Exploiting the Logic-In-Memory paradigm for speeding-up data-intensive algorithms

Exploiting the Logic-In-Memory paradigm for speeding-up data-intensive algorithms

Mario Cofano, Marco Vacca, Giulia Santoro*, Giovanni Causapruno, Giovanna Turvani, Mariagrazia Graziano

Dipartimento di Elettronica e Telecomunicazioni, Politecnico di Torino

Corso Castelfidardo 39, 10129, Torino

Abstract

In the last decades transistor scaling has driven electronics toward an extraordinary evolution. The ability to squeeze millions of transistors on a single chip makes it possible to have an incredible computational power in very small size. Many computational systems are still based on the Von Neumann architecture, where computational units and memory blocks are two separate entities. Nanometer-sized transistors enable the development of incredibly fast logic units that cannot work at full speed due to limitations in data transfer from memory. To further evolve electronic circuits, new innovative architectural solutions must be developed to overcome the main limitations of current systems. In this work, we present an architectural implementation of the *Logic-In-Memory* (LIM) concept that we characterize by considering three data-intensive benchmarks: the *odd even sort*, the *integral image* and the *binomial filter*. The architecture is synthesized on a 28nm CMOS technology and it is validated by comparing it to a previous version of the LIM structure and to conventional architectures, showing an impressive increase in performance, in terms of speed gain and power consumption reduction.

Keywords: Logic-in-Memory, Smart Memory, Parallel Architectures, Data-intensive Algorithms, Non-Von Neumann Architectures

*Corresponding author

Email address: giulia.santoro@polito.it (Giulia Santoro)

1. Introduction

The Von Neumann architecture represents the foundation of modern computational systems, on which most of the existing electronic devices are based. The Von Neumann model [1] refers to an architecture in which a processing unit communicates, through a bus, with a memory that stores both instructions and data. This simple structure has been exploited successfully for designing digital computing systems for many decades. The technological progress of CMOS systems, that has mainly taken place as a result of the transistor scaling [2][3], has, on one hand, made it possible to build more powerful computing systems, while on the other, it has stressed the so-called Von Neumann bottleneck. Indeed, the Von Neumann model suffers from a limited data communication rate between the memory and the processing unit. In turn, the effective working speed of the processing unit is limited by the memory bandwidth that it is not able to provide the required amount of data. Different solutions have been proposed in order to solve this problem. Memory hierarchy is one of them: here the idea is to have smaller and faster memories closer to the processing unit (even embedded on-chip [4]), while larger and slower memories are located far from the CPU and are accessed very few times. However, these techniques are not sufficient to definitively solve the memory bottleneck problem which gets even worse in parallel computing systems.

As a consequence, new computing paradigms are currently under development. In our work, we focus our attention on the so-called *Logic-In-Memory* (LIM) concept [5][6][7][8]. The idea behind the LIM principle is to eliminate the physical separation between logic and memory by creating a system where they can be embedded together. In particular, in this article, we present how the concept of LIM can be exploited for speeding up data-intensive algorithms. Indeed, we have optimized a smart memory system enabling a fast access to data, limiting, in turn, the memory-wall issue. Our main purpose is the demonstration of how much this architecture can improve performance in comparison

to conventional architectures. We have validated the proposed approach by selecting three widely used algorithms, however, our final aim is to exploit the same parading to solve a wider class of problems.

The paper is organized as follows: in Section 2 we briefly describe the main motivations that arise the need for fast architectures like Systolic Arrays and GPUs and we introduce our idea of LIM. In Section 3 we present prior work in the field of Logic-In-Memory, including a brief description of a previously proposed version of our LIM architecture. Then, in Section 4, we present, for the first time, a novel LIM structure called Pyramidal LIM (P-LIM). Finally, Section 5 presents the synthesis results of these architectures, whose performance are compared with other state-of-the-art computational systems in Section 6.

2. Background

Different types of processing units are available nowadays, but those that mostly put in evidence the philosophy behind the design of modern computational systems are scalar microprocessors [9]. In this type of processors, an execution unit, coordinated by a control unit, executes instructions on data stored in a dedicated memory. The concept behind microprocessors is indeed very simple, but their architecture has evolved greatly throughout the years to cope with the advancement of technology. CMOS scaling has enabled the integration of an increasing number of transistors on a single chip and the achievement of higher working frequencies. In addition, memory architectures have undergone a significant evolution that has led to the introduction of caching mechanisms coupled with hierarchical memory structures to cope with the demanding data requirements of processors. Even so, the well known "memory wall" problem [10][11] is not solved.

As depicted in Figure 1.A, the performance gap among processing units and memories is steadily increasing with CMOS technology advancements [12]. Consequently, processing systems cannot exploit their full potential. The limited bandwidth of communication buses represents, nowadays, the main bottleneck

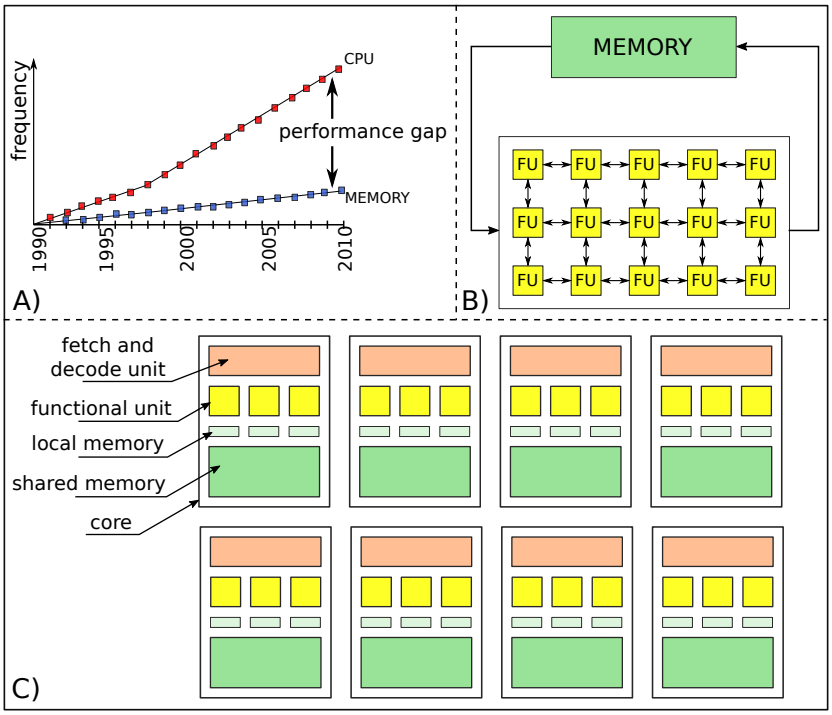


Figure 1: (A) Performance gap between logic and memory increases through the years. (B) Schematic representation of a systolic array. (C) Schematic representation of a GPU.

of processing units. As it happens in GPUs, a huge quantity of data must be continuously fetched from the main memory, slowing down the computation task. Architectures like GPUs and systolic arrays, represent the main computing models adopted nowadays to run parallel algorithms. They have been used as a starting point to conceive our LIM architectures and then, as a basis of comparison. In the following, we will present the main characteristics of these architectures, then we will introduce our conception of *Logic In Memory* as a further possible solution to the memory wall problem. The main advantage of our approach should be seen in an optimized smart memory enabling a considerable speed-up of the data fetching. In other words, we want to demonstrate that, for some types of problems, our conception of LIM can determine significant performance improvements.

2.1. Systolic Arrays

As depicted in Figure 1.B, a systolic array [13][14] is composed of several functional units (FUs) working in parallel. FUs typically execute a single operation; they can be provided with few registers to temporarily store the result of the operation depending on the type of systolic array [15]. Each FU receives data from neighboring elements or from the outside and output signals are sent to neighboring FUs or to the outside. Few key elements distinguish, therefore, systolic arrays: I) FUs work in parallel performing, usually, the same operation [16]; II) FUs are simple and, generally, they are in large number [17]; III) communication is local among FUs, easing the memory wall problem [18][19]. Systolic arrays show good performance when FUs continuously exchange data with each other, hence exploiting local interconnections, while accessing the memory very few times.

2.2. Graphic Processing Units (GPUs)

GPUs can exploit parallelism among cores and among functional units within a core. In fact, as shown in Figure 1.C, the GPU is composed of a large number of cores composed of different functional units working in parallel [20]. Typically, a single core is assigned to a thread [21][22], then, instructions are fetched and executed in parallel like in a SIMD (Single Instruction Multiple Data) structure.

Functional units can execute a wider number of operations and, in addition, they have access to their own local memory and communicate with other units not in a direct way, as in systolic arrays, but by accessing the shared memory. As a consequence, inter-FUs communication should be maintained low in order to avoid an excessive number of memory accesses. The communication with the memory is much more complex than in a systolic array, due to the different types of memories available (local, shared and global). A correct handling of memory communication can have a huge impact on GPUs performance.
GPUs are also heavily used as hardware accelerators. An algorithm can be defined suitable for a GPU if it is massively parallel and if the number of mathematical operations to be executed is higher than the number of memory ac-

cesses. This concept is expressed by the arithmetic intensity (equation 1) that is calculated as the number of mathematical operations executed by a single functional unit divided by the number of memory accesses to load data.

$$\text{Arithmetic Intensity} = \frac{\#\text{arith. operations}}{\#\text{MEM access}} \tag{1}$$

2.3. The LIM concept

When conceiving our Logic-In-Memory architecture we have tried to take into account the main features of both systolic arrays and GPUs by designing a structure that is highly parallel, programmable and that has local interconnections in order to allow direct data exchange among processing units. Moreover, the aim is to reduce the separation between logic and memory by embedding them in a single entity. We believe that there are two key concepts that distinguish a *Logic-In-Memory* system:

1. **Locality**: memory elements are distributed within the circuit. In this way, the bottleneck created by accessing an external memory is avoided. Indeed, data communication through local data exchange among local memory cells.
2. **Intelligence**: with this approach, *intelligent* memories can communicate independently with other neighboring cells providing *smart* data to its logic units.

According to the nature of the LIM architecture that we propose, not all algorithms are fitted for it. In particular, suitable algorithms should be highly parallelizable and should leverage on local interactions among neighboring processing elements. We have identified four classes of algorithms that satisfy these requirements: sorting algorithms, cryptography, mathematical problems and image processing [23][24][25]. In this work, we use the odd-even sort algorithm, the integral image and the binomial filter to test and validate our architecture because they fit perfectly the above requirements. The pseudo-code of the odd-even sort algorithm is reported in Listing 1, where `a` is the vector of length `n` to sort.

```
void OddEvenSort (T a[], int n) {
   for (int i=0; i<n; i++) {
     if (i&1) /* 'i' is odd */ {
       for (int j=2; j < n; j+=2) {
         if (a[j] < a[j-1])
           swap(a[j-1], a[j]);
       }
     }
     else {
       for (int j=1; j < n; j+=2) {
         if (a[j] < a[j-1])
           swap(a[j-1], a[j]);
       }
     }
   }
}
```

Listing 1: Odd-even sort pseudo code.

Starting from a list of elements, the algorithm compares all odd/even pairs of adjacent elements and, if the order is wrong, elements are swapped. In the next step the procedure is repeated for even/odd pairs. Then odd/even and even/odd steps are iterated until the list is sorted.

The integral image algorithm, also known as *summed area table*, is used in

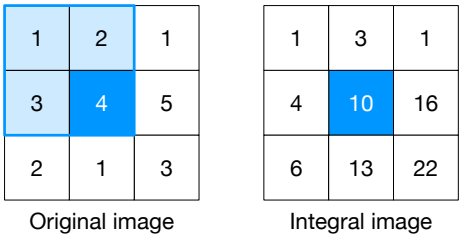


Figure 2: Integral image computation: considering the blue pixel in the original image, the correspondent integral pixel is the sum of the pixels above and to the left of it (blue rectangle).

the image processing field to compute, in an efficient and fast way, the sum of

pixel values over image sub-regions. Basically, the value of a certain pixel in an integral image is the sum of all the pixels positioned above and to the left of it, as shown in figure 2.

Binomial filters are widely used as discrete approximation of Gaussian filters in image processing applications. Given an input image, the binomial filtering is computed by applying and shifting the filter all over the image. Given, for

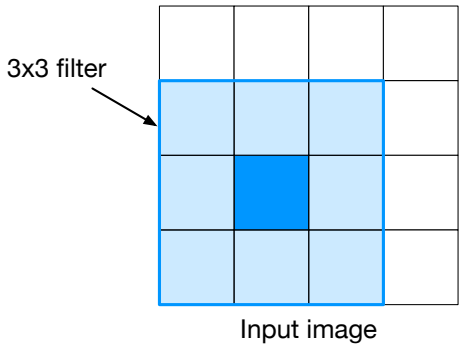


Figure 3: Binomial filtering: a 3×3 filter is applied over the input image and the new value of the central pixel (dark blue) is calculated as the weighted sum of the surrounding pixels (light blue). This computation is iterated over the whole input image.

example, a 3×3 filter as depicted in figure 3, the new value of the central pixel (dark blue) is calculated as the weighted sum of the surrounding pixels (light blue). This operation is repeated for all the pixels in the given image.

3. Related Work

The State of the Art of Logic-in-Memory is wide but the main approaches to this idea can be divided into three categories:

1. Logic and memory stacked one on top of the other exploiting the recent advances in 3D IC technology;
2. Control circuitry of the memory modified/used to perform logic or arithmetic operations;
3. Logic integrated inside memory cells or memory array used as-is to execute operations.

Works that follow the first approach [6][7][26][27] propose systems in which a standard CMOS computing unit is placed below a 3D-stacked memory (such as Hybrid Memory Cube [28]) exploiting TSV connections. These 3D-stacked architectures benefit from the wide I/O bandwidth provided by such kind of memories, significantly decreasing the memory bottleneck issue. Other works, belonging to the second category, use the computing capabilities of the logic layer of HMC to perform simple operations [29] or to design specialized hardware for graph processing as in [30]. In [31], authors add simple logic inside a generic 3D-stacked DRAM chip to perform Neural Network related operations. These works not only leverage the wide memory bandwidth but they also exploit the near-memory computing capabilities offered by 3D memories. In [32] authors modify the control circuitry of a generic resistive-based memory in order to support bulk bitwise logic operations. Similarly, in [33] authors propose a Processing-in-Memory architecture based on SOT-MRAM that can compute, efficiently, complex bit-wise operations between operands either in the same row or in the same column. The third approach exploits the intrinsic characteristics of emerging non-volatile memories to execute operations on data. In this case, since a big part of data computation is done in the memory, the main benefit is the reduced data movement and, as a result, a significant decrease in the number of memory accesses. In [34], authors propose a resistive TCAM processor to perform LUT-based computation, while in [35] resistive CAMs are used to accelerate query processing. Resistive memories are also used to perform matrix multiplications, as in [36], where one operand is stored in the ReRAM array as a resistance value while the other is given as input to the array. In [37], authors propose a modified DRAM cell to perform bulk bitwise logic operations. Being a standard DRAM technology, it can be integrated in a processor and controlled with CPU instructions. An MTJ-based (Magnetic Tunnel Junction) Logic-in-Memory system is presented in [38]. In this work, CMOS gates are combined with MTJ devices to implement non-volatile logic cells arranged in arrays to form a CAM-like structure for search operations.

In [8], [39] and [40] we presented the first version of our *Logic-In-Memory* ar-

chitecture. It uses a simple and custom logic core, allowing us to pack hundreds or thousands of processing elements inside one chip. We use a three-layer structure, where a routing plane is added to logic and memory planes to handle communication. This first version of Logic-In-Memory is called Systolic LIM (S-LIM) because its structure is similar to a systolic array. As depicted in Figure 4, the architecture is an array of identical cells that work in parallel. The big

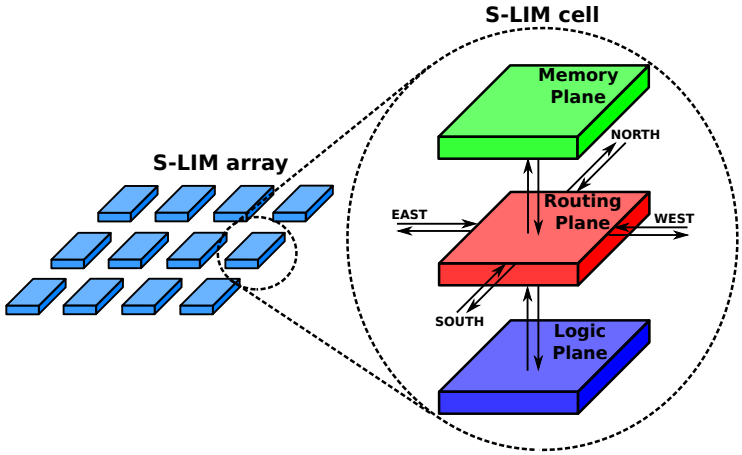


Figure 4: Schematic representation of the Systolic *Logic-In-Memory* (S-LIM).

difference with respect to Systolic Arrays is that every cell can be conceptually seen as a 3D structure, where a memory plane communicates with a logic plane and neighboring processing elements through a routing plane. It is important to underline that this 3D organization is just conceptual, the physical implementation can vary depending on the technology.

The LIM can be seen as a computing-enhanced memory in which data are not only stored but also computed. The routing plane shares some similarities with Networks-On-Chip [41][42]. Communication is only local, among neighboring cells. Inside the grid, each cell can work autonomously. Furthermore, the memory plane and the logic plane can be used independently. For example, if the memory plane of a cell is not used by the logic plane of its corresponding cell, it can be used by the logic plane of neighboring cells. The structure of the routing plane is fixed, while the logic plane and the memory size (inside the memory

plane) change accordingly to the algorithm. S-LIM operations are handled by the routing plane through messages. Each cell has four interconnection buses (north, south, east, west) to communicate with neighboring cells. For more information on the S-LIM architecture please refer to [39].

The conceived structure of the S-LIM offers several advantages in the execution of parallel algorithms, compared to systolic arrays and GPUs. (Adv. 1) Every cell can work autonomously and in parallel. (Adv. 2) Communication and synchronization are obtained using a fixed instruction set of operations. (Adv. 3) Memory and communications are local, hence, there is no bottleneck due to access times of global or shared memories. On the other hand, the S-LIM has three main limitations. (Limit. 1) The logic plane has not a fixed structure, but it must be designed each time depending on the executed algorithm. (Limit. 2) Each cell can communicate directly only with the four neighboring cells. If cells were allowed to exchange data also with non adjacent cells, the computational time could be reduced. (Limit. 3) The number of I/O pins depends on the array size: the larger, the higher the number of I/O pins required. Trying to address these problems has lead to the development of a new version of the *Logic-In-Memory* architecture, the Pyramidal *Logic-In-Memory* (P-LIM).

4. P-LIM

While the S-LIM has very good performance, it does not have an intelligent memory. During the design of the P-LIM our focus was exactly the transformation of the memory plane in an intelligent memory plane. The basic principle behind the P-LIM is what we defined as "memory pipelining". *In each cell, the logic plane is not aware of the data that it needs, it is, instead, the memory plane's duty to provide the logic with the correct data. The memory plane is therefore divided into different stages. The stage closer to the logic plane contains data that are needed immediately or shortly afterwards. Higher memory levels, farther from the logic plane, fetch data from neighboring cells or from the outside in advance, so that when the logic plane will need new data they will be*

already available, maximizing performance. The structure of the P-LIM is depicted in Figure 5. We have defined this mechanism as "3D memory pipelining"

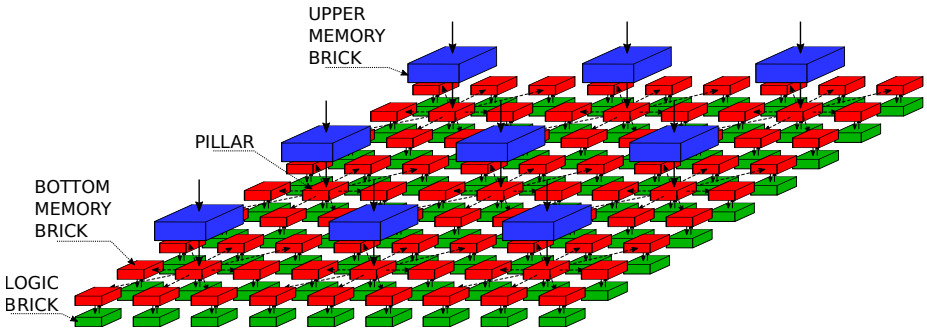


Figure 5: Architecture of the Pyramidal LIM. The number of units of the logic plane (green blocks) is the same as the number of memory cells (red blocks). The second layer instead, is composed of 9 memory elements (blue blocks).

[40] because it is similar, in principle, to the instruction pipeline of microprocessors. In microprocessors, instructions are fetched in advance, generally using predictive techniques, to keep the ALU always operational. Here, data are fetched in advance and stored in upper memory levels and then transferred to the bottom memory levels when needed. The pipeline has a 3D structure where different memory layers are stacked on a 2D array of functional units. It is important to underline that this is just a conceptual structure. The way in which it can be physically implemented is discussed in Section 5.

In the following, we define as "brick" every element of each plane. The logic plane is therefore made of "logic bricks" working in parallel, while memory planes are composed of "memory bricks". The lowest memory plane is intrinsically linked to the logic plane, so there is one memory brick for each logic brick. Upper memory layers are used only for data pipelining, so we chose to use a different mapping. In upper memory layers, there is a brick every 3×3 bricks of the bottom layer. This choice was made to simplify the whole memory structure, leading to a pyramidal memory organization, as depicted in Figure 5. In our analysis we worked with an array of 9×9 logic bricks, hence, only two memory layers are required. Increasing the number of logic bricks leads to an increase

in the number of memory layers. The main role of these "intelligent memories" is to continuously feed the logic with proper data, by fetching it in advance and masking access times to large external memories. In the P-LIM architecture, logic bricks work with data stored in the lowest memory plane, while higher memory planes fetch data in advance. For now, given that we know the exact behavior of the algorithm, the memory plane is deterministically programmed to automatically fetch data in advance. As a future development, we will implement predictive techniques for data prefetching.

It is important to underline that, while the "3D memory pipelining" concept shares some similarities with the caching mechanism, overall it is different. The caching mechanism is used to compensate the lack of a memory that is at the same time big and fast. A hierarchy of memories with increasing size and decreasing speed is used, in order to continuously fetch data to a processor, avoiding idle times. The aim of the "3D memory pipelining" concept is the same, but the implementation is different. Here the focus is on communication and intelligence. The memory becomes a smart and more complex entity that tries to fetch data wherever it is, being in a bigger and slower memory or in smaller memories of neighboring cells. In order to overcome one of the problems of the S-LIM (Limit. 3), the communication, in the P-LIM, is handled by the upper memory layer (the tip of the pyramid), limiting, therefore, the number of I/O pins. The P-LIM also addresses the other main limitation of the previous structure (Limit. 1): each logic brick is a programmable computing unit. Moreover, the behavior of each memory brick can be programmed. These choices make the structure fully programmable and, therefore, independent from the executed algorithm. Communication and data exchanges are also handled differently from S-LIM. Each logic brick can communicate only with its correspondent memory brick, but not with neighboring logic bricks. Memory bricks, instead, can exchange data also with neighboring memory bricks. In case of communication with upper or lower memory layers, only one brick can communicate. This particular brick, called pillar, is the central element of every 3x3 array of bricks (Figure 5). This choice was done to simplify the I/O reducing the number of

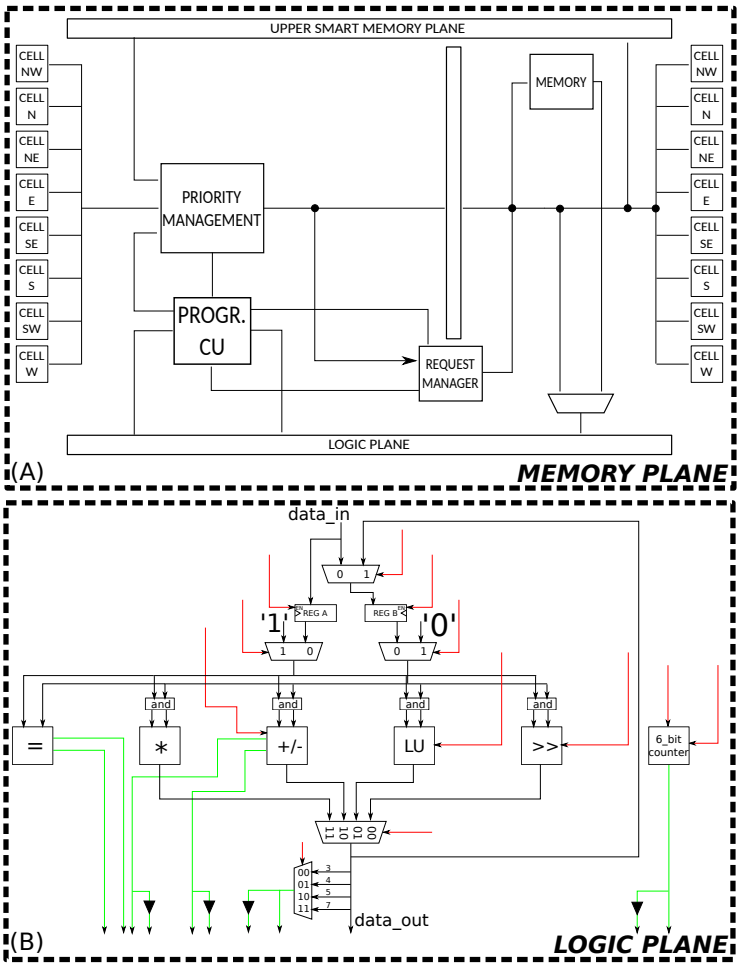


Figure 6: (A) Intelligent memory block diagram. (B) Logic brick datapath.

interconnections. Figure 6.A depicts the schematic of a memory brick. It incorporates features of the routing and memory planes of the S-LIM. It contains the *memory* used to store data, the *priority management* block used to handle conflicts in case of multiple requests and the *request management* block used to identify the operation that must be executed by a logic brick. The control unit is fully programmable. Moreover, each cell can communicate with all 8 surrounding bricks. Upper layer bricks are equal to bottom layer ones but with one important difference: the memory is split into two parts. The first one is

used to store and load data for horizontal communication, that is the exchange of requests among bricks of the same layer. The second memory portion is used to handle the vertical communication, that is the exchange of requests among bricks of different layers. Figure 6.B depicts the schematic of a logic brick. Each logic brick is an ALU which communicates only with its correspondent brick in the lowest memory layer. The memory brick provides memory to the logic element and defines the type of operation that needs to be executed. The logic element is composed of: a Logic Unit (LU), a comparator, an adder/subtractor, a multipler, a shifter and a counter. Multiplexers are used to control the data flow inside the logic brick. Several status bits are generated and sent to the correspondent memory brick. The instruction set of the P-LIM contains instructions used to exchange information among bricks, arithmetic and logic instructions and instructions for programming the control unit in each memory brick. In addition, each memory brick can be programmed independently, guaranteeing maximum flexibility.

Algorithm Mapping

Figure 7 shows how the odd-even sort algorithm is mapped on our LIM architecture. In the example reported, there is a 4×4 array, where each cell is intended as a memory-logic brick pair (cells are numbered from 1 to 16). As a first step (Figure 7.A) input data are loaded in the memory brick of each cell exploiting the 3D memory pipelining system described in section 4. Then, data must be ordered, so the actual computation starts as depicted in Figure 7.B. Suppose that data must be sorted in ascending order (cell 1 will have the smallest number and cell 16 the highest). During the first round, odd cells are in charge of the computation while even cells are in idle state. Odd cells read data from their neighboring even cells and compare them with their local data. If necessary the two data are swapped (as happens, for example, between cell 1 and cell 2, since $10 > 1$ and the sorting order is ascending). In the second computation round, odd cells are in idle while even cells perform comparisons (cells 1 and 16 are not used in this round). Comparison rounds are then iterated until all data

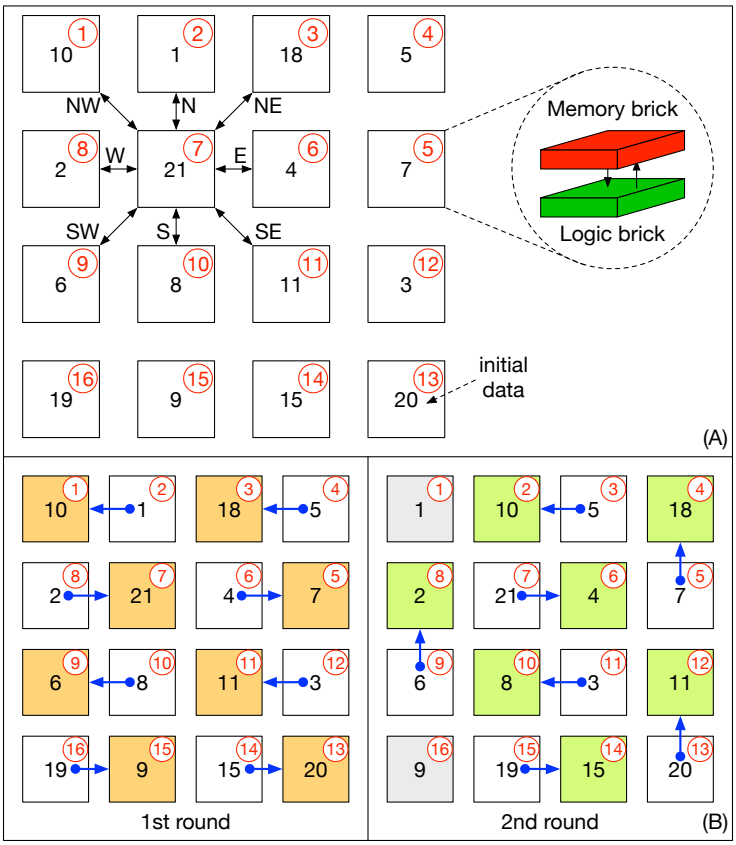


Figure 7: Odd-even sort algorithm mapping on the LIM array. Cell are numbered from 1 to 16 and each is intended as a memory-logic brick pair. (A) Data are initially loaded in the memory brick of each cell. Each cell has connections to all its neighboring cells. (B) Suppose that numbers must be sorted in ascending order (cell 1 lowest, cell 16 highest). First round: odd cells read data from their neighboring even cells, compare the two data and swap them if needed. Second round: now even cells perform the comparison. Rounds are iterated until data are sorted.

are sorted. Round after round, thanks to the flexibility of the interconnections, active cells (odd or even) use different connections (north/south/east/...) in order to make all the required comparisons.

This mapping methodology can be extended to any parallelizable algorithm (as long as the operations required are supported by the ALU of the logic plane): in fact, input data can be continuously distributed, by means of the memory

pipelining system, to the various logic bricks which process them in parallel. If required by the algorithm, intermediate data can be moved across the bricks for further processing.

5. Results

Figure 8 shows the flow followed to validate and extract data used for the comparison of P-LIM, S-LIM and the test ASIC. The architectures were described using VHDL and simulated with Modelsim [43]. We have measured, through the simulations, the number of clock cycles required by the two architectures (S-LIM and P-LIM) to execute the selected algorithms. Then, we have synthesized the architectures using Synopsys Design Compiler [44] on a STMicroelectronics 28nm CMOS technology and extracted frequency, area and power consumption values. The frequency and the number of clock cycles are used to compute the time required to run the considered algorithm for comparison purposes (Section 6). Figure 9 reports the comparisons between P-LIM and S-LIM in terms of number of clock cycles required to run the odd-even sort algorithm. Figure 9a reports the number of clock cycles required to sort

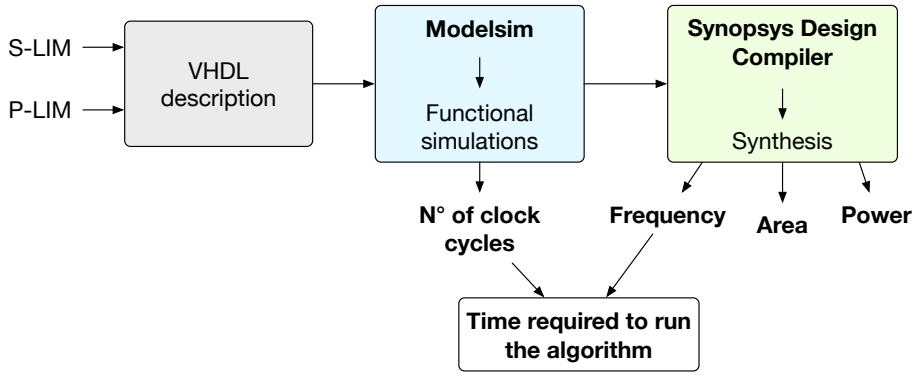
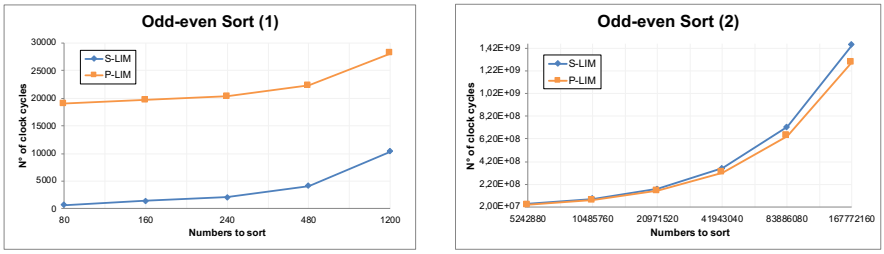


Figure 8: P-LIM, S-LIM and the test ASIC are described in VHDL, then functional simulations are run through Modelsim and the number of clock cycles that each architecture spend for executing the three algorithms is extracted. After that, the architectures are synthesized using Synopsys Design Compiler on a 28 nm technology. Maximum working frequency, area and power consumption values are extracted from synthesis results.



(a) Small quantities of numbers to sort.

(b) Large quantities of numbers to sort.

Figure 9: Number of clock cycles to execute the odd-even sort: comparison between S-LIM and P-LIM (considering a 9×9 array).

up to 1200 numbers. With up to 240 numbers to sort, the execution time is dominated by the loading time, that is the time required to load all numbers into the architecture. The difference in terms of clock cycles is quite high. With only 80 numbers to sort, the execution time for the S-LIM is around 1000 clock cycles, while the P-LIM requires nearly 20000 clock cycles. This difference can be explained easily. The P-LIM is a more complex architecture and the time required to program the circuit and to load the numbers is quite high. It is also a programmable structure while the S-LIM is tailored to the algorithm. However, when increasing the numbers to sort, as depicted in Figure 9b, the execution time of the S-LIM is higher then the P-LIM because of the overhead of data transfer that in the P-LIM is masked by the memory pipelining mechanism.

Figures 10 and 11 shows the performance comparison between S-LIM and P-LIM related to the integral image algorithm and the binomial filter, respectively. It can be clearly seen that, in both cases, as the number of input data grows, the P-LIM overtakes the S-LIM as the number of clock cycles required to execute the algorithms are lower. In particular, the higher the number of input pixels, the higher the difference in terms of clock cycles between the two architectures. This results can be explained by taking into account the nature of the algorithms considered: being data-intensive, they require a continuous stream of input data to compute, hence, part of the total number of clock cycles taken by

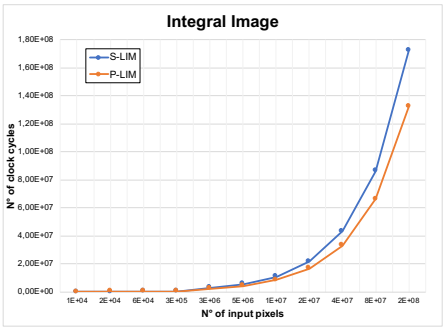


Figure 10: Integral image: performance comparison between S-LIM and P-LIM (9×9 array).

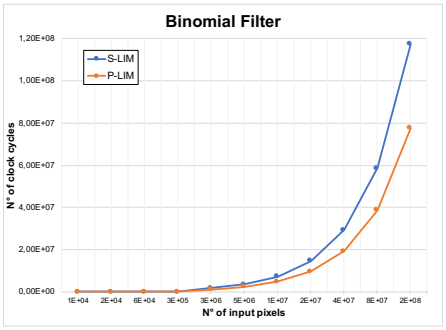


Figure 11: Binomial filter: performance comparison between S-LIM and P-LIM (9×9 array).

the execution of the algorithms are used for data transfer. The P-LIM takes advantage of its memory pipelining mechanism (as explained in Section 4) to fetch in advance the new required data and to mask the time required for the transfer. As a result, the overall number of clock cycles taken by the P-LIM is distinctly lower than the S-LIM.

Synthesis results are summarized in Table 1 (supposing an array of 9×9 logic cells). The three different S-LIM implementations refer to the three selected algorithms: 1 is for the odd-even sort, 2 is for the integral image and 3 for the binomial filter.

	S-LIM 1	**S-LIM 2**	**S-LIM 3**	**P-LIM**
Freq. [GHz]	1	1	1	1
Area $[mm^2]$	0.043	0.035	0.039	0.46
Power $[mW]$	55.3	51	52.7	330
Power density $[W/mm^2]$	1.3	1.5	1.3	0.7

Table 1: Synthesis results of 9x9 S-LIM and P-LIM arrays on a 28nm STMicroelectronics CMOS technology. S-LIM 1, 2 and 3 refer to synthesis results when considering a logic-plane tailored to the odd-even sort, the integral image and the binomial filter, respectively.

Both the S-LIM and the P-LIM are thought to be 3D structures, therefore,

they should be implemented using a 3D technology. However, since we do not have access to such a fabrication process, both architectures were synthesized on a standard planar CMOS process. As a consequence, the synthesis results obtained are far from optimal. Nonetheless, as will be discussed in Section 6, performance are pretty good also in this sub-optimal situation. The maximum frequency that can be achieved is, for both architectures, close to 1 GHz (in the integral image case the maximum working frequency of the S-LIM was slightly higher, but we have fixed it to 1 GHz in order to do a coherent comparison with the other cases). The area of the P-LIM is one order of magnitude bigger than the three S-LIM versions. This result is clearly justified by the higher complexity of the P-LIM which is fully programmable, while the S-LIM logic is tailored to a particular algorithm. Despite this difference, the P-LIM has a power consumption equal to 0.33 W which is, approximately, only 6 times as large as the S-LIM power consumption. For what concerns the power density, it is defined as the ratio between the total power dissipated by the system and the area occupation. The P-LIM has 10 times the area of the S-LIM but only 6 times the power consumption so, as a result, the overall power density of the P-LIM is lower.

The results reported in Table 1 cannot be used for comparing our architectures to the ones cited in Section 2 because these works differ from ours under various aspects, hence, the comparison would be unfair.

6. Comparison

The design and verification phase is followed by the validation phase, where the obtained results are compared to other kinds of digital architectures. The goal is to understand the quality and the value of the obtained results. In particular, we focused on two test architectures, an Application Specific Integrated Circuit (ASIC) and a GPU. ASIC were chosen because, generally, they makes it possible to reach maximum performance. GPUs are powerful parallel architectures with a complex memory hierarchy (part of it is locally embedded inside

the chip) that are heavily used as hardware accelerators. These architectures represent a right target to be used as a comparison for our P-LIM. Regarding the GPU, we have used data presented in [45], where authors have implemented the odd-even sort algorithm on an NVIDIA Quadro 6000. Unfortunately, we were not able to find GPU implementations of the integral image and the binomial filter. In addition, we designed three different versions of a custom ASIC (one for each selected application) and synthesized them on the same 28 nm technology used for the LIM. The ASIC is characterized by a conventional architecture in which a computing unit executes the sorting algorithm while fetching data from an external memory. Moreover, the test ASIC is partially parallel in order to guarantee a fair comparison with the P-LIM; in fact, it is composed of four computing units working in parallel and fetching data from four independent memories. The flow followed to design the ASIC, evaluate it and extract useful data is the same described in figure Figure 8. It is important to highlight that the test ASIC was synthesized without taking into account the memories. Instead, we assume that they are GDDR5 memories recently developed by Samsung [46]. The ASIC's working frequency obtained after synthesis is 5 GHz (without considering the memories), which is very high because of the simplicity of its architecture and it is also much higher than the working frequency of our LIM architectures. The Samsung GDDR5 memory can reach a theoretical frequency of 8 GHz that is higher than the maximum frequency at which the ASIC can work. When doing the comparison, we suppose that four GDDR5 modules, working at the maximum frequency, surround our custom circuit. Figure 12 shows the time required by the S-LIM, the P-LIM and the test ASIC to sort up to 12000 numbers. The time is given as number of clock cycles, therefore, it does not depend on the technology chosen and on the type of memory selected. The difference between the two LIM architectures and the ASIC is very high. With 12000 numbers, nearly $1.8 \cdot 10^8$ clock cycles are required by the ASIC, while the S-LIM and the P-LIM require around $1 \cdot 10^5$ clock cycles. The results of the test ASIC are obtained considering a state of the art memory that works in ideal conditions. Moreover, we are neglecting any memory access time, hence,

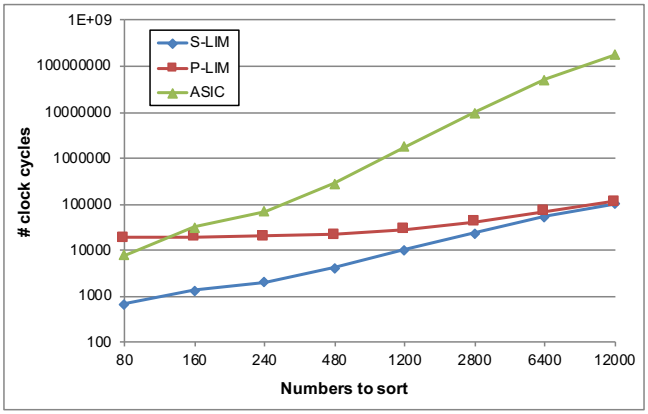


Figure 12: Comparison between S-LIM, P-LIM and ASIC implementation of the odd-even sort algorithm. Number of clock cycles axis is in logarithmic scale.

we are supposing that data are continuously sent to the circuit. Under these conditions, the ASIC works always at full speed. However, in real situations, this condition is not always met and the computing unit is slowed down by the memory. In order to take into account these assumptions, for the test ASIC we consider not only the GDDR5 memory working at 8 GHz, but also two other memory models: a GDDR5 working at 3 GHz, and a DDR3 working at 1.6 GHz. The comparison in terms of timing required to sort 2^{15} numbers is reported in Table 2. We consider three different scenarios for what concerns the ASIC, by taking into account the three different memory models cited above. In the first scenario, the working frequency of the system composed of ASIC and memory is limited by the first one (which reaches a maximum working frequency of 5 GHz). In this case the execution time needed to sort 2^{15} numbers is 268 ms. In the other two scenarios, instead, the system working frequency is limited by the the memory as these models reach a maximum frequency that is lower that the ASIC's one. The execution time grows to 447 ms and, with a DDR3 memory it further worsens, being 838 ms. As expected, the computing speed of the circuit is limited by the memory, as it commonly happens in many modern computational systems. Both the S-LIM and P-LIM architectures are way faster than the ASIC, even in a scenario where the ASIC working frequency is very high (5

Odd-Even Sort	# clock cycles	Freq. [GHz]	Time [ms]	Speed-up w.r.t. GPU
S-LIM	$2.8 \cdot 10^5$	1.0	0.28	821
P-LIM	$2.6 \cdot 10^5$	1.0	0.26	884
ASIC (GDDR5 8 GHz)	$1.3 \cdot 10^9$	5.0	268	< 1
ASIC (GDDR5 3 GHz)	$1.3 \cdot 10^9$	3.0	447	< 1
ASIC (DDR3 1.6 GHz)	$1.3 \cdot 10^9$	1.6	838	< 1
Quadro 6000	–	1.1	230	–

Table 2: Performance comparison among the two versions of LIM architectures, the test ASIC with three types of memory and the NVIDIA Quadro 6000 GPU. These results refer to a case where it is required to sort 2^{15} numbers.

GHz). Furthermore, it is very important to underline that while the test ASIC makes four memory accesses at each clock cycle, LIM architectures uses local data stored in the memory plane and access to an external memory is required only when all local data have been processed.

For what concerns the GPU, authors in [45] uses a NVIDIA Quadro 6000 based on a 40 nm technology. The processor clock is fixed at 1148 MHz and it has 6 Gb GDDR5 memory (the same model as the GDDR5 working at 3 GHz considered for the test ASIC). The GPU needs 230 ms to sort all numbers. The difference with both LIM versions is remarkable since the S-LIM needs only 0.28 ms and the P-LIM 0.26 ms to sort 2^{15} numbers. With a difference of three orders of magnitude, LIM architectures are more than 800 times faster than the GPU. The comparison between the P-LIM and the GPU is particularly relevant because they are both fully programmable and parallel architectures. The P-LIM is faster and it consumes only 0.3 W against the NVIDIA Quadro 6000 that, at full load, consumes 204 W. The benefits provided by our *Logic-In-Memory* architecture are much more evident when compared to these realistic cases. These results also demonstrate that the P-LIM is faster than the S-LIM with millions of numbers to sort, in addition to being characterized by a significantly lower

power density as shown in section 5.

Tables 3 and 4 show the performance comparison related to the integral image and the binomial filter, respectively, when the number of input data is very high (more than 2 million). For what concerns the ASIC implementations, we have fixed the target frequency to 5 GHz to make the results coherent for all the analyzed algorithms. It can be noticed that the P-LIM outperforms the

Integral Image	# clock cycles	Freq. [GHz]	Time [s]
S-LIM	$2.7 \cdot 10^6$	1.0	$2.7{\cdot}10^{-3}$
P-LIM	$2.1 \cdot 10^6$	1.0	$2.1{\cdot}10^{-3}$
ASIC (GDDR5 8 GHz)	$1 \cdot 10^{13}$	5.0	2113
ASIC (GDDR5 3 GHz)	$1 \cdot 10^{13}$	3.0	3522
ASIC (DDR3 1.6 GHz)	$1 \cdot 10^{13}$	1.6	6604

Table 3: Performance comparison among the two versions of LIM architectures and the test ASIC with three types of memory. These results refer to the integral image application where the number of input pixels is more than 2 million.

Binomial Filter	# clock cycles	Freq. [GHz]	Time [s]
S-LIM	$1.8 \cdot 10^6$	1.0	$1.8{\cdot}10^{-3}$
P-LIM	$1.2 \cdot 10^6$	1.0	$1.2{\cdot}10^{-3}$
ASIC (GDDR5 8 GHz)	$3.5 \cdot 10^{12}$	5.0	704
ASIC (GDDR5 3 GHz)	$3.5 \cdot 10^{12}$	3.0	1174
ASIC (DDR3 1.6 GHz)	$3.5 \cdot 10^{12}$	1.6	2201

Table 4: Performance comparison among the two versions of LIM architectures and the test ASIC with three types of memory. These results refer to the binomial filter application where the number of input pixels is more than 2 million.

S-LIM and all the ASIC circuits in both the applications because, as already

said before, it takes advantage of the memory pipelining mechanism to mask the latency due to data transfer. Furthermore, it is fundamental to highlight that the three applications selected leverage on the local interaction among the bricks to exchange data and reuse them as required by the algorithm. On the contrary, in all the ASIC implementations this data exchange/reuse is done by means of the memories and, as a consequence, performance are worse as a lot of time is spent just for memory accesses.

7. Conclusions

In this paper we presented an innovative architecture, called Pyramidal LIM, that implements the *Logic-In-Memory* principle. It is an evolution of the S-LIM which has good performance but it has a rather high power density. The proposed P-LIM is faster than the S-LIM, it is fully programmable (hence, it can be used as a hardware accelerator for many algorithms) and it is characterized by a lower power density.

We synthesized the two architectures on a 28nm CMOS technology and we compared the obtained performance with a test ASIC and a commercial GPU. The resulting performance gain is noticeable both in terms of speed and power consumption. As a future development, we will continue to work on the LIM concept to further enhance the architecture and to validate it by using a wider class of applications. We are also planning to increase the number of I/O connections of the P-LIM to improve performance by reducing loading times and to improve the 3D memory pipelining mechanism to make it more smart and efficient.

References

[1] M. D. Godfrey, D. F. Hendry, The Computer As Von Neumann Planned It, IEEE Ann. Hist. Comput. 15 (1) (1993) 11–21. doi:10.1109/85.194088.

[2] N. Haron, S. Hamdioui, Why is CMOS scaling coming to an END?, in:

Design and Test Workshop, 2008. IDT 2008. 3rd International, 2008, pp. 98–103. doi:10.1109/IDT.2008.4802475.

[3] J. Kawa, C. Chiang, R. Camposano, EDA Challenges in Nano-scale Technology, in: IEEE Custom Integrated Circuits Conference 2006, 2006, pp. 845–851. doi:10.1109/CICC.2006.320844.

[4] M. A. Maddah-Ali, U. Niesen, Fundamental Limits of Caching, IEEE Transactions on Information Theory 60 (5) (2014) 2856–2867. doi:10.1109/TIT.2014.2306938.

[5] J. Draper, J. Chame, M. Hall, C. Steele, T. Barrett, J. LaCoss, J. Granacki, J. Shin, C. Chen, C. W. Kang, I. Kim, G. Daglikoca, The Architecture of the DIVA Processing-in-memory Chip, in: Proceedings of the 16th International Conference on Supercomputing, ICS '02, ACM, New York, NY, USA, 2002, pp. 14–25. doi:10.1145/514191.514197.

[6] Q. Zhu, B. Akin, H. E. Sumbul, F. Sadi, J. C. Hoe, L. Pileggi, F. Franchetti, A 3D-Stacked Logic-in-Memory Accelerator for Application-Specific Data Intensive Computing, in: 2013 IEEE International 3D Systems Integration Conference (3DIC), 2013, pp. 1–7. doi:10.1109/3DIC.2013.6702348.

[7] J. Ahn, S. Hong, S. Yoo, O. Mutlu, K. Choi, A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing, in: 2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA), 2015, pp. 105–117. doi:10.1145/2749469.2750386.

[8] D. Pala, G. Causapruno, M. Vacca, F. Riente, G. Turvani, M. Graziano, M. Zamboni, Logic-in-Memory Architecture Made Real, in: 2015 IEEE International Symposium on Circuits and Systems (ISCAS), 2015, pp. 1542–1545. doi:10.1109/ISCAS.2015.7168940.

[9] N. D. Shah, Y. H. Shah, H. Modi, Comprehensive Study of the Features, Execution Steps and Microarchitecture of the Superscalar Processors, in:

2013 IEEE International Conference on Computational Intelligence and Computing Research, 2013, pp. 1–4. doi:10.1109/ICCIC.2013.6724137.

[10] X.-H. Sun, Remove the Memory Wall: From performance modeling to architecture optimization, in: Proceedings 20th IEEE International Parallel Distributed Processing Symposium, 2006, pp. 2 pp.–. doi:10.1109/IPDPS.2006.1639621.

[11] P. Jacob, A. Zia, O. Erdogan, P. M. Belemjian, J. Kim, M. Chu, R. P. Kraft, J. F. McDonald, K. Bernstein, Mitigating Memory Wall Effects in High-Clock-Rate and Multicore CMOS 3-D Processor Memory Stacks, Proceedings of the IEEE 97 (1) (2009) 108–122. doi:10.1109/JPROC.2008.2007472.

[12] C. C. Liu, I. Ganusov, M. Burtscher, S. Tiwari, Bridging the Processor-Memory Performance Gap with 3D IC Technology, IEEE Design Test of Computers 22 (6) (2005) 556–564. doi:10.1109/MDT.2005.134.

[13] H. Kung, C. Leiserson, C.-M. U. P. P. D. of COMPUTER SCIENCE., C.-M. U. C. S. Department, Systolic Arrays for (VLSI), CMU-CS, Carnegie-Mellon University, Department of Computer Science, 1978.
URL https://books.google.co.uk/books?id=pAKfHAAACAAJ

[14] S.-Y. Kung, Arun, Gal-Ezer, B. Rao, Wavefront Array Processor: Language, Architecture, and Applications, IEEE Transactions on Computers C-31 (11) (1982) 1054–1066. doi:10.1109/TC.1982.1675922.

[15] G. Causapruno, F. Riente, G. Turvani, M. Vacca, M. R. Roch, M. Zamboni, M. Graziano, Reconfigurable Systolic Array: From Architecture to Physical Design for NML, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 24 (11) (2016) 3208–3217. doi:10.1109/TVLSI.2016.2547422.

[16] W. Jin, C. N. Zhang, H. Li, Mapping multiple algorithms into a reconfigurable systolic array, in: 2008 Canadian Conference on Electrical and

Computer Engineering, 2008, pp. 001187–001192. doi:10.1109/CCECE.2008.4564726.

[17] L.-W. Chang, M.-C. Wu, A unified systolic array for discrete cosine and sine transforms, Signal Processing, IEEE Transactions on 39 (1) (1991) 192–194.

[18] H. Lim, E. E. Swartzlander, Multidimensional systolic arrays for multidimensional DFTs, in: 1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings, Vol. 6, 1996, pp. 3276–3279 vol. 6. doi:10.1109/ICASSP.1996.550576.

[19] H. Herzberg, R. Haimi-Cohen, A Systolic Array Realization of an LMS Adaptive Filter and the Effects of Delayed Adaptation, IEEE Transactions on Signal Processing 40 (11) (1992) 2799–2803. doi:10.1109/78.165667.

[20] D. Luebke, G. Humphreys, How GPUs Work, Computer 40 (2) (2007) 96–100. doi:10.1109/MC.2007.59.

[21] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, J. C. Phillips, GPU Computing, Proceedings of the IEEE 96 (5) (2008) 879–899. doi:10.1109/JPROC.2008.917757.

[22] G. Chen, G. Li, S. Pei, B. Wu, High Performance Computing via a GPU, in: 2009 First International Conference on Information Science and Engineering, 2009, pp. 238–241. doi:10.1109/ICISE.2009.634.

[23] B. Buyukkurt, W. A. Najj, Compiler generated systolic arrays for wavefront algorithm acceleration on FPGAs, in: 2008 International Conference on Field Programmable Logic and Applications, 2008, pp. 655–658. doi:10.1109/FPL.2008.4630032.

[24] K. M. Chandy, J. Misra, Systolic algorithms as programs, Distributed Computing 1 (3) (1986) 177–183. doi:10.1007/BF01661171.

[25] H. R. Myler, A. R. Weeks, The Pocket Handbook of Image Processing Algorithms in C, 1st Edition, Prentice Hall Press, Upper Saddle River, NJ, USA, 2009.

[26] D. H. Kim, K. Athikulwongse, M. B. Healy, M. M. Hossain, M. Jung, I. Khorosh, G. Kumar, Y. Lee, D. L. Lewis, T. Lin, C. Liu, S. Panth, M. Pathak, M. Ren, G. Shen, T. Song, D. H. Woo, X. Zhao, J. Kim, H. Choi, G. H. Loh, H. S. Lee, S. K. Lim, Design and Analysis of 3D-MAPS (3D Massively Parallel Processor with Stacked Memory), IEEE Transactions on Computers 64 (1) (2015) 112–125. doi:10.1109/TC.2013.192.

[27] D. Zhang, N. Jayasena, A. Lyashevsky, J. L. Greathouse, L. Xu, M. Ignatowski, TOP-PIM: Throughput-oriented Programmable Processing in Memory, in: Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing, HPDC '14, ACM, New York, NY, USA, 2014, pp. 85–98. doi:10.1145/2600212.2600213.

[28] Hybrid Memory Cube, http://www.hybridmemorycube.org.

[29] Y. Tang, Y. Wang, H. Li, X. Li, ApproxPIM: Exploiting Realistic 3D-stacked DRAM for Energy-Efficient Processing In-Memory, in: 2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC), 2017, pp. 396–401. doi:10.1109/ASPDAC.2017.7858355.

[30] G. Dai, T. Huang, Y. Chi, J. Zhao, G. Sun, Y. Liu, Y. Wang, Y. Xie, H. Yang, GraphH: A Processing-in-Memory Architecture for Large-scale Graph Processing, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (2018) 1–1doi:10.1109/TCAD.2018.2821565.

[31] L. Jiang, M. Kim, W. Wen, D. Wang, XNOR-POP: A processing-in-memory architecture for binary Convolutional Neural Networks in Wide-IO2 DRAMs, in: 2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), 2017, pp. 1–6. doi:10.1109/ISLPED.2017.8009163.

[32] S. Li, C. Xu, Q. Zou, J. Zhao, Y. Lu, Y. Xie, 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), 2016, pp. 1–6.

[33] S. Angizi, Z. He, D. Fan, 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), 2018, pp. 1–6. doi:10.1109/DAC.2018.8465706.

[34] M. Imani, T. Rosing, CAP: Configurable resistive associative processor for near-data computing, in: 2017 18th International Symposium on Quality Electronic Design (ISQED), 2017, pp. 346–352.

[35] M. Imani, S. Gupta, A. Arredondo, T. Rosing, Efficient query processing in crossbar memory, in: 2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), 2017, pp. 1–6.

[36] F. S. et al., A 462GOPs/J RRAM-based nonvolatile intelligent processor for energy harvesting IoE system featuring nonvolatile logics and processing-in-memory, in: 2017 Symposium on VLSI Circuits, 2017, pp. C260–C261.

[37] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, T. C. Mowry, Ambit: In-memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology, in: Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-50 '17, 2017, pp. 273–287.

[38] H. Jarollahi, N. Onizawa, V. Gripon, N. Sakimura, T. Sugibayashi, T. Endoh, H. Ohno, T. Hanyu, W. J. Gross, A Nonvolatile Associative Memory-Based Context-Driven Search Engine Using 90 nm CMOS/MTJ-Hybrid Logic-in-Memory Architecture, IEEE Journal on Emerging and Selected Topics in Circuits and Systems 4 (4) (2014) 460–474.

[39] M. Cofano, G. Santoro, M. Vacca, D. Pala, G. Causapruno, F. Cairo, F. Riente, G. Turvani, M. R. Roch, M. Graziano, M. Zamboni, Logic-in-Memory: A Nano Magnet Logic Implementation, in: 2015 IEEE Computer Society Annual Symposium on VLSI, 2015, pp. 286–291. doi:10.1109/ISVLSI.2015.121.

[40] G. Causapruno, Architectural Solutions for NanoMagnet Logic, Ph.D. thesis, Politecnico di Torino (2016).

[41] A. Y. Weldezion, Z. Lu, R. Weerasekera, H. Tenhunen, 3-D Memory Organization and Performance Analysis for Multi-Processor Network-on-Chip Architecture, in: 2009 IEEE International Conference on 3D System Integration, 2009, pp. 1–7. doi:10.1109/3DIC.2009.5306593.

[42] M. Martina, G. Masera, Turbo NOC: A Framework for the Design of Network-on-Chip-Based Turbo Decoder Architectures, IEEE Transactions on Circuits and Systems I: Regular Papers 57 (10) (2010) 2776–2789. doi:10.1109/TCSI.2010.2046257.

[43] Mentor Graphics, http://www.modelsim.com.

[44] http://www.synopsys.com/.

[45] F. G. Khan, O. U. Khan, B. Montrucchio, P. Giaccone, Analysis of Fast Parallel Sorting Algorithms for GPU Architectures', in: 2011 Frontiers of Information Technology, 2011, pp. 173–178. doi:10.1109/FIT.2011.39.

[46] http://www.samsung.com/semiconductor/products/dram/graphic-dram/.