ISSN : 2394-2975 (Online)
ISSN : 2394-6814 (Print)

GPGPU For Memory Optimization in Distributed Data Mining

[I]Md Shabbir Hassan, [II]Umesh Chandra
[I]Ph.D Scholar, Dept. of Computer Science, Sri Venkateshwara University, Meerut, UP, India
[II]Assistant Professor, Dept. of Computer Science, Sri Venkateshwara University, Meerut, UP, India

Abstract
The accessibility of huge datasets and rising significance of data analysis for systematic finding and innovations is generating new session of high end applications. Modern GPUs computing much these applications at very modest cost. The max computational supremacy and capability of state-of-art graphics processing units have prepared them the primary extensively manageable parallel processors with teraflops competence. To completely recognize the efficiency of general purpose computation on graphics processing units (GPGPU), two important subjects require to be considered which are how to concurrent a process into parallel work objects and distribute the loads; and how to utilize the memory in effective manner, specified its leading impact on throughput. In this paper we discuss the framework of GPGPU in data mining with its performance analysis for memory optimization.

Keywords
GPGPU; CUDA; Optimization; Performance; Mining.

I. Introduction

The obtainability of big datasets and growing significance of data analysis for logical detection and findings is making a new level of high-end applications [26]. Many devices [1] consist of applications that execute extensive computations on huge datasets. This segment of tools comprises scientific data and data mining examination. Explore new data mining processes for data analysis in scientific area has been a lively subject for many years. With large dataset dimensions, require for collaborating result from performance tools, and latest tendencies in computer science, we consider as this domain is going through a serious issue in aspect of attaining suitable execution times. From the previous years, it is quite impossible to increase the performance of system or processor by simply developing clock frequencies.

During the last one decade, many data mining methods have been developed to identify patterns, grouping, and clusters from various types of data [2]. Until various approaches emphasis on the effectiveness of mining, and other methods focus at performance enhancement. Developing distributed hierarchies has become a workable way to improve data mining performance [3].

Therefore, multi-core architectures and methods like GPUs (Graphics Processing Units) and FPGAs(Field Programmable Gate Arrays) come to be a cost effective source for better performance. Present GPUs provide an admirable performance to cost ratio for high-end applications. Moreover, the GPU computing programmability and competences remain to develop quickly. The most meaningful development has been the making of the CUDA (Compute Unified Device Architecture) by a company, NVIDIA. CUDA permits GPU software development with C language-similar structures, thus enabling the expansion of non-graphics applications on a GPU. In recent times, OpenCL is an API which appears to be developing as an cross-vendor and open level for developing computation supremacy of both GPUs and CPUs. Even former to these advances, there had been an increasing importance in the usage of GPUs for non-graphics applications [4, 5, 6, 25], as also acknowledged in the GPGPU (General Purpose computing with GPUs). There are many causes why it is required to make use of GPU computing efficiency for data mining applications. Consumers/users with a personal computer typically have a great GPU to help their graphics applications. These users can speed up their data mining executions with this GPU. In other circumstances, a classified group can be accessible for assisting high-end data executions, these groups or clusters require to have visualization proficiencies, that indicates that every point has a dominant graphics card [25]. Though, CUDA are speed up the utilization of GPUs for common purpose applications, various issues still persist in implementation of the GPUs [25]. CUDA include open parallel encoding, and plain supervision of its composite memory hierarchy. Furthermore, assigning data drive, device memory between device memory and CPU, data passing among memory hierarchies, and properties of thread network structures is clear [25]. This indicates a meaningful learning curve for developers who wish to increase the computation of their applications using the GPUs. Thus, it will obviously require to be capable to concept GPUs using a powerful and remarkable interface. In addition, as we consider in this work, application computation on GPUs can be enhanced through approaches which are not very apparent or instinctive. These kind of optimizations may be simply and spontaneously achieved using a programmed code generation system [25].

To completely recognize the efficiency of general purpose computation on graphics processing units (GPGPU), two important subjects require to be considered wisely:

1. How to concurrent a process into parallel work objects and distribute the loads
2. How to utilize the memory in effective manner, specified its leading impact on throughput.

As these two problems commonly tied together and verdict an optimum trade-off between various stages of concurrency and memory optimizations needs complete considerate of GPU hardware, developing high computing GPGPU curriculum's leftovers problem for application programmers. Additionally, GPU hardware models are developing swiftly, which marks the code advanced and set for one group like NVIDIA GTX 8800 and less optimum for the next version like NVIDIA GTX280 [27]. Our proposed explanation to these issues is to make program developers discover data-level parallelism and/or fine-grain thread-level parallelism and to usage of an enhancing technique to accomplish concurrent optimizations and memory [27]. This method, we influence an algorithm-level proficiency of program designers and at the same time release them of small-level hardware-limited performance optimizations [27].

II. Related work

1. GPGPU

The GPU is an essential constituent in product and service technologies. It was formerly developed to be a co-processor to the CPU for graphical application and games .In recent times, the GPU has been used as a hardware accelerator for several non-graphics applications, like scientific analysis, matrix computation [7], datasets [8], and distributed analytical tasks. There are primarily two types of GPU programming languages: DirectX and OpenGL as graphics APIs, and GPGPU languages: CUDA, OpenCL and CTM. CUDA is from NVIDIA. Hence CUDA supports well NVIDIA CARD. OpenCL is being managed by Khronos group. OpenCL is architecture independent language.

The previous type procedures the textures over the programmable hardware pipe, as presented in Fig. 1. Vertices and pixel processors are labouring to move the computation. So, program design with the graphs APIs can directly use the hardware properties associated to translation and visualization. Formerly, GPGPU designers used graphs APIs to map applications to the graphics interpretation machine [9]. Though, this type of mapping may be incompetent and infeasible occasionally [10].

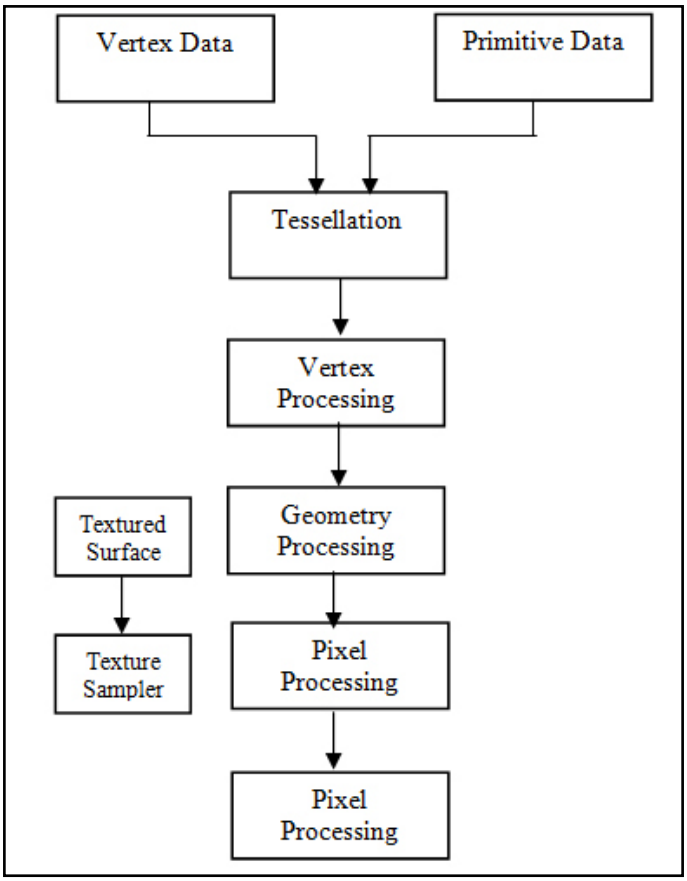


Fig.1 : The hardware Pipeline

In disparity, GPGPU programming system the GPU as a many core architecture presented in Fig.2,offer C/C++ similar features and interferences, and interpretation hardware properties for general purpose computation. For ex, CUDA revelations hardware features comprising the quick inter processor communication through the local memory, as well as the massive thread concurrency. The GPU has a huge quantity of device memory, which has large bandwidth and max access potential. Primitives as the construct blocks for higher-end applications have been projected and designed [8]. These primitives based on GPU further decrease the difficulty of GPU programming.

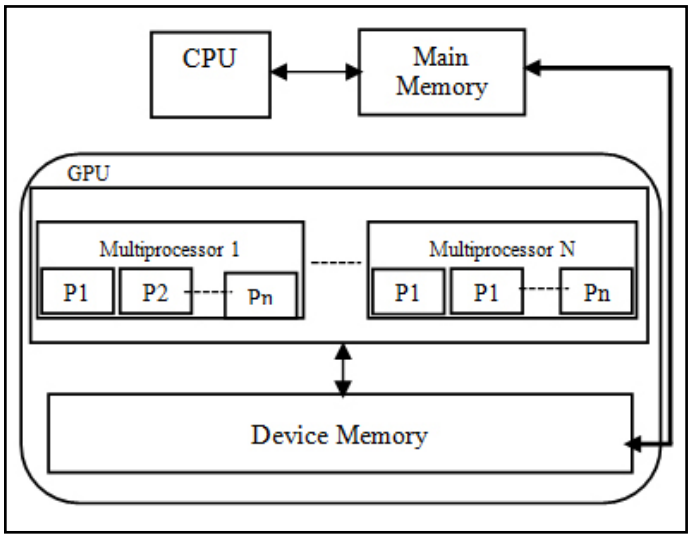


Fig. 2 : The many-core Architecture

2. Distributed Mining

The Parallel data mining is broadly deliberated in distributed database [12]. El-Hajj et al. [12] intended a distributed apriori algorithm on varied computer groups and network atmospheres using dynamic computation load supervision to handle memory restraint, attain balanced loads, and minimize computational cost and communication overhead. El-Hajj [12] offered alternates of FP Growth on computer clusters, minimum computational overhead and boost I/O use, memory, and cache. Li et al. [11] validated a linear speed-up of the FP-Growth procedure more than thousands of distributed systems using MapReduce technique Google.

Since multi-core CPUs and simultaneous multithreading (SMT) have developed as the main-stream processing unit, examiners have considered illustrative mining procedures, like Apriority[14] and k-means [15] on multi-core CPUs. The main concern is in what way completely used the thread-level parallelism (TLP) and instruction-level parallelism (ILP) on the multi-core CPU. In paper [16], enhanced FP Growth [17] via a cache-cognizant prefix tree for a tiling strategy for temporal locality and dimension vicinity and ILP. Liu et al. suggested a cache-cognizant FP array from compressing lock-free and FP-tree database tree structure algorithm for TLP. Ye et al. discovered parallelizing Bodon's tree-based Apriori algorithm with a dataset partitioning technique.

III. Operating System Issues and Challenges

In this segment, we study challenges for GPU resource management in operating systems. The below conversation is based on previous research work, and does not conclude the full range of GPU resource management.

1. GPU Scheduling

GPU setting up is possibly the main significant issue to influence the GPU in multi-threading atmospheres. Short of GPU arranging, GPU kernel process is released in FIFO (first-in-first-out) manner, Though the GPU command trigger element carries GPU command sets at their coming direction. Therefore, GPU processing come to be a non-pre-emptive process in a strong touch.

2. GPU Clustering

Additional research issues contain care for grouped various GPUs. It is a fundamental tool to utilize GPUs by High Performance

ISSN : 2394-2975 (Online)
ISSN : 2394-6814 (Print)

Computing (HPC) applications. Presently, this emerging concept designed on GPU scheduling structures and GPU resource management model to offer high-level sustenance for GPU-based communicating data cores, super-power computer, and cyber-physical machines. GPU clusters are usually hierarchical in structure. Each element is self-possessed of a slight amount of GPUs grouped on a panel. Various such elements are further grouped as a structure.

3. GPU Virtualization

GPU Virtualization is a valuable method broadly accepted in various application fields to segregate users in the structure, and mark the structure compositional and reliable. Virtualizing data hence offers the same assistances for GPU-accelerated systems. GPU virtualization provision has been offered by runtime devices [19], VMMs [20], and I/O administrators in the works. We howsoever consider that there is a problematic area for OS for maintaining GPU virtualization. As a matter of fact, virtual machines ultimately access the GPU through the device driver in the multitude OS. Therefore, at the device driver point, GPU resource management shows a dynamic place for GPU virtualization also.

4. Coordination with Runtime Engine

GPU processes are organized by GPU command sets allotted from client-space agendas. Such as, GPU kernel introductions and data replicas among the host and the device memory are initiated by a particular groups of GPU directives. Though, the operating system does not identify what kinds of GPU directives are distributed from client-space curriculums.

5. Open Source Implementation

Emerging open-source implementation is an important responsibility to share thoughts about machines tool and assist research. For example, Linux is a famous open-source software used in operating systems concept. Some of the open source device drivers of GPU are Nouveau and PSCNV, which are used for, for NVIDIA's GPUs accessible with Linux. Earlier research on Time Graph mostly used Nouveau to develop and compute an innovative real-time GPU command scheduler.

IV. Problem formulation

In contrast with earlier sequential or parallel CPU based FIM methods, this paper work is aimed for the GPU with huge thread parallelism. Furthermore, we try to recognise the general methods on executing data mining procedures on the GPU. The GPU manage occurrence calculating on communications in a bitmap while the CPU handles the trie hierarchy for outcome patterns. Such a development takes benefit of the GPU's SIMD (single instruction multiple data) massive parallelism as well as workings well with the GPU's virtualized centres and hardware-accomplished threads.

So in this paper we proposed a framework to provide a concept how to distribute a workload and optimize the memory utilization on GPGPU with performance analysis parameters.

V. Data mining based on gpu

As a unified data mining structure, Mining has the resulting factors.

1. High-level performance

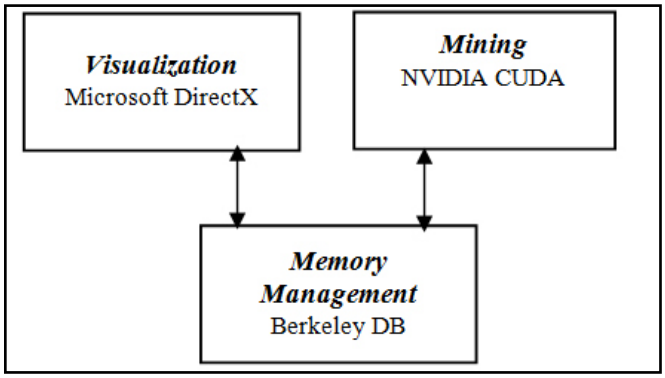
The data mining processes in GPU mining are integrated and developed as concurrent ones manipulating the parallelism of the whole system, comprising the co-handling parallelism among the CPU and the GPU, and the on-chip parallelism inside every processor. In specific, these parallel procedures are accessible to numbers of processors on the GPU.

2. I/O control organization

GPU Mining delivers a dynamic and effective I/O control structure for exploring huge volumes of data.

3. Operational visualization

Data mining is frequently a high-running and interactive procedure. Visualization assistances user to mining huge database more proficiently. GPU Mining delivers online visualization for the people to note and interrelate with the mining procedure.



The memory management module is accountable for managing the data move among the diskette, the key memory and the GPU memory. These three points of memories form a memory structure, where memory management should be sensibly designed among two end-to-end levels. For easiness and efficacy, this constituent offer bulk reads and bulk writes only, that means, reading a portion of data from the diskette to the GPU memory, and writing a large piece of data from the GPU memory to the disk.

In this work, GPU mining uses Berkeley database(Bdb) as the backend for storage the data determinedly. Related with the raw I/O APIs retrieving data in plain transcript files or structured records, Barkley DB clearly delivers the effective buffer controlling among the disk and the main memory, composed with suitable file I/O processes comprising in-place data modification. Since this module of GPU mining maintain bulk reads and writes only, we stock amount of data as a file in Bdb with a specific key. Thus, an amount of data can be retrieved or kept by the key. GPU mining based on the buffer managing from Berkeley DB offers a frivolous I/O library containing of two APIs, namely *Read Bulk* and *Write Bulk*. *Read Bulk* states a portion of data from the diskette and handovers them to the GPU memory, while *Write Bulk* yields an amount of data from the GPU memory to the disk. With these two APIs, designers can manage huge datasets short of seeing categorical data allocation and data transmission between the GPU memory, the main memory and the disk.

The mining module contains of distributed data mining systems containing grouping and common item set mining. We select GPGPU APIs to design and enhance the mining procedures due to their procedural complication. With the substructure delivered in the GPU mining, we are adding additional data mining systems such as FP-Growth and grouping.

Numbering is an essential process in data mining systems. For example, *k-means* count the various data substances related with a particular cluster, and *Apriori* count the various transactions

containing the similar element. We convert this connotation counting into counting the number of ones caused from a set of Boolean exams on the connotation. Since the connotation is commonly a binary relation, example<object, cluster>in *k-means* and <transaction, item>in *Apriori*.

The *k-means* algorithm workings in iterations. At the start, the procedure arbitrarily selects *k* of the items as the primary *centroid* for each group. In every iteration, *k-means* links every data item with its adjacent *centroid*, based on the correspondence metric. Now, it figures the different *centroids* by taking the average of all the data items in each group respectively. *K-means* dismisses once the alterations in the *centriods* are smaller than specific threshold. The data transmission among the main memory and the GPU memory is minor, when the input data is prepared on the GPU memory.

Frequent dataset mining identify groups of objects that look like in a percentage of communications, known as support, greater than a assumed threshold. The *Apriori* algorithm discovers all recurrent object sets in several moves, known a support threshold. At the opening move, it discovers the frequent objects.

VI. Result Analysis

Above approach may be further categorized by two additional standards. The primary is the ideal computation rate, which is generally the lowest time to implement all of the necessary computational work and the secondary is the lowest time to drive all data from the DRAM to the centres. The performance metrics is shown in table 1.

Table 1 : Performance Metrics

Name	Description	Unit
Texc	Expected execution time	cost
Tcomp	Computation cost	cost
Tmem	Memory cost	cost
Toverlap	Overlapped cost	cost
T'mem	*Tmem-Toverlap*	cost
Tfp	Ideal *Tcomp*	cost
Tmem_ min	Ideal *Tmem*	cost
Bserial	Benefits of eliminating serialization effects	benefit
Bfp	Benefit of improving computation efficiency	benefit

For the performance of this work, the result of proposed approach is compared with existing approach. The results clarify that the proposed work helps in minimize the cost and increase the benefits. The comparison of mining and GPU base mining on the basis of below parameters is depicted in table 2.

Table 2 : Performance Analysis

Parameters	Mining	GPU based mining
Texc	High	Low
Tcomp	High	Low
Tmem	High	Low and for high computation it may be high which are quite complex for normal computation (CPU)
Toverlap	High	Low
Bserial	Low	High
Bfp	Low	High

VII. Conclusion

In this paper, we examine terminology of GPGPU and GPU with its features and requirements. In this work we also conclude the common issues and challenges of operating system for GPU management. The discussed framework of GPGPU provide a better memory utilization with minimum computational overhead though the proposed data mining technique on GPGPU.

The proposed work can be further elongated to different data mining technique and analyse the GPGPU compiler to make an effective translation system for GPGPU applications.

References

[1] E. Randal, and Bryant, "Data-Intensive Supercomputing: The Case for DISC," Technical Report CMU-CS-07-128, School of Computer Science, Carnegie Mellon University, 2007.

[2] J. Han and M. Kamber. "Data Mining: Concepts and Techniques. Morgan Kaufmann, 2001.

[3] M. J. Zaki, "Data mining parallel and distributed association mining: A survey," IEEE Concurrency, 2010.

[4] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan, "Brook for GPUs: Stream Computing on Graphics Hardware," 2004.

[5] M. Charalambous, P. Trancoso, and A. Stamatakis, "Initial experiences porting a bioinformatics application to a graphics processor," In Panhellenic Conference on Informatics, 2005, pp.415-425.

[6] M. Christen, O. Schenk, and H. Burkhart, "General-Purpose Sparse Matrix Building Blocks using the NVIDIA CUDA Technology Platform," In First Workshop on General Purpose Processing on Graphics Processing Units, Oct 2007.

[7] Y. Yang, P. Xiang, J. Kong and H. ZHOU, "A GPGPU compiler for memory optimization and parallelism management," PLDI, ACM, June 2010.

[8] N. Govindaraju, J.Gray, R. Kumar, and D. Manocha, "GPUTeraSort: High performance graphics co-processor sorting for large database management," In SIGMOD, 2006.

[9] E. Larsen and D. McAllister, "Fast matrix multiplies using graphics hardware," 2011.

[10] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A. E. Lefohn, and T. J. Purcell, "A survey of general-purpose computation on graphics hardware," 2007.

[11] H. Li, Y. Wang, D. Zhang, M. Zhang, and E. Chang, "PFP: Parallel FP-Growth for Query Recommendation. ACM Recommender Systems," 2008.

[12] M. El-Hajj and O. Zaiane, "Parallel Leap: Large-Scale Maximal Pattern Mining in a Distributed Environment," In ICPADS, 2006.

[13] S. Hong and H. Kim, "An analytical model for GPU architecture with memory-level and thread-level parallelism awareness," In Proc. International Symposium on Computer

ISSN : 2394-2975 (Online)
ISSN : 2394-6814 (Print)

Architecture, 2009.

[14] R. Agrawal and R. Srikant. "Fast algorithms for mining association rules," In VLDB, volume 1215, 1994.

[15] M. Baskaran, U. Bondhugula, S. Krishnamoorthy, J. Ramanujam, A. Rountev, and P. Sadayappan, "Automatic Data Movement and Computation Mapping for Multi-level Parallel Architectures with Explicitly Managed Memories," In Proc. ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 2008.

[16] A. Ghoting, G. Buehrer, S. Parthasarathy, D. Kim, A. Nguyen, Y. Chen, and P. Dubey, "Cache conscious frequent pattern mining on a modern processor," In VLDB, 2005.

[17] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," In SIGMOD, 2000

[18] H.A. Lagar-Cavilla, N. Tolia, M.Satyanarayanan, and E. de Lara, "VMM-Independent Graphics Acceleratio," In Proceedings of the ACM/Usenix International Conference on Virtual Execution Environments, 2007, pp.33-43.

[19] V. Gupta, A. Gavrilovska, N. Tolia, and V. Talwar, "GViM: GPU accelerated Virtual Machines," In Proceedings of the ACM Workshop on System-level Virtualization for High Performance Computing, 2011, pp.17-24.

[20] S. Kato and Y. Ishikawa, "Gang EDF Scheduling of Parallel Task Systems," In Proceedings of the IEEE Real-Time Systems Symposium, 2009, pp.459-468.

[21] S. Kato, Y. Ishikawa, and R. Rajkumar, "CPU Scheduling and Memory Management for Interactive Real-Time Applications, "Real-Time Systems,2011.

[22] S. Kato, K. Lakshmanan, Y. Ishikawa, and R. Rajkumar, "Resource Sharing in GPU-accelerated Windowing Systems," In Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium, 2011, pp.191-200.

[23] S. Kato, K. Lakshmanan, R. Rajkumar, and Y. Ishikawa, "Time Graph: GPU Scheduling for Real-Time Multi-Tasking Environments," In Proceedings of the USENIX Annual Technical Conference, 2011.

[24] S. Kato and N. Yamasaki, "Global EDF-based Scheduling with Efficient Priority Promotion," In Proceedings of the IEEE Embedded and Real-Time computing system and applications, 2008, pp. 197-206.

[25] Wenjing Ma. "A translation system for enabling data mining applications on GPUs", Proceedings of the 23rd international conference on Conference on Supercomputing ICS 09 ICS 09, 2009.

[26] Leonid Glimcher. "Supporting load balancing for distributed data-intensive applications", 2009 International Conference on High Performance Computing (HiPC), 12/2009

[27] Yang, Yi, Ping Xiang, Jingfei Kong, and Huiyang Zhou. "A GPGPU compiler for memory optimization and parallelism management", ACM SIGPLAN Notices, 2010.