An Efficient Approach to Triple Search and Join of HDT Processing Using GPU

YoonKyung Kim, YoonJoon Lee
Computer Science
KAIST
Daejeon, South Korea
e-mail: {ykkim, yjlee}@dbserver.kaist.ac.kr

JaeHwan Lee
Computer Science
IUPUI
Indianapolis, United States
e-mail: johnlee@iupui.edu

***Abstract*— Resource Description Framework (RDF) is originally designed as a metadata data model. It has an advantage of efficient exchange between different metadata by supporting a set of common rules. To process RDF data efficiently, SPARQL (SPARQL Protocol and RDF Query Language) was introduced. In this era of emerging Web of Data, the amount and size of published RDF data have dramatically increased. Most studies so far have focused on the compression of RDF data and fast SPARQL query processing using single core CPUs. Thus, they do not utilize well current multicore environment. We in this study propose SPARQL query processing using a GPU multicore system. We focus on a search and join method using Header, Dictionary, Triples(HDT) data format and present its experimental results.**

Keywords-Resource Description Framework (RDF); HDT; SPARQL; GPGPU.

I. INTRODUCTION

The Resource Description Framework (RDF) is a standard model for data interchange on the Web [1]. It has been designed for flexible representation of the Semantic Web. It was proposed to efficiently represent connections between hyperlinks embedded along with a word or a string in web data/pages such as Wikipedia. RDF consists of three parts {*subject-predicate-object*} called triple. While *subject* represents resource, *predicate* represents relation between *subject* and *object*. For example, representing 'His name is Hoon' in RDF can be {*subject*: He, *predicate*: name, and *object*: Hoon}. As in this example, most of RDF data are string data. To efficiently process large size of RDF data, SPARQL Protocol and RDF Query Language (SPARQL) [9] was introduced, and it supports extensible value testing and constraining queries by source RDF graph. [2].

The size of published RDF datasets has dramatically increased in this era of emerging Web of Data [3]. Previous researches are mostly focused on the compression of RDF data [11] and fast SPARQL query processing. We suggest parallel processing using GPU multicore system.

This paper proposes a triple search method on HDT [3] data format using a GPU and presents its experimental result.

In Section 2, we introduce Header, Dictionary, Triples (HDT) which compresses RDF data format. RDF processing using GPU is in Section 3, and experimental results in Section 4.

II. BACKGROUND

A. Query patterns in RDF

If a query is consisted of following example, {?, hasName, ?}, it means that find every triple which has predicate 'hasName'. While this query has predicate only (subject and object are question mark), so it is called pattern ?P? or P. Likewise, {Bob, hasTitle, ?} query means that find triples that has subject 'Bob' and predicate 'hasTitle' This type of query is called pattern SP? or SP.

All queries should have at least one component that is not a question mark. So, there are 7 query patterns in RDF: S??, S?O, SP?, SPO, ?P?, ?PO and ??O. We focus on pattern ?P? in following content.

B. RDF data compression based on HDT

In the past, several studies have been proposed, in which their predominant method of RDF data compression is to use a dictionary. For the compression, entropy coding is often used to distribute short length words to more frequent bit-pattern.

The most representative studies of RDF compression are based on dictionary. RDF-3x [4] transforms triples using a dictionary, saves them with a B+ tree method, and divides queries to several partials to process the query.

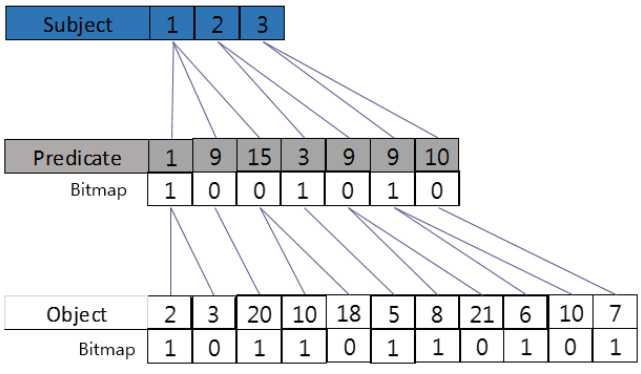

Figure 1. Triple compression of HDT.

 ISBN: 978-1-61208-408-4

Header, Dictionary, Triples (HDT) [3] transforms triple expressions (subject-predicate-object) to unique integer IDs using a lexicographically sorted dictionary. A triple is represented as a tree having subject as its root. Each subject has its own tree, so a whole set of triples are shaped like a forest. To compress triples in a forest, as shown in Figure 1, subjects, predicates, and objects are represented as bitmaps. This bitmap array is filled with zeros and ones. Zero means this element has same parent as the previous one, and one means it has a new parent. For example, the 4th predicate of Figure 1 is '3', which has subject '2', can be calculated by adding predicate bitmap values from 1 to 4. One is represented twice (at the 1st location and 4th location of the bitmap array), meaning that predicate '3' is connected with subject '2'. The bitmap array of objects is similarly encoded to grab its predicate.

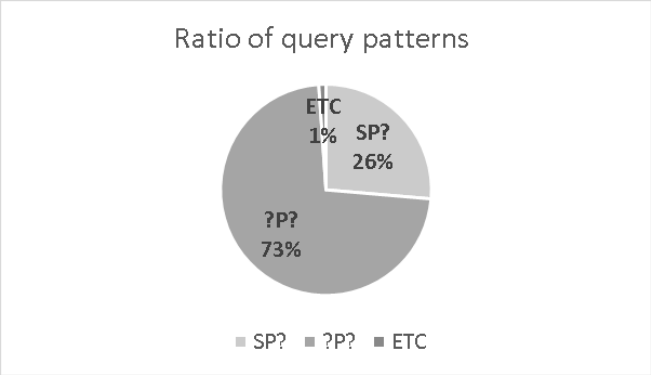

Figure 2. Query pattern ratio in SP2 benchmark.

This method of compression shows better performance when searching triples with subject (i.e., S??, SP?, S?O, SPO), and other search cases are accelerated by making index for predicate and object [4].

C. Problem Definition

Previous studies on RDF have mostly focused on a single core query processing. The disadvantage is that a triple search patterns, ?P? (P pattern) where only the predicate exists, is extremely slow. P pattern is a query that find subject and object which has a certain predicate. Unlike other components, predicate has a few variety of elements. It means that most of P pattern search results are too large to calculate using a single core.

The variance of predicates in the data is small, meaning that one predicate occurs very often. Also, the ratio of P pattern in SP2 benchmark [6] take 73% of total query triples as seen in Figure 2.

HDT makes an index for P pattern searching, but the number of the indexes is too large to process. We suggest an approach to improve this P pattern problem using GPGPU. General purpose graphic processing unit (GPGPU) is proposed in [5], focusing on its high level parallel computation power.

III. METHODOLOGY

A. Triple Pattern Searching

In the proposed approach, we modify bitmap arrays for parallel processing. The original HDT has used wavelet trees [3] to support faster search of predicate index. However, the method was optimized for single core processing. Instead, we use a GPU's scan algorithm to obtain the parent's index in a constant time. The scan algorithm constructs a position array that has the summation result of occurrences of ones from location 1 to each element's location in parallel as shown in Figure 3. In this way, predicate and object's parents (subject and predicate, respectively) can be found in a constant time using this array.

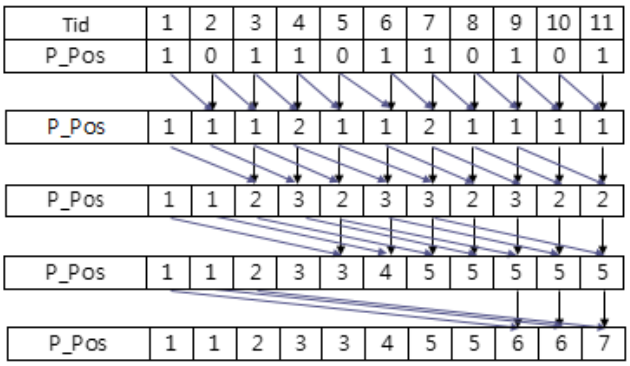

Figure 3. GPU scan algorithm for bitmap array.

After populating the position array, Figure 4 shows how this array represents predicates connected to each object. For example, object '20' (third element in object array) refers to position array (also third element in position array) which tells that '2' (second) element of predicate array is predicate of its triple. We can also find a subject using a predicate. As we allocate one object per GPU core, processing time to search a triple is constant.

Figure 4. Connection between predicate and object using object's position array.

We copy predicate, object, and object position arrays to GPU memory to search a pattern P as shown in Figure 5. After allocating all the triples to GPU cores, the search first

determines whether each object's predicate is equal to given pattern P or not. We use object array to determine triples, while others (predicate, subject) are compressed.

The object array points to the predicate array, and the predicate array points to subject array. These connections are directional, which means the predicate and subject array do not know which element of object is connected. So, the object array is needed to extract triples from the HDT data set which is compressed.

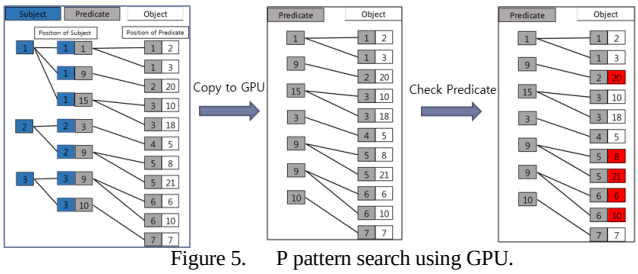

Figure 5. P pattern search using GPU.

B. Join Processing

Due to large size of RDF data, it restricts join algorithms from a result array point of view. First, if we save the result in one big array, an atomic operation of result array's pointer is needed for every new result. All other threads have to wait until the lock is released. Another method is allocating arrays to each triple, requiring N^2 memory where N is the size of input triples. It takes 1TB GPU memory for one million integer data. Considering these conditions, [7] use hash join algorithm for GPU.

With the result that GPU joins become faster with data larger than 5 million. [7] Even with data size larger than 5GB, a join hardly exceeds five million. Hence, we use a CPU sort-merge join method which original HDT uses in join processing.

IV. Experiment and Analysis

We process an experiment to compare the effect of RDF join process using GPU with previous researches (RDF-3x, original HDT).

TABLE I. Experimental Environment

CPU	AMD A10-5800K 3.80GHz
RAM	16GB
GPU	Geforce GTX 750Ti, Geforce GTX 660
Data	5GB, 1GB dataset, SP^2 benchmark
OS	Win7 in GPU, Ubuntu 12.04 in others

A. Experimental Approach

We use SP^2 benchmark [6] with several queries. Q1, Q2, Q3a, Q3b, Q3c, Q5a, and Q5b are join queries. Q10 is single O pattern query, and Q11 is single P pattern query. We exclude Q4, Q6, Q7, Q8, Q9, Q12, with following reasons. Q4 is a query for FILTER function in SPARQL, Q6, Q7 for OPTIONAL, Q8, Q9 for UNION, and Q12 for ASK. These functions are not supported in original HDT and don't have relationship with join processing. We also remove Q11's ORDERBY, LIMIT, and OFFSET to compare single P pattern search speed. All results are average of five executions while removing maximum and minimum outcomes.

We also try to use Apache Jena [10], and Bitmat [11] to compare results, but it did not work in our data set due to large size (1,5GB).

B. Experimental Results

Search speed values for each triple pattern are shown in Figure 6. The triple search using a GPU is eight times faster than other methods in P pattern, while the search speeds of other patterns are slower than those of existing methods.

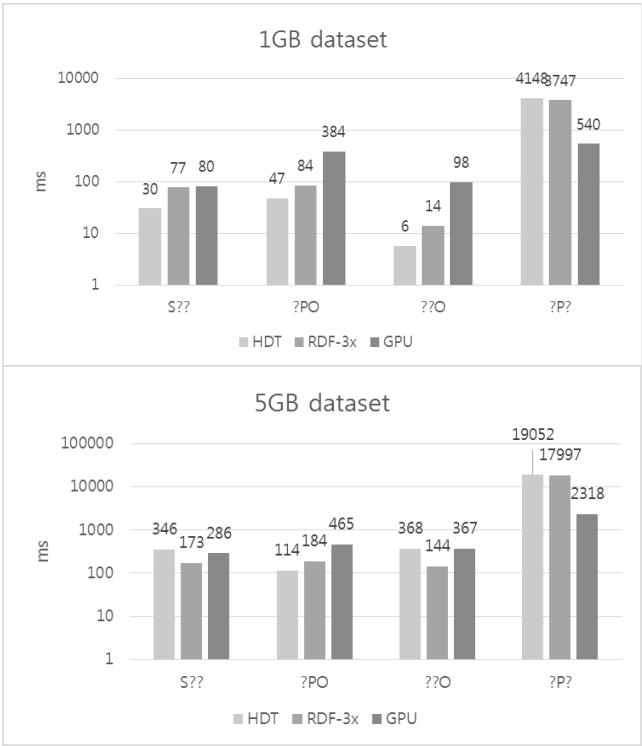

Figure 6. Speed of triple searching.

Next, we use two GPUs (shown in Table 1) to determine the number of GPU cores and memory size effect on processing speed. We use GTX 750 Ti and GTX 660, where the latter has 50% more cores than the former. Figure 7 shows the results for query processing speed. GTX660 is faster than GTX750 Ti with 5% ratio. Considering the difference in the number of cores between the two GPUS, it does not effect on GPU processing. Thus, memory transfer is critical for GPU processing. [8]

Figure 8 shows the results of SP^2 benchmark. GPU processing is slower in Q1, Q3b, Q3c, and Q10, which has result of small size. Other queries are faster. Q2's result for the original HDT is zero because it produced a wrong answer due to a program fault.

 ISBN: 978-1-61208-408-4

V. CONCLUSION

We propose an efficient RDF (HDT) query processing focusing on P pattern. Previous methods are slow based on the number of answer triples. Parallel processing using GPU increases the speed of P pattern searching. Our first approach was affected by size of memory, but not the number of GPU cores. Thus, we choose HDT to which compresses RDF data. This improved method is faster than original HDT and RDF-3x when the number of answers is large.

Our future work includes support for GPU SPARQL query processing (e.g. FILTER, UNION, and GROUPBY), and determine to use or not to use GPU based on expected speed of query processing. And a study for HDT join processing using GPU is also needed.

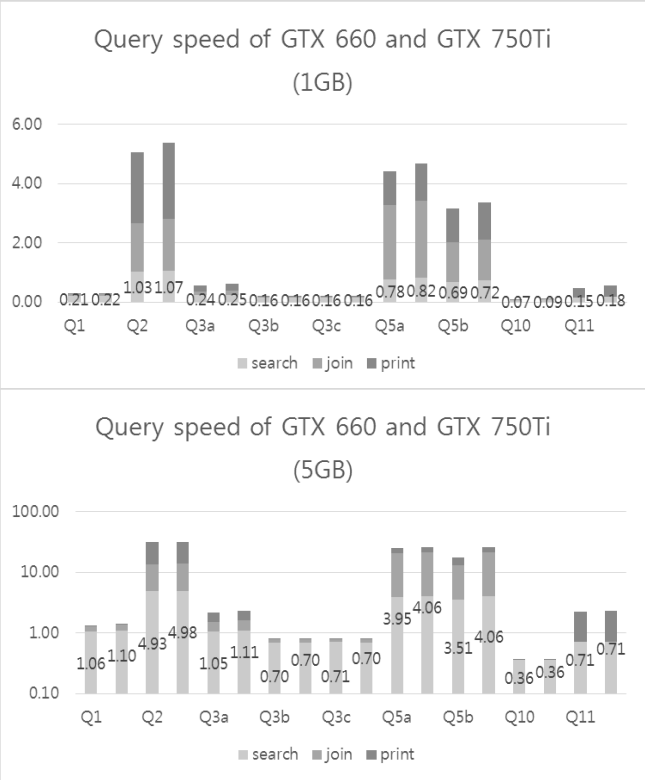

Figure 7. Query speed of GTX 660 (left), GTX 750Ti (right).

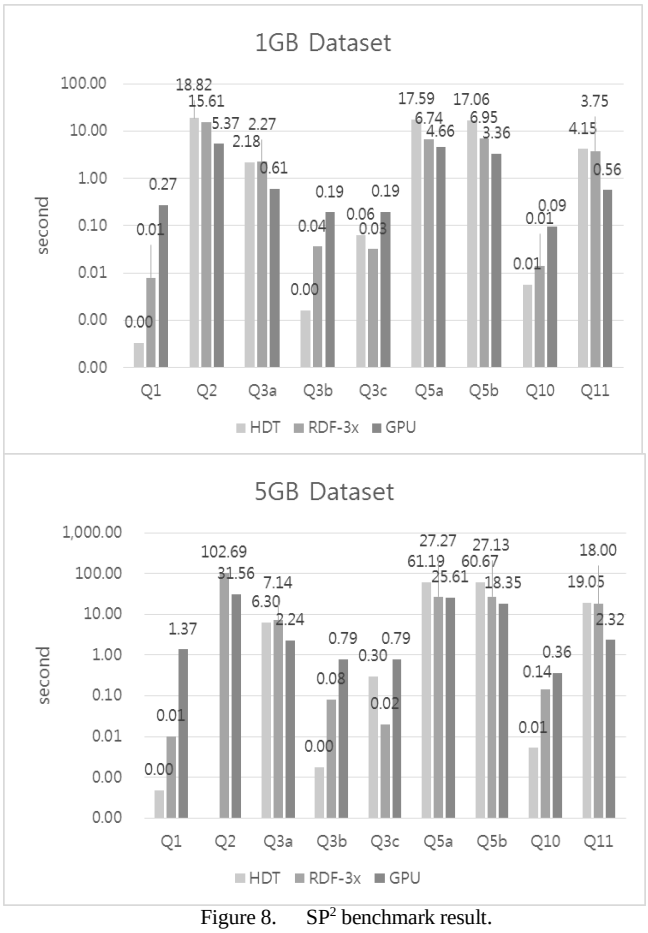

Figure 8. SP^2 benchmark result.

Acknowledgements This work was funded by Brain Pool Program, KOFST (The Korean Federation of Science and Technology Societies).

REFERENCES

[1] W3C. http://www.w3.org/RDF/, 2004 [retrieved: March, 2015].

[2] Prud'hommeaux, E., Seaborne, A. Sparql query language for rdf, http://www.w3.org/TRlrdf-sparqlquery, 2008 [retrieved: March, 2015].

[3] Martínez-Prieto, Miguel A., Mario Arias Gallego, and Javier D. Fernández. "Exchange and consumption of huge RDF data." The Semantic Web: Research and Applications. Springer Berlin Heidelberg, 2012. pp. 437-452.

[4] Neumann, Thomas, Gerhard Weikum. "The RDF-3X engine for scalable management of RDF data." The VLDB Journal 19.1 (2010): pp. 91-113.

[5] NVIDIA CORPORATION. 2013. Nvidia GPU Programming Guide for Geforce 8 and later GPUs

[6] Schmidt, Michael, et al. "SP^ 2Bench: a SPARQL performance benchmark." Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on. IEEE, 2009.

[7] Senn, Jürg. "Parallel Join Processing on Graphics Processors for the Resource Description Framework." Architecture of Computing Systems (ARCS), 2010 23rd International Conference on. VDE, 2010. pp. 1-8

[8] Dimitrov, Martin, Mike Mantor, and Huiyang Zhou. "Understanding software approaches for GPGPU reliability."

 ISBN: 978-1-61208-408-4

Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units. ACM, 2009.

[9] Quilitz, Bastian, and Ulf Leser. Querying distributed RDF data sources with SPARQL. Springer Berlin Heidelberg, 2008.

[10] Jena, Apache. "Apache jena." jena. apache. org. Available: http://jena. apache. Org, 2013 [Accessed: Mar. 20, 2014].

[11] Atre, Medha, and James A. Hendler. "BitMat: a main memory bit-matrix of RDF triples." The 5th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2009). 2009.

 ISBN: 978-1-61208-408-4