Optimizing RDF stores by coupling General-purpose Graphics Processing Units and Central Processing Units

Bassem Makni

Tetherless World Constellation, Rensselaer Polytechnic Institute
110 8th Street, Troy, NY 12180
maknib@rpi.edu
http://tw.rpi.edu/

Abstract. From our experience in using RDF stores as a backend for social media streams, we pinpoint three shortcomings of current RDF stores in terms of aggregation speed, constraints checking and large-scale reasoning. Parallel algorithms are being proposed to scale reasoning on RDF graphs. However the current efforts focus on the closure computation using High Performance Computing (HPC) and require pre-materialization of the entailed triples before loading the generated graph into RDF stores, thus not suitable for continuously changing graphs. We propose a hybrid approach using General-purpose Graphics Processing Units (GPGPU) and Central Processing Units (CPU) in order to optimize three aspects of RDF stores: aggregation, constraints checking, and dynamic materialization.

1 Problem Statement

Social media graphs and streams fit naturally with the graph structure, and the Semantic Web offers the required platform to enrich, analyze and reason about social media graphs. However from our practical experience in storing, querying and reasoning about social media streams, we encountered the following problems:

1.1 Loading continuous stream of data

Performing constraints checking at load time, with *rdfs:domain* and *rdfs:range* for example, slows down adding new triples, and may result in a long queue of triples generated from the social media streaming API, waiting to be inserted into the graph. Turning off the constraints checking, at the other hand provides faster loading but may result in integrity violations in the graph.

1.2 SPARQL query performance

Running time of aggregation queries raises significantly with the size of the graph, however these aggregate functions are the most used to analyze the social media graph by queries like: "Return the number of persons tweeting about certain hashtag".
Moreover, SPARQL queries with large results, such as the ones extracting *mentions network* or returning millions of tweets with certain hashtags, for instance to calculate their sentiments, generate timeouts. Breaking down these large queries with offset and limit is not of big help as offset requires ordered triples and ordering big number of triples timeouts as well. This observation is partially supported by the Berlin SPARQL benchmark [1], and discussed by the Semantic Web community [2].

1.3 Large-scale reasoning

One of the major advantages of Semantic Web, is the ability to reason about the data. With the Linked Data movement gaining momentum, the amount of published RDF data is increasing significantly. And reasoning about large-scale RDF data, became a bottleneck. Recent research efforts tried to scale reasoning capability by parallelizing the closure computation. Thus the entailed RDF triples are materialized by forward-chaining reasoning. We call these approaches "offline parallelization", as the RDF graph needs to be generated offline on a cluster or super-computer before being loaded into an RDF store. However these offline approaches suffer from three drawbacks:

1. Forward-chaining generates a big number of entailed triples: BigOWLIM [3], for example, generated 8.43 billion implicit statements when loading 12.03 billion triples from the LUBM(90 000) [4] benchmark. However, not all the generated triples are usable at query time, which imply slowing down the RDF store by loading unusable implicit triples. That is one of the reasons why backward and hybrid chaining are more used in RDF stores. In Backward chaining, the entailments are computed at query time, and in hybrid chaining, a small number of rules is used in forward chaining and the rest are inferred at query time.
2. Offline parallelization is not practical for continuously changing graphs, as the closure needs to be recomputed on the supercomputer and loaded again in the RDF store whenever the graph changes. That may not be the case if the change is only by adding new triples, but deleting triples require necessarily the closure re-computation when the deleted triple affect the implicit generated triples.
3. They require HPC access and skills, as the user needs to compute the closure on a cluster or super-computer before being able to load and interact with the data via SPARQL.

We are particularly interested in reasoning about dynamic, i.e. continuously changing graphs, such as the social media graphs. So the need for an "online

parallelization" approach that is integrated into the RDF store and does not require previous processing of the graph.

2 Relevancy

Reasoning about large-scale and continuously changing RDF graphs, requires at the same time an online and parallel approach.
To the best of our knowledge, GPGPU are not exploited yet to optimize RDF stores reasoning and aggregation capabilities. Our goal is to design an RDF store that uses a hybrid GPGPU-CPU architecture, in order to support large-scale reasoning and optimize aggregation queries. The design should also provide parallelization capabilities in a transparent way, so the user will interact with the GPGPU-CPU RDF store in the same way he interacts with any CPU-only RDF store, without the need for HPC skills.

3 Related Work

The literature contains three classes of relevant works to our proposal: the first class is about closure computation parallelization, the second exploits GPGPU capabilities for graphs processing and the latter is about databases optimization using GPGPU.

3.1 Parallelization of closure computation

Three particular works will be of inspiration when designing the GPGPU-CPU RDF store, which are: Scalable distributed reasoning using MapReduce [5], Parallel materialization of the finite RDFS closure for hundreds of millions of triples [6] and The design and implementation of minimal RDFS backward reasoning in 4store [7]. The first two fit into the "offline parallelization" category, and the later uses parallel implementation of backward-chaining to support reasoning for the cluster based, 4store SPARQL engine. In the first paper the authors start from a naïve application of MapReduce on RDF graphs to adding a bunch of heuristics to optimize the parallelization of the closure computation. And in the second article, the authors describe an Message Passing Interface (MPI) implementation of their embarrassingly parallel algorithm to materialize the implicit triples.

3.2 Graphs processing on GPGPU

Unlike the early GPU units which were intended for graphical processing only, the General-purpose computing on graphics processing units can be used in more flexible ways via frameworks like Open Computing Language (OpenCL) [8] and Compute Unified Device Architecture (CUDA) [9]. The main difference between GPUs and CPUs is that GPUs launch a big number of light and slow threads

that run the same code in parallel, while CPUs run one fast process in serial. The parallel reduction, is a good illustration of the use of parallel threads on GPGPU to achieve high performance results. Figure 1 from the CUDA reduction white paper [10] illustrates the use of GPGPU to calculate the sum of an array.

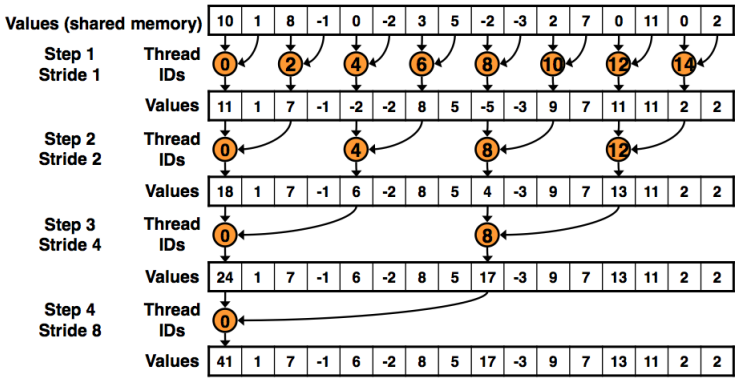

Fig. 1. Parallel Reduction: Interleaved Addressing

The literature contains many parallel algorithms that use GPGPU for fast sorting [11], DNA sequence alignment [12], etc. We focus on graphs processing algorithms, as they raise similar research problems to RDF processing, especially the data partition. In the early works on graph processing on the GPU, [13] achieved the performance of 1 second, 1.5 seconds and 2 minutes, respectively for a breadth-first search (BFS) single-source shortest path (SSSP), and all-pairs shortest path (APSP) algorithms on a 10 millions vertices graph. However these algorithms are limited to graphs with less than 12 millions vertices, as they load the whole graph into the GPGPU shared memory and they don't tackle the data partition issue. Recent research works break this scalability barrier by enabling graph data distribution. [14] use a multi-GPU architecture and balance the load of data between the different GPU devices.

3.3 Relational databases optimization on GPGPU

This research field tries to solve the similar problems that we are tackling for relational databases. In [15], the authors describe their GPU based, in-memory relational database GDB. [16] instead uses the traditional relational databases, and accelerates the SQL operations on a GPU with CUDA.

4 Research Questions

Unlike MPI and cluster based parallelization algorithms, which use the large memory available on every node, the GPGPU based algorithms are restrained by the relatively small memory of the GPU, and copying the data from the CPU to the GPU memory for parallel processing is an expensive operation. These constraints raise the following research questions when designing a GPGPU-CPU based RDF store:

Data partition One of the major constraints of GPU programming, is the small amount of dedicated graphical memory. Thus the need to swap data between the CPU memory or hard drives and the GPU memory. This need raises the research question of partitioning the RDF graph data, and the choice of the data slice that should be copied to GPU for parallel processing.

Dynamic computation As copying the data to GPU memory is time consuming, the overall speedup will depend on the parallel computation on the GPU and the data swap. If the speedup is not big, it is more efficient to compute on the CPU without the need to move the data. So the need to precompute this speedup in order to dynamically choose between CPU or GPU processing.

GPU caching In order to limit the number of memory fetches by the GPU, we need to adapt a caching mechanism to the small amount of memory, in order to maintain frequently used triples and rules in the shared memory.

Reasoning We need to select from the panoply of reasoning algorithms such as tableaux based, forward chaining, backward chaining etc. the one that is most adaptable to the GPU architecture, and can take benefit of the GPU parallelization.

5 Hypotheses

We design our GPGPU-CPU hybrid RDF store with the following hypothesis:

1. The graph structure is continuously changing, by adding new triples via social media streams. Triples can be deleted occasionally when the follower/friend relation is deleted for instance.

2. The user would interact with the RDF store in the same way he interacts with CPU based RDF stores, and the GPGPU parallelization should run in a transparent way.

6 Approach

We break down our approach into three steps related to the contributions we are promoting:

6.1 Optimizing SPARQL aggregation and ordering queries

In the first step, we are planning to integrate the CUDA reductions into the SPARQL runtime process. For example a query containing the *min* aggregator in Jena [17] is handled by the *AggMaxBase* class which go through the graph nodes and calculates sequentially the comparisons to the actual max. A GPGPU version will instead gather the whole sequence of bindings without any comparison and run a GPU reduction to get the max value in one shot.
In this step, we will get more familiar with the GPGPU programming and technical limitations, the first results of this step will allow us to tweak further steps plan.

6.2 Parallel constraints checking

One of the problems that we mentioned earlier in this paper, is the slow down of load speed, resulting from the constraints checking. We speculate that a GPGPU parallel version of constraints checking will optimize this phase and maintain the speed of inserting new coming triples from the social media streams. In this step we will get more familiar with lightweight reasoning on GPGPU.

6.3 Dynamic materialization on GPGPU

This step will be the core of the thesis work, as it raises the majority of the research questions discussed in this paper. We will use the lessons learnt from the previous steps in order to achieve parallel dynamic materialization of triples at the query time.

7 Reflections

GPGPU are being exploited by many research communities in order to provide cheap and fast parallelization of their algorithms. Successful results are achieved in bioinformatics, graphs processing, relational databases and other fields. We believe that exploiting GPGPU to optimize SPARQL queries and large RDF data processing is a fruitful research direction.
Though the problem of dynamic materialization on GPU, raises many research questions, we broke down our approach into intermediate warm-up steps before tackling this problematic.

8 Evaluation plan

We are planning to implement a GPGPU version of one, possibly two of the following RDF stores: Jena, Sesame [18] and Open Virtuoso [19]. In the evaluation phase, we plan to use SPARQL benchmarks such as the Berlin SPARQL benchmark [1] in order to compare the CPU only and the GPGPU-CPU hybrid version of each implemented RDF store.

As these benchmarks are not designed for streaming data, we will also use SRBench, the Streaming RDF/SPARQL Benchmark [20] which is specific for streaming RDF data. In [20], the authors propose a comprehensive set of queries in order to compare the main streaming engines available in the state of the art, namely $\text{SPARQL}_{\text{Stream}}$ [21], C-SPARQL [22] and CQELS [23].
Another possible benchmark to evaluate our approach, comes from the graphs processing community which is Graph500 [24].

Acknowledgements

I would like to express my deep gratitude to my advisor Prof. James Hendler and to Prof. Deborah McGuinness for their guidance in this research proposal. This work was supported in part by the DARPA SMISC program.

References

1. Bizer, C., Schultz, A.: The berlin sparql benchmark. International Journal on Semantic Web and Information Systems (IJSWIS) **5**(2) (2009) 1–24
2. semanticweb.com: Set of real-world sparql benchmark queries. `http://answers.semanticweb.com/questions/1847/set-of-real-world-sparql-benchmark-queries` (2010)
3. Kiryakov, A., Bishop, B., Ognyanoff, D., Peikov, I., Tashev, Z., Velkov, R.: The features of bigowlim that enabled the bbcs world cup website. In: Workshop on Semantic Data Management SemData VLDB. (2010)
4. Guo, Y., Pan, Z., Heflin, J.: Lubm: A benchmark for owl knowledge base systems. Web Semantics: Science, Services and Agents on the World Wide Web **3**(2-3) (2011)
5. Urbani, J., Kotoulas, S., Oren, E., Van Harmelen, F.: Scalable distributed reasoning using mapreduce. In: The Semantic Web-ISWC 2009. Springer (2009) 634–649
6. Weaver, J., Hendler, J.A.: Parallel materialization of the finite rdfs closure for hundreds of millions of triples. In: The Semantic Web-ISWC 2009. Springer (2009) 682–697
7. Salvadores, M., Correndo, G., Harris, S., Gibbins, N., Shadbolt, N.: The design and implementation of minimal rdfs backward reasoning in 4store. In: The Semanic Web: Research and Applications. Springer (2011) 139–153
8. Munshi, A., et al.: The opencl specification. Khronos OpenCL Working Group **1** (2009) l1–15
9. Nvidia, C.: Programming guide (2008)
10. Harris, M.: Optimizing parallel reduction in cuda. NVIDIA Developer Technology **6** (2007)
11. Merrill, D.G., Grimshaw, A.S.: Revisiting sorting for gpgpu stream architectures. In: Proceedings of the 19th international conference on Parallel architectures and compilation techniques, ACM (2010) 545–546
12. Trapnell, C., Schatz, M.C.: Optimizing data intensive gpgpu computations for dna sequence alignment. Parallel Computing **35**(8) (2009) 429–440
13. Harish, P., Narayanan, P.: Accelerating large graph algorithms on the gpu using cuda. In: High Performance Computing–HiPC 2007. Springer (2007) 197–208
14. Merrill, D., Garland, M., Grimshaw, A.: Scalable gpu graph traversal. In: Proceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming, ACM (2012) 117–128

15. He, B., Lu, M., Yang, K., Fang, R., Govindaraju, N.K., Luo, Q., Sander, P.V.: Relational query coprocessing on graphics processors. ACM Transactions on Database Systems (TODS) **34**(4) (2009) 21
16. Bakkum, P., Skadron, K.: Accelerating sql database operations on a gpu with cuda. In: Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units, ACM (2010) 94–103
17. McBride, B.: Jena: Implementing the rdf model and syntax specification. (2001)
18. Broekstra, J., Kampman, A., Van Harmelen, F.: Sesame: A generic architecture for storing and querying rdf and rdf schema. In: The Semantic WebISWC 2002. Springer (2002) 54–68
19. Erling, O., Mikhailov, I.: Rdf support in the virtuoso dbms. In: Conference on Social Semantic Web. Volume 113. (2007) 59–68
20. Zhang, Y., Duc, P.M., Corcho, O., Calbimonte, J.P.: Srbench: a streaming rdf/sparql benchmark. In: The Semantic Web–ISWC 2012. Springer (2012) 641–657
21. Calbimonte, J.P., Corcho, O., Gray, A.J.: Enabling ontology-based access to streaming data sources. In: The Semantic Web–ISWC 2010. Springer (2010) 96–111
22. Barbieri, D.F., Braga, D., Ceri, S., Valle, E.D., Grossniklaus, M.: Querying rdf streams with c-sparql. ACM SIGMOD Record **39**(1) (2010) 20–26
23. Le-Phuoc, D., Dao-Tran, M., Parreira, J.X., Hauswirth, M.: A native and adaptive approach for unified processing of linked streams and linked data. In: The Semantic Web–ISWC 2011. Springer (2011) 370–388
24. Murphy, R.C., Wheeler, K.B., Barrett, B.W., Ang, J.A.: Introducing the graph 500. Cray Users Group (CUG) (2010)