nodes synchronization through the network impose an overhead which makes distributed frameworks unsuitable for learning tasks where very fast response is required in real-time processing. Moreover, jobs run in isolation which increases the difficulty of implementing shared communication between intermediate processes. On the contrary, one of the advantages of GPU computing is the very small kernel launch overhead, which permits executing parallel tasks with no delay and obtain almost instant results. Nevertheless, the main drawback of GPUs, as stated in many of the research works reviewed, is the limited amount of memory which prevents to store complete big datasets. Therefore, one can observe that GPUs and MapReduce actually complement each other and hybrid solutions could benefit from the advantages of both technologies while minimizing their independent drawbacks.[134] There are two approaches, the former implements the map and reduce functions within the GPU, the latter exploits GPUs for computation in the nodes of the MapReduce cluster.

Implementation of MapReduce on GPUs

There is a number of research studies focusing on the implementation of MapReduce within the GPU. Figure 13 illustrates the architecture of the MapReduce in a multi-GPU system. Fang et al.[135] introduced Mars, a MapReduce runtime system extended to both multicores and GPUs. They also integrated Mars into Hadoop to enable GPU-acceleration for individual machines in a distributed environment. Stuart and Owens[136] presented GPMR, a stand-alone MapReduce library that leverages the power of GPU clusters for large-scale computing. They modify MapReduce by combining large amounts of map and reduce items into chunks and using partial reductions and accumulation. Experiments evaluated scalability to one and four GPUs, achieving a speedup of up to 162x and 559x, respectively.

One of their main contributions toward achieving maximum performance is the use of persistent map and reduce tasks, which minimizes the overhead of task creation, especially in iterative algorithms. Use of persistent functions avoids the overhead of reinstancing the map and reduce functions everytime they are called to compute a subset of data. Basaran and Kang[137] proposed Grex, a MapReduce framework for

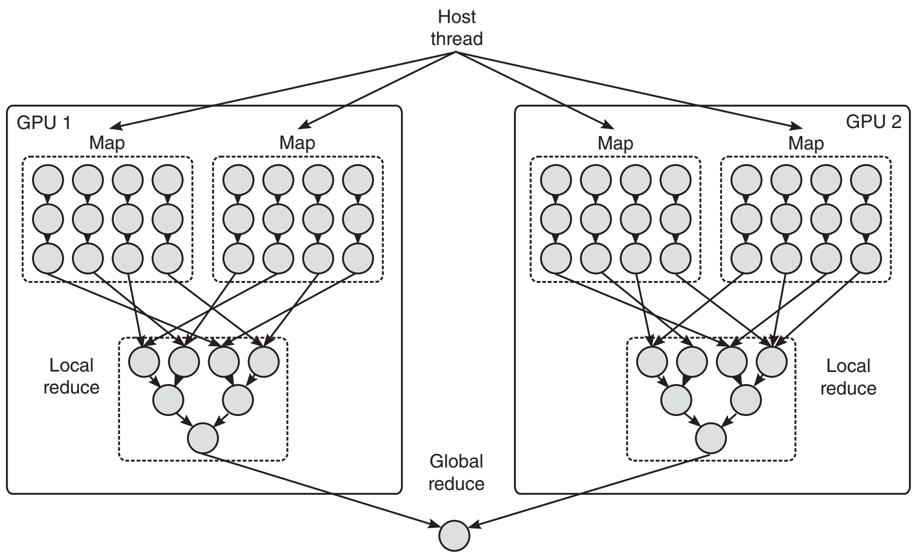


FIGURE 13 | MapReduce architecture in a multi-graphic processing unit (GPU) system.

GPUs which innovates by supporting a parallel split method to tokenize input data of variable sizes and evenly distributes data to map/reduce tasks to avoid data partitioning skews. Grex also support load balancing, lazy emit, and GPU caching, then obtaining a speedup of up to 12x as compared with Mars. Jiang et al.[14] focused on overcoming memory limitations of previous MapReduce implementations by scaling to multiple devices and by extending a pipelined version for big data.[15] They effectively handle data transfers between GPU and disk allowing programmers to write straightforward MapReduce code for big data. However, this comes at the cost of having slow long-latency data transfers between the disk and the GPU memory, which significantly impacts the performance. Qiao et al.[138] introduced MR-Graph, a customizable and unified framework for GPU-based MapReduce, which aims to improve the flexibility, scalability, and performance of MapReduce. MR-Graph efficiently explores the memory hierarchy in GPUs to reduce the data transfer overhead between execution stages and accommodate big data.

However, the implementation of the MapReduce programming model on GPUs faces important challenges. It is necessary to have a careful design of the parallel access to the critical sections by a very large number of threads and employ the appropriate mechanisms for synchronization. GPUs provide atomic operations, which are convenient to write the code faster but they are usually expensive in time. Thus, it is often preferred to write detailed code to control explicitly the parallel code and synchronization. GPUs provide synchronization barriers both at the thread-block and device levels. Moreover, it is difficult to ensure uniformly distributed workload allocation across threads to maximize GPU's occupancy.

Integration of GPUs in MapReduce Frameworks

The alternative is taking advantage of the computational power of GPUs to speedup the computation of the nodes in a MapReduce cluster. Herrero-Lopez[79] presented one of the first integrations of GPUs into MapReduce-based clusters creating a unified heterogeneous architecture that enables executing map and reduces operators on thousands of threads across multiple GPU devices and nodes, while maintaining the built-in reliability of the baseline system. Speedup of up to 163x was achieved when using four GPUs. Zhu et al.[139] integrated Hadoop MapReduce with GPUs, where Hadoop scheduled the map and reduce functions across multiple nodes, while the actual implementation of the functions in the remote nodes was accelerated on GPUs. This requires providing the function's interface to Hadoop in Java while the code is written in CUDA. Communication between the languages is done via the Java Native Interface. Niu et al.[140] combined Hadoop with GPUs to process large microarray data quality assessment and preprocessing. Their hybrid implementations allowed them to scale to data not previously computable due to high memory usage, having a speedup of 9x when scaling out to 40 nodes. Tiwary et al.[141] offloaded to the GPU the computationally intensive operations of mapping function while keeping the advantages of the HDFS, including the fault-tolerant automatic recovery. Experiments implementing the Apriori method showed 18x speedup. Shirahata et al.[16] proposed a MapReduce-based out-of-core GPU memory management technique for processing large-scale graph applications on heterogeneous GPU supercomputers, comparing the performance balance between the scale-up and scale-out approaches. Kim et al.[17] modified the Apache Mahout library to accelerate *K*-means computation using GPUs and OpenCL. The GPU is successfully capable of speeding up the algorithm but the data input/output becomes a bottleneck impacting negatively in the overall performance, limiting the speedup to up to 36x. Heldens et al.[18] presented HyGraph, a hybrid platform which delivers performance by using both CPUs and GPUs. The advantage is the dynamic scheduling of jobs into any of the compute units, providing automatic load balancing and minimizing interprocess communication overhead. Li et al.[113] introduced gcMR, a multi-GPU cluster MapReduce system for general purpose classification, providing internode and intranode parallelization.

OPEN CHALLENGES AND FUTURE DIRECTIONS

The GPU memory capacity remains as the main limitation of data mining algorithms for large-scale data despite the efforts on building pipelined models minimizing data transfer overheads. In order to scale GPU computing to terabyte-size big data, it is necessary to scale-out to multiple nodes in a cluster, mapping memory accesses to a distributed memory and file system. However, the problem is the limited bandwidth of the PCI-express bus to transfer the information to the GPU memory, and the high latency for data access time in disks and network, even using solid-state drives. Therefore, algorithms should be designed to minimize communication and make an efficient use of the memory hierarchy. Intel recently presented the Optane technology, which precisely aims at providing high-capacity storage similar to solid-state drives but at the same time offering high-throughput and low-latency as fast as main memory. The evolution of the GPU architecture in the next years promises to increase the number of cores, memory capacity, and memory bandwidth. However, data is growing at a faster pace than hardware. Multi-GPU solutions have helped to combine the computational resources and memory capacity, yet it is still insufficient to address terabyte-scale datasets. Pure distributed GPU solutions are most efficient to scale-out. However, they require to explicitly implement the workload distribution, balancing, fault-tolerance, and synchronization. On the other hand, distributed computing frameworks such as Hadoop and Spark are designed for processing big data volumes and provide automatic built-in mechanisms to distribute processing in a transparent and friendly way to the user. The main drawback is their high overhead, especially in iterative jobs, which prevents their application in real-time systems where very fast inferring is required. Integration of the MapReduce frameworks with GPU computing has become the trend in recent years to solve these problems. Researchers focused on specific algorithms and applications where *ad-hoc* optimizations considering the underlying hardware were applicable, but much work is yet to be done to facilitate an easy, efficient, and scalable general-purpose model for data mining.

NVIDIA recently presented the DGX-1 system, which shows the company's plan for future advances on GPU computing. It provides plug-and-play setup and it is engineered with groundbreaking technologies that deliver the fastest solutions for data scientists and machine learning researchers. The memory of the DGX-1 system (128 GB) is a significant step forward for increasing the data dimensionality, but the current pricing of \$149 k is prohibitive for many research groups and small industry partners. The major advantage is the integration of eight Volta GPUs in a single system communicated via high-speed NVLINK interconnect. NVIDIA NVLink is a high-bandwidth interconnect for ultra-fast communication between the CPU and GPU, and between GPUs, with a transfer rate 5–12x larger than PCIe. While multi-GPU communications via PCIe is limited by the bisection bandwidth and high latency, on an NVLINK system, however, the communication can occur in parallel because there are dedicated links between all pairs of GPUs. Therefore, it is expected to significantly increase the performance on problems where heavy data transfers among GPUs occur. The NVIDIA Volta architecture comprises 16 GB of HBM2 memory and 900 GB/second peak bandwidth, which is three times higher than the Tesla M40. The Volta architecture's computational prowess is more than just brute force: it increases performance not only by adding more SMs than previous GPUs, but by making them more efficient. Moreover, Volta Tensor cores will provide fast computation of multidimensional matrix multiplications, speeding up support vector machines, neural networks, and deep learning algorithms. Integration of application-specific integrated circuits (ASIC) is a trend for hardware aimed at particular machine learning and data mining problems. Proof of this trend, is the development of Tensor Processing Units (TPUs) by Google to speedup their TensorFlow software for machine learning.

In summary, the lessons learned from the overview on the use of GPU computing for large-scale data mining are the following:

- GPUs provide massive parallelism for large-scale data mining problems, allowing to scale algorithms to data volumes not computable by traditional approaches.
- GPUs are effective solutions for real-world real-time systems requiring very fast decision and learning, such as image recognition in autonomous driving and stock market.
- Scalability to big data is limited due to the GPU memory capacity. Multi-GPU and distributed-GPU solutions combine hardware resources to scale-out to bigger data.
- Integration of MapReduce frameworks with GPU computing overcomes many of the performance limitations and defines a promising line for future research.

ACKNOWLEDGEMENTS

This research was supported by the Spanish Ministry of Economy and Competitiveness and the European Regional Development Fund, project TIN-2014-55252-P.

REFERENCES

1. Singh D, Reddy CK. A survey on platforms for big data analytics. *J Big Data* 2014, 2:1–20.
2. Gupta P, Sharma A, Jindal R. Scalable machine-learning algorithms for big data analytics: a comprehensive review. *Wiley Interdiscip Rev Data Min Knowl Discov* 2016, 6:194–214.
3. Owens JD, Luebke D, Govindaraju N, Harris M, Krüger J, Lefohn AE, Purcell TJ. A survey of general-purpose computation on graphics hardware. In: *Computer Graphics Forum*, vol. 26. New York: Blackwell Publishing Ltd; 2007, 80–113.
4. Böhm C, Noll R, Plant C, Wackersreuther B, Zherdin A. Data mining using graphics processing units. *Trans Large Scale Data Knowl Center Syst* 2009, 5740:63–90.
5. Gainaru A, Slusanschi E, Trausan-Matu S. Mapping data mining algorithms on a GPU architecture: a study. In: *International Symposium on Methodologies for Intelligent Systems*, Warsaw, Poland. 2011, 102–112.
6. Jian L, Wang C, Liu Y, Liang S, Yi W, Shi Y. Parallel data mining techniques on graphics processing unit with compute unified device architecture (CUDA). *J Supercomput* 2013, 64:942–967.
7. Limón X, Guerra-Hernández A, Cruz-Ramírez N, Acosta-Mesa HG, Grimaldo F. A windowing strategy for distributed data mining optimized through GPUs. *Pattern Recogn Lett* 2017, 93:23–30.
8. Gaber MM. Advances in data stream mining. *Wiley Interdiscip Rev Data Min Knowl Discov* 2012, 2:79–85.
9. Lim K, Hong Y, Choi Y, Byun H. Real-time traffic sign recognition based on a general purpose GPU and deep-learning. *PLoS One* 2017, 12:1–22.
10. Zou J, Zhang H. High-frequency financial statistics through high-performance computing. In: *Conquering Big Data with High Performance Computing*, Madrid, Spain. IEEE; 2016, 233–252.
11. Dean J, Ghemawat S. MapReduce: a flexible data processing tool. *Commun ACM* 2010, 53:72–77.
12. White T. *Hadoop: The Definitive Guide*. Boston, MA: O'Reilly Media, Inc.; 2012.
13. Zaharia M, Xin RS, Wendell P, Das T, Armbrust M, Dave A, Meng X, Rosen J, Venkataraman S, Franklin MJ, et al. Apache spark: a unified engine for big data processing. *Commun ACM* 2016, 59:56–65.
14. Jiang H, Chen Y, Li K, Qiao Z, Ro W, Gaudiot J. Accelerating MapReduce framework on multi-GPU systems. *Clust Comput* 2014, 17:293–301.
15. Jiang H, Chen Y, Qiao Z, Weng TH, Li KC. Scaling up MapReduce-based big data processing on multi-GPU systems. *Clust Comput* 2015, 18:369–383.
16. Shirahata K, Sato H, Matsuoka S. Out-of-core GPU memory management for MapReduce-based large-scale graph processing. In: *IEEE International Conference on Cluster Computing*, Madrid, Spain. 2014, 221–229.
17. Kim S, Bottleson J, Jin J, Bindu P, Sakhare SC, Spisak JS. Power efficient MapReduce workload acceleration using integrated-GPU. In: *IEEE International Conference on Big Data Computing Service and Applications*, Redwood City, CA, USA. 2015, 162–169.
18. Heldens S, Varbanescu AL, Iosup A. Dynamic load balancing for high-performance graph processing on hybrid CPU-GPU platforms. In: *International Conference for High Performance Computing, Networking,*

Storage and Analysis, Salt Lake City, Utah, USA. 2016, 62–65.

19. Navarro CA, Hitschfeld-Kahler N, Mateu L. A survey on parallel computing and its applications in data-parallel problems using GPU architectures. *Commun Comput Phys* 2014, 15:285–329.
20. Navarro CA, Hitschfeld N. GPU maps for the space of computation in triangular domain problems. In: *International Conference on High Performance Computing and Communications*, Paris, France. 2014, 375–382.
21. Fang W, Lau K, Lu M, Xiao X, Lam C, Yang P, He B, Luo Q, Sander PV, Yang K. Parallel data mining on graphics processors. *Tech RepTech Rep* 2008, 1–10.
22. Ma W, Agrawal G. A translation system for enabling data mining applications on GPUs. In: *International Conference on Supercomputing*, Yorktown Heights, NY, USA. 2009, 400–409.
23. Gainaru A, Slusanschi E. Framework for mapping data mining applications on GPUs. In: *International Symposium on Parallel and Distributed Computing*, Cluj Napoca, Romania. 2011, 71–78.
24. Engel TA, Charao AS, Kirsch-Pinheiro M, Steffenel LA. Performance improvement of data mining in Weka through GPU acceleration. *Procedia Comput Sci* 2014, 32:93–100.
25. Kovács A, Prekopcsák Z. Robust GPGPU plugin development for RapidMiner. In: *RapidMiner Community Meeting and Conference*, Budapest, Hungary. 2012, 1–12.
26. Borgelt C, Kruse R. Agglomerative fuzzy clustering. In: *Soft Methods for Data Science*. New York: Springer; 2017, 69–77.
27. Braune C, Besecke S, Kruse R. Density based clustering: alternatives to DBSCAN. In: *Partitional Clustering Algorithms*. New York: Springer; 2015, 193–213.
28. Loh WK, Kim YK. A GPU-accelerated density-based clustering algorithm. In: *IEEE International Conference on Big Data and Cloud Computing*, Sydney, Australia. 2015, 775–776.
29. Huang X, Xiong L, Wang J, Ye Y, Li C. Parallel weighting K-means clustering algorithm based on graphics processing unit. *J Inform Comput Sci* 2015, 12:7031–7040.
30. Serapiao ABS, Correa GS, Goncalves FB, Carvalho VO. Combining K-means and K-harmonic with fish school search algorithm for data clustering task on graphics processing units. *Appl Soft Comput* 2016, 41:290–304.
31. Li Y, Zhao K, Chu X, Liu J. Speeding up K-means algorithm by GPUs. *J Comput Syst Sci* 2013, 79:216–229.
32. Al-Ayyoub M, Abu-Dalo AM, Jararweh Y, Jarrah M, Al Sa'D M. A GPU-based implementations of the fuzzy C-means algorithms for medical image segmentation. *J Supercomput* 2015, 71:3149–3162.
33. Takizawa H, Kobayashi H. Hierarchical parallel processing of large scale data clustering on a PC cluster with GPU co-processing. *J Supercomput* 2006, 36:219–234.
34. Teodoro G, Mariano N, Meira W, Ferreira R. Tree projection-based frequent itemset mining on multi-core CPUs and GPUs. In: *International Symposium on Computer Architecture and High Performance Computing*, Petropolis, Brazil. 2010, 47–54.
35. Li Y, Xu J, Yuan YH, Chen L. A new closed frequent itemset mining algorithm based on GPU and improved vertical structure. *Concurr Comput* 2017, 29:1–12.
36. Wang F, Yuan B. Parallel Frequent Pattern Mining without Candidate Generation on GPUs. In: *IEEE International Conference on Data Mining Workshop, IEEE*, Shenzhen, China. 2014, 1046–1052.
37. Zhou J, Yu K, Wu B. Parallel frequent patterns mining algorithm on GPU. In: *IEEE International Conference on Systems, Man and Cybernetics*, Istanbul, Turkey. 2010, 435–440.
38. Padillo F, Luna JM, Cano A, Ventura S. A data structure to speed-up machine learning algorithms on massive datasets. In: *International Conference on Hybrid Artificial Intelligent Systems*, Sevilla, Spain. 2016, 365–376.
39. Zhang F, Zhang Y, Bakos J. GPApriori: GPU-accelerated frequent itemset mining. In: *IEEE International Conference on Cluster Computing*, Austin, TX, USA. 2011, 590–594.
40. Li J, Sun F, Hu X, Wei W. A multi-GPU implementation of apriori algorithm for mining association rules in medical data. *ICIC Express Lett* 2015, 9:1303–1310.
41. Cui Q, Guo X. Research on parallel association rules mining on GPU. In: *International Conference on Green Communications and Networks*, Gandia, Spain. 2013, 215–222.
42. Cano A, Luna JM, Ventura S. High performance evaluation of evolutionary-mined association rules on GPUs. *J Supercomput* 2013, 66:1438–1461.
43. Djenouri Y, Bendjoudi A, Mehdi M, Nouali-Taboudjemat N, Habbas Z. GPU-based bees swarm optimization for association rules mining. *J Supercomput* 2015, 71:1318–1344.
44. Wu X, Kumar V, Quinlan J, Ghosh J, Yang Q, Motoda H, McLachlan G, Ng A, Liu B, Philip S, et al. Top 10 algorithms in data mining. *Knowl Inf Syst* 2008, 14:1–37.
45. Barrientos RJ, Gómez JI, Tenllado C, Matias MP, Marin M. kNN query processing in metric spaces using GPUs. In: *European Conference on Parallel Processing*, Bordeaux, France. 2011, 380–392.

46. Garcia V, Debreuve E, Nielsen F, Barlaud M. K-nearest neighbor search: fast GPU-based implementations and application to high-dimensional feature matching. In: *IEEE International Conference on Image Processing*, Hong Kong, China. 2010, 3757–3760.

47. Arefin AS, Riveros C, Berretta R, Moscato P. GPU-FS-kNN: a software tool for fast and scalable kNN computation using GPUs. *PLoS One* 2012, 7:1–13.

48. Gutierrez PD, Lastra M, Bacardit J, Benitez JM, Herrera F. GPU-SME-kNN: scalable and memory efficient kNN and lazy learning using GPUs. *Inform Sci* 2016, 373:165–182.

49. Rocha L, Ramos G, Chaves R, Sachetto R, Madeira D, Viegas F, Andrade G, Daniel S, Gonçalves M, Ferreira R. G-KNN: an efficient document classification algorithm for sparse datasets on GPUs using KNN. In: *ACM Symposium on Applied Computing*, Salamanca, Spain. 2015, 1335–1338.

50. Masek J, Burget R, Karasek J, Uher V, Dutta MK. Multi-GPU implementation of k-nearest neighbor algorithm. In: *International Conference on Telecommunications and Signal Processing*, Prague, Czech Republic. 2015, 764–767 .

51. Lin CS, Hsieh CW, Chang HY, Hsiung PA. Efficient workload balancing on heterogeneous GPUs using MixedInteger non-linear programming. *J Appl Res Technol* 2014, 12:1176–1186.

52. Cano A, Zafra A, Ventura S. Weighted data gravitation classification for standard and imbalanced data. *IEEE Trans Cybern* 2013, 43:1672–1687.

53. Espejo P, Ventura S, Herrera F. A survey on the application of genetic programming to classification. *IEEE Trans Syst Man Cybern Part C Appl Rev* 2010, 40:121–144.

54. Cano A, Zafra A, Ventura S. An EP algorithm for learning highly interpretable classifiers. In: *International Conference on Intelligent Systems Design and Applications*, Cordoba, Spain. 2011, 325–330.

55. Cano A, Zafra A, Ventura S. An interpretable classification rule mining algorithm. *Inform Sci* 2013, 240:1–20.

56. Cano A, Luna JM, Zafra A, Ventura S. A classification module for genetic programming algorithms in jclec. *J Mach Learn Res* 2015, 16:491–494.

57. Bacardit J, Llora X. Large-scale data mining using genetics-based machine learning. *Wiley Interdiscip Rev Data Min Knowl Discov* 2013, 3:37–61.

58. Franco MA, Bacardit J. Large-scale experimental evaluation of GPU strategies for evolutionary machine learning. *Inform Sci* 2016, 330:385–402.

59. Langdon WB. Graphics processing units and genetic programming: an overview. *Soft Comput* 2011, 15:1657–1669.

60. Chitty DM. Improving the performance of GPU-based genetic programming through exploitation of on-chip memory. *Soft Comput* 2016, 20:661–680.

61. Cano A, Zafra A, Ventura S. Solving classification problems using genetic programming algorithms on GPUs. In: *International Conference on Hybrid Artificial Intelligent Systems*, San Sebastian, Spain. 2010, Vol. 6077 LNAI, 17–26.

62. Cano A, Zafra A, Ventura S. A parallel genetic programming algorithm for classification. In: *International Conference on Hybrid Artificial Intelligent Systems*, Wroclaw, Poland. 2011, Vol. 6678 LNAI, 172–181.

63. Cano A, Zafra A, Ventura S. Speeding up the evaluation phase of GP classification algorithms on GPUs. *Soft Comput* 2012, 16:187–202.

64. Cano A, Olmo JL, Ventura S. Parallel multi-objective ant programming for classification using GPUs. *J Parallel Distrib Comput* 2013, 73:713–728.

65. Cano A, Zafra A, Ventura S. Parallel evaluation of Pittsburgh rule-based classifiers on GPUs. *Neurocomputing* 2014, 126:45–57.

66. Cano A, Zafra A, Ventura S. Speeding up multiple instance learning classification rules on GPUs. *Knowl Inf Syst* 2015, 44:127–145.

67. Cano A, Ventura S. GPU-parallel subtree interpreter for genetic programming. In: *Genetic and Evolutionary Computation Conference*, Vancouver, Canada. 2014, 887–894.

68. Chiu CC, Luo GH, Yuan SM. A decision tree using CUDA GPUs. In: *ACM International Conference Proceeding Series*, Ho Chi Minh City, Vietnam. 2011, 399–402.

69. Nasridinov A, Lee Y, Park Y. Decision tree construction on GPU: ubiquitous parallel computing approach. *Comput Secur* 2014, 96:403–413.

70. Grahn H, Lavesson N, Lapajne MH, Slat D. CudaRF: a CUDA-based implementation of random forests. In: *IEEE/ACS International Conference on Computer Systems and Applications*, Sharm El-Sheikh, Egypt. 2011, 95–101.

71. Jansson K, Sundell H, Bostrom H. gpuRF and gpuERT: efficient and scalable GPU algorithms for decision tree ensembles. In: *IEEE International Parallel Distributed Processing Symposium Workshops*, Phoenix, AZ, USA. 2014, 1612–1621.

72. Marron D, Bifet A, De Francisci Morales G. Random forests of very fast decision trees on GPU for mining evolving big data streams. *Front Artif Intell Appl* 2014, 263:615–620.

73. Lu Y, Zhu Y, Han M, He J, Zhang Y. A survey of GPU accelerated SVM. In: *ACM Southeast Regional Conference*, Kennesaw, GA, USA. 2014, 15:1–15:7.

74. Drozda P, Sopyła K. Accelerating SVM with GPU: the state of the art. In: *International Conference on*

Artificial Intelligence and Soft Computing, Zakopane, Poland. 2016, 624–634.

75. Li Q, Salman R, Test E, Strack R, Kecman V. Parallel multitask cross validation for support vector machine using GPU. *J Parallel Distrib Comput* 2013, 73:293–302.

76. Athanasopoulos A, Dimou A, Mezaris V, Kompatsiaris I. GPU acceleration for support vector machines. In: *International Workshop on Image Analysis for Multimedia Interactive Services*, Delft, The Netherlands. 2011.

77. Wang Z, Chu T, Choate LA, Danko CG. Rgtsvm: support vector machines on a GPU in R. arXiv preprint arXiv: 1706.05544 (2017).

78. Catanzaro B, Sundaram N, Keutzer K. Fast support vector machine training and classification on graphics processors. In: *International Conference on Machine learning*, Helsinki, Finland. 2008, 104–111.

79. Herrero-Lopez S. Accelerating SVMs by integrating GPUs into MapReduce clusters. In: *IEEE International Conference on Systems, Man and Cybernetics*, Anchorage, AK, USA. 2011, 1298–1305.

80. Yan B, Ren Y, Yang Z. A GPU based SVM method with accelerated Kernel matrix calculation. In: *IEEE International Congress on Big Data*, New York, NY, USA. 2015, 41–46.

81. Benatia A, Ji W, Wang Y, Shi F. Sparse matrix format selection with multiclass SVM for SpMV on GPU. In: *International Conference on Parallel Processing*, Philadelphia, PA, USA. 2016, 496–505.

82. Sierra-Canto X, Madera-Ramirez F, Uc-Cetina V. Parallel training of a back-propagation neural network using CUDA. In: *International Conference on Machine Learning and Applications*, Washington, DC, USA. 2010, 307–312.

83. Brito R, Fong S, Cho K, Song W, Wong R, Mohammed S, Fiaidhi J. GPU-enabled back-propagation artificial neural network for digit recognition in parallel. *J Supercomput* 2016, 72:3868–3886.

84. Li B, Zhou E, Huang B, Duan J, Wang Y, Xu N, Zhang J, Yang H. Large scale recurrent neural network on GPU. In: *International Joint Conference on Neural Networks*, Beijing, China. 2014, 4062–4069.

85. Strigl D, Kofler K, Podlipnig S. Performance and scalability of GPU-based convolutional neural networks. In: *Euromicro Conference on Parallel, Distributed and Network-based Processing*, Pisa, Italy. 2010, 317–324.

86. Juang CF, Chen TC, Cheng WY. Speedup of implementing fuzzy neural networks with high-dimensional inputs through parallel processing on graphic processing units. *IEEE Trans Fuzzy Syst* 2011, 19:717–728.

87. Li S, Dou Y, Niu X, Lv Q, Wang Q. A fast and memory saved GPU acceleration algorithm of convolutional neural networks for target detection. *Neurocomputing* 2017, 230:48–59.

88. Chen Z, Wang J, He H, Huang X. A fast deep learning system using GPU. In: *IEEE International Symposium on Circuits and Systems*, Melbourne, Australia. 2014, 1552–1555.

89. Chen C, Lee G, Xia Y, Lin W, Suzumura T, Lin C. Efficient multi-training framework of image deep learning on GPU cluster. In: *IEEE International Symposium on Multimedia*, Miami, FL, USA. 2015, 489–494.

90. Krizhevsky A, Sutskever I, Hinton GE. ImageNet classification with deep convolutional neural networks. In: *Advances in Neural Information Processing Systems*, vol. 2. Red Hook, NY: Curran Associates, Inc.; 2012, 1097–1105.

91. Fonseca A, Cabral B. Prototyping a GPGPU neural network for deep-learning big data analysis. *Big Data Res* 2017, 8:50–56.

92. Cui H, Zhang H, Ganger GR, Gibbons PB, Xing EP. GeePS: scalable deep learning on distributed GPUs with a GPU-specialized parameter server. In: *European Conference on Computer Systems*, London, United Kingdom. 2016, 4:1–4:16.

93. Del Monte B, Prodan R. A scalable GPU-enabled framework for training deep neural networks. In: *International Conference on Green High Performance Computing*, Nagercoil, India. 2016, 1–8.

94. Dong S, Kaeli D. DNNMark: a deep neural network benchmark suite for GPUs. In: *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, Austin, TX, USA. 2017, 63–72.

95. Krawczyk B, Minku LL, Gama J, Stefanowski J, Woźniak M. Ensemble learning for data stream analysis: a survey. *Inform Fusion* 2017, 37:132–156.

96. Cano A. An ensemble approach to multi-view multi-instance learning. *Knowl-Based Syst* 2017, 136:46–57.

97. Van Heeswijk M, Miche Y, Oja E, Lendasse A. GPU-accelerated and parallelized ELM ensembles for large-scale regression. *Neurocomputing* 2011, 74:2430–2437.

98. Tran HN, Cambria E. Ensemble application of ELM and GPU for real-time multimodal sentiment analysis. *Memetic Comput*. In press.

99. Arnaldo I, Veeramachaneni K, O'Reilly UM. Flash: a GP-GPU ensemble learning system for handling large datasets. In: *European Conference on Genetic Programming*, 2014, 13–24.

100. Riemenschneider M, Herbst A, Rasch A, Gorlatch S, Heider D. eccCL: parallelized GPU implementation of ensemble classifier chains. *BMC Bioinformatics* 2017, 18:371.

101. Pratx G, Xing L. GPU computing in medical physics: a review. *Med Phys* 2011, 38:2685–2697.

102. Boubela RN, Kalcher K, Huf W, Našel C, Moser E. Big data approaches for the analysis of large-scale fMRI data using apache spark and GPU processing: a demonstration on resting-state fMRI data from the human connectome project. *Front Neurosci* 2016, 9:1–8.

103. Cuomo S, Galletti A, Marcellino L. A GPU algorithm in a distributed computing system for 3D MRI denoising. In: *International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, Krakow, Poland. 2015, 557–562.

104. Kim S, Sohn HY, Chang JH, Song TK, Yoo Y. A PC-based fully-programmable medical ultrasound imaging system using agraphics processing unit. In: *IEEE Ultrasonics Symposium*, San Diego, CA, USA. 2010, 314–317.

105. Adeshina AM, Hashim R. Computational approach for securing radiology-diagnostic data in connected health network using high-performance GPU-accelerated AES. *Interdiscip Sci* 2017, 9:140–152.

106. Meier A, Gonter M, Kruse R. Artificial intelligence for developing an accident severity prediction function. *ATZ Worldwide* 2017, 119:64–69.

107. Ciresan D, Meier U, Masci J, Schmidhuber J. Multi-column deep neural network for traffic sign classification. *Neural Netw* 2012, 32:333–338.

108. Otterness N, Yang M, Rust S, Park E, Anderson JH, Smith FD, Berg A, Wang S. An evaluation of the NVIDIA TX1 for supporting real-time computer-vision workloads. In: *IEEE Real-Time and Embedded Technology and Applications Symposium*, Pittsburgh, PA, USA. 2017, 353–364.

109. You R, Kwon JW. VoNet: vehicle orientation classification using convolutional neural network. In: *International Conference on Communication and Information Processing*, Singapore, Singapore. 2016, 195–199.

110. Vasquez D, Yu Y, Kumar S, Laugier C. An open framework for human-like autonomous driving using inverse reinforcement learning. In: *IEEE Vehicle Power and Propulsion Conference*, Coimbra, Portugal. 2014, 1–4.

111. Wang N, Yeung DY. Learning a deep compact image representation for visual tracking. In: *Advances in Neural Information Processing Systems*. Red Hook, NY: Curran Associates, Inc., 2013, 1–9.

112. Cano A, Yeguas-Bolivar E, Munoz-Salinas R, Medina-Carnicer R, Ventura S. Parallelization strategies for markerless human motion capture. *J Real-Time Image Proc*. In press.

113. Li J, Chen Q, Liu B. Classification and disease probability prediction via machine learning programming based on multi-GPU cluster MapReduce system. *J Supercomput* 2017, 73:1782–1809.

114. Martinez-Angeles CA, Wu H, Dutra I, Costa VS, Buenabad-Chavez J. Relational learning with GPUs: accelerating rule coverage. *Int J Parallel Prog* 2016, 44:663–685.

115. Galvao Filho AR, Martins De Paula LC, Coelho CJ, De Lima TW, Da Silva Soares A. CUDA parallel programming for simulation of epidemiological models based on individuals. *Math Meth Appl Sci* 2016, 39:405–411.

116. Shamonin DP, Bron EE, Lelieveldt BPF, Smits M, Klein S, Staring M. Fast parallel image registration on CPU and GPU for diagnostic classification of Alzheimer's disease. *Front Neuroinform* 2014, 7:1–15.

117. de Greef M, Crezee J, van Eijk JC, Pool R, Bel A. Accelerated ray tracing for radiotherapy dose calculations on a GPU. *Med Phys* 2009, 36:4095–4102.

118. Gutierrez PD, Lastra M, Herrera F, Benitez JM. A high performance fingerprint matching system for large databases based on GPU. *IEEE Trans Inform Forensics Secur* 2014, 9:62–71.

119. Lastra M, Carabano J, Gutierrez PD, Benitez JM, Herrera F. Fast fingerprint identification using GPUs. *Inform Sci* 2015, 301:195–214.

120. Le HH, Nguyen NH, Nguyen TT. A complete fingerprint matching algorithm on GPU for a large scale identification system. In: *Information Science and Applications*. New York: Springer; 2016, 679–688.

121. Le HH, Nguyen NH, Nguyen TT. Exploiting GPU for large scale fingerprint identification. In: *Intelligent Information and Database Systems*. New York: Springer; 2016, 688–697.

122. Cappelli R, Ferrara M, Maltoni D. Large-scale fingerprint identification on GPU. *Inform Sci* 2015, 306:1–20.

123. Peralta D, García S, Benitez JM, Herrera F. Minutiae-based fingerprint matching decomposition: methodology for big data frameworks. *Inform Sci* 2017, 408:198–212.

124. Peralta D, Triguero I, García S, Saeys Y, Benitez JM, Herrera F. Distributed incremental fingerprint identification with reduced database penetration rate using a hierarchical classification based on feature fusion and selection. *Knowl-Based Syst* 2017, 126:91–103.

125. Zhang LM. Genetic deep neural networks using different activation functions for financial data mining. In: *IEEE International Conference on Big Data*, Santa Clara, CA, USA. 2015, 2849–2851.

126. Zhang S, Wang Z, Peng Y, Schmidt B, Liu W. Mapping of option pricing algorithms onto heterogeneous many-core architectures. *J Supercomput* 2017, 73:3715–3737.

127. Singh RH, Barford L, Harris F. Accelerating the critical line algorithm for portfolio optimization using GPUs. *Adv Intell Syst Comput Secur* 2016, 448:315–325.

128. Ha S, Moon BR. Fast knowledge discovery in time series with GPGPU on genetic programming. In:

Genetic and Evolutionary Computation Conference. New York: ACM; 2015, 1159–1166.

129. Ramírez-Gallego S, Krawczyk B, García S, Woźniak M, Herrera F. A survey on data preprocessing for data stream mining: current status and future directions. *Neurocomputing* 2017, 239:39–57.

130. Krawczyk B. GPU-accelerated extreme learning machines for imbalanced data streams with concept drift. In: *International Conference on Computational Science*, San Diego, CA, USA. 2016, 1692–1701.

131. HewaNadungodage C, Xia Y, Lee JJ. GStreamMiner: a GPU-accelerated data stream mining framework. In: *International Conference on Information and Knowledge Management*, Indianapolis, IN, USA. 2016, 2489–2492.

132. HewaNadungodage C, Xia Y, Lee JJ. A GPU-oriented online recommendation algorithm for efficient processing of time-varying continuous data streams. *Knowl Inf Syst* 2017, 53:637–670.

133. Chen Z, Xu J, Tang J, Kwiat K, Kamhoua C. G-Storm: GPU-enabled high-throughput online data processing in storm. In: *IEEE International Conference on Big Data*, Santa Clara, CA, USA. 2015, 307–312.

134. Rathore MM, Son H, Ahmad A, Paul A, Jeon G. Real-time big data stream processing using GPU with spark over hadoop ecosystem. *Int J Parallel Prog*. In press.

135. Fang W, He B, Luo Q, Govindaraju NK. Mars: accelerating MapReduce with graphics processors. *IEEE Trans Parallel Distrib Syst* 2011, 22:608–620.

136. Stuart JA, Owens JD. Multi-GPU MapReduce on GPU clusters. In: *IEEE International Parallel & Distributed Processing Symposium*, Anchorage, AK, USA. 2011, 1068–1079.

137. Basaran C, Kang KD. Grex: an efficient MapReduce framework for graphics processing units. *J Parallel Distrib Comput* 2013, 73:522–533.

138. Qiao Z, Liang S, Jiang H, Fu S. MR-Graph: a customizable GPU MapReduce. In: *IEEE International Conference on Cyber Security and Cloud Computing*, 2015, 417–422.

139. Zhu J, Li J, Hardesty E, Jiang H, Li KC. GPU-in-hadoop: enabling MapReduce across distributed heterogeneous platforms. In: *IEEE International Conference on Computer and Information Science*, Taiyuan, China. 2014, 321–326.

140. Niu S, Yang G, Sarma N, Xuan P, Smith MC, Srimani P, Luo F. Combining Hadoop and GPU to preprocess large Affymetrix microarray data. In: *IEEE International Conference on Big Data*, Washington, DC, USA. 2014, 692–700.

141. Tiwary M, Sahoo AK, Misra R. Efficient implementation of Apriori algorithm on HDFS using GPU. In: *International Conference on High Performance Computing and Applications*, Bhubaneswar, India. 2015, 1–7.