Proceedings of the 8th World Congress on Electrical Engineering and Computer Systems and Sciences (EECSS'22)
Prague, Czech Republic - July 28- 30, 2022
Paper No. CIST 123
DOI: 10.11159/cist22.123

A Recursive Hierarchy for Accelerator-Level Parallelism

Mihaela Malița[1], Gheorghe M. Ștefan[2]
[1]Amherst College
Amherst, MA 01002, USA
mmalita@amherst.edu;
[2]Politehnica University of Bucharest
Splaiul Independenței 313, București, RO
gheorghe.stefan@upb.ro

Abstract **-** The emergence, under the pressure of the ASICs imposed by the corporate space, of the field of Accelerator-Level Parallelism (ALP) requires a theoretical analysis to avoid the slippages that have characterized the evolutions of the last decades in the field of parallel computing. *Ad hoc* solutions imposed under time-to-market pressure have distorted the evolution of the field of parallel computing. The opportunity offered by the ALP challenge must be used to make last minute corrections in the chaotic evolution of the development of the parallel computing domain. The solution we propose is an attempt to reconsider parallelism from a double perspective. A purely theoretical one based on a mathematical model, that of the partially recursive functions proposed by Stephan Kleene, and another that emerges under the pressure of the increasingly complex applications demanded by the IT market. Our proposal consists in the hierarchical recursive structuring of ALP starting from the abstract MapScanReduce model that we have already proposed for the parallel computing.

Keywords**:** accelerator-level parallelism, heterogenous computing, parallel computing, accelerators, abstract model for parallelism.

1. Introduction

Heterogeneous computing is a consequence of the need to optimize, in terms of silicon area and energy consumed, SoCs running increasingly complex applications that require more and more intense computing. The complexity and intensity of the computation suppose a *heterogeneous computing system* in which the complex computation, executed on a host, can be segregated from the intense one, executed on one or more accelerators.

A lot of examples can be given in which the area of a SoC is dominated by accelerators that operate under the control of a host implemented through a mono- or multi-core structure. Accelerators are made in several ways. Some are dedicated circuits (for example, supporting various encoding or decoding operations) while others are programmable many-cell structures (as GPUs). An example is the M1 chip, made by Apple, in which a multi-core host is supported by a series of accelerators. Some are specially dedicated for functions implemented directly in hardware and others are multicellular programmable structures (a GPU and a Neural Engine).

The growing number of accelerators in the economy of SoCs has recently necessitated the emergence of a new concept: Accelerator-Level Parallelism (ALP). Any accelerator can be a parallel structure, but now we are faced with the parallel operation of several accelerators on the same silicon chip. It would be great if the mechanisms for structuring these accelerators, in a SoC or a network of SoCs, would be subject to a well-founded theory. We are warned in [2]:

"We assert there is as yet no "science" for debating and systematically answering basic questions for how to best facilitate broad, flexible, and effective use of multiple accelerators." (p. 36)

In [1-3] the authors invite us to contribute to the development of an appropriate theoretical environment for using the ALP concept in improving the efficiency of heterogenous computing for one- or multi-chip applications. M. D. Hill and V. J. Reddi emphasize the main problem with accelerators: the limitation due to the *von Neumann Bottleneck*. Indeed, it is easy to think of organizing a many-core system, but it is very difficult to satisfy the "hunger" of data that such a system has for certain applications, "hunger" that can only be satisfied with energy costs hard to accept. Also, the interconnection of large numbers of cells poses very difficult and costly structural organization problems in terms of area and energy.

The Gables model proposed by M. D. Hill and V. J. Reddi is a good start in developing the conceptual environment in which ALP could develop. The contribution we want to make to this concept is related to the organization of a heterogeneous

system that uses an ALP accelerator. If the host is a general-purpose system, then can't we hope that the ALP accelerator can also be configured as a general accelerator structure? Yes, we can.

In [3] are identified three major problems related to the ALP approach:

1. the computational performance of each accelerator which is strongly related with the internal memory hierarchy and the internal interconnections between cells
2. the data movement through the bottleneck between internally distributed memory in cells and the external memory
3. the bottleneck introduced by the overhead due to the communications between the accelerators.

All these three problems are addressed in this paper. In fact, it proposes a general framework in which the three issues are considered. There are *pros* and *cons* to a general solution for configuring and programming an ALP accelerator. For a start, very briefly, we can say that:

- it is worth promoting a general solution for ALP because the resulting structure will be simpler and therefore easier to design, validate, implement, test and use
- we may be reluctant to promote a general solution due to the functional diversity of the accelerators, which will sometimes prevent us from achieving maximum performance.

However, no one prevents the application of a limited general solution that can be supplemented at any time with solutions dedicated to more special functionalities.

We can maximize the chances of success of a general solution for a network of accelerators operating in parallel if we approach parallelism from a very well-founded point of view theoretically. Unfortunately, we cannot use the theoretical approaches used for parallel computation because they do not rely on abstract models derived from a mathematical model. We have described in [4] the reasons why the current solutions for parallel computing derive from *ad hoc* approaches imposed on the market by the corporate environment and not by the academic community.

In the next section we will present a hierarchical recursive model that we propose as a theoretical framework for the development of a science of ALP systems. The third section shows how to apply the recursive abstract model we proposed to substantiate the ALP approach. A final remarks section will conclude our contribution.

2. Abstract Recursive Model for Parallel Computing

When the first symmetrical MIMD engine is introduced on the computer market by Burroughs Corporation in 1962, it did so by connecting four computing systems together without any theoretical basis and having no specific solution for programming an application that could run in parallel on more than one processor. Starting form 1965, Edsger W. Dijkstra formulates [5] the first architectural concerns about specific parallel programming issues (critical region problem, semaphores, the dining philosophers problem, guarded commands). In 1974-82 abstract machine model proposals (confused with mathematical models) start to come in [6-8] after almost two decades of non-systematic experiments (started in the late 1950s) and the too early market production. No one yet really considered as mandatory a mathematical parallel computation model, although it was there waiting to be considered (see Kleene's mathematical model for computation [9]).

Under these conditions, we should not be surprised if the accelerators used in most applications achieve performance that is far from their peak performance. This shortcoming is not only due to the *von Neumann Bottleneck*; organizational and architectural mismatches of their internal structures are also to be considered. *Ad hoc* organized structures, or structures optimized for specific areas cannot be used effectively as general-purpose accelerators. Our proposal for the structure and architecture of ALP accelerators starts from the abstract model we proposed for a parallel computing system. This abstract model was configured [4], based on the partially recursive function model proposed by Stephen Kleene [9], and developed in [10-12].

2.1. The Abstract Model

The theoretical basis for ALP must start from an abstract model of parallelism because the acceleration of intense computation can only be achieved in a parallel computing system. The abstract parallelism model we have proposed is illustrated in Figure 1, where a recursively organized structure is presented. The recursion is given by the fact that eng_i is a structure of the same type as eng_{i+1} for $i = 1, 2, ...$, while eng_0 consists of a simple accumulator-based execution unit working

on the content of a register file, mem_0, of sufficiently large size (for example, of *4KB* in an FPGA implementation). Therefore, $MapScanRed_i(p(...))$ represents an *i*-th level in the recursive hierarchy, for $i = 1, 2, ...i$, with a *p*-cell MAP array where each cell is a MapScanReduce structure with (…) number of cells. For example: $MapScanRed_3(4(16(256)))$ stands for a hierarchy having on top 4 cells, each containing 16 cells of 256 elementary cells (eng_0 & mem_0) each. For $MapScanRed_1(256)$, REDUCE and SCAN networks are (pipelined) circuits, while for the other levels the actual structure can be designed using simple processors.

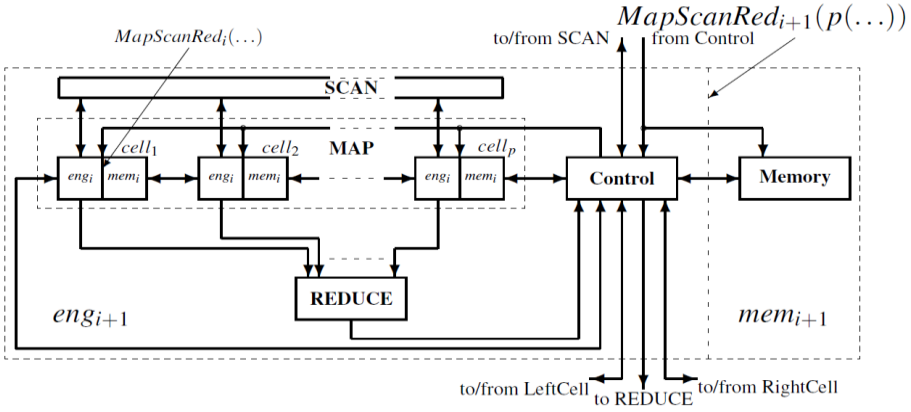

Fig. 1: Recursive abstract model for parallel computation [11].

The recursive organization we propose produces a hierarchically organized structure that can be developed at the level of a silicon chip, but can be extended on boards, racks, cabinets to the level of data centres and even beyond, to the level of global networks.

The architecture of the lowest level in the proposed hierarchy is defined by the data structure deployed in the local memories mem_0 in the MAP array, the instructions executed in each cell by eng_0, and the functions performed in the *log*-depth networks REDUCE and SCAN (see [13]). Shortly, the architecture can be defined as follows:

- data structure in the MAP array is the matrix:

$$\boldsymbol{M} = \begin{bmatrix} s_{11} & \cdots & s_{1p} \\ \vdots & \ddots & \vdots \\ s_{m1} & \cdots & s_{mp} \end{bmatrix}$$

where: vertical vectors $VV_i = [s_{1i}\ s_{1i}\ ...\ s_{mi}]$, for $i = 1, 2, ..., p$, represent the content of the local memory mem_0 in the $cell_i$, while the horizontal vectors $HV_j = [s_{1j}\ s_{1j}\ ...\ s_{mj}]$, for $j = 1, 2, ..., m$, are vectors distributed along the MAP array

- the Boolean vector: $B = [b_1\ b_2\ ...\ b_p]$ used to control the activity of each cell; **if** $b_i=1$ **then** $cell_i$ is active, **else** $cell_i$ is inactive
- in each clock cycle, from the program memory of Control are fetched two instructions, one for Control and another issued to the MAP array to be executed in each active cell; the arithmetic and logic instructions are similar in Control and in the cells of the MAP array, while the control instructions are specific for the MAP array as follows:
  - ACTIVATE: $b_i = 1$, for $i = 1, 2, ..., p$, thus activating all the cells in MAP
  - WHERE(cond): $b_i = (\ b_i\ \&\ cond_i)\ ?\ 1:0$, only the active cells where the condition *cond* is fulfilled remains active
  - ELSEWHERE: only the cells inactivated by the previous WHERE switch in the active state
  - ENDWHERE: the vector *B* switches back to the state before the previous WHERE instruction

  providing a space control in the MAP array
- the REDUCE network performs only few functions among which we list the following:
  - adds the active components of the horizontal vector provided by the MAP array

- selects the maximum value from the active components of the horizontal vector provided by the MAP array
- selects the minimum value from the active components of the horizontal vector provided by the MAP array

- the SCAN network performs, for example, permute, prefix sum, identifies the first occurrence of 1 in the *B* vector
- data transfer between the distribute memory in MAP and Memory is done transparently to the computation performed in MapScanReduce system, thus diminishing the *von Neumann Bottleneck* effect.

Versions of $MapScanRed_1(1024)$ where implemented in silicon (the last one in 2008 using *65 nm* standard process), or in FPGA technology [15].

2.2. Programming the Recursive Hierarchy of Parallel Structures Used as Accelerators

Parallel accelerator programming requires a specific approach. If, in a heterogeneous computing system, host programming involves high-level languages for which high-performance compilers are developed that are also based on the use of optimized function libraries, then for parallel accelerators it is preferable to optimize general purpose function libraries or libraries focused on specific application areas. We quote in this sense from [17]:

> *"The software span connecting applications to hardware relies more on parallel software architectures than on parallel programming languages. Instead of traditional optimizing compilers, we depend on autotuners, using a combination of empirical search and performance modelling to create highly optimized libraries tailored to specific machines."*

Each level *i* in the hierarchy can be conceived as benefiting from a kernel library of functions implemented in the hardware represented by the immediately lower level, *i-1*. The kernel library solves functions on data structures limited by the size of locally distributed mem_{i-1} memories. At level *i*, the function is implemented on data structures allowed by the size, usually larger, of the mem_i type memories. Thus, only at $MapScanRed_1\ (p)$ level, programming in the assembler will be required to develop a kernel library (advanced optimization of software libraries for current CPUs is also done in assembly languages). At any level higher than the basic one, the programming will be able to be done in a higher-level language for which we have high-performance compilers.

2.3. A Case Study: Matrix Multiplication

2.3.1 Acceleration compared with a mono-core solution

Let be a $MapScanReduce_1(p)$ accelerator. The integer multiplication (including the transfer of data) of two square matrices of $p \times p$ elements is performed in a number of clock cycles (the evaluation is based on a program written in assembly language running on a FPGA implemented version of the $MapScanReduce_1(p)$ accelerator) equal to:

$$T_{matrixMultiply} + T_{dataTransfer} = ((2p^2 + plog_2\, p + 9p + 5) + (p^2 + p))\ clockCycles$$

while the same computation is performed by a program written in C++ for a x86 mono-core architecture is performed in

$$T_{MMC++} \simeq 22p^3\ clockCycles$$

Results an acceleration, considering the two engines operating at the same frequency, of

$$\alpha \simeq 22p^3/((2p^2 + plog_2\, p + 9p + 5) + (p^2 + p))$$

The acceleration is super-linear and converges to 22p/3 for big p. The super-linear acceleration is due to the fact that in the parallel accelerator we propose, in addition to the disappearance of the third loop assumed by the algorithm (which would justify an acceleration of p times), the following processes take place in parallel:

- the control process supposed by the algorithm, in Control
- multiplication in MAP cells
- summation in the REDUCE network (its latency is avoided using a special feature added in the MAP array).

2.3.2. Energy & Area Compared with an Off-the-Shelf Solution

The same operation must be evaluated in comparison with an off-the-shelf many-core solution. Using data from [16] and our evaluation based on simulation and synthesis in the Cadence environment for our accelerator, in Table 1 is shown the comparison of a GPU with the MapScanRed accelerator in performing 1024×1024 array multiplication.

Table 1: Comparison of a GPU and MapScanRed accelerator in performing 1024×1024 matrix multiplication.

	GeForce GTX 850M	***MapScanReduce$_1$(512)***	**GeForce vs. MapScanRed**
Chip area	148 mm^2	40 mm^2	**3×** less for MapScanRed
Number of execution units	640	512	in the same range
Energy for 1024×1024 matrix multiplication	450 mJ	18.4 mJ	**24×** less for MapScanRed
Technology node	28 nm	28 nm	the same
Clock frequency	1.046 GHz	1 GHz	similar
Chip bandwidth	80 GB/sec	20 GB/sec	4× less for MapScanRed
Execution time for matrix multiplication	2 ms	2.3 ms	similar

3. Heterogenous Systems with ALP Co-Processor

The recursive definition we propose for a parallel accelerator can theoretically support ALP applications from a theoretical point of view. Instead of an unstructured organization of many accelerators on one SoC, we offer a hierarchical structure that can be more easily and efficiently controlled by a CPU as a host.

The main problem that limits the performance of a parallel structure is the limitation that occurs due to the *von Neuman* Bottleneck. To this are added the problems raised by the communication between the cells that work in parallel.

A detailed analysis of both issues can be found in [3]. Three structural configurations are examined (see figures 5, 10 and 11 of the cited paper) to evaluate the efficiency of an ALP type system. In the first, the DRAM memory, external to the SoC, is the one that supplies data to the IPs (the CPU and the accelerators). Also, communication between IPs is done through this external memory. In the second configuration, the SoC contains an internal buffer, in the form of an SRAM that attenuates the effect introduced by too frequent communication with the external DRAM memory. At the same time, communication between IPs is accelerated by the use of this SRAM. The third solution introduces a bus hierarchy that facilitates concurrent transfers in the system. From this analysis it can be seen that a hierarchical tree distribution allows the attenuation of communication problems in an ALP system both with the data source external to the SoC and between the IPs deployed on the chip.

We consider that the model of an ALP system, with a tree distribution network that is endowed in its leaves with accelerating IPs and a CPU, can have as an alternative a hierarchical tree network of accelerating IPs whose root is a CPU. One could thus conceive an alternative solution in the form of a heterogeneous computing system having as host the CPU and as accelerator a hierarchically organized ALP system.

We propose, as a promising candidate for the previously prefigured heterogeneous system, a system based on the recursive abstract model described in the previous subsection. Figure 2 shows, as an example, an instantiation of this model for a two-tier hierarchy; on the upper level are instantiated 4 cells each consisting of a parallel structure having 256 elementary cells of type (eng_0, mem_0). The *host* is HOST COMPUTER and the *accelerator* is *MapScanRed$_2$(4(256))*. In this case the accelerator consists of 4 general-purpose accelerators. The memory hierarchy is organized on four levels: HOST COMPUTER's main memory, Memory, 4 memory modules *m*, and 256 local memories mem_0 in each cell. The SCAN and REDUCE networks organized on two levels ensures, besides important functional aspects, a communication at cellular level in the two levels of the hierarchy.

What would be the *pros* and *cons* for an implementation of an ALP system in the hierarchical form we propose? We must compare the currently used configurations of ALP co-processors, represented in Figure 3, with our proposal represented in Figure 4.

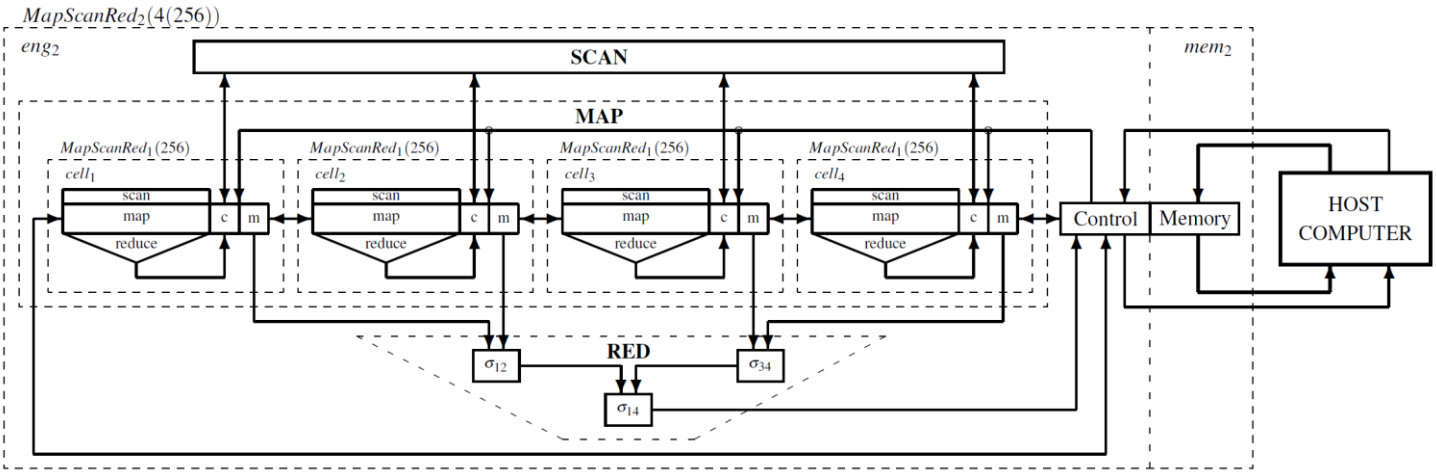

Fig. 2: Heterogenous system with an ALP co-processor implemented as a four-cell MapScanReduce parallel accelerator, where each cell consists of a 256 elementary cells MapScanReduce parallel accelerator.

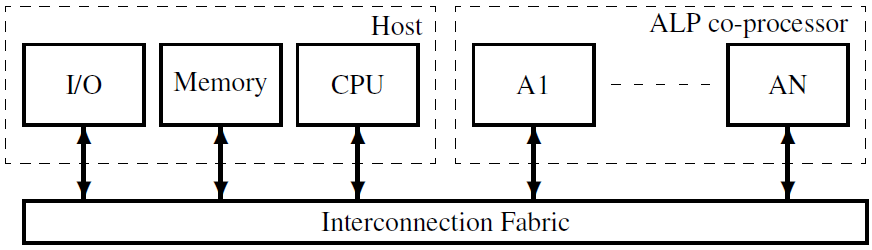

Fig. 3: Unstructured, flat organization of the N-accelerator ALP co-processor used as the accelerator part in a heterogenous system.

In the version of the heterogenous system presented in Figure 3, N accelerators, *A1, ... AN*, and Host's components, CPU, Memory and I/O, are interconnected using a standard interface, Interconnection Fabric, and all transfers are performed using Host's Memory (see in [2] the SoCs Apple's *A11 Bionic*, Qualcomm's *Snapdragon 845*, HiSillicon's *Kirin 970*, Samsung's *Exynos 9810*, or Apple's *M1* in [14]), while in our proposal, represented in Figure 4, there are solutions for data transfers which avoid the use of the Memory module from Host. Indeed, the hierarchically distributed memories M, parts in pairs MSR&M, in the structure represented in Figure 4, allow concurrent transfers between memories M deployed at different levels in hierarchy.

If two MSR&M cells in level i need to exchange data with each other, then the transfer is performed at level i+1 in the hierarchy. In this way, several data transfers can be performed concurrently in the system, resulting in a reduction in the von Neumann Bottleneck effect.

Figure 4 shows the possibility of placing an accelerator on a certain level of hierarchy, and at the same time, the accelerators can be grouped into clusters in the same cell, depending on how they interact in the system architecture. This flexibility can be used to optimize the interaction between accelerators, on the one hand, and, on the other hand, can optimize the data supply process of the effector units.

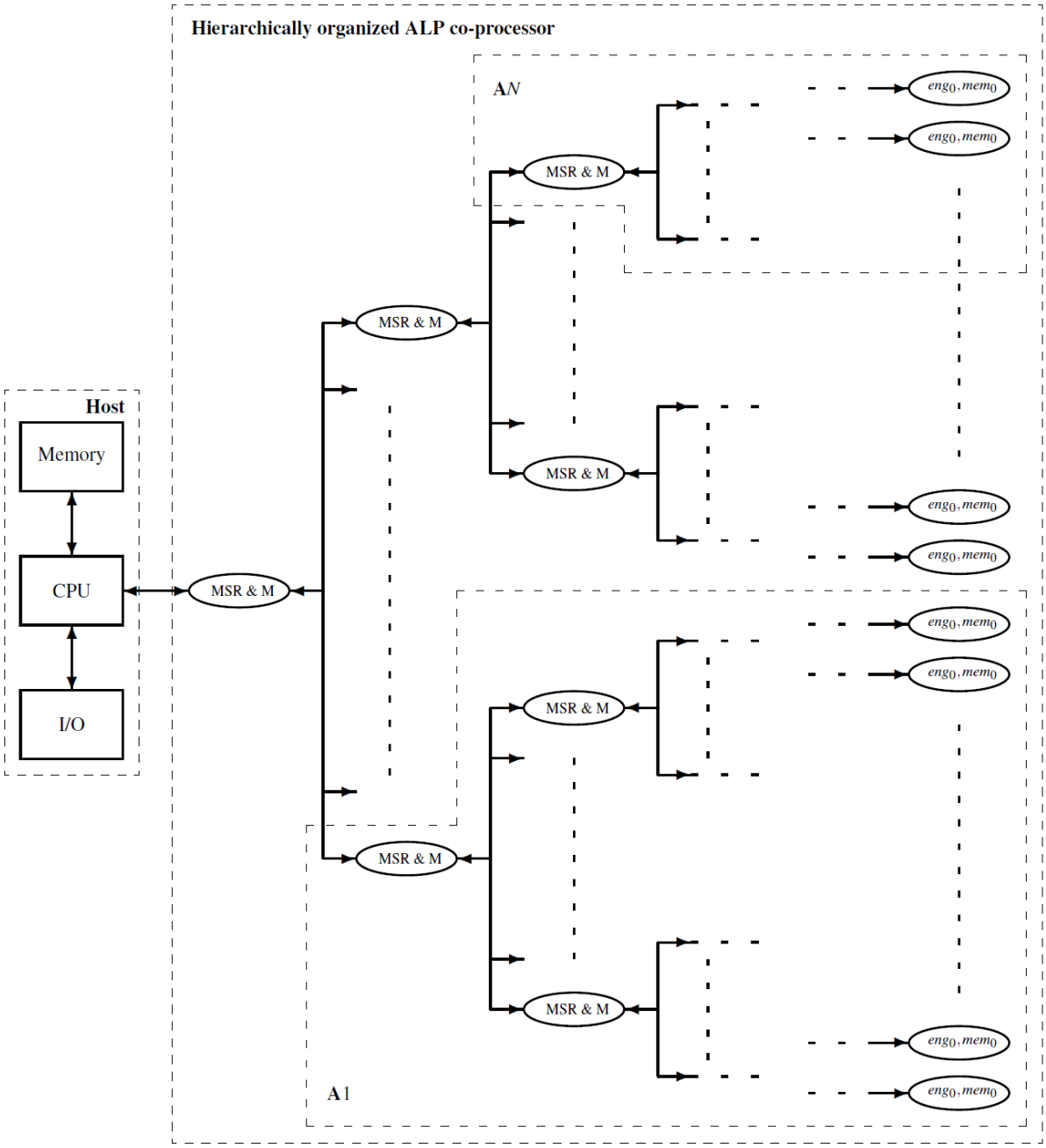

Fig. 4: Hierarchically structured N-accelerator ALP co-processor structured on *k* levels, used as the accelerator part in a heterogenous system. For example, the accelerator ***A1*** is organized on the *k-1* level in the hierarchy, while the accelerator ***AN*** is organized on the level *k-2* in the hierarchy.

4. Conclusion

The evolution of SoCs has naturally led to the emergence of the concept of ALP. At the same time, the theoretical foundation of the concept is required, a foundation that presupposes a rigorous definition of what parallel computing is. This is because the main mechanism by which acceleration occurs is parallelism. In this regard, we proposed that the structure

and architecture of ALP-type systems be based on the abstract recursive MapScanReduce model, which we proposed based on the Stephen Kleene's partially recursive function model.

The main advantages of the proposed solution are:

- the solution is of general-purpose type because it supposes programmable structures
- the acceleration obtained by MapScanReduce accelerators is significant and is obtained at low energies and on low silicon areas (as we presented in the case study: ***super-linear acceleration at 24× less energy on 3.5× less area***)
- the proposed hierarchical organization for ALP attenuates the *von Neumann Bottleneck* effect in accelerators intercommunication and data transfer between accelerators and the main memory of the heterogeneous system
- the hierarchical structure allows the functional approach of system programming through a hierarchy of function libraries that avoids the development of a complex system of compilers.

Acknowledgements

The authors received technical support, in the first attempt to impose a MapScanRed accelerator, from the following collaborators: E. Altieri, F. Ho, B. Mîțu, M. Stoian, D. Thiébaut, T. Thomson, D. Tomescu. Thanks to F. A. Gîndac for his help in evaluating the performance of the x86 architecture in the computing of the matrix product.

References

[1] V. J. Reddi and M. D. Hill, "Accelerator-Level Parallelism (ALP)", *ACM Sigarch*, Sep 3, 2019.
[2] M. D. Hill and V. J. Reddi, "Accelerator-Level Parallelism", *C. of the ACM*, vol. 64, no. 12, pp. 36-38.
[3] M. D. Hill and V. J. Reddi, "Gables: A Roofline Model for Mobile SoCs", in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 317-330, 2019.
[4] G. M. Ștefan and M. Malița, "Can one-chip parallel computing be liberated from ad hoc solutions? A computation model-based approach and its implementation", in *18th Inter. Conf. on Circuits, Systems, Communications and Computers 2014*, pp. 582-597.
[5] E. W. Dijkstra *Cooperating sequential processes*. Technical Report EWD-123, Eindhoven Univ. of Technology, 1965. [Online]. Available: https://www.cs.utexas.edu/users/EWD/transcriptions/EWD01xx/EWD123.html
[6] V. R. Pratt, M. O. Rabin, L. J. Stockmeyer, "A characterization of the power of vector machines", in *Proceedings of STOC'1974*, 1974, pp. 122-134.
[7] S. Fortune, J. C. Wyllie, "Parallelism in random access machines", in *Conference Record of the Tenth Annual ACM Symposium on Theory of Computing*, 1978, pp. 114-118.
[8] L. M. Goldschlage "A universal interconnection pattern for parallel computers" *Journal of the ACM*, 1982, **29**(4):1073-1086.
[9] S. Kleene, "General recursive functions of natural numbers", *Mathematische Annalen*, 1936, **112**(5):727–742.
[10] G. M. Ștefan, et al., "FPGA-Based Programmable Accelerator for Hybrid Processing", *Romanian Journal of Information Science and Technology*, 2016, **19**(1-2):148-165.
[11] G. M. Ștefan, "Let's consider Moore's law in its entirety", in *CAS 2021 Proceedings*, 2021, pp. 3-10.
[12] G. M. Ștefan, *Composition is the only independent rule in Kleene's model of partial recursive functions*, [Online]. Available: http://users.dcae.pub.ro/~gstefan/2ndLevel/technicalTexts/2019_Composition.pdf
[13] M. Malița, G. V. Popescu, G. M. Ștefan: "Heterogenous Computing System for Deep Learning", in Witold Pedrycz, Shyi-Chen (Eds.): *Deep Learning: Concepts and Architectures*, Springer International Publishing, pp. 287-319, 2019.
[14] Usman Saleem, "Apple M1 Chip vs. Intel x86 Processors: What Is the Difference?", *Appuals*, Dec. 19, 2020.
[15] G. M. Ștefan, *The Connex Project*, [Online]. Available: http://users.dcae.pub.ro/~gstefan/2ndLevel/connex.html
[16] *kberkay / Cuda-Matrix-Multiplication*, [Online]. Available: https://github.com/kberkay/Cuda-Matrix-Multiplication
[17] K. Asanovic, et al., "A View of the Parallel Computing Landscape", *C. of the ACM*, vol 52, no. 10, pp. 56-67, 2009.