Journal of Systems Architecture 61 (2015) 185–196

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

FPGA acceleration of semantic tree reasoning algorithms

Jesús Barba *, María José Santofimia, Julio Dondo, Fernando Rincón, Julián Caba, Juan Carlos López

School of Computer Science, University of Castilla-La Mancha, Fermín Caballero Building, Paseo de la Universidad, 4, 13170 Ciudad Real, Spain

ARTICLE INFO

Article history:
Received 17 November 2013
Received in revised form 1 September 2014
Accepted 15 January 2015
Available online 7 February 2015

Keywords:
Dynamic recursive data structures
Hardware acceleration
Reconfigurable logic
Artificial Intelligence
Reasoning algorithms

ABSTRACT

Semantic trees are a particular type of trees widely used in the representation of the concepts and their relations. Therefore, a computational model of the reality can be built and processed by Artificial Intelligence algorithms to infer knowledge, make decisions, etc. In this work, the design of a hardware component to accelerate reasoning operations on semantic trees by means of an FPGA based platform is presented. The target application is common-sense reasoning where marker-passing algorithms work on semantic tree structures; the core of the Scone Knowledge-Based system.

On top of the functionality to be implemented, a strategy to deal with the implementation in reconfigurable hardware of dynamic and recursive data structures has been envisioned. Since lists, graphs or trees are the cornerstone in the modelling of computer friendly solutions for complex problems; this proposal contributes to reduce the breach between the software and silicon domains.

As a result, an optimized micro-architecture of an FPGA accelerator for marker-passer algorithms integrated into a heterogeneous computing platform, and a smart data mapping procedure have been delivered. The design has been prototyped on a Xilinx ML507 board and compared to an equivalent software implementation, showing a significant reduction in execution times.

1. Introduction

The use of complex data structures is almost unavoidable when facing the implementation of computer friendly algorithms. These artifacts support software developers in the modeling process of key concepts in the domain of the problem. In this sense, the mental process of abstracting the real world, devising it as a collection of data structures and operations to perform on them, is determinant in the success and efficiency achieved by the developer of software algorithms.

The use of specialized hardware can help to speed up the manipulation of these kinds of recursive data structures. In general, the manipulation of these data structures in software is costly in terms of latency due, for example, to the irregular memory access patterns which spoil cache locality benefits.

Nevertheless, there are several factors to take into account before facing the implementation of hardware accelerators in this context. For example, the manipulation of graphs, lists or trees is control-oriented whereas FPGA circuits exhibit the best results for data-oriented applications. Also, the design of circuits for recursive dynamic data types is not straightforward due to the static nature of the silicon.

Hardware architects should count on building blocks or templates for implementing complex data types structures (so far, stack or FIFO components are the most and only representative example of standardized hardware module), just as software developers relies on the concept of pointer. A pointer stores a reference to memory (address) where actual data is stored. Therefore, pointers enable the implementation of indexed and indirect addressing modes, which are of great utility to build complex data structures. These structures implement a predefined organization of data in memory to be easily retrieved and interpreted later by a routine running in the processor. Moreover, pointers support dynamic growth of program variables, for example adding new node elements to a list structure.

In this article, we present the design of a hardware component to accelerate reasoning operations on semantic trees and an FPGA based platform to support a scalable implementation of the Scone Knowledge-Based system [1]. Scone is an open-source project that provides an expressive, easy to use, scalable and efficient approach for accomplishing search and inference operations on semantic trees. Scone, which inspires this work, proposes a *marker-passing* strategy to extract the implicit knowledge.

A semantic tree is a specialization of a tree, which is a collection of nodes hierarchically arranged, where each node points to one or

* Corresponding author. Tel.: +34 926 29 53 00 x3708.
E-mail addresses: jesus.barba@uclm.es (J. Barba), mariajose.santofimia@uclm.es (M.J. Santofimia), juliodaniel.dondo@uclm.es (J. Dondo), fernando.rincon@uclm.es (F. Rincón), julian.caba@uclm.es (J. Caba), juancarlos.lopez@uclm.es (J.C. López).

http://dx.doi.org/10.1016/j.sysarc.2015.01.001
1383-7621/

more nodes at the next level of the hierarchy. Nodes in semantic trees represent facts of the world (a concept) and links represent relations between concepts. Semantic trees are widely used in the representation and modeling of knowledge in the Artificial Intelligence field. Inference or reasoning processes use algorithms that basically go through the semantic tree in order to select the node or nodes that match the search premises.

The final goal is to provide a high performance, scalable and flexible platform for our target application. At the same time, a solution for the implementation of recursive data structures in this context is provided. This work helps to reduce the semantic gap between hardware and software which is preventing many application domains from taking advantage of the use of acceleration units implemented as digital circuits.

The implementation of the Scone algorithms has been performed using a Xilinx ML507 prototyping board, with a Virtex-5 LX110T FPGA (*Field Programmable Gate Array*) chip. The process does not only imply the design of the microarchitecture, but also the study of the memory technology available in the FPGA, in order to optimize the memory organization. Also, a keen representation in memory of the semantic network is provided contributing to achieve the efficiency goal.

The remainder of this paper is organized as follows: Section 2 refers to previous work regarding hardware implementations of algorithms related to complex, dynamic and recursive data structures. Section 3 describes the marker-passer algorithm and the reasoning process based on the use of semantic trees. We continue with Section 4 where an overview of the proposed platform is depicted, focusing on the integration with legacy Scone Engines and scalability possibilities of our approach. A description of the core architecture and the design decisions taken to accelerate the marker-parser computation are given in Section 5. Section 6 contains implementation results, finishing the article with a summary of the benefits and future work in Section 7.

2. Related work

In the first half of this section, a selection of works concerning ad-hoc implementations of complex, recursive data structures in hardware is presented. We focus specially on those proposals that make it use of reconfigurable hardware as the target platform since it is the driving technology in our work as well.

The first three works deal with graph representation in hardware, and the implementation of the corresponding algorithms. Since, in many application domains, the problem is represented using complex networks of vertices and edges, a considerable number of researches have been attracted to this field of interest.

For example, [2] elaborates a framework for large graph manipulation in hardware. Graph data, which cannot be partitioned and locally processed, is stored lineally in off-chip memories. The processing units (called Graph Processing Elements) access the shared memory architecture through a low latency crossbar. The architecture is accompanied by a design flow that enables the automatic extraction of the processing elements and customization of the memory access infrastructure.

Ahmed et al. propose in [3] a static approach for a hardware implementation of a graph where vertices are gates and the edges are the wires that connect the gates. The graph is first represented by an adjacency matrix and later on mapped to the logic network. Several implementations of core algorithms such as graph reachability and shortest path computation are also presented in this work. Despite of the great efficiency achieved, the main drawback of this proposal is the inability to grow at run time. Huelsbergen, on the contrary, presents a dynamic approach for graph representation and manipulation in hardware which admits modifications on the topology, since vertices and edges might be inserted or deleted [4].

Chandra and Sinnen explore in [5] the implementation of a priority queue module to boost the performance of the Prim's algorithm, a common graph operation for computing the minimum spanning tree. What actually differentiates this work from others is the abstraction provided to software developers to easily integrate the proposed solution by means of a Java to hardware interface. Thus, the usage of the embedded hardware priority queue is transparent to the eyes of the programmer.

The articles just analysed above perform a general approach to the problem, so their results can be reused in more specific domains. However, most of the compiled works in this section focus more on the optimization of the proposal for the target application domain rather than in generalization.

Besides graph implementation and its manipulation through hardware accelerators, decision trees are the other main topic tackled by the majority of works in the state of the art. It is worth noticing that decision trees are broadly used as a way of supporting a variety of tasks such as classification, decision taking processes, optimization, image processing and many others. However, in these works usually very little information is provided to the reader about the implementation details of such data structures. As it has been mentioned before, the problem of mapping decision trees and their algorithms to a hardware circuit is hidden behind the use case.

In [6], Decision Tree classification for data mining support is improved my means of a hardware computational kernel that speeds up the most demanded operations of the algorithms. Single oblique or nonlinear decision trees are the subject of a more generic approach developed by Struharik in [6]. Both, [5,6] use FPGA based reconfigurable computing for their designs. However, the ability to support flexible and scalable designs, even at run time, that is inherent to these devices is not exploited in these proposals. More recently, Saqib et al. [7] published a pipelined architecture for each of the engine processes that handle the data to be classified in parallel. In this case, the scalability of the architecture depends on the capability of the high speed communication channel in charge of delivering data at a rate equal to the throughput of the processing nodes.

Decision Trees are also present in the core of *Random Forest* classifiers or *K*-means clustering. In [8] a systolic architecture to speed up the training stage of a Random Forest classifier, using a very similar approach to the one carried by Nayaranan et al. in [5], is presented. In this research line, Van Essen et al. [9] optimize the generation of the set of decision trees so the utilization of memory resources is the best possible. This is one of the main limiting factors in FPGA or GP-GPUs implementations. Van Essen makes a comparison of the performance achieved by the FPGA against GP-GPU and OpenMP for multi-core architecture. FPGA implementation wins but needing more hardware resource due to the limited amount of on-board memory.

Concerning *K*-means clustering, tree structures are used in order to reduce the computation time. This strategy is present in [10–12]. In [10], the authors focus on the filtering algorithm where an FPGA performs the kd-tree traversal. A centroid visiting module is depicted in [11] where a hierarchical memory architecture maintains the binary tree structures. FPGAs are used in these works since *K*-means algorithms exhibit little control and high degree of parallelism.

Regarding concrete applications of hardware boosted decision tree classifiers, [13,14] combine silicon and highly optimized VLSI circuits for gas identification. These works do not make it use of FPGAs since the resulting circuit must be embodied in a small sensor, and size and power consumption are the main driving factors. The architecture of the solution presented in [13] is optimized for

power consumption by means of the elimination of costly computations in the decision process. The work in [14] describes a parameterizable, fault tolerant hardware implementation of trees classifiers based on a custom VLSI chip and a CPLD chip for the same application. Another interesting application where decision tree classifiers are involved is body part recognition in the Microsoft Kinect vision pipeline. Oberg et al. explore in [15] a hardware implementation of the Forest Fire pixel classification algorithm using a FPGA. Finally, it is worth mentioning the work of Jiang and Prasanna [16] where a multi-field packet classifier is implemented in a FPGA device. The inference system uses a rule-based approach which relies itself on the use of a decision tree algorithm. Several optimizations are proposed so the amount of memory needed is kept in reasonable levels to be suitable for reconfigurable chips.

From this point, a review of previous proposals on tailored hardware architectures and custom machines for Artificial Intelligence (AI) reasoning procedures is presented. Lately, the activity on this subject has clearly declined since Moore's law makes it available computer grids and supercomputers with hundreds and thousands of cheap and powerful processors, and almost an unlimited amount of memory. Therefore, the implementation of AI algorithms over large semantic databases mostly rely on this kind of highly distributed platforms. The bulk of the optimization work in order to speed up execution times stems in efficient data structure implementations and effective load balancing and synchronization mechanism. TROJAN [17] is a representative example of this kind of systems.

Fahlman was the first to propose a custom hardware architecture for his Scone Knowledge-Based system. The NETL [18] machine is rather a concept than a real computation system. NETL is a network of hardware components that represents the concepts and relations of the semantic tree. Each component has an internal memory to store marker values and the propagation logic attending to the defined marker-passing algorithms. This configuration suffers from scalability issues, mainly due to the fact that every processing element in the data path is associated to a single element of the semantic tree.

With the evolution of technology, some massively parallel architectures with specialized processing units appeared. However, communication networks in these approaches were still fixed or not scalable. The Connection Machine [19] is a SIMD computer with thousands of microprocessors arranged in a regular and multidimensional hypercube. Together with the microarchitecture, there is a software layer that includes an instruction set for semantic network processing. The Semantic Network Array Processor [20] is devoted to natural language processing. SNAP adds the capability to modify the connections between processing units at run time. This feature adds some degree of flexibility compared to other proposals. SNAP can be also used for semantic network processing since it actually implements an extended model of the marker-passing strategy. The Parallel Network Wave Machine (PNWM) [21] combines data and control distribution (sending the instructions to be executed to each of the different nodes) to implement a decentralized architecture for high performance semantic network processing. A language called *Wave* has been developed to ease the programming of this data flow computer.

Little work has devised the use of FPGA and reconfigurable devices for AI reasoning techniques. Only GraphStep [22] proposal by deLomirier et al. prospects the development of specialized processing engines in Xilinx Virtex-2 FPGAs for ConceptNet spreading activation mechanism. A Network-on-a-Chip connects all the graph-processing engines together. GraphStep uses regular BRAMs to store the graph data and is able to perform one operation per node and cycle thanks to the pipelining structure of the processing engines.

To finish, it is worth mentioning the COGENT [23] project and the IXM2 [24] parallel associative processor. Both works are an example, in their respective application domains, of novel hardware facilities that help to reduce execution times. For example, the COGnitive Engine Technologies processor is a recirculating parallel computer model devoted to facilitate efficient processing of large graphs. A processing node or CAGE is composed by a COGENT processor and a local memory where graph information is stored. A centralized control is in charge of distributing data, command and collects results from the individual CAGEs. IMX2 makes it use of associative memories for fast indexing of graph data as we do in our approach. But IXM2 is not able to handle data set hierarchies for massive marker activation.

As a conclusion, we may say that there are many contributions that try to offer a benefit to Artificial Intelligence systems by means of the use of hardware accelerators or custom computation systems. So far, the goal is to improve response times through local optimizations on the reasoning algorithms rather than offer a comprehensive solution to the problem. The majority of the proposals scale little or nothing and lack of flexibility and run-time adaptability. The work presented in this article takes advantage from the use of reconfigurable logic as a viable alternative to implement a system-level approach that overcomes the limitations found on other related works. Our solution provides a custom hardware platform together with services and methods for smart data distribution and decentralized control.

3. Semantic trees and implicit knowledge extraction

The basic form of a semantic tree (also called semantic network) is a graph in which each node represents a concept and the vertices depict the relations between concepts. This abstraction is widely used in knowledge representation because of its simplicity and the opportunities it brings into light for automatic manipulation in knowledge-oriented systems.

Knowledge in semantic trees adopts a hierarchical shape with the more general concepts in higher levels and the more precise ones (even individuals) at the leaves. In this sense, a semantic tree basically allows knowledge categorization. Fig. 1 illustrates an example of a semantic tree.

The real potential of knowledge-based systems is in their capability to extract what is called *implicit knowledge* from a semantic network. For example, it can be easily inferred that "if Dumbo *is-a* elephant and *color-of* elephants is grey, then Dumbo must be grey!". Recalling Fig. 1, it can be noticed that there is nothing like Dumbo being grey. However the implicit knowledge found in categorization give us such information.

This inference process, which is natural to humans, is frequently implemented in knowledge-based systems as the intersection of sets of concepts in the semantic tree. Each concept set is obtained through computation of closures of various transitive relations. To the question, "what color Dumbo is?" the process would follow these steps:

1. First, all the concepts related to Dumbo upwards (through the *is-a* relation) need to be selected. All the visited nodes (Elephant, Mammal, Animal and Object) are tagged with a marker. Let us say "■" marker (see Fig. 1).
2. Second, for each concept in the (■) set computed above, select all the nodes (upwards) related to them through a *color-of* relation and tag them with the "●" marker. In our examples, just the "Grey" concept will be marked.
3. Third, all the colors in the semantic tree have to be selected, downwards from the color concept. Red and Grey nodes are tagged with the "♦" mark.

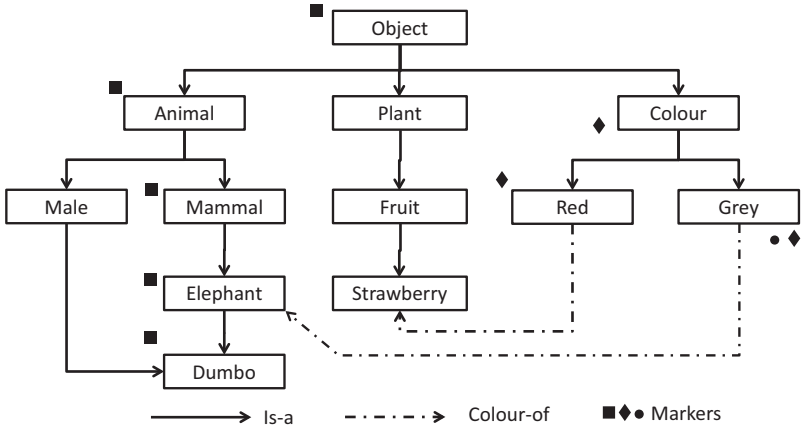

Fig. 1. Example of a semantic tree and result of marker propagation to determine the color of *Dumbo*.

4. Finally, the answer to the initial question would be the intersection of the sets computed in steps two and three. That is, the only node with tags "♦" and "●": "Grey".

3.1. Marker-passing algorithms

The strategy depicted above is called *marker-passer* because the implemented mechanism to build sets of concepts. Each abstraction of the semantic tree (i.e. nodes and links) is endowed with a bit mask dedicated to propagate information during the deduction process. Such propagation process is based on the transitive propriety of the relations between nodes.

Thus, the extraction of the implicit knowledge hidden behind the declarative knowledge, which represents the semantic tree, is based on three operations: *upscan*, *downscan* and *select*. Algorithm 1 shows an adaptation of the original algorithm for the *upscan* operation. The goal is to provide the reader with the understanding about the complexity and general procedure of these primitives.

Algorithm 1. Upscan operation pseudocode.

```
Upscanning(nodeN,markM,relationR)
1: Nodes2Visit = EmptySet
2: Insert(Nodes2Visit,nodeN)
3: Do
4:    ActualNode = First(Node2Visit)
5:    If isNotMarked(ActualNode,relationR) Then
6:       Mark(ActualNode,markM)
7:       Parents = GetParentsOf(ActualNode,relationR)
8:       Node2Visit = Union(Nodes2Visit,ParentsOf)
9:    End If
10: While Nodes2Visit! = EmptySet
```

Given a starting node, the semantic tree is traversed towards upper levels, visiting the current node parents. This process is repeated until all nodes from the *nodes to visit* set have been visited. The condition in line 5 prevents the algorithm from propagating the marker through nodes that have already been visited. The pseudocode for the *downscan* operation is almost identical to the one shown in Algorithm 1 but the function in line 7 is replaced with *GetChildsOf*. Such function, given a node *N* and a relation *R*, computes the set of nodes which are children of *N* for that relationship *R*. The role of the upscan and downscan operations is mainly the construction of the node set used in subsequent analysis.

Selection and *intersection* of node sets can be implemented almost straightforwardly by only analyzing the values of the bit mask for a certain node. For example, to assert the membership of a node *N* to a set *S*, it is only necessary to check whether the corresponding bit is activated in the node mask. Set intersections are computed just by calculating the logic *and* operation of the bit masks related to the sets under consideration.

3.2. Scone Knowledge-Based system

Scone [1] is an open-source system that efficiently manages the task of extracting implicit knowledge from a knowledge base. Developed by Dr. Scott E. Fahlman and his research group at Carnegie Mellon University, Scone implements a marker-passing approach that allows querying the knowledge base about individuals or types that exhibit some set of properties.

Although marker-passing algorithms cannot perform some functions present in other systems such as theorem-prover, their simplicity enables faster execution times. Thus, the Scone system is an interesting candidate to be embedded in handheld devices or real time applications. The time spent exploring the facts of the knowledge-base is constant regardless the size of the data base. Scone's marker-passing algorithms are extremely optimized providing high performance results. These algorithms were originally designed for the NETL machine introduced in Section 2 which implies a massively parallel computer to get the best figures.

Among the features that differentiate Scone from other systems is the ability to easily add new knowledge to a system (several manual and automatic means have been developed), exception handling or modeling of multiple contexts. The context mechanism models alternative paths in the reasoning process that depends on variables such as time, place. Hypothetical, counter-factual states or figurative language are some examples.

4. Scone reasoning FPGA platform

In this work, key concepts of the Scone Knowledge-Based system have been analyzed from the hardware perspective and implemented as an FPGA-based platform for fast reasoning. The goal is to provide a scalable and optimal solution based on the use of reconfigurable hardware and a smart data representation and distribution. The resulting platform exploits coarse-grain parallelism on the reasoning process, enabling concurrent operations on several regions of the semantic networks.

The proposed platform is composed by one or more *RHCs* (Reasoning Hardware Core) that are the hardware cores in charge of accelerating marker-passing functions. They have a specialized datapath and control logic to this endeavor. Since we are using dynamic reconfigurable devices for the prototypes, the number of

RHCs may vary depending on the punctual needs of the applications.

Each RHC has a local memory where a subset of the semantic tree is stored. This is the current data set the RHC is working on. Usually, the size of the semantic network surpasses the capacity of these local memories which, then, act as a cache of a part of the whole knowledge base. The second memory level is played by a DDR2 memory which also holds the bitstream for later incarnations of new RHC components.

A MicroBlaze processor runs a small control software routine which is in charge of the synchronization of the operations within the System-on-Chip (SoC) and the communications with a PC. There is no other operating system running in the platform but a little software layer (Xilkernel) implementing basic services such as task scheduling, thread support and communication with input/output peripherals (i.e., Ethernet interface).

The PC runs a client software that forwards Scone requests to the SoC and also is able to load the semantic tree information in a remote way. The MicroBlaze control software receives UDP messages and interprets the commands embedded in them. From the PC Scone Engine user point of view, there is no change in the way he has to interact with Scone. Scone Engine's java libraries which implement the search and inference functionality have been extended to support communication with the FPGA platform.

Fig. 2 represents all the above mentioned elements integrated in the Scone Reasoning FPGA Platform. A XUPV5 prototyping board (based on a Virtex5 LX110T chip) has been used to implement the configuration depicted in Fig. 2.

4.1. Scalability and flexibility

In Section 2, flexibility and scalability were identified as a usual weak point in most of the proposals. In this work, both properties are assured because of the use of dynamic reconfigurable logic and a distributed marker propagation mechanism.

Dynamic reconfiguration technology enables on-demand growth of the platform capabilities. Whenever it is necessary, new instances of the RHC can be instantiated at run-time. Thus, the number of processing cores can be adapted depending on the size of the semantic tree region that it is going to be traversed simultaneously. Also, the size of the semantic tree depends on the application under execution. Then, our platform can be sized accordingly to the real needs, avoiding unnecessarily wasting power consumption due to unused RHC cores.

Concerning the optimized marker propagation mechanism, a distributed and decentralized method has been developed. Since each RHC holds a part of the semantic tree, the goal is to start the execution of the reasoning algorithm as soon as possible. In order to reduce the overall latency of the *upscan* and *downscan* operations, the semantic tree data must be partitioned and deployed among several RHCs.

4.1.1. Dynamic reconfiguration management

Partial reconfiguration is a feature present in some of the FPGA devices in the market. This characteristic allows run-time variations of the hardware architecture by re-programming the configuration memory of the reconfigurable chip. Therefore, an adaptation of a system, according to the current needs, is performed adjusting the resources to the actual demand.

The work proposed here resorts to the infrastructure for dynamic reconfiguration management for Xilinx platforms developed by Dondo et al. [26] in order to make the Scone hardware platform an entity that scales well through the addition of new processing cores when it is required.

Following, a brief description of the process and supporting architecture devised in [26] is provided to the reader. The goal is to offer a global solution to the main challenges that make it possible to take full advantage of partial and dynamic reconfiguration. The main working lines are: hardware support to reduce the reconfiguration latency, workflow and tools based on a high level model to ease the development of dynamic reconfigurable systems.

The *Reconfiguration Manager* (*ReM* from now on) is a hardware core, coupled to the MicroBlaze processor through a Fast Serial Link interface. The *ReM* is connected directly to the ICAP which is the Xilinx core in charge of managing the access to the FPGA configuration memory. Therefore, the communication bandwidth with the ICAP is increased, reducing the time needed to reconfigure a specific area. As it can be seen in Fig. 3, the ReM core is also directly connected to the DDR memory which stores the bitstreams through a *Native Port Interface*; no software intermediaries nor MicroBlaze action are required in the FPGA programming process.

The *RController* keeps a register of the available cores that can be instantiated in a free reconfigurable area and where the programming bitstreams are located in the DDR memory, among other information concerning persistency and management of the core status, which is out of the scope of this work. When the RController receives a command to load or to evict an instance of a component, it orchestrates all the actions required through the *Factory*. The Factory is in charge of moving data from/to the DDR memory and configuration memory, playing the role of a specialized DMA controller. Operations on the DDR can be overlapped with operations on the ICAP, which leads to a maximum bandwidth of 180 MB/s.

The interfacing with the MicroBlaze processor is handled through messages received using the *Fast Serial Link*. The *MicroBlaze Interface* decodes such messages generated by the embedded software when a petition from the PC running the Scone Engine arrives.

The whole process, thus, implies the following steps: (1) The PC sends an Ethernet message which is received by the MicroBlaze embedded software; (2) the Ethernet message is decoded and a petition to the Reconfiguration Manager is made through the FSL interface; (3) the FSL message is received and interpreted by the ReM module which triggers the operation (RController); and, finally, (4) the necessary memory transactions are performed between the DDR and the ICAP through the Factory component. These steps are taken just once to deploy the desired configuration of ScopeIP components before any reasoning algorithm starts. So, the time invested in this configuration stage is negligible in comparison with the operation time of the configuration once it has been set up. Moreover, the bitstreams for the RHC modules are pre-loaded in DDR memory and the reconfiguration time for one module is approximately 12.45 ms.

As to the layout of the reconfigurable areas, we have followed a conservative approach since there is only one type of component to be instantiated. The reconfigurable region is divided into areas of fixed size, shape and placement up to a total number of four in this prototype due to the resource constrains imposed by the chosen platform. The interface with the static region is fixed and it is composed by the signals that connect the core to the PLB (*Processor Local Bus*) bus.

4.1.2. Smart data distribution

Replication has no value on its own without a mechanism that distributes the data among the processing cores in order to take a benefit from the parallel architecture. The FPGA Scone platform makes it available several RHC modules that potentially can process parts of the semantic tree concurrently. Therefore, a smart data distribution strategy must be envisioned.

Semantic tree partitioning can be performed by branches (*horizontal partitioning*) or levels (depth or *vertical partitioning*). Fig. 4 shows how these criteria can be applied to split the semantic

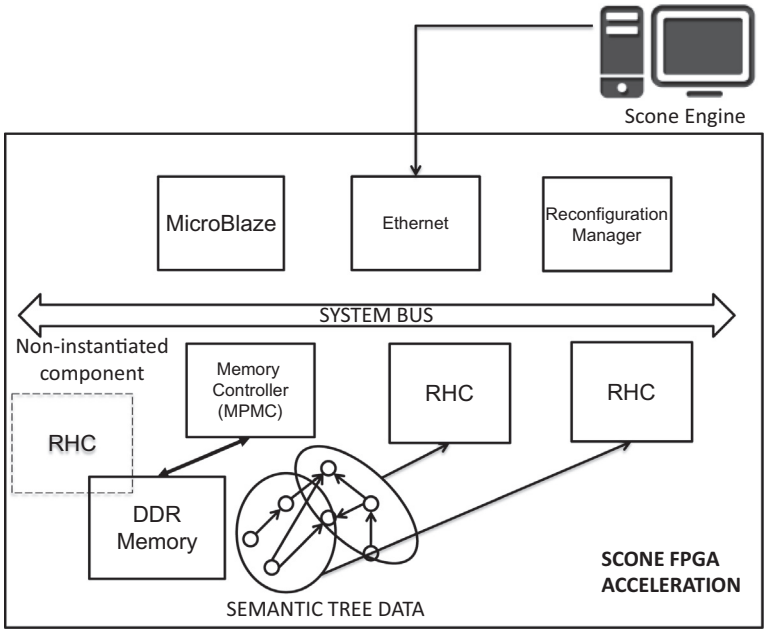

Fig. 2. Architecture of the Scone FPGA reasoning platform.

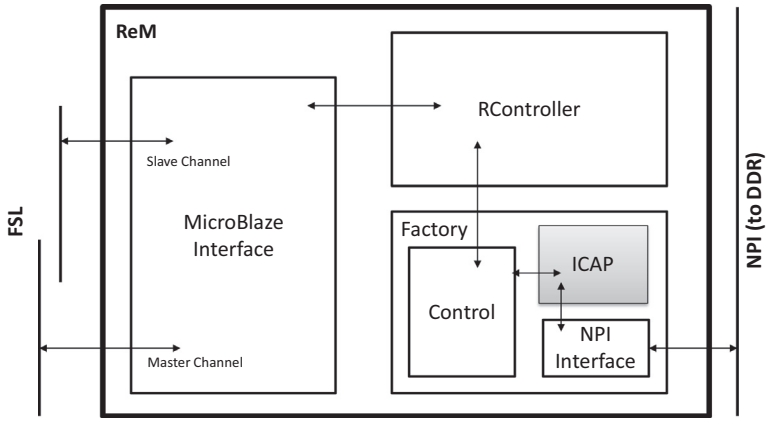

Fig. 3. Main components for the Reconfiguration Manager co-processor.

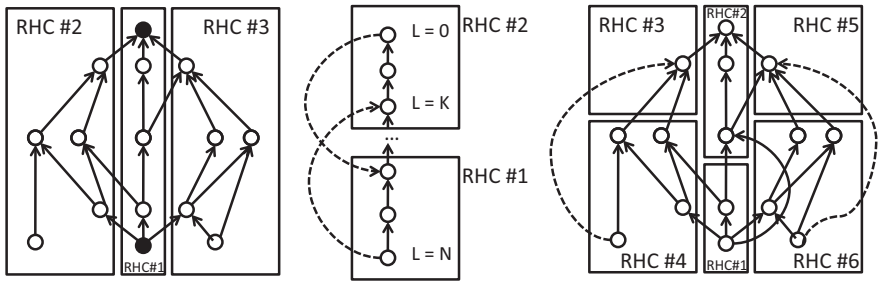

Fig. 4. Semantic tree distribution alternatives: branch (left); vertical with forwards links (center); and a combination of both (right).

tree in *N* clusters that, in turn, can be assigned to *N* different RHCs. The time necessary to complete the marking-passing algorithms on a cluster mainly depends on the total number of links and nodes that must be visited. Thus, reducing the size of the clusters to process is an objective. However, semantic tree partitioning must be performed thoughtfully to favor parallel propagation of the markers.

The configuration depicted on the left side of Fig. 4. Is an example of horizontal partitioning for early marker propagation. Any upscan/downscan operation starting at one of the black nodes of

the semantic tree (RHC #1) initiates two parallel executions of the upscan/downscan on RHC #2 and RHC #3. Activation takes place whenever the procedure reaches what is called an *edge link*, that is, a link that references an entity residing in an external (different) RHC.

Vertical partitioning (center configuration in Fig. 4) reduces the depth of the cluster, and therefore its size, with the consequent aforementioned benefits. In this case, it is necessary to add extra knowledge to the semantic network in the form of *forward links*. A forward link (represented with dashed lines in Fig. 4) is an edge link that captures the transitive property of a given relation. Their function is to trigger a soon start of marker propagation on another RHC.

Last scenario (right configuration in Fig. 4 combines the advantages of both techniques that lead to constant processing times regardless the size of the semantic tree. It is worth mentioning that a firewall mechanism has been also implemented to prevent duplicate activations due to overlapping node visitation. The RHC control logic is able to detect such situation, stopping the propagation of any marker that has been set/clear before.

As mentioned before, each semantic tree partition is assigned to a different RHC module. Actual activation implies a bus transaction where the master is the RHC core source of the edge/forward link and the slave is the destination. This means an extra cost in terms of communications due to arbitration and transmission cycles. But this overhead is significantly reduced because our platform is based on the principles, tools and infrastructure provided by the OOCE SoC middleware [27]. OOCE allows hardware to hardware transactions by means of special bus wrappers (which are part of the RHC architecture, see Fig. 5) which means no software involved in the control and synchronization process. The whole activation process can be performed in only 3 bus cycles without bus congestion. A separate study should be carried out in order to determine the relation of the number of activation operations over the number of internal RHC computations that might lead to bus saturation levels with an impact on the algorithm performance. The results of this study could help to improve and drive the partition and data distribution procedure.

5. RHC architecture

This section focuses on the explanation of the micro-architecture for the hardware core designed to accelerate the three basic operations of the marker-passing algorithm; the *Reasoning Hardware Core* (RHC).

Fig. 5 shows a high level block diagram of the main elements conforming the RHC's architecture. Three main subsystems can be identified, which are briefly introduced underneath:

- *System integration logic.* It is the bus wrapper in charge of isolating the core logic of the kernel of the RHC from the implementation details of the physical protocol. This wrapper does not only interpret bus transactions and fires the specific requests to the *Central Control*, but it is also responsible for loading and updating the content of the different RHC[1] memories.
- *Computation of the node set and links to visit.* This subsystem includes the *CAM Control Logic*, the *Semantic Tree CAM* and the logic intended to filter and compute the actual node set to visit in the next iteration of the upscan or downscan operation.
- *Marker propagation.* It is a pipelined datapath that, given a node set and links to visit, it updates the marker bits and gets the identifiers of the nodes to visit in the next iteration of the upscan and downscan operations. For a select operation, it iterates through the marker data stored in internal memories and retrieves those node identifiers whose mask bits have been activated.

5.1. Representation of the semantic tree: the ChildOf and ParentOf functions

This subsection is devoted to provide an insight of the implementation of the *ChildOf* and *ParentOf* functions referenced in the upscan and downscan algorithms (recall Algorithm 1).

The semantic tree network of nodes and links is represented in a tabular format which is more suitable for hardware processing. Each semantic tree node has a unique value identifying it. Relationships between nodes are translated into table entries, qualified by the relationship type and the direction of such relation (i.e. A-node or B-node using Scone terminology). Fig. 5 graphically represents the method [25] that computes the parents or children of a given node, as explained below.

In the RHC, the *Semantic Tree CAM* (STC) stores the flat representation as described above. STC is implemented using a Ternary CAM which allows the specification of *don't care bits* in the search patterns. Therefore, to retrieve all the relations in which a node *N* *is-father* (play the role of B-node), it is only necessary to place *do not care bits* in the A-node field and *N* in the B-node field of the search pattern (and, of course, the right code for the relation we are looking for). The reader can guess that a similar approach works for the case of an *is-child* primitive (swapping the search pattern fields). In summary, and following the example depicted in Fig. 6 asking for the parents of *Node8* in the *Is-as* hierarchy will result on a match operation with "*Is-a|Node8|XXX*" as the input pattern mask. Querying for the children of *Node2* of a *Color* needs the following pattern "*Colour|XXX|Node2*".

Configuring the STC output not to be encoded, the match output of the CAM is a bitmask with one bit representing a memory position. If the bit *B* is set to 1, it means that entry *B* satisfies the primary condition (is-father or is-child). Hence, the match register is used to index a Block Ram (BRAM) that contains a copy of the STC content plus the marker bits.

5.2. CAM Control

The CAM Control logic is in charge of generating STC search patterns and node sets to be visited in the next iteration of the current operation. The former operation is carried out by the technique described before whereas the latter needs to be explained in more detail.

As it has been previously mentioned, STC output is used to index a BRAM containing the marker data. The basic approach to solve this problem in hardware (a shift register and a counter implementing an address generator) would be non-optimal in time. Thus, an improved indexation mechanism is developed based on: (1) a reduction of the range of addresses to be generated (see Section 5.3) and; (2) a reduction of the number of iterations needed to complete the algorithm.

To accomplish (2), the CAM Control uses a network of registers and logic grouping several STC outputs. This strategy reduces the number of iterations and, therefore, the number of times the BRAM needs to be indexed. Also, the number of actual accesses to BRAM per iteration will be higher, rising the productivity of the indexation process.

Each iteration starts reading the Nodes2Visit FIFO until it is empty. This results on various consecutive queries to the STC in order to determine the child or parent nodes to visit in the next iteration, depending on the command under execution. The outputs of consecutive queries to the STC are 'ored' in a

[1] For the sake of simplicity, the relationships and main connections between elements have been omitted from the diagram supporting this feature.

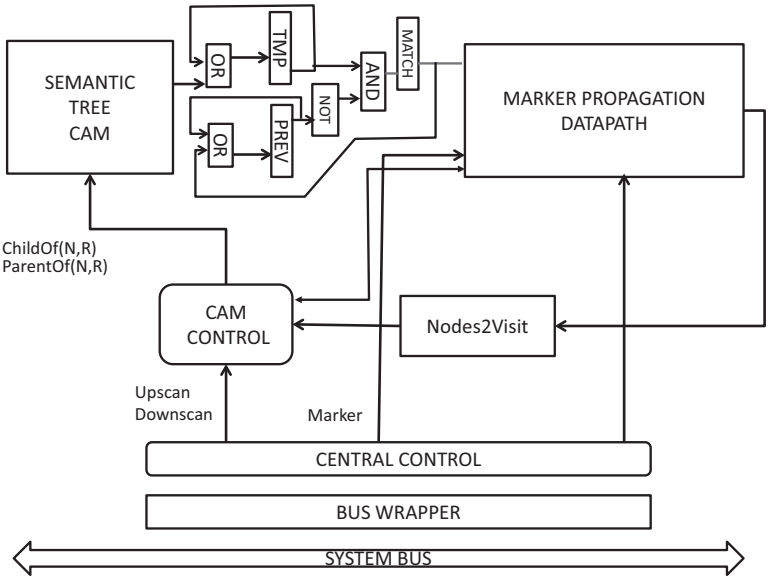

Fig. 5. Reasoning Hardware Core internal block diagram.

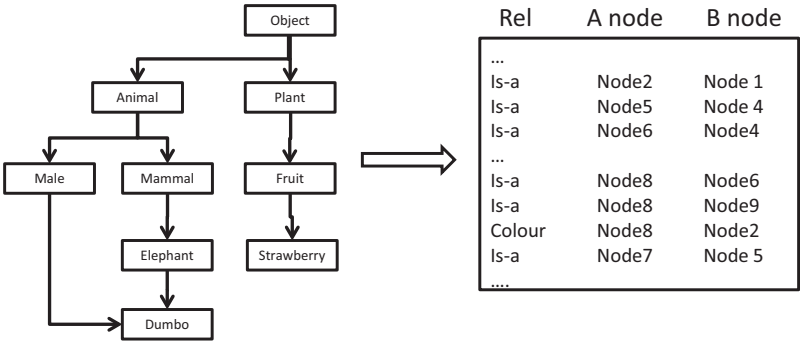

Fig. 6. Mapping a semantic tree to a flat memory structure.

temporary register, implementing the "Union" function (line 8 of Algorithm 1).

In addition, to prevent repeated visits to a same node in different iterations, the history of previous accesses to BRAM is kept in the *match prev register*. The current computed set of BRAM positions to read is filtered using this information when a batch of STC accesses is completed. The following expressions summarize such optimization filters where *i* stands for the value of the current iteration of the execution of a reasoning operation.

$$match_i = OR(STC\_output_j) \text{ for each } j \text{ in } Nodes2Visit$$
$$match_{i+1} = NOT(match\_history_i) \text{ AND } match_i$$
$$match\_history_{i+1} = match\_history_i \text{ OR } match_i$$

The CAM Control does not take part in the execution of the *select* operation.

5.3. Marker propagation datapath

The marker propagation stage updates the marker bits for all the memory entries computed by the CAM Control logic. It is also responsible for inserting into the *Nodes2Visit* FIFO the identifiers of the child or parent nodes (depending on the operation under execution). The CAM Control retrieves from the Nodes2Visit FIFO the information needed to generate the batch of STC queries for the current iteration.

The input parameters for marker propagation are: the marker mask (bits to activate), the command under execution (upscan, downscan or select), the match register (set of addresses to access) and the historical log of previous accesses (previous match register). Fig. 7 shows the diagram of the segmented datapath developed for the marker propagation process. Three stages can be identified (from left to right): *indexation* or memory access generator, *propagation* and *marker updating*. In the picture, the second and third stages somehow meld since they share the same dual port BRAM memory; propagation to read and marker updating to write.

Before starting the marker propagation, it is necessary to load the match and previous match values into the first BRAM memory. This BRAM's content is used in the indexation or memory access generator stage. The Index BRAM (IBRAM) geometry will have a depth equal to the width of the match register (one bit per memory address) and a width of two bits. When a read access is performed on the IBRAM, the two bit word contains one bit from the match input and one bit from the previous match input (i.e. `ibram[j] = match[j] & prev[j]`). Thus, when the IBRAM is written, the bits of both input registers are interleaved. In Fig. 6 the reader can notice that there is an asymmetry in the data port width for read and write operations. This feature is supported by some memory technologies as the one used in the implementation of the prototype. In this case, a wider data port for write operations allows completing the initialization of the IBRAM in fewer cycles.

Once the IBRAM has been initialized, the normal operation of the datapath can proceed. The indexation mechanism uses two address registers (the *Front Index* and the *Rear Index*) to access the IBRAM. The RIndex is initialized to start from the highest

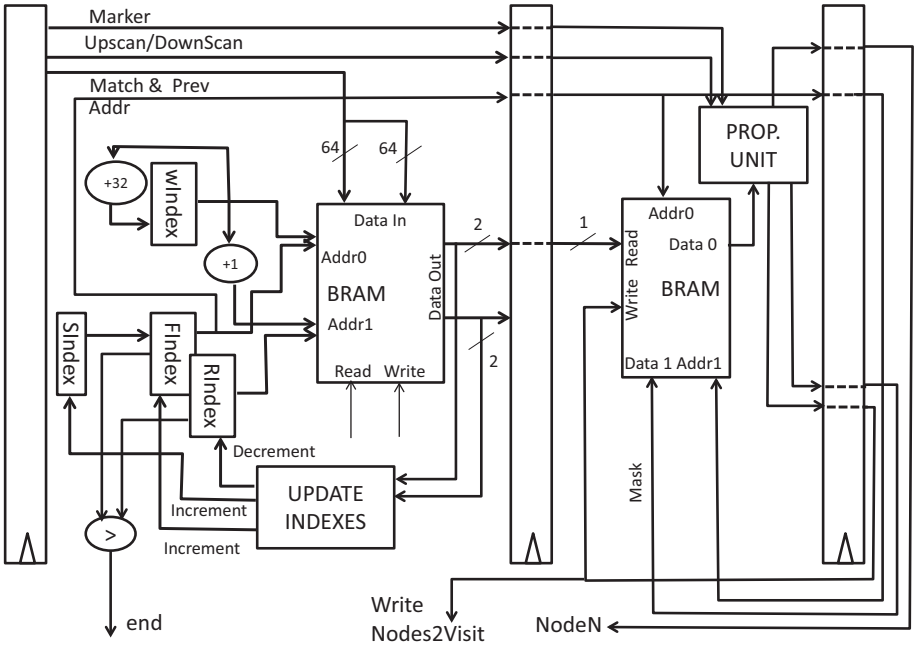

Fig. 7. Marker propagation pipeline.

memory address and every cycle it is decremented until the first 1 is found for the match bit (`ibram[RIndex][0] == 1`). Therefore, the RIndex value indicates the last address to access accordingly to the match register value for this iteration (stop condition). This optimization avoids unnecessary cycles to check match bits for more memory accesses.

Another optimization implemented in the proposed indexation mechanism uses the *Start Index* register which tracks the position of the first bit with a value of 0 (that is, it has not been visited before) in the history of match values. For each iteration, FIndex is loaded with the value of Start Index, avoiding unnecessary memory checks. Fig. 8 shows how Start and RIndex are used to reduce the total accesses to the memory since the valid range of the memory map shrinks. This technique is valid since the flattening (Section 5.1) of the semantic tree is performed using a depth first traversal of the structure and, hence, placing the nodes related within a hierarchy on consecutive memory addresses.

When `ibram[FIndex][0] == 1`, an access to the Sematic Tree BRAM (STBRAM) takes place at the next execution cycle. The *Propagation Unit* analyzes the semantic tree entry retrieved from memory and determines, based on the current operation and the marker mask, whether the memory entry needs to be updated (if the marker bit has not been activated yet – line 5 of Algorithm 1) or not. The Propagation Unit generates the write commands to the Nodes2Visit FIFO and the STBRAM that will take place in the next execution cycle.

Finally, it is worth sketching out how the *select* operation works. In this case, there is no need of IBRAM initialization since all the memory map needs to be explored. Thus, the indexing stage generates all the possible address values to read the STBRAM.

The Propagation Unit only generates write commands for the Nodes2Visit FIFO when the following condition evaluates to true `marker == (marker && stbram[i].bitmask)`.

6. Implementation results

The XUPV5 board from Xilinx has been the device chosen for the implementation of a prototype of the Reasoning Hardware Core presented here. It is based on a Virtex5 LX110T chip, equivalent to four million logic gates with dynamic partial reconfiguration capability.

The RHC core is connected to the Processor Local Bus (PLB) and a MicroBlaze soft processor which runs a small program which communicates with the RHC peripheral to launch commands, load semantic tree data, read results, etc.

In this prototype, the RHC is able to store a semantic tree with a maximum of 1 K relations and 512 concepts. Four different types of relations can be defined between the nodes and the reasoning process can support eight different marks or sets to be combined. These features delimit the number of bits necessary to represent and store such information in memory. In this version of the platform, we have been conservative in order to maintain in reasonable levels the memory requirements for our tests.

As a result, all the memories in the design have a depth of 1 K words and 20 bits width (2 bits to codify the relation code and 9 bits $\times$ 2 to codify the node identifier) for the STCAM. The IBRAM is 2 bit width as mentioned in the previous section and the STBRAM is 30 bit width (20 bits + 8 bits for the marker mask + 2 control bits). All the RAM memories in the design are true dual-port memories and configured in READ FIRST operating mode.

The STCAM memory is the most demanding component from the resources point of view and conditions the maximum clock frequency the RHC is able to achieve. The STCAM is a SRL16E-based CAM with a 16 clock cycle write latency and one cycle latency for search operations. STCAM implements a basic ternary mode for search operations and the match address output is multi-match unencoded (many-hot).

The whole design has been synthetized using the 12.1 ISE Design Suit Embedded Edition with the following results (Table 1):

The evaluation of the FPGA based Scone reasoning platform presented here needs to be compared to the software implementation of the Scone system. The set of tests has been design with the following strategy in mind: reproducing similar scenarios than the ones undertaken in [1]:

- *Test #1.* A semantic network of 1024 elements is traversed and marked from top to down. Each link must be visited at least once, more if a node has two or more parents. This operation is called *"Downscan thing"* since each concept in the semantic network *is-a* thing.

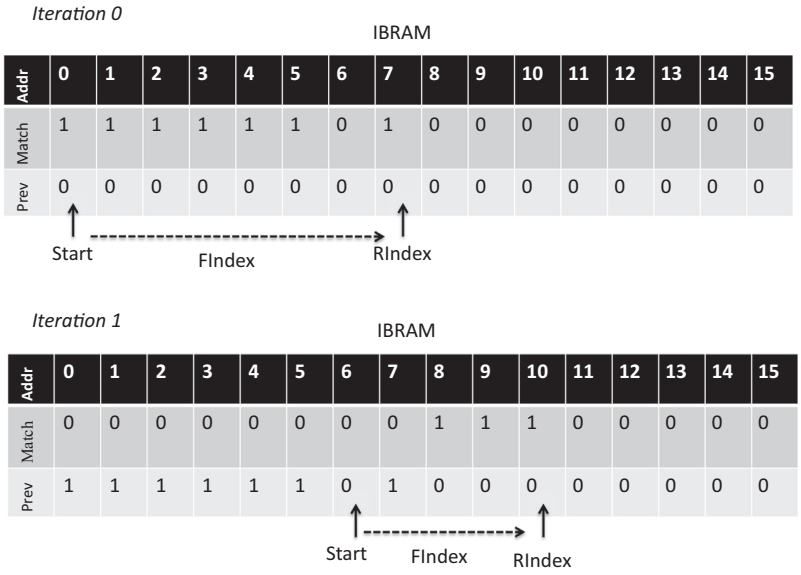

Fig. 8. Indexation registers.

Table 1
RHC synthesis results.

Freq	SRL16	Flip flops	LUTs	Slices	BRAM
107 MHz	10,240 (57%)	1259 (2%)	14,935 (22%)	4049 (23%)	4 (36 kbit)

Table 2
Execution time performance.

Test	SW time (ms)*	HW time (μs)*	Speedup
#1 Downscan the semantic network tree (1 K elements)	43	92	467×
#2 Mark & intersect 2 sets with 10 members, one winner	2	423	4.7×
#3 Mark & intersect 3 sets with 10 members, one winner	3.6	476	7.6×

* Mean execution time of relevant Scone reasoning operations in our prototype and Dell workstation. 500 executions of each test were carried out.

- *Test #2.* In this test, the semantic network has only 64 elements. Two downscan operations are performed first in order to mark each of the ten elements belonging to a group. Then, a query to the system is made in order to retrieve the only element which has both marks active (the winner).
- *Test #3.* The same procedure as test 2 but considering three groups and a semantic network size of 128 elements.

The hardware platform under test implements a single RHC and no reconfiguration was needed. Also, it is assumed that the semantic tree data has already been loaded into the RHC memories. Therefore, the HW execution times shown in Table 2 are not affected by communication or topology changes during run-time. The goal was to evaluate the performance of the RHC architecture in an optimal configuration. Moreover, the reader must notice that reconfiguration takes place before (and just once) the system is able to accept commands, so the influence of this feature does not change the results obtained. Concerning communications overhead, it is highly dependent on the semantic tree data and the partition strategy. Therefore, in this case, it is necessary an in depth study of the algorithms used to divide the semantic tree and insert the forward links.

As to the software side, the computer in which the software runs is a Dell XPS 8300 workstation with 8 GB of DDR3 memory, an Intel i7-2600 processor at 3.4 GHz and a 64-bit GNU/Linux (kernel version 2.6.16) operating system.

The analysis of the results from Table 2 rapidly focuses on the important speedup difference between test #1 and the rest. There are three factors that combine to explain the leap:

- First, there is a penalty for the software execution in test #1 probably derived from the memory management performed by the Scone software engine (cache misses). On the contrary, the hardware execution makes its best since the whole semantic tree is stored in BRAM memories and each access takes a couple of clock cycles.
- Second, for tests #2 and #3 the hardware execution has an extra penalty due to the time to send/receive and process the UDP control messages to/from the necessary to drive two (three in test#3) downscanning and one select operations.
- Third, the size of the data set is much bigger in test #1 than in test #2 and #3 (more than a thousand elements compared to a few tens) so the percentage of computation time spent in the RHC over the rest of operations (mainly communication with the PC and hardware/software interfacing) increases. Therefore, in test #1, there is an extra opportunity to take advantage of hardware acceleration.

These preliminary results indicate that our approach is more suitable for operations over larger data sets, trying to reduce the communication with the Scone Engine running in the PC. In this regard, it would be necessary to build a batch-like system so the set of commands to be executed by the hardware platform is sent just once, avoiding the need of sending several UDP packets via the Ethernet interface.

7. Conclusions

In this paper, the design of a hardware unit to accelerate semantic tree operations for common-sense reasoning applications was presented. The algorithm utilized was the *marker-passing* strategy proposed for the Scone Knowledge-Based system. The

implementation has been performed using a Xilinx ML507 prototyping board, with a Virtex-5 LX110T FPGA (*Field Programmable Gate Array*) chip and compared to an equivalent software implementation.

The optimized micro-architecture together with a smart data mapping strategy allows our solution to achieve a significant reduction in execution times of marker-passing algorithms. On top of this, scalability and flexibility are assured by means of the adoption of reconfigurable logic technology which makes it our proposal suitable for a wide range of needs. Also, the architecture and methods exposed can be generalized so it can be applied to other algorithms that make it use of recursive structures such as general trees or graphs.

It is observed that memory usage is the main limiting factor in the proposed architecture. Since reconfigurable devices have a relatively small amount of in-chip memory blocks, most of the FPGA resources are devoted to the components that have to store the semantic tree data.

Therefore, further research must be performed in order to reduce the memory needs of the studied algorithms or envision new ways to encode the working dataset. In our vision, this is the first step to follow in order to make it feasible the embodiment of such kind of reasoning capabilities in ubiquitous environments. This ultimate goal is motivated by the need of real-time context-aware processing using embedded boards with reconfigurable logic support. Lately, a wide variety of hybrid platforms combining microcontrollers and FPGA chips are available in the market which represents an alternative to other approaches exclusively based on software (which lack of computational power or are power hungry). This works aims to set up the basis for future work so this new technology can be applied in new scenarios and applications away from the classical high performance, mainframe ecosystems.

Acknowledgments

This research was supported by the Spanish Ministry of Science and Innovation under the project DREAMS (TEC2011-28666-C04-03), and by the Spanish Ministry of Industry and the Centre for the Development of Industrial Technology under the project ENERGOS (CEN-20091048).

References

[1] S. Fahlman, Marker-passing inference in the scone knowledge-base system, in: First International Conference on Knowledge Science, Engineering and Management (KSEM'06), Springer-Verlag (Lecture Notes in AI), 2006.
[2] B. Betkaoui, D.B.T. Thomas, W. Luk, N. Przulj, A framework for FPGA acceleration of large graph problems: graphlet counting case study, in: Proceedings of the IEEE International Conference on Field-Programmable Technology (FPT 2011), New Delhi, India, December 12–14, 2011.
[3] I. Ahmed, M.A.U. Rahman, S. Alam, N. Islam, Implementation of graph algorithms in reconfigurable hardware (FPGAs) to speeding up the execution, in: Proceedings of the 4th International Conference on Computer Sciences and Convergence Information Technology (ICCIT'09), November 24–29, 2009.
[4] L. Huelsbergen, A representation for dynamic graphs in reconfigurable hardware and its application to fundamental graph algorithms, in: Proceedings of the 2000 ACM/SIGDA 8th international symposium on field programmable gate arrays (FPGA'00), ACM, New York, NY, USA, 105–115.
[5] R. Chandra, O. Sinnen, Improving application performance with hardware data structures, in: Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on, 19–23 April 2010, pp. 1, 4. doi: 10.1109/IPDPSW.2010.5470740.
[6] R. Narayanan, D. Honbo, G. Memik, A. Choudhary, J. Zambreno, An FPGA implementation of decision tree classification, in: Proceedings of the conference on Design, automation and test in Europe (DATE'07), 2007, pp. 189–194.
[7] R.J.R. Struharik, L.A. Novak, evolving decision Trees in Hardware, J. Circuits Syst. Comp. 18 (6) (2009) 1033–1060.
[8] F. Saqib, A. Dutta, J. Plusquellic, P. Ortiz, M. Pattichis, Pipelined decision tree classification accelerator implementation in FPGA (DT-CAIF), in: Computers, IEEE Transactions on, no. 99, pp. 1,1, 0. doi: 10.1109/TC.2013.204.
[9] C. Cheng, C.-S. Bouganis, Accelerating random forest training process using FPGA, in: Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on September 2013.
[10] B. Van Essen, C. Macaraeg, M. Gokhale, R. Prenger, Accelerating a random forest classifier: multi-core, GP-GPU, or FPGA?, in: Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on pp. 232, 239, May 2012.
[11] F. Winterstein, S. Bayliss, G.A. Constantinides, FPGA-based *K*-means clustering using tree-based data structures, in: Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on, September 2013.
[12] T.-W. Chen, S.-Y. Chien, Flexible hardware architecture of hierarchical *K*-means clustering for large cluster number, in: Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, vol. 19, no. 8, pp. 1336, 1345, August 2011.
[13] Q. Li, A. Bermak, A low-power hardware-friendly binary decision tree classifier for gas identification, J. Low Power Electron. Appl., 1 (2011) 45–58.
[14] A. Bermak, D. Martinez, A reconfigurable hardware implementation of tree classifier based on a custom chip and CPLD for gas sensors applications, Proceedings of the IEEE TENCON, Chiang Mai, Thailand 1 (2004) 32–35.
[15] J. Oberg, K. Eguro, R. Bittner, A. Forin, Random decision tree body part recognition using FPGAs, in: Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on pp. 330, 337, August 2012.
[16] W. Jiang, V.K. Prasanna, Large-scale wire-speed packet classification on FPGAs, in: Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays (FPGA'09), ACM, New York, NY, USA, 219–228, 2009. doi: 10.1145/1508128.1508162.
[17] C.-W. Lee, C.-H. Huang, L.T. Yang, S. Rajasekaran, Distributed path-based inference in semantic networks, J. Supercomput. 29 (2) (2004) 211–227. doi:10.1023/B:SUPE.0000026852.08638.96.
[18] S.E. Fahlman, NETL: A System for Representing and Using Real-World Knowledge, MIT Press, Cambridge, MA, 1979.
[19] S.H. Chung, D.I. Moldovan, Modeling semantic networks on the connection machine, J. Parallel Distributed Comput. 17 (1993) 152–163.
[20] H. Kitano, D. Moldovan, Semantic Network Array Processor as a massively parallel computing platform for high performance and large-scale natural language processing, in: Proceedings of the 14th Conference on Computational Linguistics (COLING'92), Stroudsburg, PA, USA, 1992.
[21] P. Sapaty, I. Kocis, A parallel network wave machine, Proceedings of the Third International Workshop on Parallel Processing by Cellular Automata and Arrays (1987) 267–273.
[22] T.F.J., DeHon, A., GraphStep: a system architecture for sparse-graph algorithms, in: Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2006, pp. 143–151.
[23] J.F. Bogdanowicz, J. Granacki, COGENT – an innovative architecture for cognitive processing, in: Proceedings of the 10th Annual Workshop on High Performance Embedded Computing, Lexington, MA, USA, 19–21 September 2006, pp. 19–21.
[24] T. Higuchi, K. Handa, N. Takahashi, T. Furuya, H. Iida, E. Sumita, O. Oi, H. Kitano, The IXM2 parallel associative processor for AI, Computer 27 (11) (1994) 53, 63. doi: 10.1109/2.330048.
[25] N.H. Minsky, Representation of binary trees on associative memories, in: presented at Inf. Process. Letters, 1973.
[26] J. Dondo, J. Barba, F. Rincón, F. Moya, J.C. López, Dynamic objects: supporting fast and easy run-time reconfiguration in FPGAs, J. Syst. Arch. 59 (1) 2013, ISSN 1383-7621, doi:10.1016/j.sysarc.2012.09.001.
[27] J. Barba, F. Rincón, J. Dondo, F. Moya, J.C. López, A comprehensive integration infrastructure for embedded system design, Microprocessor Microsyst. 36 (5) (2012) 383–392 (Elsevier).

Jesús Barba received the MS and PhD degrees in Computer Engineering Diploma from the University of Castilla-La Mancha (UCLM), Spain, in 2001 and 2008 respectively. He is working as Associate Professor with the Department of Information and Systems Technology (TSI) since 2001, lecturing in the area of Computer Architecture and Networks. He is part of the ARCO research group, located at the School of Computer Science (UCLM, Spain), Since 2000 Dr. Barba has been involved in several projects related with advanced compilers for EPIC architectures, co-design of heterogeneous systems and CAD tools for Hw/Sw integration of reconfigurable systems. These two research lines converge in his PhD. dissertation where a system level heterogeneous middleware for reconfigurable embedded systems is proposed.
Currently, he is overlapping his research activity with his position as a member of the Directive board at the School of Computer Science (UCLM, Spain) since 2012. Among the open research lines and interests it is worth mentioning the following: "Low cost, low power consumption reconfigurable systems for ubiquitous computing", "Reconfigurable computing platforms for AI algorithms", "Heterogeneous Distributed Embedded Computing" and "High-level Synthesis Tools".

María José Santofimia received the Ph.D degree in Computing Engineering by the University of Castilla-La Mancha, the degree of Technical Engineer in Computer Science in 2001 from the University of Córdoba (Spain); the Master's degree on Computer Security from the University of Glamorgan (Wales, UK) in 2003; and the degree of Engineer in Computer Science in 2006 from the University of Castilla-La Mancha (Spain). She is currently working as a Teaching Assistant in the University of Castilla-La Mancha, where she is also addressing her research interests towards achieving intelligent systems with real intelligence.

Julio Dondo completed his graduate studies in Electrical-Electronic Engineering in National University of San Luis- Argentine- in 1996, and receives the PhD degree from the University of Castilla-La Mancha in 2010, where he is currently working as an Teaching Assistant. His research interests include the integration of complex embedded systems, reconfigurable computing, heterogeneous distributed systems, high performance computing and Reconfigurable Grid Computing.

Fernando Rincón got his degree in Computer Science from the Autonomous University of Barcelona in 1993. In 1998 he moved to the University Of Castilla-La Mancha, where he received his PhD in 2003.
He is currently an Associate Professor at the School of Computer Science in the UCLM, and Head Of the Technologies and Information Systems Department. His actual research interests include heterogeneous distributed systems, design automation, and FPGA-based dynamically reconfigurable systems.

Julián Caba received the B.S. and M.S. degrees in Computer Science from University of Castilla-La Mancha, Spain, in 2006 and 2009, respectively. He is currently pursuing the Ph.D. degree from the Department of Technologies and Information Systems.
While pursuing the Ph.D. degree, he has been working with verification methodologies, like UVM, to verify hardware components. His current research interests include high-level synthesis, verification tools and methodologies and reconfigurable computing.

Juan Carlos López received the MS and Ph.D. degrees in Telecommunication Engineering from the Technical University of Madrid in 1985 and 1989, respectively. From September 1990 to August 1992, he was a Visiting Scientist in the Department of Electrical and Computer Engineering at Carnegie Mellon University, Pittsburgh, PA (USA). His research activities center on embedded system design, distributed computing and advanced communication services. He is author of different articles, conference papers and book chapters, and has led different national and international projects in these areas and has served as program committee member, session chair and reviewer in the main international conferences on design automation. From 1989 to 1999, he has been an Associate Professor of the Department of Electrical Engineering at the Technical University of Madrid. Currently, Dr. López is a Professor of Computer Architecture at the UCLM where he served as Dean of the School of Computer Science from 2000 to 2008, holding now the Indra Chair. He is and has been member of different panels of the Spanish National Science Foundation and the Spanish Ministry of Education and Science, regarding the Information Technologies research programs.