High Level Synthesis of SPARQL queries

Marco Minutoli* Vito Giovanni Castellana† Antonino Tumeo†
Pacific Northwest National Laboratory
Richland, WA, USA
*marco.minutoli@pnnl.gov †vitoGiovanni.Castellana@pnnl.gov ‡antonino.tumeo@pnnl.gov

***Abstract*—Resource Description Framework (RDF) databases naturally maps to labeled, directed graphs. SPARQL is a query language for RDF databases that expresses queries as graph pattern matching operations. GEMS is a RDF database that, differently from other solutions, employs graph methods at all levels of its stack. Graph methods are inherently task parallel, but they exhibit an irregular behavior. In this poster we discuss an approach to accelerate GEMS with reconfigurable devices. The proposed approach automatically generates parallel hardware implementations of SPARQL queries using a customized High Level Synthesis (HLS) flow. The flow has been enhanced with solutions to address limitations of graph methods with conventional HLS approaches, enhancing extraction of *TLP* and management of concurrent memory operations. We have validated our approach by synthesizing and simulating seven queries from LUBM. The hardware accelerators generated with our approach provide an average speed up of 2.1 with respect to their serial version.**

I. Introduction

The exponential growth in data availability is driving the need for effective methods to gain insight from them, for both business and governments. Challenges reside not only in the data set size, but also in the requirement to process them in real-time, enabling analysts to obtain actionable results and take effective decisions.

The *RDF* data model has seen a significant uptake as a way to organize these often poorly structured data. RDF represents data as subject-predicate-object triples, and it naturally maps to a directed labeled graph. SPARQL is a common query language for RDF datasets that describes queries as graph pattern matching operations. Graph database Engine for Multithreaded Systems (GEMS) [1] implements an RDF database on commodity clusters by mainly employing graph methods. GEMS is organized in three main components: the SPARQL-to-C++ translator, the Semantic Graph Library (SGLib), and the Global Memory and Threading (GMT) runtime. The SPARQL-to-C++ translator parses the input query and builds an Intermediate Representation (IR). The translator optimizes the IR to find an optimal query plan. Finally, it generates graph pattern matching routines in C++ that employs SGLib's Application Programming Interface (API). SGLib provides primitives to load and access the database, to perform graph traversal, and to manipulate the data structures. SGLib is built on top of the GMT runtime. Graph algorithms are task parallel but, unfortunately, they present fine grained, unpredictable data accesses, and are synchronization intensive. This reduces the efficiency of the memory subsystem and the network. Furthermore, graphs are data structures difficult to partition on distributed memory systems without generating load imbalance or hotspots. In summary, they determine an irregular behavior, while modern High Performance Computing (HPC) clusters are optimized for regular workloads. GMT overcomes some of these limitations by implementing a global address space across all cluster nodes, tolerating remote data access latency through lightweight software multithreading, and improving network utilization through message aggregation.

Field Programmable Gate Arrays (FPGAs) appear promising platforms to accelerate emerging workloads on HPC clusters. Previous works [2], [3] have already investigated solutions to accelerate queries (e.g., in SQL) on relational databases.

In this abstract, we present a solution and preliminary results for the acceleration of SPARQL queries on RDF databases through FPGAs. Our solution is based on GEMS and Bambu[4], an open source HLS tool. HLS provides automatic translation of codes in high level languages, such as C or C++, to Hardware Description Languages (HDLs). HDLs are then translated into netlists and loaded on the reconfigurable devices.

We have built a customized SPARQL-to-C translator to generate C implementations of queries that Bambu can synthesize. Moreover, we have developed an alternative graph API that does not use GMT. Bambu processes the output of the translator and applies code transformations to better expose Thread Level Parallelism (TLP). We have extended its synthesis flow to automatically synthesize custom accelerators that implement TLP in hardware through resource replication. The majority of HLS techniques adopts the Finite State Machine with Datapath (FSMD) as the target architecture. While very effective in exploiting Instruction Level Paralellism (ILP), this computational model is not efficient with TLP. To overcome the limitation of the FSMD model, we have adopted an alternative architecture design that handles concurrency through an adaptive, distributed, parallel controller [5] (PC). The PC consists of a set of interacting elements that control execution of operations and tasks, and their mapping to resources, through a token-passing mechanism, providing dynamic scheduling. In opposition to the static scheduling of a FSMD, the PC easily allows managing operations with unknown latency.

Parallel execution of tasks interacting with a shared memory space requires a memory architecture able to support concurrent memory operations. To solve this challenge, the template architecture of the synthesis flow includes an adaptive Memory

Table I
PERFORMANCE RESULTS

	Serial			T=4, M=4			Speed up	
	Latency (#Cycles)		Max Freq. (MHz)	Latency (#Cycles)		Max Freq. (MHz)		
	LUBM-1	LUBM-40		LUBM-1	LUMB-40		LUBM-1	LUMB-40
Q1	5,339,286	1,082,526,974	130.34	5,176,116	1,001,581,548	113.37	1.03	1.08
Q2	141,022	7,359,732	143.66	54,281	2,801,694	130.11	2.60	2.63
Q3	5,824,354	308,586,247	121.27	1,862,683	98,163,298	114.53	3.13	3.14
Q4	63,825	63,825	143.20	42,851	42,279	122.97	1.49	1.51
Q5	33,322	33,322	133.92	13,442	13,400	138.31	2.48	2.49
Q6	674,951	682,949	136.76	340,634	629,671	113.26	1.98	1.08
Q7	1,700,170	85,341,784	131.98	694,225	35,511,299	106.71	2.45	2.40

Table II
SYNTHESIS RESULTS

	Serial		T=4, M=4		Area Ov.	
	LUTs	Slices	LUTs	Slices	LUTs	Slices
Q1	5,600	1,802	13,469	4,317	2.40	2.39
Q2	2,690	8,24	5,280	1,607	1.96	1.95
Q3	5,525	1,775	13,449	4,308	2.43	2.43
Q4	3,477	1,073	7,806	2,399	2.24	2.24
Q5	2,785	848	5,750	1,738	2.06	2.05
Q6	4,364	1,369	10,600	3,426	2.43	2.50
Q7	6,194	1,943	15,002	4,953	2.42	2.55

Interface Controller (MIC)[6]. The MIC dynamically maps memory operations to multiple, distributed and/or multi-ported memories. It also provides atomic memory operations, exposed as "special" resources to the *HLS* flow. The PC dynamically schedules accesses of the parallel hardware tasks to the various memory ports, and naturally manages operations with variable latency, such as atomic memory operations.

We have evaluated our approach by synthesizing seven queries[7] from the LUBM benchmark[8]. We executed queries on two different data-sets: LUBM-1 (100,573 triples), and LUBM-40 (5,309,056 triples). We synthesized two different configurations of the architecture: a serial one, and one with 4 hardware kernels. In both cases, the memory architecture exposes 4 memory channels. Table I and Table II respectively report the simulation results (performance in clock Cycle) and the synthesis results (occupation on the FPGA), targeting a Xilinx Virtex 7 device. With our approach, the parallel implementation provides from 1.03 to 3.01 times the performance of the serial implementations depending on the query (average of 2.1). Cases with low performance gains depend on the structure of the query and the data-set.

REFERENCES

[1] V. Castellana, A. Morari, J. Weaver, *et al.*, "In-memory graph databases for web-scale data", *Computer*, vol. 48, no. 3, pp. 24–35, 2015.

[2] J. Casper and K. Olukotun, "Hardware acceleration of database operations", in *FPGA'14: 2014 ACM/SIGDA International Symposium on Field-programmable Gate Arrays*, 2014, pp. 151–160.

[3] R. Halstead, B. Sukhwani, H. Min, *et al.*, "Accelerating join operation for relational databases with fpgas", in *FCCM 2013: IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines*, 2013, pp. 17–20.

[4] *Bambu: A Free Framework for the High-Level Synthesis of Complex Applications*. [Online]. Available: http://panda.dei.polimi.it.

[5] V. G. Castellana and F. Ferrandi, "An automated flow for the high level synthesis of coarse grained parallel applications", in *FPT 2013: International Conference on Field-Programmable Technology*, 2013, pp. 294–301.

[6] V. G. Castellana, A. Tumeo, and F. Ferrandi, "An adaptive memory interface controller for improving bandwidth utilization of hybrid and reconfigurable systems", in *DATE 2014: Design, Automation and Test in Europe*, 2014, pp. 1–4.

[7] B. Shao, H. Wang, and Y. Li, "Trinity: a distributed graph engine on a memory cloud", in *ACM SIGMOD International Conference on Management of Data*, ACM, 2013, pp. 505–516.

[8] *SWAT Projects – the Lehigh University Benchmark (LUBM)*. [Online]. Available: http://swat.cse.lehigh.edu/projects/lubm.