Exploring the Possibility of using A GPU While Implementing Pipelining to Reduce the Processing Time in the ETL Process

Dr. Deepshikha Aggarwal
Department of Information Technology
Jagan Institute of Management Studies
Delhi, India
deepshikha.aggarwal@jimsindia.org

Abstract: To accumulate data at one place and to make it suitable for strategic decisions we need a data warehouse system. This requires an extract, transform and load (ETL) software, which extracts data from various resources, transform it into new data formats according to required information needs, and then load it into desired data structure(s) such as a data warehouse. Such softwares take enormous time for the purpose which makes the process very slow. To deal with the problem of time taken by ETL process, parallel processing is utilised. In this paper we have proposed the use of pipelining for the parallel processing and explored the possibility of using a GPU for the process. When a computer process does not contain high parallelism it works well on CPU which contains less number of cores. Whereas the processes that contain high degree of parallelism CPU is less efficient and each independent code runs on separate core of GPU. This paper gives the basic idea of the parallel computing and also gives a simple comparison between the usage of GPU vs CPU. By comparison and analysis, we have reached a conclusion that GPU is suitable for processing large scale data parallel load where high level of parallelism is required to be run on multiple processors , however, the CPU is more suitable for processing low level parallel computing applications..

Keywords: *ETL, pipelining, GPU, Parallel computing, Data warehousing*

*****

I. INTRODUCTION

In today's world businesses demand softwares to process large data volumes at the lightning speeds. Everybody wants to have information and reports on a single click. In this fast changing competitive environment, data volumes are increasing exponentially and hence there is an increasing demand on the data warehouses to deliver available information instantly. Additionally, data warehouses should generate consistent and accurate results.[1] In data warehouse, data is extracted from operational databases, ERP applications, CRM tools, excel spread sheets, flat files etc. It is then transformed to match the data warehouse schema and business requirements and loaded into the data warehouse database in desired structures. Data in the Data warehouse is added periodically – monthly, daily, hourly depending on the purpose of the data warehouse and the type of business it serves [2]. Since the process of loading data into the data warehouse requires a huge amount of data processing, different techniques have been proposed to enhance the ETL process. This paper explores the concept of pipelining and possibility of using the processing capacity of a GPU for the purpose.

.

II. OVERVIEW OF ETL PROCESS

ETL is a continuous and frequent part of a data warehouse process, ETL processes must be automated and a well-designed. An ETL system is necessary for the accomplishment of a data warehousing project. ETL is the process that involves Extraction, Transformation and Loading.

A. Extraction

The first part of an ETL process is to extract the data from various source systems. Most data warehousing projects consolidate data from different sources. These systems may use a different data organization and format. The process of data extraction is one of the very important tasks as it also involves data profiling and data cleaning. What goes into the data warehouse determines what quality of results are going to be obtained after data analysis.

B. Transformation

The transform phase applies a set of instructions or functions to the extracted data so as to convert different data formats into single format which can be loaded into the data warehouse. Some common transformations which may be required in the data are:

- Selecting only certain columns to load (Which is needed in decision or selecting null columns not to load)
- Translating coded values (e.g., if the source system stores M for male and F for female, but the warehouse stores 1 for male and 2 for female)
- Encoding free-form values (e.g., mapping "Male",”M " and "Mr" onto 1)
- Deriving a new calculated value (e.g., Age=CurrentDate-DOB)
- Joining together data from multiple sources (e.g., lookup, merge, etc.)
- Summarizing multiple rows of data (e.g., total sales for each Month)
- Generating surrogate key values
- Transposing or pivoting (turning multiple columns into multiple rows or vice versa)
- Splitting a column into multiple columns (e.g., putting a comma-separated list specified as a string in one column as individual values in different columns)

C. Loading

This phase loads the transformed data into data warehouse. The execution of this phase varies depending on the need of organization eg. If we are maintaining the records of sales on yearly bases then data in the data warehouse will be reloaded after a year. So the frequency of reloading or updating the data depends on type of requirements and type of decisions that we are going to make. This phase directly interact with the database, the constraints and the triggers as well, which helps in improving the overall quality of data and the ETL performance.

III. PARALLEL PROCESSING IN ETL

Parallel computing is the simultaneous use of more than one CPU or processor core to execute a program or multiple computational threads. Parallel processing makes programs run faster because there are more engines (CPU s or Cores) running it. In order to improve the performance of ETL software, parallel processing may be implemented. This has enabled the evolution of a number of methods to improve the overall performance of ETL processes when dealing with large volumes of data.

There are 3 main types of parallelisms as implemented in ETL applications:

- Data: By splitting a single sequential file into smaller data files to provide parallel access.
- Pipeline: Allowing the simultaneous running of several components on the same data stream..
- Component: The simultaneous running of multiple processes on different data streams in the same job.

All three types of parallelism are usually combined in a single job.[2][3]. Parallel processing is one of the most explored field of research in the area of computer science as it’s applications can make the processing by computers faster and faster.

IV. ABOUT GPU (GRAPHICAL PROCESSING UNIT)

GPUs were initially designed to work with images and graphical databases but due to their parallel processing capability they can be used for numerous other applications that require high speed parallelism. GPU-accelerated computing is the use of a graphics processing unit (GPU) together with a CPU to accelerate scientific, analytics, engineering, consumer, and enterprise applications[4] . The latest generation of GPUs, consisting of hundreds of stream processing units are capable of supporting thousands of concurrently executing threads, with zero-cost hardware controlled context switching between threads.

GPU-accelerated computing offers unprecedented application performance by offloading compute-intensive portions of the application to the GPU, while the remainder of the code still runs on the CPU. From a user's perspective, applications simply run significantly faster.

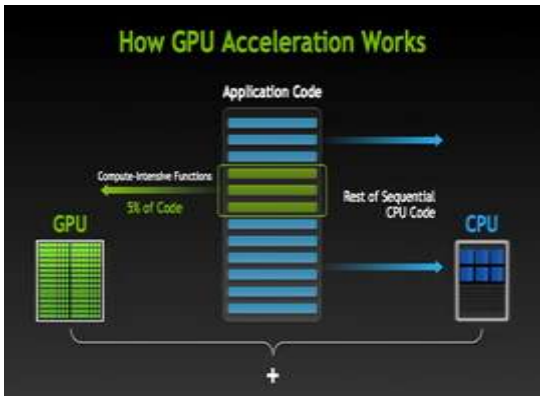

Figure 1. Acceleration using GPU

V. DIFFERENCE BETWEEN A CPU AND GPU

A simple way to understand the difference between a CPU and GPU is to compare how they process tasks. A CPU consists of a few cores optimized for sequential serial processing while a GPU has a massively parallel architecture consisting of thousands of smaller, more efficient cores designed for handling multiple tasks simultaneously. GPUs have thousands of cores to process parallel workloads efficiently.

 ISSN: 2321-8169
333 – 337

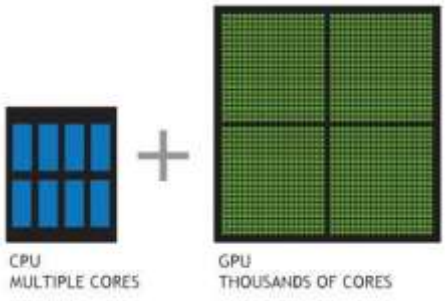

Figure 2. GPU vs CPU

VI. THE CURRENT STUDY USING THE CONCEPT OF PIPELINING

To perform Extraction, transformation and loading for a Data warehouse , we are using Pipelining. Pipelining is a technique where the complete process is divided into various segments and all these segments can work simultaneously.

Clock pulse	Seg 1	Seg 2	Seg 3	Seg 4	Seg 5	Seg 6	Seg 7	Seg 8	Seg 9
1	M1								
2	M2	M1							
3	M3	M2	M1						
4	M4	M3	M2	M1					
5	M5	M4	M3	M2	M1				
6	M6	M5	M4	M3	M2	M1			
7	M7	M6	M5	M4	M3	M2	M1		
8	M8	M7	M6	M5	M4	M3	M2	M1	
9	M9	M8	M7	M6	M5	M4	M3	M2	M1
10		M9	M8	M7	M6	M5	M4	M3	M2
11			M9	M8	M7	M6	M5	M4	M3
12				M9	M8	M7	M6	M5	M4
13					M9	M8	M7	M6	M5
14						M9	M8	M7	M6
15							M9	M8	M7
16								M9	M8
17									M9

Table 1: Pipelining the ETL Process

If we perform the ETL process in such a way where we have divided the whole process into segments and then run the segments in a pipelined manner, we will be able to save a lot of clock cycles and hence will be able to speed up the complete process. The following calculations show the speeding. process

Speeding up of the ETL process is given by:

S= (N*tn) / (k+N-1)tp

Where,

N = no. Of tasks

tn= Time to execute a task in case of Non-Pipeline

k = no. of segments.

tp = Time to execute a task in case of Pipeline

Assuming that the time takesn to process a sub operation in each segment be equal to tp= 20ns. The pipeline has k= 9 segments and execute n= 100 tasks in sequence. The pipeline system will take (k+n-1)tp = (3+99)*20 = 2040 ns to complete the task. Assuming that tn= ktp = 9*20 = 180 ns, a non pipeline system requires nktp= 100*180 = 18000 ns to complete 100 tasks. The speed ratio is equal to 18000/2040 = 8.82 .So we can conclude that the speed up of the process using pipelining is approximate 9 times.

VII. COMPARISON BETWEEN PIPELINING AND PARALLEL PROCESSING

The current study is about the possibility of using the concept of pipelining in the process of data cleaning. To perform data cleaning using pipelining, the entire process of data cleaning is divided into segments and these segments run in a pipeline in different clock cycles using the single processor. We have divided the process into 9 segments and it speeds up the process around 9 times. If parallel processing is used, we would require 9 processors to run 9 segments in parallel and it would also speed up the process 9 times only but by using multiple processors. So we deduced that the speedup of the process through pipelining is as effective as parallel processing but using only one processor which makes it more cost effective. Still if we want to use parallel processing, we can attain it by using a single CPU with a multi core processor.

Assuming 100 tasks divided into 9 segments

Pipelining process:

Number of processors/ cores required = 1

Execution of multiple segments is done simultaneously using different clock cycles

Segments used in current study = 9

Speed up using pipelining = >8 times (approx.)

Parallel processing (for the same 9 segments used in current study):

Number of processors/ cores required = 9 (one for each segment)

Speed up using parallel processing = 9 times (because of the 9 processors working in parallel)

The results were compared using different number of segments and processes and we concluded that if we continue to divide our tasks into segments and compare the

results of speed up using pipelining and parallel processing, the difference is not substantial. To attain this level of parallel processing, multiple processor CPUs can be used. The difference between speed up of pipelining and parallel processing would arise if we do not divide the tasks into segments and execute the 100 tasks taken in the study in parallel. In that case, the speed up will be 100 times if we could run all the tasks in parallel using 100 processors. But this would be possible only if the tasks are not dependent on each other and are capable of executing independently.

VIII. POSSIBILITY OF USING GPU

When we consider the possibility of using a GPU for the current scope of this study, we find that the GPU which generally consists of hundreds of processors that run processes in parallel, is not essential if we do not have a large number of processes to run in parallel. We conducted a research on the possible implementation of GPU in this study and concluded that the high speed parallelism of GPUs is most suited for graphics and highly parallel applications. In our current study, we are concentrating only on data cleaning that can be performed by using a single processor through pipelining and therefore, the investment in a GPU is not justified. The CPU works clock by clock and GPU works core by core. The individual performance of GPU core is lower than that of a CPU core. GPU core is designed to handle parallel processing faster. GPUs are designed for graphical processes and highly parallel tasks. A GPU consists of hundreds of processors that work in parallel. The proposed framework is based on pipelining which uses a single processor to perform data cleaning by dividing the process into segments and running these segments in different clock cycles on a single processor. Since the proposed framework divides the process of data cleaning in 9 segments, the hundreds of processors present in the GPU will not be required. In case of not enough data parallelism, GPU overhead is higher than the benefit. GPU consists of hundreds of processors/ cores and if the processes are few, the cost of GPU will not be justified.

IX. LIMITATIONS OF USING GPU IN THE CURRENT STUDY

Limitations of GPU in the current scope of our research also includes the difficulty in programming. The code has to be converted into the GPU format whereas in pipelining the traditional coding works. The architecture of GPU is still in its developmental stage. Data cleaning process can have different stages that may be dependent on each other where the output of one task acts as input for the next task. In such a scenario even the GPU will not speed up the process. In case of pipelining, we have divided the process of data cleaning into segments and these segments can work on a single processor through pipelining. The speedup has been calculated and it was found that the speedup is proportional to the number of segments. The speedup is almost equal to the number of segments created. In case of parallel processing, the speedup is equal to the number of processors. So through pipelining also we are able to speed up the process as much as we could have done using parallel processing but the advantage with our method is we are saving the cost of 8 extra processors that would be needed to attain the same level of speedup in case of parallel processing where different processors are used to run different segments.

X. EXPANSION OF SCOPE OF STUDY USING GPU

In case we want to expand the scope of work consider the data coming from multiple sources, and then each individual core of the GPU can use pipelining to clean up the data from each data source. In that case, pipelining will be done on each different processor and that will be done in parallel. That time the speed will increase depending on the number of processors used. We will need as many processors as the number of data sources and each processor would perform data cleaning using individual pipelining as described in our existing study. We can use as many processors as the number of data sources and each processor will perform data cleaning on each data source through pipelining. In this case if we assume data is coming from 20 data sources and we use 20 processors. Within each processor pipelining is used for data cleaning. We assume the same 9 segments for each process as considered in our study, then each processor will speed up the data cleaning process by approximately 8 times and we are using 20 processors so total speedup will be around 160 times. But even in that case a CPU with multicore processor is enough and the GPU is not required until we are involving the cleaning for graphical data as the multi-processor CPU will be able to process multiple data sources in parallel. Our current study on data cleaning is limited to nominal and textual data, and therefore we can perform the data cleaning by using a single processor. In case of graphical data cleaning, GPUs might be required. Another possibility of using GPUs for data cleaning can be by using parallel processing for different data sources as well as different segments. In this arrangement, more number of processors will be required and GPU will be a suitable choice. For example if the data is coming from 20 data sources and we perform data cleaning using the 9 segments per cleaning process as used in the pipelining and execute these segments parallel using separate cores on the

GPU, we would need 20*9=180 processors and it will speed up the process 180 times.

XI. TO IMPLEMENT GPU IN THE CURRENT STUDY

If the number of parallel tasks is considerable, GPU is needed.
Assuming the data coming from multiple sources and data cleaning is to be performed on each data.
Number of data sources = 20
Number of segments for data cleaning for each data source = 9
Total number of segments = 180
If we use parallel processing on GPU,
Total number of cores required = 180 (each core would perform pipelining individually)
Speed up of data cleaning = 180 times
If the same number of segments are executed using pipelining,
The number of processors required = 20 (one for each data source)
Speed up by each processor = >8 times
Total speed up =>160 times
In case there were no segments and we had to run 100 tasks on each of the 20 data sources for cleaning,
Number of tasks = 2000
The number of cores = 2000
Then we can justify the need of GPU. But if we do not have so many independent tasks, the need for GPU is not justified.

XII. CONCLUSION

Thus, we have deduced that by using substantially lesser number of processors, the processing speeds can be increased considerably by using segmentation and pipelining. Therefore, the use of GPU cannot be recommended only on it's capability of high level parallel processing but other factors need to be considered before investing in it. The potential of parallel processing can be utilized only when the processes are capable of running independently. Moreover we need to justify the cost of using a GPU including the hardware and programmers' cost before recommending it for use in a particular application as the main characteristic of GPU is not just parallel processing but the presence of hundreds of cores that need to be put to effective use and justify the cost of a GPU. The best use of GPU is for high level parallel processing where hundreds of tasks execute in parallel and graphical processes that require large number of cores.

REFERENCES

[1] Li, B., and Shasha, D., "Free Parallel Data Mining", ACM SIGMOD Record, Vol.27, No.2, pp.541-543, New York, USA (1998).

[2] Anand, S. S., Bell, D. A. and Hughes, J.G., "EDM: A general framework for data mining based on evidence theory", Data and Knowledge Engineering, Vol.18, No.3, pp.189-223 (1996).

[3] Agrawal, R., Imielinsk, T. and Swami, A., "Database Mining: A Performance Perspective", IEEE Transaction Knowledge and Data Engineering, vol. 5, no. 6, pp. 914-925 (1993).

[4] K. Fatahalian and M. Houston, "A closer look at GPUs," Communications of the ACM, Vol. 51, No. 10, October 2008.

[5] Z. Fan, F. Qiu, A. Kaufman, S. Yoakum-Stove, "GPU Cluster for High Performance Computing,", in Proc. ACM/IEEE conference on Supercomputing, 2004.

[6] D. Tarditi, S. Puri, and J. Oglesby, "Accelerator: Using data-parallelism to program GPUs for general-purpose uses", in Proc. 12th Int. Conf. Architect. Support Program. Lang. Oper. Syst., pp. 325-335, Oct. 2006.

[7] John D. Owens, Mike Houston, David Luebke and Simon Green," GPU Computing Graphics Processing Units-powerful, programmable, and highly parallel-are increasingly targeting general-purpose computing applications", In the procd. Of IEEE Xplore, Vol. 96, no. 5, May 2008.

[8] Danilo De Donno, Alessandra Esposito, Luciano Tarricone, and Luca Catarinucci, "Introduction to GPU Computing and CUDA Programming: A Case Study on FOlD" In the procd. Of IEEE Antennas and Propagation Magazine, Vol. 52, No.3, June 2010.

[9] J. N. William and J. Dally, "The GPU Computing Era," IEEE Trans. Image Process, vol. 10, no. 5, pp. 767-782, May 2010.

[10] K. Korotaev, "Hierarchical CPU Schedulers for Multiprocessor Systems,Fair CPU Scheduling and Processes Isolation," IEEE Trans. Electron Devices, vol. ED-11, pp. 34-39,Jan 2005.

[11] Q. Hou, X. Sun, and K. Zhou et al., "Memery-Scalable GPU Spatial Hierarchy Construction," IEEE Trans.Computer Engineering, vol. 4, pp. 189-193, Apr 2011.

[12] F. Cui and C. Cheng et al., "Accelerated GPU Computing Technology for Parallel Management Systems," IEEE Trans. Image Process, vol. 10, no. 5, pp. 255-259, May 2010.

[13] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU Computing," IEEE Trans. Neural Networks, vol. 5, pp. 334-339, Oct 2010.