Processing Markov Logic Networks with GPUs

Carlos Alberto Martínez-Angeles[1], Inês Dutra[2], Vítor Santos Costa[2], and Jorge Buenabad-Chávez[1]

[1] Departamento de Computación, CINVESTAV-IPN
Av. Instituto Politécnico Nacional 2508, México D.F. 07360, México.
[2] Departamento de Ciência de Computadores, CRACS INESC-TEC LA and Universidade do Porto, Rua do Campo Alegre 1021, 4169-007, Porto, Portugal
camartinez@cinvestav.mx, jbuenabad@cs.cinvestav.mx, {ines,vsc}@dcc.fc.up.pt

Abstract. Graphics Processing Units (GPUs) are being widely used to improve performance of machine learning and logic programming systems. Next, we propose using this technique to improve the performance of Markov logic programs. In this paper we focus on the first step of the inference phase, the *grounding* of first-order logical formulas composing a Markov network. Our system, Tu2GPU, is based on the state-of-the-art high-performance Tuffy system. We compare Tu2GPU's performance to that of the Alchemy, Tuffy and RockIt systems on three widely used applications. Results show that Tu2GPU is up to 15x times faster than the other systems.

Keywords: Statistical Relational Learning, Markov Logic, Markov Logic Networks, Datalog, Parallel Computing, GPUs

1 Introduction

Markov logic "combines first-order logic and Markov networks. A knowledge base in Markov logic is a set of first-order [logic] formulas with weights" [2]. The structure of such formulas and their weights is managed as a Markov network, such that they establish *soft constraints*: worlds that violate a formula are less likely but still possible. In contrast, formulas in standard first-order logic are hard constraints: a world that violates a formula is not possible.

Markov logic networks have been widely adopted, and various systems have been developed with it including: the system by We *et al.* to refine Wikipedia's Infobox Ontology [10]; Riedel and Meza-Ruiz's system to carry out collective semantic role labelling [7]; and in Natural Language Processing (NLP) [2, p. 97].

We claim that GPU processing can significantly expedite grounding, the most time consuming step of MLN processing which has been found to take up to 96% of the execution time of some applications [4]. To verify our hypothesis, we designed Tu2GPU, a Markov Logic system based on: Tuffy [4], YAP Prolog [8], and GPU-Datalog, a GPU-based engine that evaluates Datalog programs [3]. We compare the performance of Tu2GPU to that of Alchemy [2], Tuffy and RockIt [5], with three applications: information extraction, entity resolution and relational classification. The performance of Tu2GPU is on par or better than the other systems for most applications.

2 Related Work

The wide adoption of Markov logic for various types of applications has fostered the development of various systems and research on improvements. Alchemy was the first Markov logic system implementation [2]. It is one of the most complete systems, including various algorithms for inference following a top-down approach and various techniques for learning weights and structure.

Tuffy was developed by Feng Niu *et al.* [4]. It relies on Postgresql relational database management system (RDBMS) to perform inference. Tuffy follows a bottom-up approach to solve the grounding step. This allows the grounding to be expressed as SQL queries which, combined with query optimization by the RDBMS, allows Tuffy to complete the grounding faster than Alchemy [4].

RockIt is a recent system by Noessner *et al.* [5]. It treats the inference problem as an integer linear programming problem and includes a new technique called cutting plane aggregation (CPA) which, coupled with shared-memory multi-core parallelism during most of the inference, allow RockIt to outperform all other systems. Other work on Markov logic relevant to ours include [1] and [6].

3 Markov Logic

We present in this section some Markov logic concepts through an example program. The example is presented using Datalog syntax. Markov logic programs and Datalog programs consist of a finite number of facts (Extensional Database [9]) and rules (Intensional Database), except that Datalog rules have no weights attached. Figure 1 shows the *Smokers* Markov logic program [2], which estimates the probability of people having cancer based on who their friends are and whether or not their friends smoke. In Markov logic, a knowledge base (KB) is a set of first-order logic formulas with weights [2]. The *Smokers* KB is shown at the top of Figure 1: in English and in first-order logic (on the left); in clausal form in Datalog syntax; and the weights of the formulas. The bottom of Figure 1 shows: (on the left) some *evidence* (known facts) as a set of records in two relational tables, Fr (Friends) and Sm (Smokes); some processing of the rules; and the results stating the probability that John, Anna and Bob may have cancer considering the given facts.

Rules can be created by domain experts, or learned from training data; Inductive Logic Programming (ILP) has been used for this purpose [2, p. 52]. Weight assignment is difficult because different formulas correlate with each other; hence it is almost always learned from training data [1, 2]. It is common to use the inference phase with an already configured KB and a number of facts in relational tables, as in our experiments in Section 5.

4 Our GPU-based Markov Logic Platform

Our platform Tu2GPU was designed to accelerate the grounding step, as this is often the most time consuming. Its main components are: the Tuffy Markov

```
ENGLISH AND FIRST-ORDER LOGIC            CLAUSAL FORM (Datalog syntax)            WEIGHT
-------------------------------------------------------------------------------------------
Friends of friends are friends:
   Fr(x,y) & Fr(y,z) => Fr(x,z)          Fr(x,z) :- Fr(x,y), Fr(y,z)                 0.7
Smoking causes cancer:
   Sm(x) => Ca(x)                        Ca(x) :- Sm(x)                              1.5
If two people are friends and one
smokes, then so does the other:
   Fr(x,y) & Sm(x) => Sm(y)              Sm(y) :- Fr(x,y), Sm(x)                     1.1

EVIDENCE                PROCESSING OF VALID GROUNDINGS                          RESULTS
-------------------------------------------------------------------------------------------
Fr(John,Anna)           Fr(John,Anna) & Fr(Anna,Bob) => Fr(John,Bob)       0.92 Ca(John)
Fr(Anna,Bob)            Sm(Anna) => Ca(Anna)                               0.59 Ca(Anna)
Fr(Gary,Frank)          Fr(John,Anna) & Sm(John) => Sm(Anna)               0.58 Ca(Bob)
Sm(John)
                        PROCESSING OF INVALID GROUNDINGS
                        --------------------------------
                        Fr(John, Anna) & Fr(Gary, Frank) => {}
                        Sm(Gary) => Ca(Gray)
                        Fr(Gary, Frank) & Sm(Gary) => Sm(Frank)
```

Fig. 1. The Markov logic *Smokers* example. Fr() is short for Friends(), Sm() for Smokes() and Ca() for Cancer().

logic system [4], the YAP Prolog system [8] and GPU-Datalog [3]. The latter evaluates Datalog programs with a bottom-up approach using GPU kernels that implement the relational algebra operations *selection, join* and *projection*. For GPU-Datalog to be able to run Markov logic programs, its original version was extended with: management of stratified negation; improved processing of built-in comparison predicates; processing of OR logic in addition to AND logic (to simplify specifying SQL queries and to improve their processing); and an interface to communicate directly with PostgreSQL.

Figure 2 shows the interaction between the main modules of our platform in running a Markov logic program. Tuffy is called first, receiving three input files: i) the evidence (facts) file; ii) the MLN program file; and the queries. Tuffy starts by creating a temporary database in PostgreSQL to store the evidence data and partial results. It then parses the program and query files in order to determine predicates and to create a (relational) table for each predicate found. Tables are then loaded with the evidence data.

Original Tuffy would then start the grounding phase. In Tu2GPU, this phase is performed by GPU-Datalog, but, as Tuffy uses OR-logic to specify a program, we first translate it to Datalog AND-logic and syntax. Then the Datalog program is sent to YAP, using a Java-Prolog interface, to compile it into a numerical representation (NR) where each unique string is assigned a unique integer id. YAP then sends the program NR to GPU-Datalog to process the grounding. By using an NR, our GPU kernels show relatively short and *constant* processing time because all tuples in a table, being managed as sets of integers, can be processed in the same amount of time. Tuffy also uses an NR for evidence loaded in the database; this simplified extending it with GPU processing.

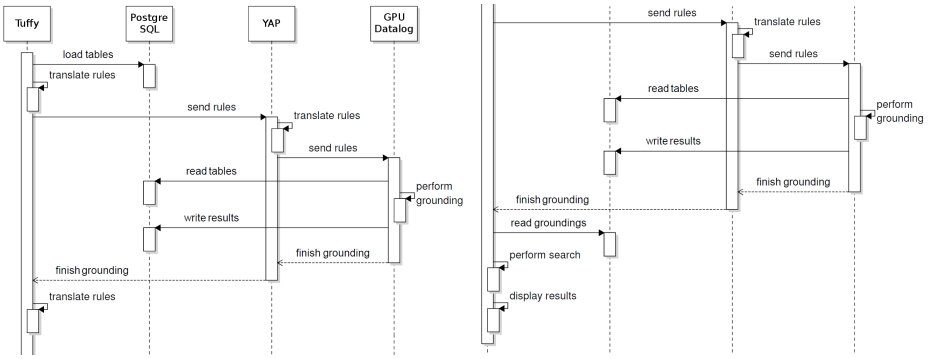

Fig. 2. Tu2GPU-Datalog modules running a Markov logic program.

Tuffy and Tu2GPU use the Knowledge-Based Model Construction [4] (KBMC) algorithm to determine those clauses, facts and rules that are relevant to a query. The *relevant* facts are in the database and GPU-Datalog reads them from there to perform the first step (of two) of the grounding process: determining the *active facts*, i.e., facts whose truth value *flips* (changes from true to false or vice versa) during the grounding process. GPU-Datalog writes back to the database those active facts whose truth value changed to true. The second step determines the *active rules*, rules that can be violated (i.e., if their truth value becomes false) by flipping zero or more active facts or subgoals relevant to the rules. For this step, Tu2GPU translates the program rules from SQL into Datalog, and YAP into the NR used by GPU-Datalog. When GPU-Datalog finishes the grounding, it writes the found *active rules* to the database. At the end of both grounding steps, Tuffy searches for the most likely world of the MLN and displays the result.

5 Experimental Evaluation

This section describes our experimental evaluation of the performance of Tu2GPU compared to that of the systems Alchemy [2], Tuffy [4] and RockIt [5].

5.1 Applications and Hardware-software Platform

We used the following applications available with the Tuffy package. Table 1 shows some of their characteristics. For two of them (ER and RC), more tuples were randomly generated to test the systems with bigger data (right column).

- *Entity Resolution (ER)*: a simple, recursive MLN to determine if a person has cancer based on who his/her friends are and their smoking habits.
- *Relational Classification (RC)*: classifies papers into 10 categories based on authorship and on the categories of other papers it references.
- *Information Extraction (IE)*: given a set of citations divided in tokens, rules with constants are used to extract structured records.

Application	Inference rules	Evidence relations	Tuples in relations Original	Random
ER	3	3	8	(310,000)
RC	15	4	82,684	(441,074)
IE	1024	18	255,532	(na)

Table 1. Applications Characteristics.

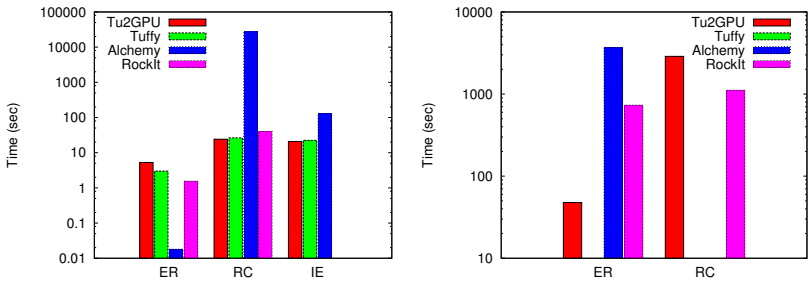

Fig. 3. Performance of the systems with original (left) and random (right) datasets. Note that the graphs are in log. scale.

We ran our experiments in the following hardware-software platform. **Host**: an AMD Opteron 6344, 12 cores CPU, with 64 GB DRAM. **GPU**: a Tesla K40c, 2880 CUDA Cores, with 12 GB GDDR5 memory and CUDA Capability 3.5. **Software**: CentOS 7, PostgreSQL 9.5 and CUDA Toolkit 7.0.

5.2 Results

Figure 3 (left) shows the performance of the systems using the original datasets in the Tuffy package. Our system was the fastest in 2 out of the 3 applications, but only by a few seconds relative to standard Tuffy. Alchemy was the fastest in ER because the dataset is small and does not incur overhead setting up a database. Figure 3 (right) shows the performance of the systems with our randomly generated datasets. For ER, our system was 15 times faster than RockIt and 77 times faster than Alchemy. Tuffy did not finish the grounding after more than 3 hours. However, RockIt was 2.5 times faster than our system for RC. Both Tuffy and Alchemy did not finish after more than 5 hours. We were unable to execute IE in RockIt, hence the empty space in the graph.

We performed a detailed analysis to determine why our system performed so well in ER and so poorly in RC. For ER, our random data combined with its recursive clauses, generates many more recursive steps, 24 vs 2 in the original data. Each recursive step creates new tuples that need to be evaluated again. In our system, approximately 1,000,000 new tuples where generated in each iteration, most of them to be latter discarded by our duplicate elimination kernel. Since our system was designed around these recursive applications, grounding

was finished rather quickly while other systems struggled with costly joins that do not capitalize on parallel processing.

In RC, the number of recursive steps was 2 for both datasets. We hence analyzed the execution times of each part of our system. Both groundings take about 2 minutes to complete, loading data and other tasks take 30 seconds, but the search phase takes an astounding 43 minutes. In contrast, the times for ER are about 8 seconds for both groundings, 21 seconds for data loading and other tasks, and 16 seconds for the search. In the search phase, ER, despite being a bigger dataset, uses only 252,249 tuples, while RC uses 5,586,900 tuples.

6 Conclusions

We have presented a system that accelerates the grounding step in MLNs by combining Tuffy with our GPU-Datalog engine. Its performance is on par or better than other well-known MLN systems. Our results show that the benefit of performing the grounding phase on the GPU outweighs the overhead of using a database and of GPU I/O, even for rather small datasets. Our system can be greatly improved by also performing the search step of the inference phase in the GPU. This would require the parallelization of a SAT solver. There are several available in the literature and we could adapt one to our needs.

Acknowledgements. Carlos gratefully acknowledges grants from Cinvestav-IPN.

References

1. K. Beedkar et al. Fully parallel inference in markov logic networks. In *15th GI-Symposium Database Systems for Business, Technology and Web*, Germany, 2013. Bonner Kllen.
2. P. Domingos and D. Lowd. *Markov Logic: An Interface Layer for Artificial Intelligence*. Morgan and Claypool, 1st edition, 2009.
3. C. A. Martínez-Angeles et al. Relational Learning with GPUs: Accelerating Rule Coverage. *Intl. J Parallel Programming*, IN PRESS, 2015.
4. F. Niu et al. Tuffy: Scaling Up Statistical Inference in Markov Logic Networks Using an RDBMS. *Proc. VLDB Endow.*, 4(6):373–384, March 2011.
5. J. Noessner et al. Rockit: Exploiting parallelism and symmetry for MAP inference in statistical relational models. *CoRR*, abs/1304.4379, 2013.
6. S. Riedel. Improving the accuracy and efficiency of map inference for markov logic. In *Proceedings of the 24th Annual Conference on Uncertainty in AI (UAI '08)*, pages 468–475, 2008.
7. S. Riedel and I. Meza-Ruiz. Collective semantic role labelling with markov logic. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning*, pages 193–197, Stroudsburg, 2008. Association for Computational Linguistics.
8. V. Santos-Costa et al. The YAP Prolog system. *TPLP*, 12(1-2):5–34, 2012.
9. J. Ullman. *Principles of Database and Knowledge-Base Systems, Volume I*. Computer Science Press, 1988.
10. F. Wu and D. S. Weld. Automatically refining the wikipedia infobox ontology. In *Proceedings of the 17th International Conference on World Wide Web*, pages 635–644, New York, 2008. ACM.