DOI: https://doi.org/10.15276/aait.09.2026.13
UDC 004.383:004.272.3

Analytical modeling of OAM-based programmable hardware accelerators for data-invariant algorithms execution

Viktor A. Melnyk[1), 2)]
ORCID: https://orcid.org/0000-0002-5046-8002; viktor.melnyk@kul.pl. Scopus Author ID: 57200786767
[1)] The John Paul II Catholic University of Lublin. 14, Al. Racławickie. 20-950 Lublin, Poland
[2)] Lviv Polytechnic National University,12, Stepan Bandera Str, Lviv, 79013, Ukraine

ABSTRACT

Relevance. The growing demand for high-performance computing in domains such as signal processing and scientific modeling necessitates efficient hardware accelerators capable of predictable and scalable execution of tasks. Algorithms with data-invariant structure, characterized by fixed execution schedules and static data dependencies, represent an important class of workloads that can benefit from specialized architectural solutions. Ordered Access Memory (OAM)-based programmable hardware accelerators offer deterministic and conflict-free parallel data access, making them a promising approach, however, systematic analytical methods for their performance and resource evaluation remain insufficiently developed. **Aim and objectives.** The aim of the scientific research highlighted in this paper is to develop a unified analytical framework for modeling and evaluation of OAM-based programmable hardware accelerators. The objectives include formalization of memory requirements, computational throughput, and execution time, as well as analysis of architectural trade-offs for different accelerator structures. **Methods used.** The study employs analytical modeling based on matrix representation of data, instructions, and indices, and derives formal expressions for basic system characteristics as functions of architectural parameters and algorithm properties. A unified execution time model is developed, incorporating preparation, computation, and output phases, with explicit consideration of memory access parallelism and overlapping operations. Comparative analysis is conducted for dual-memory and algorithm-adaptive memory structures. **Results.** The proposed framework provides models for estimating data, instruction, and index memory requirements, computational throughput, and execution time. It enables systematic exploration of the architectural parameter space and reveals trade-offs between memory organization, degree of parallelism, and performance. The analysis shows that adaptive memory structures reduce redundant data movement, leading to improved execution time and more efficient resource utilization compared to dual-memory designs. **Conclusions.** The novelty of the work lies in the development of a unified analytical modeling framework that jointly captures memory organization, computational structure, and execution dynamics of OAM-based accelerators. The practical value of the results consists in enabling early-stage, platform-independent evaluation and optimization of accelerator structures for data-invariant algorithms execution, supporting informed design decisions without the need for time-consuming simulation or prototyping.

Keywords: Programmable hardware accelerator; parallel architecture; ordered access memory; high-performance computing

***For citation*: Melnyk V. A. "Analytical modeling of OAM-based programmable hardware accelerators for data-invariant algorithms execution". Applied Aspects of Information Technology. 2026; Vol.9 No.2: 185–199. DOI: https://doi.org/10.15276/aait.09.2026.13**

INTRODUCTION

The increasing demand for high-performance computing in domains such as signal processing, scientific computing, and real-time data analytics has driven the development of specialized hardware accelerators. In particular, algorithms characterized by data-invariant structures, where the sequence of operations and data dependencies remain unchanged regardless of input values, represent an important class of computational problems. A data-invariant algorithm is an algorithm in which the execution schedule, including the sequence of operations, control flow, and data dependencies, is fully determined a priori by the problem size and structure, and remains independent of the input data values. Examples include fast Fourier transforms (FFT) and other orthogonal transforms [1], sorting networks [2], and various stencil-based computations [3].

Traditional programmable hardware accelerators often face challenges related to irregular memory access patterns, resource contention, and inefficient utilization of parallel processing units [4], [5]. These issues become especially pronounced when attempting to scale performance while maintaining predictable execution behavior.

To address these limitations, Ordered Access Memory (OAM)-based programmable hardware accelerators have been proposed as an alternative architectural paradigm [6], [7], [8]. The key idea behind OAM is to enforce a predefined, deterministic order of memory accesses, which eliminates access conflicts and enables efficient parallel data processing. Previous research has demonstrated the feasibility of OAM-based architecture, proposed conflict-free memory

organization techniques, and introduced optimizations aimed at improving overall efficiency.

Despite these advances, existing works primarily focus on architectural design and structural optimization, while the problem of systematic performance evaluation remains insufficiently explored. In particular, there is a lack of unified analytical models that would allow designers to estimate memory requirements, execution time, and processing throughput at early design stages. In contrast to simulation-based approaches, analytical modeling enables fast estimation of system characteristics, explicit identification of parameter dependencies, and scalable exploration of the architectural design space without the need for time-consuming experimental evaluation.

This paper addresses this gap by proposing a comprehensive framework for analytical modeling of performance, memory requirements and execution timing parameters of OAM-based programmable hardware accelerators. The framework enables quantitative evaluation of system characteristics based on architectural parameters and algorithm properties. It provides a foundation for design space exploration and supports the development of efficient accelerator architecture tailored to data-invariant algorithms.

RELATED WORKS AND PROBLEM STATEMENT

The design of hardware accelerators for high-performance computing has been extensively studied, with a particular focus on FPGA-based and application-specific architectures [9], [10], [11]. A significant body of research addresses dataflow and streaming models, which aim to exploit parallelism by organizing computations as directed graphs of operations [12], [13]. While these approaches can achieve high performance, they often rely on dynamic scheduling and may suffer from resource contention and unpredictable memory access behavior [14.].

In contrast, architectures targeting data-invariant algorithms leverage the static nature of computation graphs to enable deterministic execution [15]. Such approaches are closely related to static scheduling techniques and synchronous dataflow models, where execution order and communication patterns are known at design time [16.], [17]. These properties enable more efficient hardware implementations but require specialized memory organization and control mechanisms.

The Ordered Access Memory (OAM) paradigm [18] represents a distinct direction in this context. Earlier works introduced the concept of OAM-based programmable hardware accelerators, demonstrating how ordered memory access can be used to ensure conflict-free parallel data processing [6], [7]. Subsequent research focused on eliminating access conflicts through structured memory organization and on improving architectural efficiency by optimizing data flow and processing unit utilization [8].

However, existing studies predominantly emphasize architectural design aspects and empirical evaluation. The problem of analytical modeling of system characteristics, such as memory requirements, execution time, and performance scalability, has received comparatively little attention. In particular, there is a lack of generalized models that can be applied across different OAM-based architectural configurations and algorithm classes.

Beyond OAM-specific research, analytical performance modeling has been widely explored in the context of parallel computing systems. Classical approaches include throughput and latency models [19], as well as roofline-based analyses that relate computational intensity to memory bandwidth [20]. While these models provide useful high-level insights, they do not capture the specific structural properties of OAM-based architecture, such as deterministic access ordering and tightly coupled memory-compute interaction.

The framework proposed in this paper extends existing research by introducing a unified analytical model tailored specifically to OAM-based programmable hardware accelerators. It bridges the gap between architectural design and performance evaluation, enabling systematic analysis and optimization of accelerators for data-invariant algorithms.

OAM-BASED ACCELERATOR ARCHITECTURE OVERVIEW

Ordered Access Memory-based programmable hardware accelerators are designed to efficiently execute algorithms with data-invariant structures, where the sequence of operations and data dependencies are known a priori and do not depend on input values. The key idea underlying the proposed architecture is the use of memory subsystems that enforce a deterministic and index-driven data access order, enabling conflict-free parallel processing [21].

1. Principle of Operation

In the OAM-based approach, a program is represented as a set of three components: data, instructions, and indices. All components are stored in dedicated OAM blocks as ordered matrices, where the position of each element is determined by its associated index. The indices define the placement of elements in output data structures and ensure correct alignment between data and operations during execution.

Computation is performed in a synchronous, stepwise manner. At each clock cycle, a row of data and a corresponding row of instructions are read in parallel from their respective OAM blocks and supplied to the arithmetic logic unit (ALU). The ALU executes multiple operations concurrently, and the resulting data are written back to memory together with their indices. Intermediate and final results are stored in pre-allocated locations determined by index values.

A key property of this execution model is that data are accessed only when all required operands are available, ensuring deterministic behavior and eliminating runtime memory conflicts. For algorithms with data-invariant structure, the number of execution cycles is equal to the number of algorithm steps.

The generalized structure of the OAM-based accelerator is shown in Fig. 1.

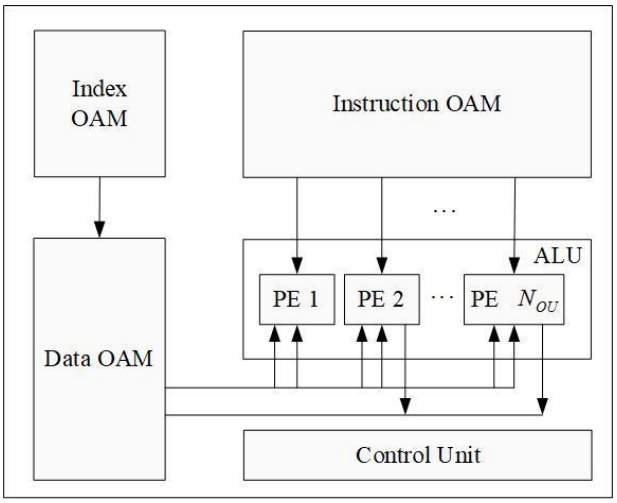

Fig. 1. **Generalized OAM-based accelerator structure**
Source: **compiled by the author**

This structure consists of the following main components:

- Data OAM blocks, used for storing input, intermediate, and output data;
- Index OAM blocks, storing indices associated with data elements;
- Instruction OAM, storing the sequence of operations to be executed;
- Arithmetic Logic Unit (ALU), composed of multiple parallel operating units (or PEs – processing elements);
- Control Unit, responsible for synchronization, mode control (read/write), and routing of data streams.

The ALU includes a set of parallel processing elements, the number of which is determined by the degree of parallelism required by the target algorithm and the available hardware resources. Ideally, the number of processing elements matches the maximum number of operations that can be executed in parallel at any step of the algorithm, enabling full exploitation of parallelism.

Instructions and data are processed in vector or matrix form. When sufficient hardware resources are available, all operations corresponding to a given algorithm step are executed in a single clock cycle. Otherwise, operations are distributed across multiple cycles, resulting in a matrix-like execution model, where instruction and data matrices are processed row by row.

2. Dual-Memory Structure

A baseline implementation of the OAM-based accelerator [6] employs two data memory blocks operating in an alternating (ping-pong) manner. One memory block is used for reading data to the ALU, while the other is used for writing intermediate results.

During execution, data flow between the two memory blocks across successive algorithm steps. At each step, a row of data is read from the active memory block, processed by the ALU according to the corresponding instruction row, and the results are written to the alternate memory block. After each step, the roles of the memory blocks are swapped.

This organization enables continuous data processing without read-write conflicts within a single memory block.

However, it introduces several limitations:

- the need for data transfers between memory blocks at each step;
- redundant movement of unused data when only a subset of elements is required;
- the necessity to adapt memory organization to varying data matrix dimensions.

To reduce hardware complexity, bus-based implementations can be used, where shared input/output buses replace multi-port switching

structures [7]. Such designs reduce resource consumption but preserve the same execution principles.

3. Structure with Algorithm-Adaptive Memory Organization

To overcome the limitations of the dual-memory approach, an enhanced structure with an algorithm-adaptive number of data memory blocks can be employed [8]. In this structure, the number of memory blocks corresponds to the number of algorithm steps, and each block is sized according to the volume of data processed at the corresponding step.

In contrast to the dual-memory structure, intermediate results are written directly to the memory block associated with the next step in which they are required. This eliminates redundant data transfers and avoids the need to reorganize memory contents between steps.

The execution proceeds as follows. After loading instructions and indices, input data are distributed across memory blocks according to their future usage in the algorithm. At each step, the control unit activates the corresponding memory block for reading, and the ALU processes the data according to the instruction stream. The results are then routed to appropriate memory blocks for subsequent steps or to the final output memory.

This structure provides a higher degree of correspondence between the hardware configuration and the algorithm, resulting in improved performance and reduced execution time. However, it requires increased memory resources and a more complex control mechanism.

4. Architectural Trade-offs

The presented architecture illustrates different trade-offs between hardware complexity, memory usage, and performance.

The dual-memory structure offers a simpler design and lower memory requirements but may suffer from redundant data movement and reduced efficiency for algorithms with irregular data utilization. In contrast, the adaptive memory structure minimizes unnecessary data transfers and achieves higher execution efficiency by aligning memory organization with algorithm structure, at the cost of increased hardware resources.

These structural differences directly impact key system characteristics such as memory requirements, throughput, and execution time. In the following Sections, a formal analytical framework is developed to quantify these characteristics and support design space exploration of OAM-based accelerators.

MEMORY REQUIREMENTS FORMAL MODELING

Designing of OAM-based programmable hardware accelerators requires estimation of memory resources needed to store data, instructions, and index structures. Due to the matrix-based representation of program components and deterministic execution model [6], memory requirements can be analytically derived as functions of architectural parameters and algorithm characteristics.

1. Data Memory Requirements

In OAM-based architecture, data are stored and processed in matrix form. Three categories of data must be considered:

- input (initial) data;
- processed data (including intermediate results);
- output (final) data.

Memory requirements for input data can be estimated taking into account that input data are supplied as a matrix $D^{in} \in \mathbb{R}^{N_r^{in} \times N_c^{in}}$. The required memory size is:

$$M_{in} = N_c^{in} \cdot N_r^{in} \cdot w_d \,, \qquad (1)$$

where N_c^{in} is the number of input channels; N_r^{in} is the number of input cycles; w_d is the data word width.

Similarly, let's estimate memory requirements for processed and output data. Processed data (input and intermediate) form the matrix $D^{proc} \in \mathbb{R}^{N_r^{proc} \times N_c^{proc}}$.

The required memory size is:

$$M_{proc} = N_c^{proc} \cdot N_r^{proc} \cdot w_d \,, \qquad (2)$$

where N_c^{proc} is the number of output channels for the processed data submission to processing elements, N_r^{proc} is the number of input cycles.

This memory must support both reading and writing operations during execution.

Output data are represented as $D^{out} \in \mathbb{R}^{N_r^{out} \times N_c^{out}}$, requiring:

$$M_{out} = N_c^{out} \cdot N_r^{out} \cdot w_d \,, \qquad (3)$$

where N_c^{out} is the number of output channels for the results output form the accelerator; N_r^{out} is the number of output cycles.

The total memory required for storing all data is:

$$M_{data} = M_{in} + M_{proc} + M_{out} . \quad (4)$$

The data memory requirement scales linearly with:

- the number of channels (N_c), reflecting the degree of parallelism;
- the number of cycles (N_r), reflecting algorithm depth.

Thus, increasing parallelism improves performance but proportionally increases memory demand. This trade-off is fundamental for design space exploration.

2. Instruction Memory Requirements

Instructions are stored in matrix form $C \in \mathbb{R}^{N_r^c \times N_c^c}$, where N_r^c is the number of instruction loading cycles, N_c^c is the number of instruction channels.

The required instruction memory size is:

$$M_{instr} = N_c^c \cdot N_r^c \cdot w_c , \quad (5)$$

where w_c is the instruction word width.

Due to possible reuse of instructions, the matrix of executed instructions C^{exec} may differ from the stored instruction matrix.

The total number of executed instructions is:

$$| C^{exec} | = N_r^{exec} \cdot N_c^{exec} , \quad (6)$$

where $N_c^{exec} = N_{PE}$ is the number of processing elements, N_r^{exec} is the number of execution cycles.

If the accelerator implements the possibility of multiple execution of one instruction, then when calculating the memory size required to store executed instructions, it is necessary to take into account that the matrix of executed instructions may be larger than the matrix of instructions, since instructions can be executed, and accordingly read from memory, multiple times.

This leads to:

$$N_r^{exec} \geq N_r^c . \quad (7)$$

As a result, the effective instruction memory requirement may be significantly lower than the total number of executed operations. This introduces an implicit compression effect, where a relatively small instruction set can drive large-scale computations.

The instruction word width depends on the number of supported operations:

$$w_c = int\lceil \log_2 N_{op} \rceil , \quad (8)$$

where N_{op} is the number of operation types supported by the processing elements, $int\lceil . \rceil$ denotes a larger integer.

Instruction memory scales with the number of distinct operations rather than total execution length. This property distinguishes OAM-based accelerators from conventional architectures and contributes to their efficiency in data-invariant workloads.

3. Index Memory Requirements

Index structures in OAM-based architecture define the ordering and placement of data elements in memory.

Three categories of indices are considered:

- indices of input data;
- indices of processed and output data;
- auxiliary index structures (indices of indices).

The index matrix for input data I^{in} has the same dimensions as the input data matrix:

$$M_{idx}^{in} = N_c^{in} \cdot N_r^{in} \cdot w_{idx}^{in} . \quad (9)$$

The index width depends on the number of input data elements:

$$w_{idx}^{in} = int\lceil \log_2 (N_c^{in} \cdot N_r^{in}) \rceil . \quad (10)$$

Indices of processed and output data are stored in matrix $I^{proc/out}$, with size:

$$M_{idx}^{proc/out} = N_c^{proc} \cdot N_r^{proc} \cdot w_{idx}^{proc/out} , \quad (11)$$

where:

$$w_{idx}^{proc/out} = int\lceil \log_2 (N_{data}^{proc/out}) \rceil , \quad (12)$$

and $N_{data}^{proc/out}$ is the total number of intermediate and output data elements.

Additional index structures (e.g., indices of indices) are required for indexing indices in the indices OAM.

Their memory requirements can be expressed similarly:

$$M_{idx}^{aux} = N_c^{aux} \cdot N_r^{aux} \cdot w_{idx}^{aux} . \quad (13)$$

The total memory required for index storage is:

$$M_{idx} = M_{idx}^{in} + M_{idx}^{proc/out} + M_{idx}^{aux} . \quad (14)$$

Unlike data memory, index memory exhibits logarithmic scaling with respect to the number of data elements (applies to index size, not the indices number):

$$M_{idx} \propto N \cdot \log_2 N . \quad (15)$$

This property ensures that index overhead remains relatively moderate even for large problem sizes.

At the same time, index structures enable:

- deterministic data access;
- conflict-free parallel execution;

• flexible mapping between algorithm structure and hardware resources.

Thus, index memory can be treated as a key enabler of the OAM paradigm.

4. Scaling Laws

The developed memory models reveal the following scaling properties.

1. Linear scaling of data memory: $M_{data} \propto N$.
2. Linear scaling of instruction memory (with respect to program size): $M_{instr} \propto N_{instr}$.
3. Quasi-linear scaling of index memory: $M_{idx} \propto N \log N$.
4. Trade-off between parallelism and memory usage: increasing the number of channels improves performance but increases memory requirements proportionally.

These relationships form the basis for parametric optimization of OAM-based accelerators and will be used in the next Section to derive performance models and design trade-offs.

COMPUTATIONAL THROUGHPUT FORMAL MODELING AND PARAMETRIC ANALYSIS

1. Processing Structure and Degree of Parallelism

The computational throughput of an OAM-based programmable hardware accelerator is primarily determined by the structure of the ALU and the degree of parallelism it has to provide.

The ALU consists of N_{PE} parallel processing elements, each capable of executing one operation per clock cycle. Each PE typically has two input operands and one output, reflecting the binary nature of most arithmetic and logical operations.

Accordingly, the number of data channels is defined as:

• number of input data channels:

$$N_{in}^{proc} = 2 \cdot N_{PE} , \quad (16)$$

• number of output data channels:

$$N_{out}^{proc} = N_{PE} , \quad (17)$$

• number of instruction channels:

$$N_c^{exec} = N_{PE} . \quad (18)$$

Thus, the degree of parallelism is directly proportional to the number of processing elements:

$$P = N_{PE} . \quad (19)$$

In the ideal case, N_{PE} matches the maximum number of operations that can be executed in parallel at any step of the algorithm, enabling full utilization of available parallelism.

2. Peak Throughput and Effective Throughput

Assuming that each processing element performs one operation per clock cycle, the peak computational throughput of the accelerator can be expressed as:

$$\Theta_{peak} = \frac{N_{PE}}{T_{clk}} , \quad (20)$$

where T_{clk} is the clock period.

This expression corresponds to the maximum number of operations executed per time unit under ideal conditions, i.e., when all processing elements are fully utilized and data are continuously supplied without stalls. However, in practice the effective throughput depends on the actual execution schedule defined by the matrix of executed instructions C^{exec}.

The total number of executed operations is:

$$N_{ops} = N_r^{exec} \cdot N_c^{exec} , \quad (21)$$

and the total execution time is:

$$T_{exec} = N_r^{exec} \cdot T_{clk} . \quad (22)$$

Thus, the effective throughput is:

$$\Theta_{eff} = \frac{N_{ops}}{T_{exec}} = \frac{N_r^{exec} \cdot N_c^{exec}}{N_r^{exec} \cdot T_{clk}} = \frac{N_{PE}}{T_{clk}} . \quad (23)$$

This shows that under ideal conditions, effective throughput equals peak throughput.

3. Throughput under Resource Constraints and Bottleneck Analysis

When the available number of processing elements is less than the required parallelism of the algorithm (due to, for instance, resource limitations), operations must be distributed over multiple cycles.

Let N_{op}^{max} be the maximum number of operations in a single algorithm step. Then the number of execution cycles per step becomes:

$$K = \left\lceil \frac{N_{op}^{max}}{N_{PE}} \right\rceil . \quad (24)$$

In this case, the effective throughput is reduced to:

$$\Theta_{eff} = \frac{N_{op}^{max}}{K \cdot T_{clk}} \leq \frac{N_{PE}}{T_{clk}} . \quad (25)$$

Thus, insufficient hardware parallelism leads to a proportional decrease in throughput.

The achievable throughput of the accelerator is determined by the slowest subsystem.

Two main limiting factors can be identified.

1. Compute-Bound Regime. The system is compute-bound when $\Theta_{peak} \leq \Theta_{mem}$, where Θ_{mem}

is the maximum throughput supported by memory bandwidth. In this regime all processing elements are fully utilized, performance scales with N_{PE}, and increasing memory bandwidth has no effect.

2. Memory-Bound Regime. The system is memory-bound when $\Theta_{peak} > \Theta_{mem}$. In this case processing elements are underutilized, throughput is limited by data delivery rate, and increasing N_{PE} does not improve performance.

The memory subsystem must supply data and instructions at a rate sufficient to sustain parallel execution.

The required data bandwidth is:

$$B_{data} = (2 \cdot N_{PE} + N_{PE}) \cdot w_d = 3 \cdot N_{PE} \cdot w_d \text{ , (26)}$$

where $2 \cdot N_{PE}$ accounts for input operands, N_{PE} accounts for output results.

Additionally, instruction bandwidth must satisfy:

$$B_{instr} = N_{PE} \cdot w_c \text{ .} \quad (27)$$

To sustain peak throughput, the memory subsystem must satisfy:

$$B_{mem} \geq \max(B_{data}, B_{instr}) \text{ .} \quad (28)$$

The OAM-based memory organization ensures that this bandwidth can be utilized efficiently due to deterministic access patterns and absence of conflicts.

The choice of memory organization in accelerator affects achievable throughput. It has to be taken into account that dual-memory structure, presented in [6], requires full data transfers between memory blocks at each step and usually introduce redundant data movement. Effective throughput may be reduced here due to unnecessary memory traffic. Adaptive memory structure eliminates redundant data transfers and aligns memory structure with algorithm steps. It also improves data locality and reduces bandwidth requirements. As a result, adaptive structure is more likely to operate in the compute-bound regime and achieve higher effective throughput.

It should be noted that the proposed analytical models assume that the communication infrastructure, including data buses and interconnects, provides sufficient bandwidth to sustain the required data transfer rates between accelerator's inputs, memory and processing elements, and outputs. In particular, the bus width is assumed to be matched to the data word size and the number of parallel data channels, ensuring that data movement does not become a performance bottleneck. In practical implementations, especially in bus-based architectures, bandwidth constraints may affect performance and should be taken into account during detailed hardware design and experimental evaluation.

4. OAM-Based Accelerators Properties

The developed computational throughput model highlights several key properties of OAM-based accelerators.

1. Linear scalability with processing elements: $\Theta \propto N_{PE}$.

2. Strong dependence on memory bandwidth, which determines the transition between compute-bound and memory-bound regimes.

3. High efficiency of parallel execution, enabled by deterministic scheduling, absence of memory conflicts, and tight coupling between memory and computation.

These properties make OAM-based accelerators particularly well-suited for data-invariant algorithms, where predictable execution patterns allow near-optimal utilization of hardware resources. The derived model provides a basis for evaluating architectural trade-offs and will be further used in the next Section to analyze execution time and system-level performance.

EXECUTION TIME FORMAL MODELING

1. Execution Phases Decomposition

The execution of a program in an OAM-based programmable hardware accelerator can be decomposed into three sequential phases:

1) program preparation phase (data, instructions, and indices loading);

2) processing phase (execution of operations);

3) output phase (result extraction).

Accordingly, the total execution time is defined as:

$$T_{total} = T_{prep} + T_{proc} + T_{out} \text{ ,} \quad (29)$$

where T_{prep} is the preparation phase time, T_{proc} is the processing phase time, and T_{out} is the output phase time. This decomposition reflects the operational structure of OAM-based accelerators and enables independent modeling of each phase.

2. Program Preparation Phase Timing

The preparation phase includes loading instructions into instruction memory, loading input data into data memory, and loading indices into index memory. Let's estimate the duration of the program preparation phase for proposed accelerator structures.

In structures with two data memory blocks [6], [7] instructions, indices, and input data can be loaded partially in parallel.

The preparation time is determined by the longest of these operations:

$$T_{prep}^{(2)} = \max\left(T_{instr}, T_{idx}^{in}, T_{idx}^{proc}, T_{in}\right), \quad (30)$$

where:

$$T_{instr} = N_r^c \cdot T_{clk},$$

$$T_{idx}^{in} = N_r^{in} \cdot T_{clk},$$

$$T_{idx}^{proc} = N_r^{proc} \cdot T_{clk},$$

$$T_{in} = N_r^{in} \cdot T_{clk}.$$

Assuming synchronous operation with a global clock, all loading processes are aligned to the same cycle duration.

For structure with an algorithm-adaptive number of memory blocks [8], two loading strategies are possible.

1. With sequential data indices loading into indices memory blocks. In this case the preparation time is equal to the one with two data memory blocks and bus interconnect:

$$T_{prep}^{(adapt,seq)} \approx T_{prep}^{(2)}. \quad (31)$$

2. With parallel data indices loading. If index memory blocks are loaded in parallel, preparation time reduces to:

$$T_{prep}^{(adapt,par)} = \max(T_{instr}, T_{idx}^{max}, T_{in}), \quad (32)$$

where T_{idx}^{max} is the maximum loading time among all index blocks.

Thus, the adaptive structure enables reduction of preparation time by exploiting parallel loading of index structures. This advantage becomes more pronounced for large-scale problems with multiple processing stages.

3. Processing Phase Timing

The processing phase consists of reading rows of processed data from OAM, executing operations in the ALU, and writing intermediate and final results back to memory blocks.

Let:

- N_r^{proc} be the number of processing cycles (or data input cycles to ALU's PEs (rows of D^{proc}));
- T_{mem} be the memory access cycle time;
- T_{alu} be the ALU operation cycle time.

In a basic execution model, which implies execution without overlapping operations, the execution time is:

$$T_{proc} = N_r^{proc} \cdot (T_{read} + T_{alu} + T_{write}), \quad (33)$$

where T_{read} is a duration of a cycle of the processed data row reading from the data memory, T_{write} is a duration of a cycle of the results row writing from the ALU outputs to the data memory.

Assuming symmetric memory access: $T_{read} = T_{write} = T_{mem}$, we can estimate the execution time as follows:

$$T_{proc} = N_r^{proc} \cdot (2T_{mem} + T_{alu}). \quad (34)$$

Since a key feature of OAM-based architecture is the possibility of overlapping memory and computation operations, the pipelining:

- reading of the next data row,
- execution of the current operation,
- writing of the previous result,

will lead to the overlapped execution model, where the execution time reduces to:

$$T_{proc} = N_r^{proc} \cdot \max(T_{mem}, T_{alu}). \quad (35)$$

In synchronous systems, a global clock is used:

$$T_{clk} = T_{mem} + T_{alu}. \quad (36)$$

Thus, the execution time can be approximated as:

$$T_{proc} = N_r^{proc} \cdot T_{clk}. \quad (37)$$

This also impacts the architecture. In adaptive memory structure, the number of rows of the matrix of processed data, and accordingly the number of cycles of supplying this data to the ALU, is equal to the ratio of the number of ALU inputs to the amount of data used at the given step of the algorithm:

$$N_r^{proc} = \frac{N_{data}^{step}}{N_{in}^{proc}}, \quad (38)$$

where N_{data}^{step} is the number of data elements used in a given algorithm step, N_{in}^{proc} is the number of ALU input channels.

At the same time, this number in structures with two blocks of data memory is equal to the ratio of the amount of data used in the current and all subsequent steps of the algorithm to the number of ALU inputs, and therefore is larger, as a result of which the speed of algorithm execution in these structures is lower. Compared to dual-memory structures, this reduces N_r^{proc}, leading to faster execution.

4. Output Phase Timing

The output phase consists of reading final data from OAM and transferring them to the output interface.

The output time is determined by the number of rows in the output data matrix:

$$T_{out} = N_r^{out} \cdot T_{mem} . \quad (39)$$

Since output operations are purely memory-bound, their duration depends only on memory access characteristics.

5. Unified Execution Time Model

Combining all phases, the total execution time is:

$$T_{total} = T_{prep} + N_r^{proc} \cdot T_{clk} + N_r^{out} \cdot T_{mem} . \quad (40)$$

The proposed model enables systematic estimation of execution time using the following steps.

1. Determine preparation time: $T_{prep}^{(2)}$ for dual-memory structure or $T_{prep}^{(adapt)}$ for adaptive structure.
2. Compute processing time as:

$$T_{proc} = N_r^{proc} \cdot T_{clk} . \quad (41)$$

3. Compute output time as:

$$T_{out} = N_r^{out} \cdot T_{mem} . \quad (42)$$

4. Compute total execution time as:

$$T_{total} = T_{prep} + T_{proc} + T_{out} . \quad (43)$$

The developed execution time model reveals several important properties:

- deterministic timing behavior, enabled by static scheduling and OAM-based memory organization;
- pipeline efficiency, achieved through overlapping memory and computation operations;
- strong dependence on data matrix dimensions, particularly N_r^{proc};
- performance advantage of adaptive structure, which reduce redundant data transfers and minimize processing cycles.

These properties confirm that OAM-based accelerators can achieve predictable and high-performance execution for data-invariant algorithms.

COMPARATIVE ANALYSIS OF ACCELERATOR STRUCTURES

This Section presents a comparative analysis of two main structural variants of OAM-based programmable hardware accelerators: the dual-memory (2-block) structure, and the algorithm-adaptive memory structure.

We will consider a data-invariant algorithm characterized by:

- processed data matrix $D^{proc} \in \mathbb{R}^{N_r^{proc} \times N_c^{proc}}$;
- maximum number of parallel operations per step N_{op}^{max};
- number of processing elements N_{PE}.

Both structures execute the same computational workload but differ in memory organization and data movement patterns. The comparison is performed in terms of execution time, computational throughput, memory usage, and scalability, based on the analytical models developed in previous Sections.

1. Execution Time Comparison

From the previous Section, the total execution time for a dual-memory structure is:

$$T_{total}^{(2)} = T_{prep}^{(2)} + N_r^{proc,(2)} \cdot T_{clk} + N_r^{out} \cdot T_{mem} \quad (44)$$

where $N_r^{proc,(2)}$ is influenced by repeated data transfers between memory blocks with inclusion of unused or redundant data in processing streams.

For the adaptive structure the total execution time is:

$$T_{total}^{(adapt)} = T_{prep}^{(adapt)} + N_r^{proc,(adapt)} \cdot T_{clk} + N_r^{out} \cdot T_{mem} . \quad (45)$$

Due to elimination of redundant data movement:

$$N_r^{proc,(adapt)} \leq N_r^{proc,(2)} , \quad (46)$$

and thus, under typical execution conditions and for algorithms with non-uniform data utilization across computation steps:

$$T_{total}^{(adapt)} < T_{total}^{(2)} . \quad (47)$$

The inequality holds under the assumption that the amount of data required at each algorithm step varies and that the dual-memory structure processes and transfers the full data set between steps, including elements not used in subsequent computations. In contrast, the adaptive structure stores and processes only the data required for each specific step, eliminating redundant data movement. This leads to reduced execution time in the adaptive structure.

2. Throughput Comparison

From the Section discussing computational throughput formal modeling, peak throughput for both structures is:

$$\Theta_{peak} = \frac{N_{PE}}{T_{clk}} . \quad (48)$$

However, effective throughput differs due to memory behavior. In the dual-memory structure

effective throughput is reduced due to redundant data transfers (which takes time) and respectively increased number of processing cycles:

$$\Theta_{eff}^{(2)} = \frac{N_{ops}}{T_{total}^{(2)}}. \quad (49)$$

Improved data locality in the adaptive memory structure leads to:

$$\Theta_{eff}^{(adapt)} = \frac{N_{ops}}{T_{total}^{(adapt)}} \quad (50)$$

with $\Theta_{eff}^{(adapt)} > \Theta_{eff}^{(2)}$.

Also, it is important to note that dual-memory structures are more likely to operate in the memory-bound regime due to excessive data movement, while adaptive structure tend toward the compute-bound regime, as memory bandwidth is utilized more efficiently.

3. Memory Usage Comparison

Total data memory size in the dual-memory structure can be estimated as:

$$M_{data}^{(2)} \approx 2 \cdot M_{proc}\ , \quad (51)$$

due to duplication of data across two memory blocks. This ensures advantages in lower control complexity and simpler memory addressing. However, the disadvantages are redundant storage and inefficient use of memory capacity if the amount of data used differs at different steps of the algorithm.

Total data memory size in the adaptive memory structure can be estimated as:

$$M_{data}^{(adapt)} = \sum_{k=1}^{S} M_k\ , \quad (52)$$

where S is the number of algorithm steps. The advantages here are that the memory is allocated according to actual data usage, and the redundant storage is eliminated. The disadvantages include increased total memory in worst-case scenarios and more complex control logic.

Index memory size for both structures can be estimated as:

$$M_{idx} \propto N\log N\ . \quad (53)$$

However, adaptive structure may require additional index structures for routing and increased index management overhead.

4. Summary of Trade-offs

The main trade-offs between the accelerators' two structures are summarized in Table 1.

The comparative analysis demonstrates that the choice of structure may significantly affect system performance. The dual-memory structure provides a simple and robust baseline solution, suitable for systems with limited hardware resources or strict latency constraints and high number of the algorithm execution steps. In contrast, the adaptive memory structure enables reduced execution time, higher effective throughput, and better utilization of memory bandwidth.

These advantages make it particularly suitable for large-scale data-invariant computations, where efficient handling of intermediate data is critical. At the same time, the increased complexity of control logic and memory organization must be carefully managed during design.

Table 1. **Comparison of OAM-based accelerators structures**

Feature	Dual-memory structure	Adaptive memory structure
Memory organization	Fixed (2 blocks)	Algorithm-dependent
Execution time	Higher	Lower
Throughput	Moderate	High
Data movement	Redundant	Minimal
Memory size	Lower (2 blocks)	Higher (algorithm-dependent number of blocks)
Memory utilization	Lower efficiency	Higher efficiency
Control complexity	Low	High
Scalability	Limited	High

Source: **compiled by the author**

DISCUSSION OF RESULTS

1. Applicability of OAM-Based Accelerators

The results presented in this paper indicate that OAM-based programmable hardware accelerators are particularly effective for a class of data-invariant algorithms, where the structure of computation and data dependencies remain fixed regardless of inputvalues. Such algorithms are characterized by deterministic execution flow, statically defined data access patterns, absence of data-dependent branching, and predictable communication between processing elements.

Typical representatives of this class include:

- structured signal processing algorithms [1], [22];
- matrix and tensor transformations [23];
- sorting networks [2], [21], [24];
- stencil-based computations [3], [25].

For these workloads, the OAM paradigm provides very important architectural requirements, particularly conflict-free memory access, deterministic scheduling of operations, and efficient exploitation of parallelism.

As shown in Sections on computational throughput and memory requirements formal modelling, these properties allow the accelerator to achieve high utilization of processing elements and provide near-peak computational throughput. Moreover, the execution time is predictable and analyzable. Thus, OAM-based architecture is particularly well-suited for applications where performance predictability and regularity of computation are critical.

Despite its advantages, the OAM-based approach has inherent limitations when applied to algorithms with data-dependent behavior [25], [26].

Such algorithms involve:

- conditional branching;
- irregular memory access patterns;
- dynamic data dependencies.

In these cases, the static ordering of memory accesses enforced by the OAM model becomes restrictive. The main limitation is that the fixed access order in OAM does not accommodate dynamically changing execution paths. As a result, for highly irregular workloads, conventional architectures based on dynamic scheduling or general-purpose processors may be more suitable.

2. Scalability Considerations

The scalability of OAM-based accelerators is one of their key strengths, but it is subject to several practical constraints. Firstly, adaptive memory structure provides better scalability compared to dual-memory designs due to better alignment between algorithm structure and hardware resources. However, scalability comes at the cost of increased control complexity and more sophisticated memory management.

As shown in Section on computational throughput formal modelling, computational throughput scales linearly with the number of processing elements: $\Theta \propto N_{PE}$. This scaling holds under the condition that sufficient parallelism is available in the target algorithm.

From Section on memory requirements formal modelling, memory requirements exhibit the following scaling behavior:

- data memory:

$$M_{data} \propto N \, . \tag{54}$$

- index memory:

$$M_{idx} \propto N \log N \, . \tag{55}$$

While data memory grows linearly, index memory introduces additional overhead. However, due to logarithmic dependence, this overhead remains manageable even for large problem sizes.

A critical factor limiting scalability is memory bandwidth. As shown in Section on computational throughput formal modelling, the required bandwidth scales as:

$$B_{req} \propto N_{PE} \, . \tag{56}$$

If memory bandwidth does not scale proportionally, the system transitions from a compute-bound to a memory-bound regime, limiting performance gains.

3. Comparison with Existing Modeling Approaches

The proposed analytical framework can be compared with several widely used approaches for performance modeling and design of hardware accelerators, including Roofline-based models, high-level synthesis (HLS) estimation techniques, and dataflow-oriented architectures.

Roofline models provide a high-level characterization of system performance by relating computational throughput to memory bandwidth [27], [28]. While this approach is effective for identifying whether a system is compute-bound or memory-bound, it does not capture the structural properties of OAM-based architectures, such as deterministic data access ordering, index-driven data placement and usage, and tightly coupled interaction of memory with PEs. In contrast, the proposed model explicitly incorporates these architectural features and enables detailed estimation of execution time and memory requirements at the level of algorithm structure.

HLS tools offer performance estimation based on behavioral descriptions of algorithms, supported by simulation and synthesis reports [29], [30]. These approaches are practical for implementation-oriented design but typically require iterative refinement and tool-specific optimization. The analytical model proposed in this paper complements HLS methodologies by providing early-stage, platform-independent estimation of system characteristics, which can guide architectural decisions prior to synthesis.

Dataflow architectures and models represent computations as directed graphs and exploit parallelism through streaming execution [31], [32]. While they are well-suited for parallel processing, they often rely on dynamic scheduling and may experience resource contention and irregular memory access patterns. In contrast, OAM-based accelerators enforce a static execution schedule and deterministic memory access order, eliminating conflicts and enabling predictable performance, which is directly reflected in the proposed analytical models.

Thus, the proposed framework can be viewed as complementary to existing approaches, providing a specialized modeling methodology tailored to deterministic, data-invariant computations.

FUTURE WORKS

Several directions for further research are identified to extend and validate the presented results.

The first is the development of a programming model for OAM-based accelerators. This includes the design of representative programs for selected classes of data-invariant algorithms, as well as the evaluation of corresponding quantitative characteristics of different accelerator structures, such as memory requirements, execution time, and achievable parallelism.

The second is the experimental validation of the proposed analytical models. This involves implementation of OAM-based accelerator architecture on FPGA platform, followed by synthesis, simulation, and empirical measurement of performance, memory usage, and timing characteristics. Such experiments will enable verification of the accuracy of the derived models and identification of potential deviations caused by hardware-level effects.

The third is a comparative evaluation of the proposed accelerators with existing hardware architectures, including FPGA-based and GPU-based solutions. The comparison should be performed in terms of performance, resource utilization, scalability, and efficiency for representative data-invariant algorithms execution.

In addition, further research may focus on automated design space exploration based on the proposed analytical models. The integration of these models into computer-aided design tools would enable efficient selection of architectural parameters under given performance and resource constraints.

Finally, an important direction is the extension of the proposed framework to partially data-dependent algorithms, investigating the limits of applicability of the OAM paradigm and possible hybrid execution models that combine deterministic scheduling with limited dynamic behavior.

These directions will contribute to transforming the proposed analytical framework into a practical methodology for the design, optimization, and evaluation of OAM-based programmable hardware accelerators.

CONCLUSIONS

This paper proposes a comprehensive analytical framework for estimating memory requirements, execution time, and performance of OAM-based programmable hardware accelerators targeting data-invariant algorithms. The proposed approach extends previous research on OAM architecture by moving from structural design considerations to formal performance and resource modeling.

The main contribution of this work is the development of a unified set of analytical models that enable design-time estimation of system characteristics. In particular, the paper introduced:

- a formal model for estimating memory requirements, covering data, instruction, and index storage and revealing their scaling behavior;

- a computational throughput model that captures the relationship between processing parallelism, memory bandwidth, and achievable performance;
- a detailed execution time model that decomposes program execution into preparation, processing, and output phases, explicitly accounting for pipelined operation and overlapping of memory and computation;
- a comparative analysis of structural variants, highlighting the trade-offs between dual-memory and adaptive memory organizations.

A novelty of the proposed framework lies in its ability to jointly analyze memory organization, computational structure, and execution dynamics within a single consistent model. This enables systematic exploration of the design space and provides insights into the interaction between parallelism, memory bandwidth, and execution efficiency.

The results demonstrate that OAM-based accelerators can achieve high performance and predictable execution behavior for data-invariant algorithms due to deterministic memory access patterns and conflict-free operation. At the same time, the analysis identifies important design trade-offs and practical limitations, particularly in terms of memory bandwidth and applicability to data-dependent workloads. The presented models form a foundation for the systematic design of programmable hardware accelerators based on the OAM paradigm.

REFERENCES

1. Shalaby, N. "Efficient Parallel FFTs for different computational models". *PP*. 1997.
2. Batcher, K. E. "Sorting networks & their applications". In *AFIPS Conference Proceedings*. *Thomson Book Company* 1968; 32: 307–314.
3. Datta, K., et al. "Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures". In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*. 2008. p. 1–12. DOI: https://doi.org/10.1109/SC.2008.5222004.
4. Segura, A., Arnau, J.-M. & Gonzalez, A. "Irregular accesses reorder unit: Improving GPGPU memory coalescing for graph-based workloads". *Journal of Supercomputing*. 2023. DOI: https://doi.org/10.48550/arXiv.2007.07131
5. Dehne, F. & Yogaratnam, K. "Exploring the limits of GPUs with parallel graph algorithms". 2010. DOI: https://doi.org/10.48550/arXiv.1002.4482.
6. Melnyk, V. & Melnyk, A. "Ordered access memory based programmable hardware accelerator parallel architecture". In *Proceedings of the 15th International Conference on CAD Systems in Microelectronics (CADSM 2019)*. 2019. p. 1–5. DOI: https://doi.org/10.1109/CADSM.2019.8779249.
7. Melnyk, V. & Melnyk, A. "Parallel conflict-free ordered access memory based programmable hardware accelerator structure". In *Proceedings of the 9th International Conference on Advanced Computer Information Technologies (ACIT 2019)*. 2019. p. 179–182. DOI: https://doi.org/10.1109/ACITT.2019.8779928.
8. Melnyk, V., Melnyk, A. & M. Rahma. "Efficient OAM-Based programmable hardware accelerator architecture." In *Proceedings of the 14th International Conference on Advanced Computer Information Technologies (ACIT 2024)*. 2024. p. 683–687. DOI: https://doi.org/10.1109/ACIT62333.2024.10712504.
9. Betz, V., Rose, J. & Marquardt, A. "Architecture and CAD for Deep-Submicron FPGAs". *New York: Springer*. 1999.
10. Hauck, S. & DeHon, A. "Reconfigurable computing: The theory and practice of FPGA-based computation". *San Francisco: Morgan Kaufmann*. 2007.
11. Mishra, P. & Dutt, N. "Processor description languages". *San Francisco: Morgan Kaufmann*. 2008. DOI: https://doi.org/10.1016/B978-0-12-374287-2.X5001-0.
12. Melnyk, A. & Melnyk, V. "Personal supercomputers: Architecture, design, application". *Lviv Polytechnic National University Publishing*. 2013.
13. Lee, E. A. & Messerschmitt, D. G. "Synchronous data flow". *Proceedings of the IEEE*. 2005: 75 (9): 1235–1245.
14. Dubois, M., Annavaram, M. & Stenström, P. "Parallel computer organization and design". *Cambridge: Cambridge University Press*. 2012.

15. Parhi, K. K. “VLSI digital signal processing systems: Design and implementation”. *New York: Wiley*. 1999.

16. Lee, E. A. & Parks, T. M. “Dataflow process networks”. *Proceedings of the IEEE*. 1995: 83 (5): 773–801.

17. Sriram, S. & Bhattacharyya, S. “Embedded Multiprocessors: Scheduling and Synchronization”. *Boca Raton: CRC Press*. 2009. DOI: https://doi.org/10.1201/9781420048025.

18. Melnyk, A. “Ordered access memory and its application in parallel processors”. *Advances in Cyber-Physical Systems*. 2017; 2 (2): 54–62. DOI: https://doi.org/10.23939/acps2017.02.054.

19. Hennessy, J. L. & Patterson, D. A. “Computer architecture: A quantitative approach”. *6th ed. Morgan Kaufmann*. 2019.

20. Williams, S., et al. “Roofline: An insightful visual performance model for multicore architectures”. *Communications of the ACM*. 2009: 52 (4): 65–76. DOI: https://doi.org/10.1145/1498765.1498785.

21. Melnyk, A. “Computer memory with parallel conflict-free sorting network-based ordered data access.” *Recent Patents on Computer Science*. 2014; 9: 67–77. DOI: https://doi.org/10.2174/2213275907666141021234845.

22. Blahut, R. E. “Fast algorithms for signal processing”. *Cambridge: Cambridge University Press*. 2010. – Available from: https://assets.cambridge.org/97805211/90497/frontmatter/9780521190497_frontmatter.pdf.

23. Usevich, K., Li, J., & Comon, P. “Approximate matrix and tensor diagonalization by unitary transformations: convergence of Jacobi-type algorithms”. *SIAM Journal on Optimization*. 2020; 30 (4): 2998–3028.

24. Dubey, A. “Stencils in scientific computations.” In *Proceedings of the Second Workshop on Optimizing Stencil Computations (WOSC ’14)*. 2014. p. 57. DOI: https://doi.org/10.1145/2686745.2686756.

25. Aguilar, J. & Leiss, E. “Data dependent loop scheduling based on genetic algorithms for distributed and shared memory systems”. *Journal of Parallel and Distributed Computing*. 2004; 64 (5): 578–590. DOI: https://doi.org/10.1142/S0129626405002118.

26. Tong Zhang. “Data dependent concentration bounds for sequential prediction algorithms.” In *Learning Theory (COLT 2005)*. Berlin: Springer. 2005; 3559. DOI: https://doi.org/10.1007/11503415_12.

27. Williams, S., Waterman, A. & Patterson, D. “Roofline: An insightful visual performance model for multicore architectures”. *Communications of the ACM*. (2009); 52 (4): 65–76. DOI: https://doi.org/10.1145/1498765.1498785.

28. Panagou, I.-M., Gkeka, M. R., Patras, A., Lalis, S., Antonopoulos, C. D. & Bellas, N. “FPGA roofline modeling and its application to visual SLAM”. In *Proceedings of the 32nd International Conference on Field-Programmable Logic and Applications (FPL 2022)*. 2022. p. 130–135. DOI: https://doi.org/10.1109/FPL57034.2022.00030.

29. Basalama, S. & Cong, J. “Stream-HLS: Towards automatic dataflow acceleration”. In *Proceedings of the 2025 ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA ’25)*. 2025. p. 103–114. DOI: https://doi.org/10.1145/3706628.3708878.

30. Molina, R., Costa, V., Crespo, M. & Ramponi, G. “High-level synthesis hardware design for FPGA-based accelerators: Models, methodologies, and frameworks”. *IEEE Access*. 2022; 10: 90429–90455. DOI: https://doi.org/10.1109/ACCESS.2022.3201107.

31. Wan, Y., Xie, X., Yi, L., Jiang, B., Chen, J. & Jiang, Y. “Pflow: An End-to-End heterogeneous acceleration framework for CNN inference on FPGAs”. *Journal of Systems Architecture*. 2024; 150: 103113. DOI: https://doi.org/10.1016/j.sysarc.2024.103113.

32. Li, R. “Dataflow & Tiling strategies in Edge-AI FPGA accelerators: A comprehensive literature review”. *arXiv*. 2025. DOI: https://doi.org/10.48550/arXiv.2505.08992.

Conflicts of Interest: The author declares that he has no conflict of interest regarding this study, including financial, personal, authorship, or other interests, which could influence the research and its results presented in this article.

Received 12.01.2026
Received after revision 12.03.2026
Accepted 19.03.2026

DOI: https://doi.org/10.15276/aait.09.2026.13
УДК 004.383:004.272.3

Аналітичне моделювання ПВД-базованих програмовних апаратних прискорювачів для виконання алгоритмів з інваріантною до даних структурою

Мельник Віктор Анатолійович [1) 2)]
ORCID: https://orcid.org/0000-0002-5046-8002; viktor.melnyk@kul.pl. Scopus Author ID: 57200786767
[1)] The John Paul II Catholic University of Lublin, 14, Al. Racławickie. 20-950 Lublin, Poland
[2)] Lviv Polytechnic National University, 12, Stepana Bandery Street. Lviv, 79013, Ukraine

АНОТАЦІЯ

Актуальність. Зростаюча потреба у високопродуктивних обчисленнях у таких галузях, як обробка сигналів і наукове моделювання, зумовлює необхідність створення ефективних апаратних прискорювачів, здатних забезпечити передбачуване та масштабоване виконання задач. Алгоритми з інваріантною до даних структурою, які характеризуються фіксованим порядком виконання та статичними залежностями за даними, становлять важливий клас задач, ефективність виконання яких можна підвищити застосуванням спеціалізованих архітектурних рішень. Програмовні апаратні прискорювачі на основі пам'яті з впорядкованим доступом (ПВД) забезпечують детермінований та безконфліктний паралельний доступ до даних, що робить їх перспективним підходом, однак систематичні аналітичні методи оцінювання їх продуктивності та ресурсних характеристик залишаються недостатньо розвиненими. **Мета і завдання.** Метою наукового дослідження, висвітленого в цій статті, є розроблення уніфікованої аналітичної основи для моделювання та оцінювання програмовних апаратних прискорювачів на основі ПВД. До завдань належать формалізація вимог до пам'яті, обчислювальної продуктивності та часу виконання, а також аналіз архітектурних компромісів для різних структур прискорювачів. **Методи.** У роботі використано аналітичне моделювання на основі матричного подання даних, інструкцій та індексів, а також отримано формальні вирази для базових характеристик системи як функцій архітектурних параметрів і властивостей алгоритмів. Розроблено узагальнену модель часу виконання, яка включає етапи підготовки, обчислення та виведення результатів, з явним урахуванням паралелізму доступу до пам'яті та перекриття операцій. Проведено порівняльний аналіз структур із двома блоками пам'яті та з адаптивною до алгоритму кількістю блоків пам'яті. **Результати.** Запропоновані моделі для оцінювання вимог прискорювачів до пам'яті даних, інструкцій та індексів, продуктивності та часу виконання завдань. Вони дозволяють системно досліджувати простір архітектурних параметрів і виявляти компроміси між організацією пам'яті, ступенем паралелізму та продуктивністю. Показано, що структури з адаптованою до алгоритму кількістю блоків пам'яті зменшують надлишкові переміщення даних, що дає змогу скоротити час виконання алгоритму та ефективніше використовувати ресурси порівняно зі структурами з двома блоками пам'яті. **Висновки.** Наукова новизна роботи полягає у розробленні уніфікованої аналітичної моделі, яка комплексно враховує організацію пам'яті, обчислювальну структуру та динаміку виконання задач прискорювачем на основі ПВД. Практичне значення результатів полягає у забезпеченні можливості раннього платформонезалежного оцінювання та оптимізації архітектури прискорювачів для виконання алгоритмів з інваріантною до даних структурою, що сприяє прийняттю обґрунтованих проєктних рішень без необхідності трудомісткого моделювання або апаратного прототипування.

Ключові слова: програмовний апаратний прискорювач; паралельна архітектура; пам'ять із впорядкованим доступом, високопродуктивні обчислення

ABOUT THE AUTHOR

Viktor A. Melnyk - Doctor of Engeneering Sciences, Professor, Department of Social and Technical Sciences, John Paul II Catholic University of Lublin, Al. Racławickie 14, 20-950 Lublin, Poland, Professor, Department of Information Technologies Security, Lviv Polytechnic National University, 12, St. Bandera Str. Lviv, 79013, Ukraine
ORCID: https://orcid.org/0000-0002-5046-8002; viktor.melnyk@kul.pl. Scopus Author ID: 57200786767
***Research fields*:** Computer systems architecture research and design; IP cores design; high-performance reconfigurable computer systems design; computer data protection; cryptographic processors design and implementation; wireless sensor network security

Мельник Віктор Анатолійович - доктор технічних наук, професор факультету Соціальних і технічних наук, Люблінський католицький університет ім. Івана Павла II, Al. Racławickie 14, 20-950 Lublin, Польща. Професор кафедри Безпеки інформаційних технологій, Національний університет «Львівська політехніка», вул. Ст. Бандери, 12. Львів, 79013, Україна