Bitmap Index: a Processing-in-Memory reconfigurable implementation

M. Andrighetti[1], G. Turvani[1], G. Santoro[1], M. Vacca[1], M. Ruo Roch[1], M. Graziano[1], and M. Zamboni[1]

Politecnico di Torino, Department of Electronics and Telecommunications
giovanna.turvani@polito.it

Abstract. During the years, microprocessors went through impressive performance improvement thanks to technology development. CPUs became able to process great quantities of data. Memories also faced growth especially in density, but as far as speed is concerned the improvement did not proceed as the same rate. Processing-in-Memory (PIM) consists in enhancing the storage unit of a system, adding computing capabilities to memory cells, partially eliminating the need to transfer data from memory to execution unit. In this paper, a PIM architecture is presented for bulk bitwise operation mapped on the Bitmap Index application. The architecture is a memory array with logical computing abilities inside the cells. The array is a configurable modular architecture distributed in different banks, each bank is able to perform a different operation at the same time. This architecture has remarkable performance being faster than other solutions available in literature.

Keywords: Processing-in-memory, Bitmap Index, Reconfigurable architecture

1 Introduction

Nowadays, data-intensive applications, such as image processing and databases ones, must process big amounts of data. This is a consequence of the speed improvement obtained throughout the years thanks to technology scaling. However, memory development did not follow the same path, resulting in a much slower performance increase. This disparity reduces the overall computing capability of the system, as memory is not able to provide data as fast as CPU demands them. This issue is known as *memory wall* or *Von Neumann bottleneck*. A possible solution to this problem is to nullify the distance between processor and memory, removing the cost of data transfer and creating a unit which is capable of storing information and performing operations at the same time. This concept is called Processing-in-Memory (PIM). There are many different approaches in literature to the Processing-in-Memory idea.

People have exploited new emerging technologies, such as NML (Nano Magnetic Logic) [3] and *Magnetic Random Access Memory* (MRAM), a non-volatile memory that uses Magnetic Tunnel Junctions (MTJs) as its basic element.

Thanks to their storage and logic properties, MTJs can be used to implement hybrid logic circuits with CMOS technology ideal for a PIM architecture [7]. Another widely explored technology is *Resistive RAM*, a non-volatile memory that exploits a resistive component (metal-insulator-metal structure) to store information. ReRAM arrays are usually found in crossbar structures that enable the implementation of matrix-vector multiplication, commonly used in neural networks applications. One example of such implementation is PRIME [4], an architecture aimed at accelerating Artificial Neural Networks, which are based on operations that perfectly fit the crossbar structure. While the previous proposals shaped their approach on a particular technology, others worked on an architectural perspective, independently from the technology itself. Some tried to narrow the physical distance between memory and computation unit by stacking them on a 3-Dimensional structure, enhancing the available bandwidth by connecting the layers through True Silicon Vias [5]. Anyhow, it should be noticed that in this case even if the two units (memory and logic) are moved very close to each other, they are still distinct components. Another possible approach is to creates a system composed of an host processor surrounded by several HMC-based (Hybrid Memory Cube) units, composed of multiple memory layers stacked on a logic layer [10]. A different solution is to slightly modify the circuits controlling the memory to implement simple logic operations inside the memory array, such as Ambit[9], an in-memory accelerator which exploits DRAM technology. The DRAM array is slightly modified to perform AND, OR and NOT operations. Other examples are presented in [6, 2, 1]. Among the many proposals provided by literature, one of the best fitting representative of the PIM concept is presented in [8]. In this work the proposed architecture is a memory array where the cell itself is capable of performing several logical operations on the stored value.

In this paper, we propose a different solution of Processing-in-Memory, presenting an architecture shaped around the application of Bitmap Indexing, thus suitable for bulk bitwise operations. The proposed architecture is a memory array in which each cell is able to both store information and to be configured to execute simple logical operations such as AND, OR and XOR. The array is also distributed into banks and each bank is able to work both independently and with other banks, solving different queries, achieving flexibility and an high degree of parallelism. Since the structure is modular it can be built with as many banks as needed. The architecture synthesized is an array of 8,512 KB, distributed on 16 banks. The technologies used are CMOS 45nm and 28nm. The results obtained highlight great potential as the synthesized structure can reach a maximum throughput of 2.45Gop/s and 9.2Gop/s for 45nm and 28nm respectively and it is noticeably faster than other solutions presented in literature.

2 The Architecture

The Processing-in-Memory paradigm requires that logic and storage elements are merged together. This paradigm is particularly suited for all those algorithms that need to perform huge amount of simple operations on data stored. To demonstrate the advantages that the PIM approach can provide, we choose to implement an architecture able to solve the Bitmap indexing problem. The Bitmap indexing is an important algorithm often used in database management systems.

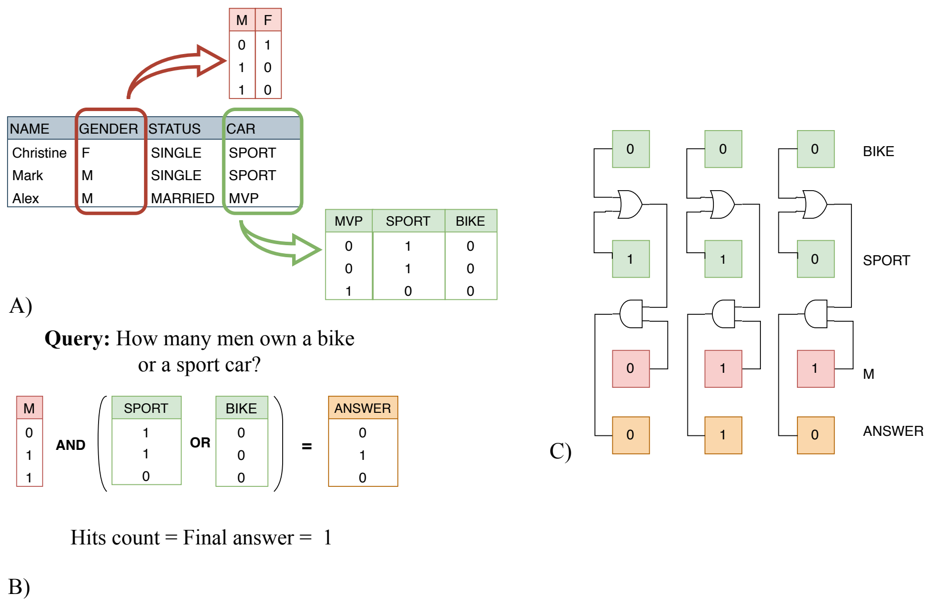

Fig. 1. A) Given a table, bitmap indexing transforms each column in as many bitmap as the number of possible key-values for that column B) In order to answer a query logic, bitwise operations are to be performed C) Practical scheme of the execution of the query.

The Bitmax Indexing is an algorithm used to identify, inside a database, entries that have specific characteristics. For example, inside the database of Fig. 1.A the query consist in the identification of how many man own a motorbike or a sport car. To reach this goal each feature is indexed using a binary representation. The gender column, for example, is divided in two sub-columns, one representing the male gender and one representing the female gender. Then each sub-column is represented using single bits. For example the first entry of the database is a female, so the M column contains ’0’, while the F column contains ’1’ (see Fig. 1.A). Searching for a specific query inside such database means performing simple logic operations between each sub-column, as depicted in Fig. 1.B.

In our architecture, instead of memorizing the database inside the memory following the same structure proposed in Fig. 1, we memorize the transpose of the matrix of bit representing the database. With this solution every row of the memory contains a column representing a specific feature. As a consequence

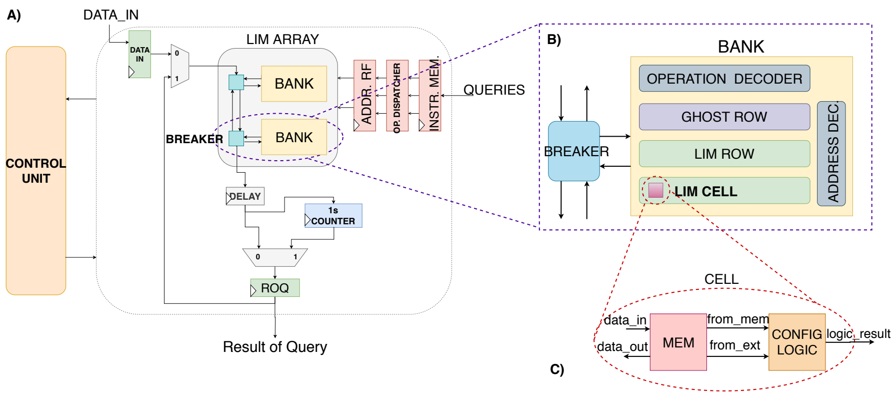

Fig. 2. A) Overview of the complete architecture. B) Structure of the duo Bank-Breaker C) Insight of the PIM cell

to search for a specific query in the database it is necessary to execute logic operations between subsequent rows of the memory (Fig. 1.C). To reach this goal we have designed a memory cell that consists of a memory element and a configurable logic block (Fig. 2.C, more details on the implementation will be given in Section 3).

Fig. 2 provides an overview of the complete architecture. The core part is represented by a memory storing the database. To give more flexibility to the structure the memory array is divided in banks. The circuit can be used as a standard memory if configured in that way. Otherwise it is possible to perform logic operations on stored data and to implement the Bitmap Indexing algorithm. When a query is executed all the banks in the array can eventually be activated in parallel, performing different logic operations on different rows in the bank. This is the biggest advantage of the proposed architecture because it is possible to perform a logic operation on all the data stored inside a memory bank in parallel, leading to a huge speedup in the execution of the algorithm. As depicted in Fig. 2.A the memory array is surrounded by additional logic circuits and a control unit. For space reason we cannot describe the details of each block. The control is used to guarantee the correct execution of the algorithm according to the input queries. The instruction memory block is used to collect the queries to execute. It consists in a register file having as many registers as the number of the banks in the array. The operation dispatcher is in charge of blocking any old query. Also, since a query can take place between any couple of addresses in the array, it necessary to send the addresses to their respective bank. Thus the operation dispatcher reorders the addresses and then the address register file sends them to their own bank.As in Fig. 2.B each memory bank contains also ghost memory rows used to store temporary results. To handle all the configuration signals needed to manage the correct execution, two decoders are needed inside each bank. The first one configures the logic operation to execute, sending it to the right row. The second was inserted to control addresses, data flow inside the bank and select between PIM and standard memory mode. The breaker block is used to enable the communication among different banks. This structure is flexible and can be easily reconfigured to implement other algorithms.

3 Results and conclusions

To evaluate the performance of the structure, a circuit composed by a 8,512 KB PIM array, distributed on 16 banks with 16 bit data size, was implemented. Then, the architecture, implemented in VHDL (VHSIC Hardware Description Language), was tested with Modelsim and later synthesized with Synopsys Design Compiler using 45 nm BULK and 28 nm FDSOI CMOS technologies (table 1). In this first implementation the storage elements were synthesized as latches, instead of designing a custom memory cell. As a consequence the results here presented can be greatly improved by designing a custom memory-logic cell.

Parameter	**45nm**	**28nm**
Total area $[mm^2]$	2.33	1.058
f_{CLK} $[MHz]$	153.4	574.7
Total Power $[mW]$	49.7	14.07

Table 1. Synthesis results for 45nm and 28 nm CMOS technologies

	$f = A \cdot B$	$f = A \cdot (\overline{B} \cdot C)$
Pinatubo[6]	5	9
RIMPA[2]	3	5
PIMA[1]	1	3
PIM	1	2

Table 2. Clock cycles comparison for a single query execution

Table 1 highlights the synthesis results. As it can be noticed the architecture is very efficient, it is capable of high clock speed but at the same time has a low power consumption.

One of the main goal this paper aimed to fulfill is the high level of concurrency. This was accomplished thanks to the internal organization of the array, that is distributed on banks which are capable of working both independently and with each other, providing flexibility in the position of the operands that are called to act in the query. To execute a simple query only one cycle is required (Table 2).

The maximum throughput achievable is $throughput_{max} = f_{CLK} \cdot N_{ops}$. Assuming to execute a different query in each of the 16 available banks, a maximum throughput of 2.45Gop/s and 9.2Gop/s for 45nm and 28nm can be reached. Table 2 highlights the comparison of the proposed architecture with the state of the art in terms of clock cycles required for an operation. Our architecture is always faster than the other solution proposed in literature.

It should be taken into account that even with multiple parallel operations the clock cycles required would remain constant, achieving the throughput mentioned above, meaning also that the maximum degree of parallelism reachable is equal to the number of the available banks. Moreover, thanks to its modular structure, the architecture is meant to be easily scaled to bigger dimensions and with as many banks as needed. It could also be possible to develop a 3D structure in order to increase performance. The architecture could be easily modified to implement other types of operations. In conclusion, this architecture demonstrates that a Processing-in-Memory approach leads to a great improvement of performance. The architecture here proposed achieve very good performance and has enough flexibility to be adapted to several different algorithms.

References

1. Angizi, S., He, Z., Fan, D.: Pima-logic: A novel processing-in-memory architecture for highly flexible and energy-efficient logic computation. In: 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC). pp. 1–6 (June 2018)
2. Angizi, S., He, Z., Parveen, F., Fan, D.: Rimpa: A new reconfigurable dual-mode in-memory processing architecture with spin hall effect-driven domain wall motion device. In: 2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI). pp. 45–50 (July 2017)
3. Causapruno, G., Riente, F., Turvani, G., Vacca, M., Roch, M.R., Zamboni, M., Graziano, M.: Reconfigurable systolic array: From architecture to physical design for nml. IEEE Transactions on Very Large Scale Integration (VLSI) Systems PP(99), 1–10 (2016)
4. Chi, P., Li, S., Xu, C., Zhang, T., Zhao, J., Liu, Y., Wang, Y., Xie, Y.: Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. In: 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA). pp. 27–39 (June 2016)
5. Kim, D.H., Athikulwongse, K., Healy, M.B., Hossain, M.M., Jung, M., Khorosh, I., Kumar, G., Lee, Y.J., Lewis, D.L., Lin, T.W., Liu, C., Panth, S., Pathak, M., Ren, M., Shen, G., Song, T., Woo, D.H., Zhao, X., Kim, J., Choi, H., Loh, G.H., Lee, H.H.S., Lim, S.K.: Design and analysis of 3d-maps (3d massively parallel processor with stacked memory). IEEE Transactions on Computers 64(1), 112–125 (Jan 2015)
6. Li, S., Xu, C., Zou, Q., Zhao, J., Lu, Y., Xie, Y.: Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories. In: 2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC). pp. 1–6 (June 2016)
7. Matsunaga, S., Hayakawa, J., Ikeda, S., Miura, K., Endoh, T., Ohno, H., Hanyu, T.: Mtj-based nonvolatile logic-in-memory circuit, future prospects and issues. In: 2009 Design, Automation Test in Europe Conference Exhibition. pp. 433–435 (April 2009)
8. Santoro, G., Turvani, G., Graziano, M.: New logic-in-memory paradigms: An architectural and technological perspective. Micromachines 10(6) (2019), `https://www.mdpi.com/2072-666X/10/6/368`
9. Seshadri, V., Lee, D., Mullins, T., Hassan, H., Boroumand, A., Kim, J., Kozuch, M.A., Mutlu, O., Gibbons, P.B., Mowry, T.C.: Ambit: In-memory accelerator for bulk bitwise operations using commodity dram technology. In: Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture. pp. 273–287. MICRO-50 '17, ACM, New York, NY, USA (2017), `http://doi.acm.org/10.1145/3123939.3124544`

10. Zhang, D., Jayasena, N., Lyashevsky, A., Greathouse, J.L., Xu, L., Ignatowski, M.: Top-pim: Throughput-oriented programmable processing in memory. In: Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing. pp. 85–98. HPDC '14, ACM, New York, NY, USA (2014), `http://doi.acm.org/10.1145/2600212.2600213`