IRSMG: Accelerating Inexact RDF Subgraph Matching on the GPU

Junzhao Zhang[1,3], Bingyi Zhang[1,3], Xiaowang Zhang[1,3,*], and Zhiyong Feng[2,3]

[1] School of Computer Science and Technology, Tianjin University, Tianjin, China
[2] School of Computer Software, Tianjin University, Tianjin, China
[3] Tianjin Key Laboratory of Cognitive Computing and Application, Tianjin, China
∗ Corresponding author: xiaowangzhang@tju.edu.cn

Abstract. Many approaches have been proposed to solve subgraph matching problem based on filter-and-refine strategy. The efficiency of those existing serial approaches relies on the computational capabilities of CPU. In this paper, we propose an RDF subgraph matching algorithm based on type-isomorphism using GPU since GPU has higher computational performance, more scalability, and lower price than CPU. Firstly, we present a concurrent matching model for type-isomorphism so that subgraph matching can be tackled in a parallel way. Secondly, we develop a parallel algorithm for capturing our proposed concurrent matching model and implement a prototype called IRSMG using GPU. Finally, we evaluate IRSMG on the benchmark datasets LUBM. The experiments show that IRSMG significantly outperforms the state-of-the-art algorithms on the CPU.

1 Introduction

Subgraph matching, also known as subgraph isomorphism, is a widely known NP-complete problem [3]. The rapid growth of RDF data and high complexity of SPARQL query language together make a significant challenge to process RDF subgraph matching efficiently. To deal with subgraph matching over larger graphs, many algorithms explore basic graph pattern to achieve better performance [1]. Since those algorithms are designed for the architecture of CPU, their efficiencies rely on the computational capabilities of CPU. However, the clock rate of CPU has almost reached its peak recently. Compared to CPU, GPU provides a higher level of parallelism by reducing the complexity of a single compute unit and thus they are referred to as massively parallel hardware [2].

In this paper, we propose a parallel RDF subgraph matching on GPU and implement a prototype called IRSMG (*I*nexact *R*DF *S*ubgraph *M*atching on the *G*PU). where inexact RDF subgraph matching is a generalization of the exact RDF subgraph matching [5]. The major contributions of our work are summarized as follows:

– We propose a concurrent RDF subgraph matching model by extending type-isomorphism for supporting massively parallel processors.

– We develop a GPU-based parallel matching algorithm for embedding our proposed matching model on GPU.
– We implement and evaluate our proposal IRSMG on the benchmark datasets LUBM.

The experiments show that IRSMG significantly outperforms the state-of-the-art methods approximately 4 times.

2 Concurrent matching model of subgraph matching

Type-isomorphism Let $G = (V_G, E_G, \mu, \sigma)$ be a semantic graph (as an object graph) and $G_P = (V_P, E_P, \mu, \sigma)$ be a connected semantic graph (as a subject graph), where labeling functions map to a common labeling alphabet; L_V and L_E are sets of discrete symbols; $L_{V_P} \subseteq L_{V_G}$ and $L_{E_P} \subseteq L_{E_G}$. Let $W_P = (v_0^P, e_0^P, v_1^P, \ldots, e_{k-1}^P, v_k^P)$ be a k-edge walk over G_P, where $v_i^P \in V_P$ and $e_i^P \in E$.

Then a k-edge walk $W_G = (v_0^G, e_0^G, v_1^G, \ldots, e_{k-1}^G, v_k^G)$ over G is a *type-isomorphic match* [5] iff $C_\mu(v_i^P, v_i^G) = 1$ for $j = 0, \ldots, k$ and $C_\sigma(e_j^P, e_j^G) = 1$ for $j = 0, \ldots, k-1$.

Type comparator $C_\mu(v_a, v_b)$ returns true if the labels for v_a and v_b match, which also goes well for $C_\sigma(e_i, e_j)$.

Basic graph pattern (BGP) query and its concurrent strategy BGP query [4], as the basic form and main subset of SPARQL query, is a query based on BGP. Other (graph) patterns in SPARQL such as union pattern, optional pattern, can be converted to BGP with additional algebra operation.

Concurrently, BGP query can be decomposed into two basic operations: *mapping* and *join* [**?**]. Suppose that *tp* is a triple pattern in BGP query, which matches all the triples in RDF graphs, and let E_{tp} be the candidate triple set that satisfy the condition in *tp*, this process of screening is called *mapping*. If a type-isomorphic walk with k length, the mapping set has k separate portions while all those sets have no relationship. Note that E_{tp}^i and E_{tp}^{i+1} have the vertex whose *id* is the same. That is to say, the two triples shape a walk. If all E_{tp}^i can conduct a walk with k-length then all the type-isomorphism of BGP query in RDF data graph are found.

Because the GPU cannot support dynamic memory allocation on the device memory during the execution of the GPU code, we will employ plain arrays as the main data structure. As to RDF graphs, each triple is composed of subject, predicate, and object. All elements in triples are string values (i.e. URIs and literal objects). When parsing a triple, both subject and object are treated equivalently while the predicate is treated separately. The dictionary is built to convert string values to integers. Therefore, we get a new quintuple: a set of label values associated with each vertex (also referred to subject and object) and another set of label values associated with each edge (predicate).

Join Algorithm in GPU Join operation merges matched triples or partial matched graphs into a matched graph. The output of mappings is delivered to join if a BGP contains one or more shared variables. If a BGP has no shared variables, there is no reason to proceed next join operation. As to type-isomorphism, if two matched triples have a shared variable, they can be merged into a walk.

In order to make full use of parallelization advantages, the join process contains three phases: mapping parallel, sort by join *id*, and reducing duplicate. When mapping parallel, we set flag left or right according to the shared variable's position in triples. The flag left or right contributes to reducing unnecessary computation in the stage of reducing duplicate. GPU's single instruction multiple data architecture [2] contributes to accelerating cartesian product parallel.

Algorithm 1 IRSMG: Find all type-isomorphism matches using GPU

Input: S, the set of triples in RDF, one for each edges in graph.
T_p, stored as an ordered list of k segments.
Output: P, the set of matching walks in RDF.
1: **InitWalks:** Read a segment S_g from S.
If it matches the first segment (S_1, P_1, O_1) from W_P,
then convert S_g into a candidate walk of length 1.
2: **for t = 2 to k**
3: **ExtendWalks:** Read a segment S_g from S. If it matches the first segment (S_t, P_t, O_t) from W_P, then convert S_g into a candidate walk of length t.
Also read a candidate walk from the most recent candidate list.
4: **Use bitonic sorting to order candidate walk**
5: **Reduce duplicate:**
6: (Iterate over records with a given vertex key, sorted into two groups.)
7: Group 1, candidate walks of length $t-1$ from the *ExtendWalks*.
8: Group 2, segments S_g from the *ExtendWalks* mapper.
9: For each vertex key:
10: **loop** over all members of *CandWalkList*
11: If Cartesian product is non-null, then append S_g to make a candidate walk of length t.
12: **end loop**

GPU-based Algorithm Algorithm 1 starts with *InitWalks*, which chooses segments from an RDF graph S that matches the first segment of the pattern W_p. The loop beginning at line 2 runs a matching and join operation at each iteration to extend candidate walks by one segment. The mapper emits segments and walks with vertex ID as key. Matching is implemented in line 3. A bitonic sort (line 4) [2] is used to sort candidate walks by shared variables vertex key. The remaining parts of Algorithm 1 (line 8 to line 12) perform cartesian product operation to reduce the duplicate. IRSMG iterates through W_p one segment at a time with constructing all candidate walks in M that match the segments considered so far. Clearly, the set of full length directed paths in M generates the set of all type-isomorphic walks.

3 Experiments and evaluations

Our experiments were performed on a PC with a GTX590 GPU and an Intel Quad-Core CPU 2.66GHz running Ubuntu 14.04 64-bit. The main memory is 2GB, and the device memory of the GPU is 1536MB. We employ LUBM [4] as the benchmark dataset in our experiments to compare the performance with its CPU implementation in different dataset scale.

The experimental results are shown in Figure 1. With the increasing scale of data, the query time increases and the speedup grows up to four times. If computation is done on the GPU, data must be copied over the PCI express bus to the device and results have to be copied back. That explains the reason that queries on CPU is faster than queries on GPU when the scale of dataset is small.

As the increase of data scale, data transfer time between the device and the host takes smaller proportions of the whole process time, thus the query time increases while the growth rate becomes slower and slower.

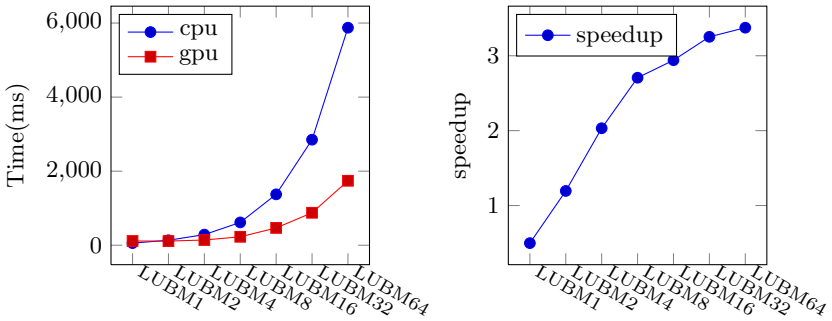

Fig. 1: Querying running time comparison and SpeedUp

Acknowledgement. This work is supported by the program of the National Key Research and Development Program of China (2016YFB1000603) and the National Natural Science Foundation of China (NSFC) (61672377).

References

1. Berry, J.W., Hendrickson, B., Kahan, S., & Konecny, P.: Software and algorithms for graph queries on multithreaded architectures. In: *Proc. of IPDPS 2007*, pp.1–14, 2007.
2. CUDA best practices guide. v5. 5. NVIDIA, May 2013.
3. Garey, M.R. & Johnson, D.S.: Computers and intractability: A guide to the theory of NP-completeness. W.H. Freeman and Company, 1979.
4. Perez, J., Arenas, A., & Gutierrez, C.: Semantics and complexity of SPARQL. *ACM Transactions on Database Systems*, 34(3):30–43, 2009.
5. Plantenga, T.: Inexact subgraph isomorphism in MapReduce. *Journal of Parallel and Distributed Computing*, 73(2): 164–175, 2013.

[4] `http://swat.cse.lehigh.edu/projects/lubm/`