Octantis: An Exploration Tool for Beyond von Neumann architectures

Octantis: An Exploration Tool for Beyond von Neumann architectures

1st Andrea Marchesin
Dept. of Electronics and Telecomm.
Politecnico di Torino
Turin, Italy
andrea.marchesin@polito.it

2nd Giovanna Turvani
Dept. of Electronics and Telecomm.
Politecnico di Torino
Turin, Italy
giovanna.turvani@polito.it

3rd Andrea Coluccio
Dept. of Electronics and Telecomm.
Politecnico di Torino
Turin, Italy
andrea.coluccio@polito.it

4th Fabrizio Riente
Dept. of Electronics and Telecomm.
Politecnico di Torino
Turin, Italy

5th Marco Vacca
Dept. of Electronics and Telecomm.
Politecnico di Torino
Turin, Italy

6th Massimo Ruo Roch
Dept. of Electronics and Telecomm.
Politecnico di Torino
Turin, Italy

7th Mariagrazia Graziano
Dept. of Electronics and Telecomm.
Politecnico di Torino
Turin, Italy

8th Maurizio Zamboni
Dept. of Electronics and Telecomm.
Politecnico di Torino
Turin, Italy

***Abstract*—Nowadays, the modern electronic systems are facing an important limitation in terms of performance, known as *von Neumann bottleneck*. It affects the communications between two crucial elements, the CPU and the memory, which suffer from a saturation in bandwidth. Many solutions are currently under investigation and among them the concept of Logic-in-Memory (LiM) has been introduced: a memory enriched in its array of computational elements which enable the implementation of a flexible distributed processing system. The current work introduces Octantis, a High-Level Synthesizer useful for the exploration of LiM architectures. The proposed software analyzes an input algorithm described in standard C language and identifies which LiM architecture would implement it better. At its output, the synthesized solution is provided together with a test-bench, to properly characterize it, in terms of performance, spatial occupation and power consumption. Many algorithms have been successfully synthesized by Octantis and some of the results achieved will be discussed along the document.**

***Index Terms*—High-Level Synthesis (HLS), Logic-in-Memory (LiM), von Neumann bottleneck**

I. Introduction and Background

Retracing the history of the evolution of electronic devices, Moore's law has strongly influenced the growth of their computational capabilities. However, today's scientific community is called upon to tackle the so called *Von Neumann bottleneck*, where such a prosperous behavior is gradually undermined by a bandwidth limitation between the CPU and the Memory. Therefore, new computational paradigms are currently under investigation, to exceed these challenges. Among the different promising solutions, the *Logic-in-Memory (LiM)* concept has been recently proposed. Simple logic elements are introduced inside the same Memory array, which is now capable of performing partial computation without considering the CPU. This enables, consequently, a reduction of the load on the channel through which they exchange information. An intense research activity has been carried out over the years and important results have been achieved in [1], [2]. For the implementation of LiM structures, not only traditional technology has been considered, but also beyond-CMOS ones [3].

The aim of this paper is to present Octantis, a *High-Level Synthesis (HLS)* tool for the exploration of Logic-in-Memory architectures. The concept of HLS has been refined in the last decades and various approaches have been adopted. However, its design flow has remained almost constant [4], as the program represents essentially a compiler, a software which is able to transform an algorithm through different forms, and from which it inherits the overall structure [5]. In particular, these tools receive as an input an algorithm described by means of a High-Level language *(e.g. C and C++)* that has to be implemented in an Integrated Circuit. Hence, at their output, a customized hardware solution is provided. They are typically employed for the definition of regular designs, like *Intellectual Property (IP)* blocks and Memories and they are widely diffused throughout the Electronic Industry [6]. As Logic-in-Memory units consist in quite regular array structures, a High-Level Synthesis tool represents a good candidate for the agile exploration and design of those architectures.

Currently, there are many commercial High-Level Synthesizer, among which *Xilinx' "Vivado"* [7] and *Mentor Graphics' "Catapult"* [8]. However, there are many other

research tools made available for free, as *"LegUp"* [9] from the *University of Toronto* and *"Bambu"* [10] from *Politecnico di Milano*. Therefore, these programs have been considered as a reference for the definition of Octantis' structure. Then, it has been developed to bring to completion the synthesis of LiM Units. Therein lies the innovative nature of the tool and that sets it apart from the alternative solutions present in the state-of-the-art. The introduced High-Level Synthesizers are valuable for the definition of highly optimized *Application Specific Integrated Circuits (ASICs)*, which refer to many design techniques belonging to traditional schemes of electronic computation. Octantis inherits part of these synthesis strategies, but at the same time it introduces others which shall endeavour to design specifically leading-edge architectures, as the Logic-in-Memory arrays are.

Octantis has been developed for the study of Logic-in-Memory architectures at Politecnico di Torino. More in detail, Octantis is conceived to work in pair with another tool, called DExIMA [11], which represents a *simulator* for generic Logic-in-Memory architectures. Hence, Octantis and DExIMA merge in a *unicum* for the effective exploration of new Logic-in-Memory implementation possibilities, providing to a designer a complete architecture, fully characterized in terms of *performance*, *spatial occupation* and *power consumption*, starting form an input algorithm. A high level representation of the framework under development is depicted in Figure 1.

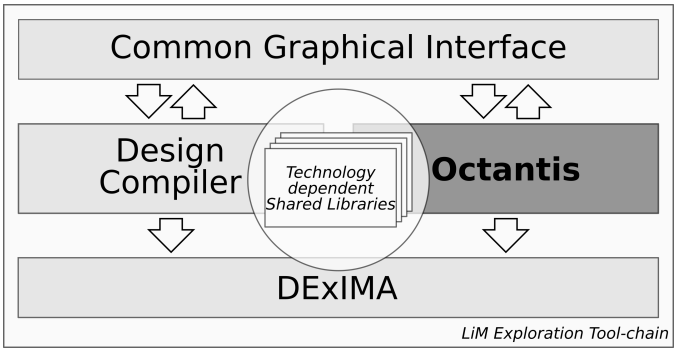

Fig. 1. Schematic of the Tool-chain under development at Politecnico di Torino for the exploration of innovative Logic-in-Memory Systems.

II. Octantis

The presented tool is capable of exploring Logic-in-Memory architectures from an input algorithm, expressed in standard C language, providing the description and the test-bench of a possible implementation. The tool requires also a *configuration file* to adapt the behavior of the synthesizer according to the constraints and specifications of the designer. Octantis is based on the *LLVM Compiler Infrastructure*, an open-source framework for the flexible definition of new compilers, or extensions for the existing ones [12]. The Clang library has been considered as the front-end of the entire compilation process, letting most of the Octantis' modules the roles of *optimizer* and *back-end*. The workflow of Octantis is depicted in Figure 2, where the main elements of the synthesis process are reported.

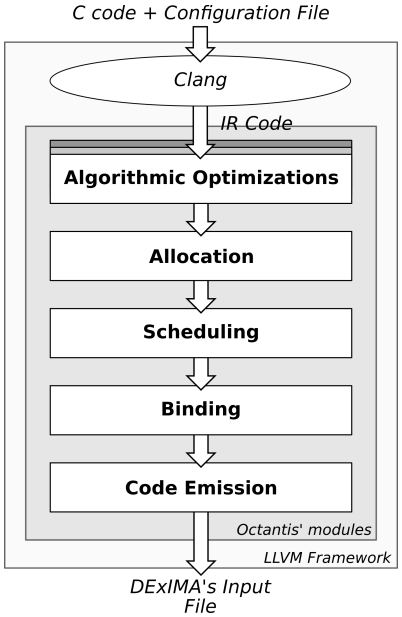

Fig. 2. Octantis' working principle. The whole program is integrated inside the LLVM Framework, from which it inherits Clang and some other compilation functions. At the output, the DExIMA's input file contains both the synthesized LiM Architecture and its Test-bench.

The algorithm provided at the input of Octantis, is firstly processed by the front-end component, translated into the *Intermediate Representation (IR)* which is, as its name suggests, a less abstract form of the input algorithm. However, the description remains generic enough to let the back-end component to easily adapt it into a specific hardware implementation. In particular, Clang verifies the general correctness of the input code, from a semantic and a syntactic perspective. As these operations are considered standard for a specific input language, the integration of Clang into Octantis has been conceived to be easily extended and reusable.

Starting from the IR of the input algorithm, Octantis performs a series of *optimizations*, executed in pipeline and standardized for the LLVM Infrastructure in *Passes*. These functions are useful to apply algorithmic transformation to the original algorithm to better express and accent the characteristics that could benefit from a Logic-in-Memory implementation. Then, the *back-end compilation* catches on, which is useful to finalize the translation of the algorithm into a real Logic-in-Memory architecture. As this process is complex and many operations are performed, the back-end component has been subdivided into different modules, according to the *modularity principle*. At the output of the program, the results of the logic synthesis are expressed into a format compatible with the simulator engine DExIMA, another tool capable of performing a full characterization of

the Logic-in-Memory solution, providing information about the *spatial occupation*, the *maximum operating frequency* and the *static and dynamic power consumption*.

In particular, Octantis provides as output two data structures:

- A *complete hardware description* of the synthesized Logic-in-Memory Array;
- A *test-bench* to estimate the performance of the proposed solution.

The detail of the working flow are discussed in the following, where the implementation choices are detailed. The compilation phase of the input C code into the Intermediate Representation by Clang is not reported as it represents a standard process. Hence, particular attention is given only to the subsequent elaborations, the ones proper to the synthesis of Logic-in-Memory architectures.

A. The input Optimizations

First, the Intermediate Representation code, directly obtained by Clang, is elaborated by means of various algorithmic optimization techniques, some of which inherited by the LLVM framework (e.g. *simplifycfg* and *licm Passes* [12]), others specialized to exploit the massive parallel computational capabilities that Logic-in-Memory architectures can provide. As the most important technique to describe parallel codes is represented by *loops*, an effective *loop pipeline* optimization has been introduced. This is a common operation performed by the most diffused High-Level Synthesizers [13], which allows to execute in parallel the different iterations of a loop, provided that the memory dependencies are not too binding. Therefore, a speedup can be introduced in the execution performance of the same algorithm.

B. The Allocation process

The optimized algorithm is then analyzed to extract the main needed resources to implement it effectively. In particular, the input configuration file is parsed, where the designer has imposed the hardware constraints for the synthesis process. Among them, there are parameters regarding the specifications on the data to be processed, as the *word length* (i.e. the parallelism of the data stored inside the memory). Others refer to hardware limitations, as the *maximum dimension* of the final LiM array. The obtained information is then organized in a proper data structure, useful for the subsequent compilation processes.
For the purpose of simplifying the definition of possible constraints, when Octantis starts the user is prompted if a template for a configuration file should be generated. In this way, the designer has to modify only the parameters related to the synthesis constraints, leaving the others to their default values.

C. The Scheduling process

The different instructions belonging to the algorithm are here scheduled in time. In this first version of Octantis, it has been decided to favour the performance of the synthesized solutions, therefore an *As-Soon-As-Possible (ASAP)* algorithm has been implemented. A specific time slot is assigned to each input operator in which it has to be executed and a suitable data structure is organized to gather this information.

D. The Binding process

Starting from the data structures collected during the previous steps, the instructions are mapped into physical hardware resources. During this process, also *target-oriented* optimizations are implemented, which try to exploit at best the peculiarities of the Logic-in-Memory paradigm. An example of these optimizations consists in the condensation of various operations that have to be executed into memory rows enriched in logic ports. Indeed, the Logic-in-Memory arrays can be composed of both traditional memory rows and memory rows with integrated logic. The more the rows are equipped with computational capabilities, the greater the speed up of the execution time of the algorithm.
Two data structures are here produced, describing: one the Logic-in-Memory array, expression of a complete synthesized architecture, and the other the *Finite-State-Machine (FSM)*, useful for characterizing its behaviour over time. At this point, the synthesis process is ended and the obtained architecture is ready for the last stage, which translates the results into a specific output language.

E. The Code Generation process

The two data structures just defined, i.e. the LiM Array and the associated FSM, are considered for the generation of a DExIMA's input file, ready for the simulation. This process consists of a translation, which can be substituted to bring compatibility to other target languages, oriented for the hardware description (e.g. VHDL or Verilog).

In conclusion, the synthesis ends with a complete Logic-in-Memory architecture implementing the input algorithm. This synthesis process is optimized for the exploitation of the Logic-in-Memory paradigm and provides results *correct by definition*. In fact, the algorithm is translated step-by-step in order to guarantee a one-way equivalence between the source information and the elaborated one. This aspect is important for the *validation* procedure of the synthesized architecture, which can be also tested through a test-bench which is automatically generated by the same tool.

III. The XNOR-Net: a Case Study

With a view of detailing the behavior of Octantis and how a designer can approach to the tool, a case study is here discussed. The description of the produced data will be preparatory to better understand the effective output of the program and, consequently, the results presented in the next Section. It is emphasized that the ease of use of Octantis is independent of the complexity of the input algorithm, hence the designer has to follow the same steps argued in the following.

In particular, the XNor-Net, described in [14], is synthesized. The circuit consists of an array of Logic-in-Memory cells which integrate XNor logic gates. This is an hardware component useful to perform the convolution operation inside the presented *Binarized Convolutional Neural Network*. Inside the memory array, the input data are stored together with a fixed weight which has to be applied through the XNor operation to each of them. This procedure represents an approximation of the multiplication. The obtained results are then externally accumulated to extract the needed features.

First, the input C code which describes the circuit has to be defined. An example of this algorithmic representation is reported in Figure 3. The designer has also to characterize Octantis' configuration file. In this case study, the only information requested for the correct synthesis of the final circuit is the dimension of the *word-length*. Considering the specifications of the reference article, a word length of *5 bits* has been imposed as constraint.

```
//Code for the implementation of a XNor Net
void XNor_Net(){

    //Allocation of the weight
        unsigned weight;
    //Allocation of the matrix for the input data
        unsigned dataMatrix[5];
    //Allocation of the rows for the output results
        unsigned outData[5];

    //Execution of the Xor operations on the data
    for(int i=0;i<5;++i)
                outData[i]=~(weight^dataMatrix[i]);

}
```

Fig. 3. Tested input code for the LiM implementation of the *XNOR Net*.

The synthesis process can be now launched and both the information about the LiM architecture and the related FSM are detailed. On the terminal window where Octantis is under execution, the final results of the synthesis are summarized, including eventual error messages and suggestions to possible corrections. Specifically, a logic representation of the obtained solution is depicted in Figure 4, while the related details have been gathered inside Table I.

TABLE I
INFORMATION ABOUT THE LIM STRUCTURE PROVIDED BY OCTANTIS.

Row Number	Address	Word Length	Integrated Logic	Input Connect.	Scheduling Time
1	000	5	—	—	0
2	001	5	XNor	0000	0
3	010	5	XNor	0000	0
4	011	5	XNor	0000	0
5	100	5	XNor	0000	0
6	101	5	XNor	0000	0

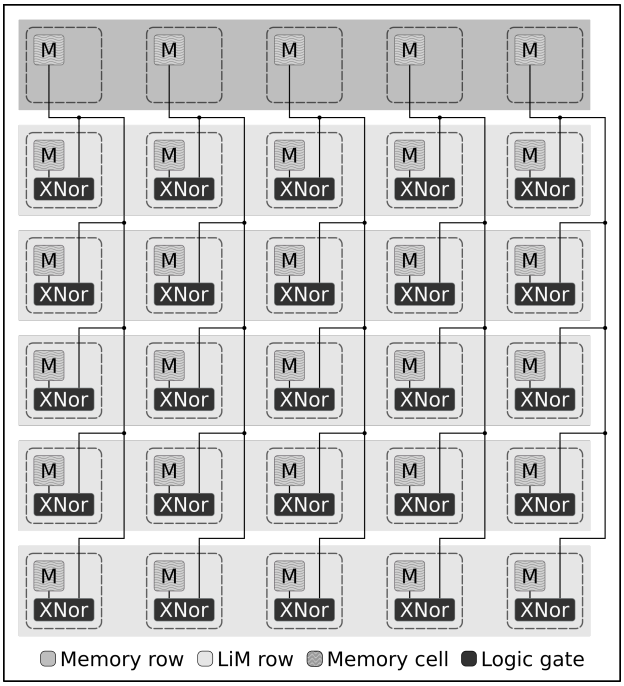

Fig. 4. Graphical representation of the synthesized XNor-Net. Only the internal communication channels are reported in order to highlight the flow of information inside the LiM array.

Considering the implemented ASAP scheduling algorithm, the synthesized architecture is coherent with what could be expected. In particular, assuming that the memory initialization phase has been performed before the execution of the algorithm starts, the XNor Net performs the associated operations within a single clock cycle *(as it could be seen in the Table, looking at the assigned Scheduling Time)*.

IV. RESULTS

In order to prove the effectiveness of the tool, different algorithms have been successfully synthesized. In addition to the example proposed in the previous section, two more architectures with different complexities are discussed. In particular, two recent works [15] and [16] have been taken in consideration as a benchmark for the Octantis' synthesis capabilities. However, simplified architectures have been modeled before proceeding. In fact, they represent very optimized ASIC circuits which allow to configure their behavior in order to implement different nuances of the algorithms that they are meant.

For each work, an algorithm in standard C has been defined and then processed by means of Octantis. The obtained results have been analyzed to verify the quality of the synthesis process and particular attention has been given to the quantity of logic integrated inside the memory array and the execution time. This information has been subsequently compared with the results achieved and discussed in the cited works. The results of the conducted tests are gathered inside Table II, where also the data derived from the case study are included.

TABLE II
RESULTS OF THE CONDUCTED TESTS ON OCTANTIS.

Implemented Algorithm	Synthesized Architecture			Reference Architecture		
	Memory dimensions	*Integrated Logic*	*Exec. time*	*Memory dimensions*	*Integrated Logic*	*Exec. time*
X-NOR Net [14]	25 bits	XNor: 25	1 T_{clk}	25 bits	XNor: 25	1 T_{clk}
Bitmap Indexing [15]	144 bits	And: 128 Or: 128 Xor: 128 Mux 3-to-1 16 bits: 8	T_{exe} [(a)]	144 bits	And: 128 Or: 128 Xor: 128 Mux 3-to-1 16 bit: 8	T_{exe} [(a)]
Convolutional Neural Network [16]	136 bits	Full/Half-Adder: 64 Mux 2-to-1 8 bits: 9	T_{exe} [(a)]	120 bits	Full/Half-Adder: 48 Mux 2-to-1 8 bits: 9	T_{exe} [(a)]

[(a)] As the reference architectures are configurable and so many algorithms can be implemented, the execution time is expressed in a parametric way.

All the synthesis processes have produced Logic-in-Memory architectures functionally equivalent to the original implementations. The first one matches also structurally, while the other one have featured an overhead of the needed hardware resources due to the different optimizations implemented, those of Octantis versus those of the Author. As regards the timing information, both the solutions need the same time, in terms of clock cycles, to execute the implemented algorithms.

V. CONCLUSIONS

With this paper, the working principle of Octantis has been described, bringing out its basic scheme and its functionalities. Along this discussion, also importance has been given to the environment in which the program is located and to the contribution it provides to a designer in the exploration of possible Logic-in-Memory solutions. The synthesis performed from an input high-level algorithm generates logical structures, independent from a particular technology. Hence, it allows the study of different types of final implementations, from the traditional to the newest ones, today under investigation by researchers.

Octantis aims to be a valid guide during the exploratory phase of Logic-in-Memory architectures, a promising solution for how electronics could evolve in the coming years.

REFERENCES

[1] K. Yang, R. Karam, and S. Bhunia, "Interleaved logic-in-memory architecture for energy-efficient fine-grained data processing," in *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2017, pp. 409–412.

[2] M. Cofano, M. Vacca, G. Santoro, G. Causapruno, G. Turvani, and M. Graziano, "Exploiting the logic-in-memory paradigm for speeding-up data-intensive algorithms," *Integration*, vol. 66, pp. 153–163, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016792601830556X

[3] G. Santoro, G. Turvani, and M. Graziano, "New logic-in-memory paradigms: An architectural and technological perspective," *Micromachines*, vol. 10, no. 6, 2019. [Online]. Available: https://www.mdpi.com/2072-666X/10/6/368

[4] G. Martin and G. Smith, "High-level synthesis: Past, present, and future," *IEEE Design Test of Computers*, vol. 26, no. 4, pp. 18–25, July 2009. [Online]. Available: http://dx.doi.org/10.1109/MDT.2009.83

[5] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison-Wesley Longman Publishing Co., Inc., 2006.

[6] A. Takach, "High-level synthesis: Status, trends, and future directions," *IEEE Design Test*, vol. 33, no. 3, pp. 116–124, June 2016.

[7] "Vivado design suite." [Online]. Available: https://www.xilinx.com/products/design-tools/vivado.html

[8] "Catapult high-level synthesis and verification." [Online]. Available: https://eda.sw.siemens.com/en-US/ic/catapult-high-level-synthesis

[9] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, T. Czajkowski, S. Brown, and J. Anderson, "Legup: An open-source high-level synthesis tool for fpga-based processor/accelerator systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, 09 2013.

[10] C. Pilato and F. Ferrandi, "Bambu: A modular framework for the high level synthesis of memory-intensive applications," in *2013 23rd International Conference on Field programmable Logic and Applications*, Sep. 2013, pp. 1–4.

[11] N. Piano, "Dexima: a design explorer for in-memory architectures," Master's thesis, Politecnico di Torino, 2019. [Online]. Available: https://webthesis.biblio.polito.it/12547/

[12] *LLVM's Analysis and Transform Passes*. [Online]. Available: https://llvm.org/docs/Passes.html

[13] S. Lahti, P. Sjövall, J. Vanne, and T. D. Hämäläinen, "Are we there yet? a study on the state of high-level synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 5, pp. 898–911, May 2019.

[14] A. Coluccio, M. Vacca, and G. Turvani, "Logic-in-memory computation: Is it worth it? a binary neural network case study," *Journal of Low Power Electronics and Applications*, vol. 10, no. 1, p. 7, Feb 2020.

[15] M. Andrighetti, G. Turvani, G. Santoro, M. Vacca, A. Marchesin, F. Ottati, M. Ruo Roch, M. Graziano, and M. Zamboni, "Data processing and information classification—an in-memory approach," *Sensors*, vol. 20, no. 6, p. 1681, Mar 2020. [Online]. Available: http://dx.doi.org/10.3390/s20061681

[16] G. Santoro, G. Turvani, and M. Graziano, "New logic-in-memory paradigms: An architectural and technological perspective," *Micromachines*, vol. 10, no. 6, p. 368, May 2019. [Online]. Available: http://dx.doi.org/10.3390/mi10060368