ROMANIAN JOURNAL OF INFORMATION SCIENCE AND TECHNOLOGY
Volume 23, Number 2, 2020, 115-126

Architectural Features for Artificial Intelligence & Blockchain in the Nano-Era

Mihaela Maliţa[1], Răzvan Mihai [2], and Gheorghe M. Ştefan[2]

[1]Smith College, Northampton, MA
[2]Politehnica University of Bucharest
[2]Email: gheorghe.stefan@upb.ro

Abstract. Artificial Intelligence and Blockchain are among the most computationally demanding domains. They request tremendously powerful computation engines capable to deliver *fast*, *cheap* and *energy aware* solutions. These two technological domains provide the safe computational environment for making intelligent decisions related to complex issues. Here we present the functional aspects and the structural and architectural requirements best suited for implementing the technological environment which allows an *emerging safe & intelligent world* – a world dominated by technologically assisted consensual decisions and self-enforced regulations. Finally, we assert the improvements required from the emerging nano-technologies. These are focused mainly on the interconnection problems raised by the current implementations of big cellular one-chip systems.

1. Introduction

The functional approach in the domains involving emergent nano-technologies must start from high level requirements defined at the market level. The current technology driven approach must shift gradually toward a market driven approach. In this context, the architectural level is of maximal importance by its intermediary position. A resurrected technology Artificial Intelligence (AI) – and an emergent technology – Blockchain (BC) – *form a very promising couple* for the near future. Indeed, in our increasingly complex world, *intelligent* decisions should be taken only in a *safe* environment. *Complexity-Intelligence-Safety* represents the magic triad we face in our emergent technologically dominated world. Under these conditions, requirements appearing at the highest functional level need to be reflected at the deepest structural levels. In this paper we will try to investigate the space between function and structure, with a special emphasis on the architectural level.

What are the challenges raised by these technologies? Both, involve a *big* amount of computation to be executed as *fast* as possible. The price for "big & fast" is large chips and energy waste. The compromise to be looked for is in the architectural domain. Therefore, our investigation is an architectural one. We have big chips with a huge computational competence, but

they are not able to use all their computational power in solving real problems. Rarely the peak performance of an accelerator is achieved in real applications.

In the next section we present some aspects of the new applications using both AI and BC which provide advanced tools capable to safely manage complex problems. The safety and complexity are supported by specific highly intensive computation. The third section emphasizes the specific architectural requirements for designing computing systems able to deal with AI and BC using energy aware and area efficient solutions. The fourth section describes the structural solutions. Some preliminary evaluations are provided in the fifth section. We conclude with the main improvement expected at the nano-technological level.

2. Where and How AI & BC are Involved

BC and AI have presently become the two most talked about technologies having the potential to disrupt almost any imaginable vertical industry. Both technologies meet the definition of general-purpose technologies (GPT) as defined by Bresahan and Trajtenber [2] (cited in [3]). One of the key characteristics of a GPT is that it catalyzes innovation and that it is complementary to other emerging technologies. While each technology comes with its own degree of complexity, they both seemed to have passed the hype phase and are now moving on to the productivity/maturity phase on the Gartner[1] technology development scale.

BCs reputation, as a general-purpose technology, is largely due to its first application – Bitcoin – a peer-to-peer cash system proposed by Satoshi Nakamoto back in 2008 [6]. Nakamoto presented a solution to the double-spending problem using a peer-to-peer distributed timestamp server. The term distributed has since been arguably considered an underlying feature of an authentic BC architecture, in line with the original intent of the Bitcoin creator. When this technology deserves our attention? The answer is provided in [17]:

> *Blockchain technology addresses the scenario in which a collection of entities (for example, individuals or companies) want to participate in a communal system but do not trust each other or any third party to operate the system single-handedly.*
> *...*
> *A good place to start is by posing the following questions:*
>
> 1. *Does the system require shared governance?*
> 2. *Does the system require shared operation?*
>   *...*
> 3. *Is it necessary to audit the system's provenance?*
> 4. *Is it necessary to prevent malicious data deletion?*

When it comes about challenges and limitations, in the same paper, the main price paid for applying BC technology is emphasized:

> *Users must acquire hardware and expend electricity to participate in consensus, the real-world cost of which can be tremendous. For example, it was estimated that as*

[1] Gartner, Inc, founded in 1979, is a global research and advisory firm providing for senior leaders across the enterprise with the indispensable business insights, advice and tools they need to achieve their mission-critical priorities and build the organizations of tomorrow.

> *of April 2018 the energy consumed by Bitcoin miners alone was equivalent to the power usage of almost 5.5 million U.S. households.*

AI has made an impressive comeback in recent years thanks to significant advances in computing power. The most famous implementation of AI includes personalized user recommendations, advertisements targeting, or Machine Learning (ML) algorithms to prevent fraud. There are other applications in logistics, data mining, medical diagnoses, or automotive, to name just a few.

AI relies on large amount of data sets in order to train and improve its algorithms. Hence, the quality and accuracy of collected data are fundamental for its development. At the same time, BC is well-known for its capability to handle immutable, tamper free and consensus validated data. It is precisely at this point where BC and AI technology intersect and complement one another.

In contrast to BC, AI is at the present time highly centralized in the hands of corporations such as Facebook, Google or Amazon who have been able to collect impressive amounts of data sets. Combining these two technologies has the potential not only to improve the quality and reliability of input data necessary for ML algorithms, but to democratize the computational power of AI in a distributed peer-to-peer system such as the one proposed by BC. Artificial Intelligent Blockchains are already implemented (see KafkaBlockchain open source library as a starting point for securing enterprise applications). Last, but not least, is the problem of the (consensual) voting:

> Electronic voting *is a challenging problem that is often asserted to benefit from block-chain technology's properties. Shared governance could be used to ensure multiple parties (the government, nongovernmental organizations, international watchdogs) can work together to ensure an election is legitimate. Auditability is important in providing evidence to the electorate that the election was fair. [17]*

Indeed, the current mode of making decisions can be fundamentally reformed by using a well weighted combination of BC and AI technologies [22]. The safety of BC combined with the competence of AI can provide the tool able to well temper our too untempered world.

3. Architectural Requirements for AI & BC

The two technologies we envisage – AI and BC – suppose quite different architectural requirements, but we will accommodate them on the same physical organization, even if at the end the low level functions will be used to differentiate few structural embodiments.

3.1. AI Architecture

There are two distinct approaches in AI:

- One is explicitly rule-based, developed usually in a Lisp-like language (Lisp, Scheme, Clojure, etc.). The mechanisms are based on explicitly identified rules in the targeted domain. It works only when the rules are explicitly identifiable and expressible in a high level language.
- Another – currently in vogue – involving ML based on various techniques, exemplified by the followings :

Unsupervised Learning Algorithms: *"tell me what patterns exist in my data"*.

1. Markov Models (MM) and Hidden Markov Models (HMM)
2. Clustering (k-Means, Mean-Shift, Density-Based, ExpectationMaximization using Gaussian Mixture Models, Hierarchical, Conceptual)
3. Principal Component Analysis
4. Association Rules
5. Social Network Analysis

Supervised Learning Algorithms: *"use the patterns in my data to make predictions"*.

1. Regression Analysis
2. k-Nearest Neighbors
3. Support Vector Machine
4. Decision Tree
5. Random Forests
6. Deep Convolutional Neural Networks (DCNN)

Reinforced Learning Algorithms: *"use the patterns in my data to make predictions, and improve these predictions as more results come in"*.

1. Multi-Armed Bandits

working for complex realities where it is hard to emphasize explicitly all the rules governing that realities.

In a real complex application both techniques are used. For example, a real time driving assistance system use a DCNN-based system to detect the situation and rule-based system to decide according to the situation.

3.1.1. Rule-Based AI Architecture

The functional programming languages (Lisp, Scheme, Clojure, etc.) are developed starting from the lambda-calculus proposed by Alonzo Church [7]. The main distinct operations performed in Lisp at the lowest hardware level are searches, inserts and deletes in lists, plus a very efficient implementation of a huge stack. In sequences of data, where the previous operations are performed, it is possible to implement one or more stacks. On a sequence of symbols $S =< s_1\ s_1\ \ldots\ s_n >$ organized as a list of lists, we apply operations such as:

- `SEARCH(name)`: all the occurrences of the sub-sequence `name` are located
- `INSERT(name)`: in the *first* located place the sub-sequence `name` is inserted
- `DELETE`: the s-expression (an atom or a list) from the *first* located place is deleted
- `READ`: the s-expression from the *first* located place is accessed moving the access to the next one.

A stack is directly implemented in any place of the sequence S using `INSERT` (for push) and `DELETE` (for pop) in a fix point (the top of the stack). A sketch of such a Lisp implementation is to be found in [16] [18] [19].

3.1.2. ML-Based AI Architecture

The main functions used in implementing ML work also on *sequences*, now organized as *arrays*. They are exemplified by, but not limited to:

- convolution, pooling, and fully connected neural layers for DCNN
- Hadamard matrix operations for MM and HMM
- predicated vector and reduction operations for various forms of clustering

All these functions are implemented on vectors:

$$V_i = [s_{i1}s_{i2}\ldots s_{in}]$$

$$B = [b_1 b_2 \ldots b_n]$$

for $i = 1,\ldots,m$, as the following predicated:

- vector operations:
$$s_{ij} <= b_j \; ? \; OP(s_{ij}, s_{kj}) \; : \; s_{ij}$$
for $j = 1,...,n$, where OP is an arithmetic or logic operation applied on the components where $b_j = 1$
- reduction operations:
$$redOP(V_i) = OP_{j=1}^{n}(b_j \; ? \; s_{ij})$$
for $j = 1,...,n$, where OP is an associative operation; it provides a scalar starting from the selected components of a vector
- scan operation:
$$s_{ij} <= (j \leq n/2) \; ? \; s_{i(2\times j))} \; : 0$$
for $j = 1,...,n$, which return a vector containing the even components of the vector.

Both, rule-based and ML-based architectures can have associated parallel structures because they operate on sequences of data. Depending on the way the sequences are organized – list or arrays – the associated functionality is specific, but there are a lot of overlaps. Due to these overlaps the same architecture can be efficiently defined for both of them.

Preliminary, it seems we need a sort of SIMD architecture (a map level) which must be complemented with scan and reduce features.

3.2. BC Architecture

The mining process is the most time and energy consuming process associated with the BC technology. It consists in hashes reconsidered until the requested condition is fulfilled. For the mining process the implementation is embarrassingly parallel. Two types of computations are involved:

- map computations performing hashing functions (for example SHA256) on blocks of data each prefixed with a different `nonce`

- reduce operations to find for what nonce the condition is fulfilled.

A many-cell hardware receives a block of data and distributes it in each cell. The nonce is selected randomly and incremented in each cell with the index of the cell. A SIMD-like program runs. At the end, the cell where the condition is fulfilled, if any, sends back its index using the reduction network.

Preliminary, seems we need a sort of SIMD architecture complemented with reduce features.

4. State of the Art

The main drawbacks of the current hardware solutions for AI and BC are [14]:

1. the programmable solutions, represented by CPU, Intel's Many Integrated Core (MIC), GPU, DSP, have inappropriate parallel architectures unable to use all their computational power for the specific computation requested by AI and BC
2. the ASIC solutions do not have the flexibility requested to be efficiently adapted for the various forms of AI and BC,
3. the FPGA-based solutions are expensive, consume much energy and require hardware design specific knowledge.

The general purpose multi-core architectures such as MICs provide a pretty good use of the peak performance (an average of $\simeq 45\%$) but the computation per Watt is low ($\simeq 21\ GFLOP/Watt$). The reduced number of cores (less than 100) requests a simple control, allowing, for some applications, the use of $> 80\%$ from its peak performance. But, for most of the applications, we are unable to use more than 25% from their peak performance. Some of them use only a small percentage of their full performance. The architecture of the general purpose computers is designed for a wide specter of applications, while for intense applications there are specific requirements. The general purpose processors waste too much resources for 32-bit floating point computations, while usually the CNN computation requires small integer arithmetic for inferences and accepts 16-bit float operations for training [13] [5].

The oxymoronic, so called "general purpose" graphic processing unit (GPGPU) is a tempting "of-the-shelf" solution. It is a many-core parallel processor for intense computations, but, in the same time, is dangerously used far from its application domain. The actual performance related to the peak performance is lower compared with the general purpose multi-core architectures ($\simeq 34\%$) due to the difficulties involved in control and data transfer for hundred or thousands of execution units. The energy use is only a little improved ($\simeq 22.36\ GFLOP/Watt$). The GPUs optimized for deep learning are designed with distinct physical resources for integer, 32-float, 64-float, and tensor operations lowering the area efficiency [4] [10] [8].

Specific ASIC's, such as Tensor Processing Unit (TPU), do not have enough flexibility to support the high variety of applications. Only for a small part of the applications their huge computational power is fully activated. For most of the applications (90%, according to [9]) no more than 13.4% from the peak performance is used. Only for one application, deployed in less than 5% of the applications, 93% from the peak performance is activated. The very big number

of arithmetic systolic units can not be easy activated efficiently for the variety of functions we are dealing with real applications [10] [11] [24].

We conclude about the state of the art of the hardware support for AI and BC:

- The cache-based memory hierarchy does not work for intense computation because the high predictability of the program and data flow does not need the specific features of a cache memory.

- The architectural inadequacy is the main issue, because a general purpose architecture, a graphic oriented architecture or a simple systolic circuit can not be adapted to the specific requirements of the intense computational domains of AI and BC. And, adding the energy saving criteria makes the problem much difficult.

5. The Structure of Our Proposed System

The structure we propose is based on the mathematical computational model of partial recursive functions proposed in 1935 by Stephen Kleene [12] (in the same year Alan Turing published its seminal paper). The structure has a Map-Scan-Reduce organization, already described in few versions in [14] [15] [20] [21]. The solution we propose is an accelerator as part of a hybrid computing system (see Figure 1). HOST is a general purpose (mono- or multi-core) CPU which runs the program which uses a hardware implemented function library running on the accelerator. The HOST processor loads the program(s) describing function(s) to be accelerated, and orders data transfers between ACCELERATOR and External Memory when needed. When the program runs, it sends the necessary calls to the accelerator.

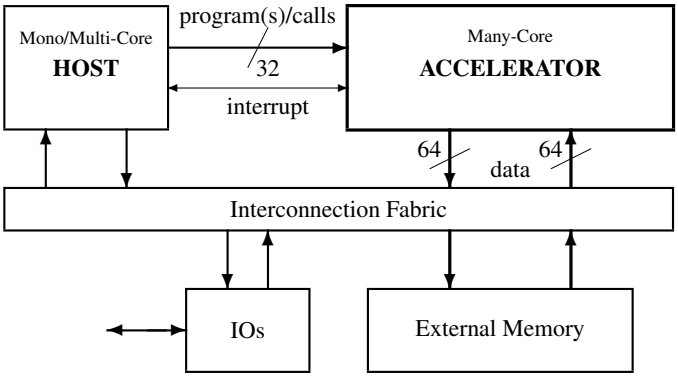

Fig. 1. Heterogenous computing system.

The structure of the accelerator (see Figure 2) is centered on a linear array of n cells, MAP, which receives, in each clock cycle through the *log*-depths network DISTRIBUTE, an instruction issued by CONTROLLER to be executed in all its active cells. Two loops are closed over MAP: one through the *log*-depths network SCAN, and another through: *log*-depths network REDUCE, CONTROLLER and *log*-depths network DISTRIBUTE.

In each cell there is an (accumulator centered) execution unit, eng_j, and a local data memory, mem_j, for $j = 1,\ldots,n$. Thus, along the cells are distributed the previously defined *horizontal vectors* V_i, for $i = 1,...,m$. The content of each local memory represents a *vertical vector*:

$$W_j = [s_{1j}s_{2j}\ldots s_{mj}]$$

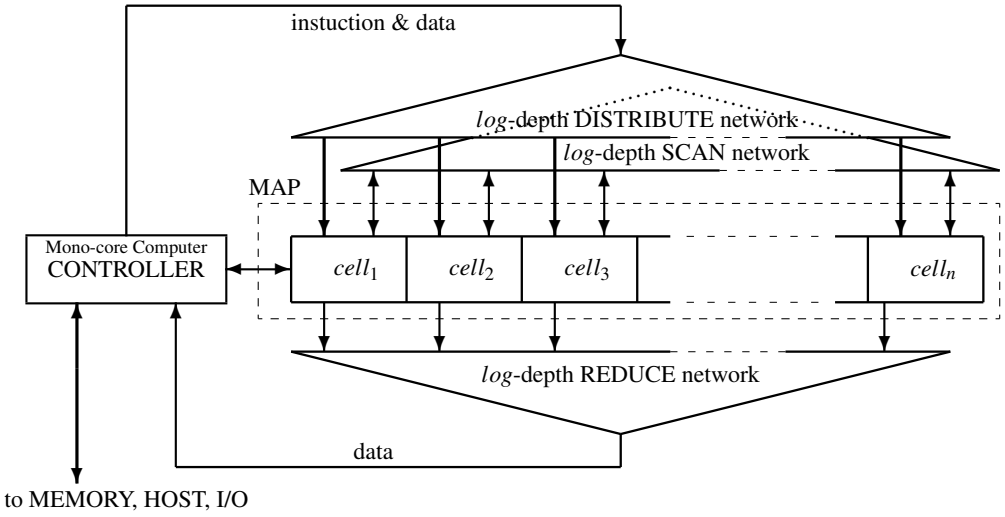

Fig. 2. The structure of the accelerator [14].

Besides these vectors are implemented:

- the accumulator vector:
$$ACC = [acc_1 acc_2 \ldots acc_n]$$
because each cell operates having one operand (the content of the accumulator register) acc_i, and another provided by the instruction, by the local memory or by the controller's accumulator *acc*.

- the Boolean vector used to select the active cells:
$$B = [b_1 b_2 \ldots b_n]$$
it is used to partition the cells in two categories: active cells, for $b_i = 1$, and idle $b_i = 0$

- the cell index vector:
$$IX = [1\ 2\ \ldots\ n]$$
is a constant vector used to identify each cell and to part them in various classes defined by simple arithmetic operations.

Besides the standard logic and arithmetic instructions executed on vectors in the active cells (where $b_i = 1$), in the MAP array are performed specific global operations involving all calls. Some of them are exemplified by the followings:

- spatial selections functions used to part the active cells in two categories. For example:
  - `WHERE(cond)`: $b_i \Leftarrow b_i \,\&\, cond_i \,?\, 1 \,:\, 0$
  - `ENDWHERE`: restores the vector B to the value modified by the previous `WHERE`
- search and modify operations:
  - `SEARCH(x)`: $b_i \Leftarrow (acc_i == x) \,?\, 1 \,:\, 0$
  - `CSEARCH(x)`: $b_i \Leftarrow (acc_i == x) \,\&\, (b_{i-1} == 1) \,?\, 1 \,:\, 0$;
  - `INSERT(x)`: $acc_i \Leftarrow next_i \,?\, acc_{i-1} \,:\, (first_i \,?\, x \,:\, acc_i)$
    where[2]:
    $$NEXT = [next_1, next_2, \ldots, next_n] = prefixOR(B) >> 1$$
    $$FIRST = [first_1, first_2, \ldots, first_n] = prefixOR(B) \,\&\, NOT(NEXT)$$
    are Boolean vectors provided by the SCAN module (see Figure 2)
  - `DELETE(x)`: $acc_i \Leftarrow (i == p) \,?\, 0 \,:\, ((next_i \,||\, first_i) \,?\, acc_{i+1} \,:\, acc_i)$

  are used to identify sequences of symbols in a string and operate modifications on the string in the located position
- global shift operations applied on horizontal vectors
- scan operations:
  - `POOL`: $acc_i \Leftarrow i \leq p/2 \,?\, acc_{2\times i} \,:\, 0$
  - `FIRST`: is an "unrequested" (i.e., computed automatically with a latency in $O(log\, n)$) Boolean vector with 1 on the first active cell position
  - `NEXT`: is an "unrequested" (i.e., computed automatically with a latency in $O(log\, n)$) Boolean vector with 1 on all cells following the first active cells
- reduction operations:
  - `REDADD`: $acc \Leftarrow \Sigma_1^p(b_i \,?\, acc_i \,:\, 0)$ – returns the sum of acc_i from the active cells
  - `REDMAX`: $acc \Leftarrow MAX_1^p(b_i \,?\, acc_i \,:\, 0)$ – returns the the maximum value of acc_i from the active cells
  - `REDOR`: $acc \Leftarrow OR_1^p b_i$ – returns 1 if at least one cell is active.

  where acc is the controller's accumulator. The execution time of the reduction operations is in $O(log\, n)$.

The size of an n-cell ACCELERATOR is in $O(n)$. The latency of its global loops is in $O(log\, n)$.

The accelerator is supposed to be implemented in few versions in order to match better to the requirements of the actual applications. Roughly speaking, three versions could be envisaged. They are differentiated mainly in the type of arithmetic implemented in the execution units of each cell, as follows:

[2] *prefixOR* is the scan function which computes prefixes of the logic function OR. It has as argument a Boolean vector, and returns a Boolean vector.

1. floating point arithmetic (oriented toward the training processes)
2. integer arithmetic (oriented toward the inference processes
3. integer arithmetic with rotate and shift functions instead of multiplication (oriented toward the SHA256-like encryption)

This approach fits perfect the FPGA technology, but can be considered also for big-market products.

6. Evaluation

The proposed architecture is validated by three versions implemented in silicon. Besides these implementations, a 28nm version is evaluated for 2048 32-bit cells with $m = 1024$. Running at 1 GHz and 85^oC the silicon die of $9.2 \times 9.2mm^2$ is powered at 12 Watt.

The performance in implementing DCNN for ML is evaluated in simulations [14]. The acceleration provided for all types of layers is in $O(n)$. For some operations the constant associated to the magnitude order is > 1. For example, the multiplication of a matrix with a vector, besides the parallelism introduced at the MAP level, benefits from the parallelism between: multiplication in MAP, addition in REDUCE and the loop control in CONTROL.

Regarding SHA256, used in BC, the performance (evaluated in Mega Hashes per Second per Watt, *MH/sec/Watt*) on the previously described 28nm implementation is 46.8 $MH/sec/Watt$, which is $3.37\times$ the performance of a Nvidia chip produced in 28nm [23].

There are problems with algorithms running sequentially in $O(n)$ on data of size in $O(n)$ or, worse, in $O(n^2)$. The transfer time for the $O(n)$ sized data is also in $O(n)$ (the "von Neumann Bottleneck" emphasized for the first time by John Backus [1]). Therefore, sometimes the execution time is limited by the data transfer time. Thus, executing *only one* inner product of two vectors has the time complexity in the same magnitude order for both, sequential or parallel computation. Only, for example, the multiplication of two matrices benefits from parallel acceleration, because the sequential time is in $O(n^3)$ for $n \times n$ matrices, and both, parallel computation and data transfer are performed in $O(n^2)$.

Sometimes, the "von Neumann Bottleneck" issue is solved with the price of an increased latency. For example, a fully connected layer in a DCNN supposes multiplying a matrix with a vector. If we load the matrix for each evaluation in the stage it describes, then the transfer time, in $O(n^2)$, is higher then the computation time which is in $O(n)$. The solution is to stay on each stage of the DCNN many steps in order to avoid many weight matrices load. Obviously, the price is the latency. If the latency can be afforded, then we have a solution, else we have a problem.

For some AI algorithm (such as hierarchical clustering) the latency introduced by the reduction and scan loops cannot be avoided (as we did for matrix-vector multiplication). Therefore, accelerating internal data transfer is crucial in acceleration the computation.

7. Concluding Remarks

From the emergent nano-technologies, going under 7 *nm*, for the physical embodiment of the proposed hybrid computing system, we request support for improving or adding the following features to the accelerator part:

On chip dynamic memory buffer big enough to avoid as much as possible data transfers between MEMORY and ACCELERATOR (see Fig. 1). If an internal buffer has the output word big enough to feed one edge of the array of cells with data in one cycle, then the data in and out of the array of cells would be accelerated $\sqrt{n}$ times. A big matrix reused in a program can be stored in this buffer and reloaded faster and with a lower energy consumption.

Multi-chip module with a memory organized as a chip-stack having on top the processing chip will reduce the energy and time for transferring data between MEMORY and ACCELERATOR. Indeed, reducing the chip connections' capacitance both energy and time are saved.

Faster internal connections to reduce the latency of the two global loops inside ACCELERATOR. There are sometimes algorithmic limitations (imposed mainly by the functions expressed as partial functions) due to the latency imposed by the reduction or scan functions: we must wait for the result of a reduction or a scan to continue the computation.

Summing up : in the current stage of the technology it is relatively ***easy to perform the computation***, but there are embarrassingly high ***interconnecting issues*** between the various parts of the computing system.

Acknowledgements - The authors got support from the technical contributors to the development of the *ConnexArray*^{TM} technology: Emanuele Altieri, Frank Ho, Bogdan Mîţu, Marius Stoian, Dominique Thiébaut, Tom Thomson, Dan Tomescu.

References

[1] BACKUS J., *Can Programming Be Liberated from the von Neumann Style? A Functional Style and its Algebra of Programs*, Communications of the ACM, 21, pp. 613-641, August 1978.

[2] BRESNAHAN T. F., TRAJTENBERG M., *General Purpose Technologies: Engines of Growth?*, Journal of Econometrics, 65(1), pp. 83-108, 1995.

[3] CATALINI C., GANS J. S., *Some Simple Economics of the Blockchain*, 2016. [Online]. Available[3]: `https://www.nber.org/papers/w22952.pdf`

[4] CHETUR, S., WOOLLEY, C., et al., *cuDNN: Efficient Primitives for Deep Learning*, 2014. [Online]. Available: `https://arxiv.org/pdf/1410.0759.pdf`

[5] GEORGANS, E., AVANCHA, S., et al., *Anatomy of high-performance deep learning convolutions on SIMD architectures*, SC18, 2018. [Online]. Available: `https://arxiv.org/pdf/1808.05567.pdf`

[6] NAKAMOTO S., *Bitcoin: A Peer-to-Peer Electronic Cash System*, 2008. [Online]. Available: `https://bitcoin.org/bitcoin.pdf`

[7] CHURCH A., *An unsolvable problem of elementary number theory*, The American Journal of Mathematics 58, pp. 345-363, 1936.

[8] HARRIS, M., *Mixed-Precision Programming with CUDA 8*, 2016. [Online]. Available: `https://devblogs.nvidia.com/mixed-precision-programming-cuda-8/`

[3] For all web references, the last access was made on 01/01/2020.

[9] HENNESSY J. L., PATTERSON, D. *Computer Architecture. A Quantitative Approach*. Sixth Edition, Morgan Kaufmann, 2019.

[10] JOOUPPI, N. P., YOUNG, C., PATIL, N., PATTERSON, D., et al., *In-Datacenter Performance Analysis of a Tensor Processing Unit^{TM}*, 44th International Symposium on Computer Architecture (ISCA), Toronto, Canada, June 26, 2017. [Online]. Available: https://drive.google.com/file/d/0Bx4hafXDDq2EMzRNcy1vSUxtcEk/view

[11] KENNEDY, P., *Case Study on the Google TPU and GDDR5 from Hot Chips 29*, 2017. [Online]. Available: https://www.servethehome.com/case-study-google-tpu-gddr5-hot-chips-29/

[12] KLEENE S., *General recursive functions of natural numbers*, Mathematische Annalen, 112(5), pp. 727-742, 1936.

[13] KUMAR, A., TRIVEDI, M., *Intel Scalable Processor Architecture Deep Drive*, 2017. [Online]. Available: https://en.wikichip.org/w/images/0/0d/intel_xeon_scalable_processor_architecture_deep_dive.pdf

[14] MALIŢA M., POPESCU G. V., ŞTEFAN G. M., *Heterogenous Computing System for Deep Learning*, in Witold Predricz, Shyi-Ming Chen (eds), Deep Learning: Concepts and Architectures, Springer, pp 287-319, 2019.

[15] MALIŢA M., ŞTEFAN G. M., THIÉBAUT D., *Not multi-, but many-core: Designing integral parallel architectures for embedded computation*, ACM SIGARCH Computer Architecture News, 35(5), pp. 32-38, Dec. 2007.

[16] MÎŢU B., MÎŢU C., *Toy LISP Interpreter on a Connex Memory Machine*, Journal of Universal Computer Science, vol. 2, no. 5, pp. 427-438, 1996.

[17] RUOTI S., KAISER B., YERUKHIMOVICH A., CLARK J., CUNNINGHAM R., *Blockchain Technology: What Is It Good For?*, Communications of the ACM, January 2020, Vol. 63 No. 1, pp. 46-53, January 2020. [Online]. Available: https://cacm.acm.org/magazines/2020/1/241715-blockchain-technology/fulltext

[18] ŞTEFAN G. M., MALIŢA M., *Chaitin's ToyLisp on a Connex Machine*, Journal of Universal Computer Science, vol. 2, no. 5, pp. 410-426, 1996. [Online]. Available: http://www.jucs.org/jucs_2_5/chaitin_s_toylisp_on/Stefan_G.pdf

[19] ŞTEFAN G., *The Connex Memory: A Physical Support for Tree/List Processing*, The Roumanian Journal of Information Science and Technology, Vol.1, Number 1, pp. 85-104, 1998.

[20] ŞTEFAN G. M., MALIŢA M., *Can one-chip parallel computing be liberated from ad hoc solutions? A computation model-based approach and its implementation*, 18th Inter. Conf. on Circuits, Systems, Com. and Comp., pp 582-597, 2014.

[21] ŞTEFAN G. M., et al., *The CA1024: A Fully Programmable System-On-Chip for Cost-Effective HDTV Media Processing, Stanford University: Hot Chips: A Symposium on High Performance Chips*, August 2006. [Online]. Available: https://youtu.be/HMLT4EpKBAw at 35:00.

[22] ŞTEFAN G. M., MIHAI R., *IT Driven Distributed Consensus for an Integrated Globalized World*, ROMJIST, Volume 21, Number 2, pp. 114-128, 2018.

[23] ***, *MSI GeForce RTX 2070 Ventus 8GB GDDR6 Review (Turing TU106 GPU)*, 2018. [Online]. Available: https://www.geeks3d.com/20181130/msi-geforce-rtx-2070-ventus-8gb-gddr6-review-turing-tu106-gpu/#4_15

[24] **** *Cloud TPU. System Architecture*, 2019. [Online]. Available: https://cloud.google.com/tpu/docs/system-architecture