INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2026, VOL. 72, NO. 3, PP. 1–7
Manuscript received March 10, 2026; revised July 2026. doi: 10.24425/ijet.2026.1721

Scalable FPGA hardware architecture for parallel reduct computation in big datasets Using rough sets

Maciej Kopczynski

***Abstract*—Rough set theory, originally proposed by Z. Pawlak, constitutes an important framework for data analysis and processing in intelligent systems. As contemporary computational environments generate increasingly large datasets, the efficiency of data processing has become a central concern. Data reduction represents a key mechanism for improving computational performance. In the context of rough sets, such reduction is achieved through the elimination of redundant information using reducts. Existing reduct-generation algorithms are predominantly software-based, which entails several inherent limitations, including fixed word-length constraints and overhead associated with instruction fetching and data manipulation. These factors contribute to comparatively low execution performance. Hardware-oriented approaches offer substantially higher processing throughput. This study introduces an FPGA-based hardware solution incorporating a softcore CPU, designed for parallel computation of reducts in large datasets. The proposed solution was evaluated on two real-world datasets executed directly within the FPGA environment, with dataset sizes ranging from 1 000 to 1 000 000 objects. For benchmarking purposes, a corresponding implementation in C was executed on a standard PC. Processing times for both hardware and software variants were recorded and analyzed.**

***Keywords*—rough sets; FPGA; reduct; parallel calculation**

I. Introduction

Major challenge in the development of efficient decision support systems is the time required to process input data, particularly when dealing with large-scale datasets, commonly referred to as *big data*. This term encompasses extensive, heterogeneous, and dynamically evolving collections of data whose analysis is computationally demanding yet highly valuable, as it often reveals previously inaccessible patterns or knowledge. A dataset may be regarded as ”large” when it exceeds the capabilities of standard analytical tools and widely accessible computational resources [14]. The point at which this occurs depends on the complexity of the algorithms employed and may range from megabytes to terabytes.

Numerous techniques have been proposed to address the computational burden posed by such datasets, with particular emphasis on knowledge discovery and data reduction. Rough set theory represents one such approach, offering a mechanism for reducing the dimensionality of the data by eliminating attributes that do not contribute essential information. Introduced by Z. Pawlak [18], rough set theory extends classical set-theoretic notions to accommodate uncertainty and incompleteness. Two concepts play a pivotal role in this framework: *reduct* and the *core* [20]. A reduct is a minimal subset of attributes that preserves the classification capability of the full dataset, while the core consists of attributes that are indispensable and present in every possible reduct.

Field Programmable Gate Arrays (FPGAs) provide a reconfigurable hardware platform whose architecture can be adapted to implement custom digital logic, typically using hardware description languages such as VHDL. Their flexibility, together with substantial potential for fine-grained parallelism, makes FPGAs well suited for accelerating rough set computations. As a result, they can serve as high-performance computational units in both embedded and general-purpose systems operating on large datasets.

The primary objective of this study is to develop an FPGA-based architecture for reduct computation, employing specialized processing units capable of executing operations in parallel. Compared with conventional software-based implementations, the proposed solution achieves significantly higher computation speeds, often by several orders of magnitude.

Prior research has predominantly concentrated on software-oriented solutions or algorithms with low computational complexity. For instance, [1] presents a feature extraction algorithm with linear-time complexity tailored to specific data domains. Other software solutions exploit parallel processing frameworks such as MapReduce, as illustrated by the parallel rough set approximation method in [24]. Additional approaches include Binarized Neural Networks (BNNs) optimized for memory and computational efficiency [4], as well as hardware–software hybrid accelerators for sparse convolutional neural networks [6]. Notably, an FPGA-based accelerator for a Deep Convolutional Neural Network implementing the AlphaGo Policy Network is reported in [7].

Several hardware implementations of rough set techniques have also been proposed. A sample processor for generating decision rules from decision tables is introduced in [17], while [12] proposes a processor based on cellular network concepts following earlier work in [13]. A hardware mechanism for minimizing large logic functions using discernibility matrices is presented in [8]. Although [16] offers efficient heuristics for computing rough set approximations and reducts, the proposed approaches remain software-based. More recent hardware-

The research was carried out within project no. WZ/WI-IIT/3/2023 at Bialystok University of Technology and financed from the research subsidy of Bialystok University of Technology.

M. Kopczynski is with Faculty of Computer Science, Bialystok University of Technology, Bialystok, Poland (e-mail: m.kopczynski@pb.edu.pl).

level advances include an FPGA implementation of the LEM2 algorithm [15] and RTL-level modules for computing lower and upper approximations [23]. Paper [22] describes design of exact reduct rough set hardware accelerator.

The author's earlier contributions include a conceptual rough set processor [21], hardware-supported reduct computation units [9], and scalable modules for processing larger datasets in core [10] and reduct [11] computation. A two-stage reduct computation algorithm was additionally proposed in [5].

The present work introduces a new hardware-assisted approach enabling multiple parallel reduct computations for large datasets. The proposed architecture extends previous research described in [9] and [10]. The earlier implementation in [9] demonstrated high performance for small datasets (up to 107 objects) residing entirely within FPGA memory, but scalability was limited by on-chip storage. The subsequent solution in [11] enabled processing of larger datasets but was restricted to a single computational core.

The remainder of this paper is structured as follows. Section II introduces the principles of reduct computation in rough set theory and outlines the proposed algorithm. Section III presents the architecture of the hardware system, detailing its components and data flow. Section IV reports the experimental evaluation, including dataset characteristics and configuration details for both hardware and software environments. Finally, Section V summarizes the findings and outlines directions for future research.

II. Theory of Operation

This section presents a comprehensive overview of the algorithm developed for generating reducts on FPGA. It also introduces the formal concept of a reduct within rough sets including all essential definitions. The section concludes with an explanation of the fundamental operation mechanism.

A. Definition of Reduct

Consider a DT decision table $(U, A \cup d)$, where U denotes the universe of objects, A represents the set of conditional attributes, and d defines decision attribute. Let the cardinality of the universe be $|U| = n$, where $n > 0$ is a natural number.

For any subset $B \subseteq A$ and any attribute $a \in B$, the indiscernibility relation $IND(B)$ is defined as a binary relation over U that groups together objects indistinguishable with respect to all attributes in B:

$$IND(B) = \{(x, y) \in U \times U : \forall_{a \in B}; a(x) = a(y)\} \quad (1)$$

We introduce the following notions:

- an attribute $a \in B$ is superfluous if removing it does not change the induced indiscernibility relation: $IND(B) = IND(B \setminus a)$. In other case a is called relevant.
- a set B is independent if every attribute in B is relevant.

A reduct is any subset $R \subseteq A$ that satisfies two requirements:

1) Independency — the set R is independent, and
2) Preservation of indiscernibility — $IND(R) = IND(A)$.

Thus, a reduct allows the partitioning of the universe induced by the full attribute set A, meaning that the attributes in $A \setminus R$ are redundant from the perspective of classification.

Since multiple subsets of A may satisfy the reduct conditions, the collection of all reducts of A is denoted by $Red(A) = \{R_1, \ldots, R_r\}$ [18], [20].

B. REDUCT-PHIDM Algortihm

A widely used approach to computing reducts relies on the discernibility matrix. Let U denote the universe of objects, A the set of conditional attributes, and d the decision attribute, with $|U| = n$ for some natural number $n > 0$.

The REDUCT-PHIDM algorithm (**REDUCT** **P**arallel **H**ardware **I**ndirect **D**iscernibility **M**atrix) is founded on identifying the maximal frequency of occurrence of individual conditional attributes. Traditional discernibility-matrix methods require storing the full matrix DM as a two-dimensional structure of $|U| \times |U|$ size, which becomes impractical for large datasets due to excessive memory consumption. Such an approach is infeasible for FPGA execution, where memory resources are limited.

This constraint lead to the design of a hardware-oriented solution—REDUCT-PHIDM. The algorithm partitions the dataset into m segments distributed across multiple independent memory blocks, each supplying data to dedicated processing modules. These memory segments are processed sequentially or in parallel depending on configuration. For clarity, the pseudocode illustrates the simplest version of the REDUCT-PHIDM consisting of a single $subRED$ module supported by two memory units: a shared block RAM_{cmn} accessible to all modules, and a dedicated block RAM_1 (the first of the RAM_n units for an individual $subRED$). More details are presented in Section III.

The pseudocode for the algorithm is presented below:

REDUCT-PHIDM Algorithm

INPUT: decision table $DT = (U, A \cup \{d\})$, two natural numbers $n, m > 0$
OUTPUT: reduct $R \subseteq A$

```
1:  R ← ∅
2:  for a ∈ A do
3:    for b ∈ A do
4:      counts(b) ← 0
5:    end for
6:    for cnt_1 ← 0 to m − 1 do
7:      RAM_cmn ← {x ∈ U : x_{cnt_1·n} to x_{(cnt_1+1)·n−1}}
8:      for cnt_2 ← cnt_1 to m − 1 do
9:        RAM_1 ← {x ∈ U : x_{cnt_2·n} to x_{(cnt_2+1)·n−1}}
10:       for x ∈ RAM_cmn do
11:         for y ∈ RAM_1 do
12:           if d(x) ≠ d(y) then
13:             for c ∈ A \ R do
14:               if c(x) ≠ c(y) then
15:                 counts(c) ← counts(c) + 1
16:               end if
```

17: **end for**
18: **end if**
19: **end for**
20: **end for**
21: **end for**
22: **end for**
23: $redAttr \leftarrow \{b \in A \setminus R : counts(b) = max\{counts(a) : a \in A \setminus R\}$
24: $R \leftarrow R \cup \{redAttr\}$
25: **end for**

The REDUCT-PHIDM algorithm takes a decision table DT as input and produces a reduct R as output. The procedure begins by initializing R as an empty set. The outer loop in line 2 is responsible for iterating over the dataset a number of times equal to the number of conditional attributes. Lines 3–5 reset all entries in the $counts$ vector, which is used to track the frequency of occurrence for each attribute.

The nested loops in lines 6 and 8 select specific partitions of the decision table. The DT is divided into m segments, each containing n objects. Lines 7 and 9 load the selected segments into the FPGA's on-chip RAM blocks. Loops in lines 10 and 11 then extract pairs of objects from the loaded segments for comparison. Line 12 evaluates whether the decision attribute values of objects x and y differ; only when they belong to distinct decision classes, the algorithm proceeds with further processing.

The loop in line 13 covers all conditional attributes not yet included in the created reduct R. For each attribute, if objects x and y differ in value (line 14), the entry in the $counts$ vector is incremented by one. Once all object comparisons are completed, line 23 selects the attribute with the highest occurrence count. This attribute in $redAttr$ is appended to the reduct R (line 24).

In more advanced configurations, where multiple $subRED$ modules operate in parallel, the REDUCT-PHIDM loads additional partitions of DT into the various RAM_n memory blocks, what is covered in line 9. Likewise, comparisons of decision and conditional attribute values occur between the object taken from RAM_{cmn} and the objects stored in each RAM_n block across all instantiated $subRED$ units, what is covered by lines 12–18.

III. Hardware Implementation

This section provides a detailed description of the hardware implementation of the REDUCT-PHIDM algorithm, including the complete system architecture, its constituent submodules, and the primary data flows.

A. System Architecture

The system architecture deployed on the DE-3 development board consists of the following components:

- *Stratix III FPGA* — the primary computational unit, integrating all hardware logic, including the embedded processor and the dedicated blocks for reduct computation,

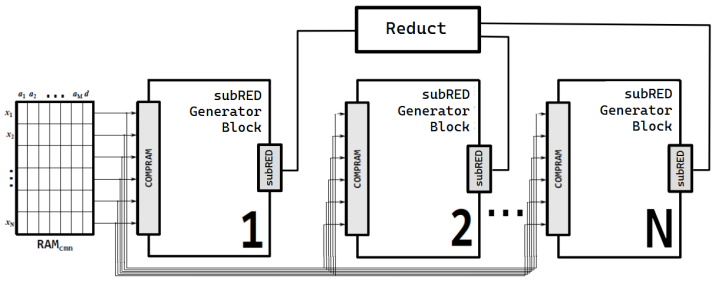

Fig. 1: System overall architecture.

- *DDR2 RAM* — external memory accessed by the NIOS II softcore processor used for support software, the complete dataset retrieved from the Secure Digital memory card, and all intermediate and final results,
- *SD Flash Memory* — used for persistent storage of datasets in binary form.

The FPGA serves as the core of the system, implementing the control logic, dataflow coordination, and the *subRED* modules that perform parallel generation of parts of reducts. The overall FPGA-based system architecture is illustrated in Fig. 1.

The system is composed of the following components:

- RAM_{cmn} — a memory block containing the portion of the decision table that is compared against the data segments stored in the individual $subRED$ Generator Blocks;
- $subREDGeneratorBlock$ — a computational unit (described in detail later in this section) responsible for performing comparisons and calculations between two selected parts of the decision table.

Algorithm is executed by control software running on the NIOS II softcore processor. During each iteration of lines 6–22 of the REDUCT-PHIDM algorithm (see Section II-B), the RAM_{cmn} block is loaded with a segment of the decision table (line 7). Each $subRED$ block likewise stores its own designated segment of the table. A $subRED$ module computes attribute occurrence counts according to the procedure outlined in lines 10–20. When multiple instances of the $subRED$ block are instantiated, the operations corresponding to lines 10–20 can be replicated and executed concurrently, enabling substantial parallelism and reducing overall computation time.

B. Submodule Design

The REDUCT-PHIDM algorithm (Subsection II-B) constructs a reduct using the discernibility matrix [18], [20]. The hardware realization of these operations is encapsulated within the $subRED$ module. Its architecture, illustrated in Fig. 2, is composed of the following elements:

- *Attribute Router with Masking* — a dedicated routing unit that distributes signals derived from the discernibility matrix to the ones counters. It supports attribute masking, enabling selected attributes to be excluded from processing. This behavior corresponds to the condition $a \in A \setminus R$ in line 23 of the algorithm.

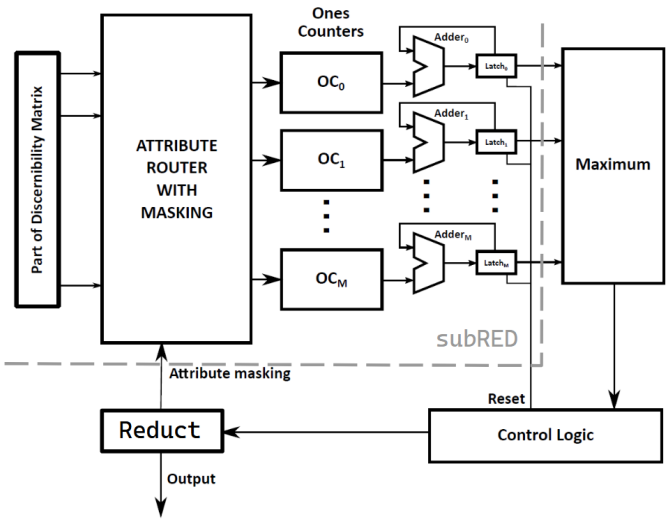

Fig. 2: Block diagram of the $subRED$ module.

- $OC_0, \ldots, OC_M$ — a set of ones counters, each responsible for counting the number of '1' bits associated with a specific attribute in the discernibility matrix. Each OC_i is assigned to exactly one attribute.
- $Adder_0, \ldots, Adder_M$ — accumulation units that sum the counts returned by the ones counters, corresponding to the operation performed in line 15 of the algorithm. Every $Adder_i$ is dedicated to a single attribute.
- $Latch_0, \ldots, Latch_M$ — storage elements used to retain intermediate occurrence counts for each attribute. One $Latch_i$ is provided per particular attribute.

The following components, also depicted in Fig. 2, are shared across all instantiated $subRED$ modules:

- *Maximum* — a module that identifies the attribute with the highest accumulated occurrence count,
- *Control Logic* — module responsible for generating control signals, including attribute selection, latch reset after each computation cycle, and forwarding the most frequent attribute to the *Reduct* block,
- *Reduct* — the register holding the final reduct produced by the system.

As each segment of the discernibility matrix is processed, the *ones counters*, together with the *adders* and *latches*, accumulate the total occurrence count for every attribute represented in the matrix. After all processing cycles have been completed, the *Maximum* module shared among $subRED$ blocks identifies the attribute with the highest accumulated count. This information is provided to the control logic, which then forwards the selected attribute to the *Reduct* register and subsequently marks it as ignored for the remaining iterations. When all attributes have been masked out, the content of the *Reduct* register constitutes the final reduct, and the computation terminates.

Attribute selection and masking are executed within the Attribute Router with Masking (ARM) module. Each entry of the discernibility matrix is delivered to the ARM, where the relevant signals are provided depending on the attribute currently evaluated and subsequently processed by an AND-based masking stage. When the bit corresponding to an attribute is asserted in the *Red* register, the AND suppresses all matrix entries related to that attribute. The resulting masked outputs are then propagated to the *Ones Counters*, which handle the subsequent stages of computation.

IV. Experiments

This section presents the results of the experimental study, including descriptions of the data, methodology, the experimental conditions, and a discussion of the obtained results.

A. Data

This work employs two benchmark datasets: the Poker Hand [2] and a medical dataset characterizing children diagnosed with insulin-dependent diabetes mellitus [19].

The Poker Hand dataset, sourced from the UCI Machine Learning Repository [3], comprises 1 000 000 instances, each representing a five-card hand taken from a standard 52-card deck. Every card is specified by its suit and rank, resulting in ten conditional attributes per instance. Decision attribute identifies the resulting poker combination, covering ten classes arranged according to decreasing frequency: no combination, one pair, two pairs, three of a kind, straight, flush, full house, four of a kind, straight flush, and royal flush.

Type 1 diabetes mellitus is a chronic metabolic condition marked by impaired insulin production, requiring therapeutic insulin supplementation. The corresponding dataset contains 107 patient records, described by twelve conditional attributes, including physical examination findings and laboratory measurements, and one decision attribute representing the presence or absence of microalbuminuria. The original dataset is presented in [19], with an extended software-based evaluation provided in Chapter 6 of [20].

For experimental purposes, the Poker Hand dataset was downsampled to construct subsets of sizes between 1 000 and 500 000 objects, while maintaining the original distribution of decision classes. Conversely, the diabetes dataset was expanded to sizes between 1,000 and 1,000,000 objects by replicating records from the source data.

All datasets were subsequently converted into a binary format. For both datasets, each attribute value was coded using 4 bits. Consequently, each object occupied 44 bits in the Poker Hand dataset and 52 bits in the diabetes dataset. To comply with FPGA memory alignment requirements, every object was padded to a 64-bit word by appending zero-valued bits.

B. Results

The software implementation of the REDUCT-PHIDM algorithm described in Section II-B was developed in the C programming language. It should be emphasized that the implementation utilized only a single CPU core to provide a fair basis for comparison between the PC and FPGA solutions. The experiments were conducted on a workstation equipped with 32GB of RAM and a 4-core Intel Core i7-1185G7 3.0GHz processor, running the Windows 11. The source code was compiled using the GNU GCC 12.1 compiler with the `O2` compilation setting.

Quartus II 13.1 was used for the compilation code in VHDL language. The synthesized solution was deployed on a Terasic DE-3 evaluation board containing a Stratix III EP3SL150F1152C2N FPGA device. The FPGA operated at a 50MHz clock frequency derived from the on-board oscillator, which was used for all sequential components of the design. The implemented REDUCT-PHIDM algorithm corresponds to the version presented in Section II-B.

The NIOS II softcore processor, together with the embedded-system peripherals, was generated using the Qsys 13.1 IDE. The accompanying NIOS II software was implemented in C and compiled via the NIOS II Software Build Tools for Eclipse.

Timing measurements were performed using a LeCroy WaveSurfer 104MXs-B oscilloscope (1GHz BW, 10GS/s sampling). For execution times, that could not be measured by oscilloscope, on-chip hardware timers instantiated within the FPGA were employed.

It should be noted that the PC clock frequency is approximately $\frac{clk_{PC}}{clk_{FPGA}} = 60$ times higher than the FPGA system clock. All experiments were conducted using the datasets described in Section IV-A, with sizes ranging from 1 000 to 1 000 000 objects. In every case, preprocessing (binary transformation and discretization) was performed on the PC.

Table I presents the execution times of the hardware (t_H) and software (t_S) implementations of REDUCT-PHIDM algorithm for both datasets. The hardware solution in the first configuration employed a single instance of the $subRED$ generator block. Segments of the dataset were stored in RAM_{cmn} and RAM_1.

The last two columns of the tables show the acceleration factors with (C_{clk}) and without (C) correction for the clock-frequency difference between the both types of implementation. The abbreviations used for dataset size are $k = 10^3$ and $M = 10^6$.

Table II presents the execution times for the hardware (t_H) and software (t_S) implementations for the configuration, where the hardware system employs two instances of the $subRED$ module. The corresponding segments of the dataset are distributed across RAM_{cmn}, RAM_1, and RAM_2.

Table III presents the execution times for the hardware (t_H) and software (t_S) implementations for the configuration, where the hardware system utilizes four instances of the $subRED$ module. The corresponding dataset segments were stored in RAM_{cmn}, RAM_1, RAM_2, RAM_3, and RAM_4.

The different hardware configurations require following FPGA resources:

- 24 782 Logical Elements (LE) for a configuration using one $subRED$ block,
- 39 408 Logical Elements (LE) for a configuration using two $subRED$ blocks,
- 48 718 Logical Elements (LE) for a configuration using four $subRED$ blocks.

The Stratix III FPGA provides a total of 113 600 Logical Elements, and the above values covers resources used by the hardware modules and the NIOS II softcore processor.

Figure 3 shows the relationship between the number of objects and the execution time for both types of implementation and configuration of $subRED$ modules. Axes are presented on a logarithmic scale.

TABLE I: Execution time for both types of implementation for REDUCT-PHIDM algorithm (one $subRED$ block).

Objects	t_H	t_S	$C = \frac{t_S}{t_H}$	$C_{clk} = 60\frac{t_S}{t_H}$
—	[s]	[s]	—	—
Poker Hand dataset				
1k	0.051	0.527	10.310	618.578
2.5k	0.217	2.727	12.578	754.691
5k	0.926	11.327	12.237	734.211
10k	3.483	43.922	12.609	756.516
25k	20.584	271.724	13.201	792.038
50k	79.121	1 112.276	14.058	843.472
100k	365.181	4 477.226	12.260	735.618
250k	2 199.903	27 374.271	12.443	746.604
500k	8 504.588	110 877.710	13.037	782.244
1M	31 088.792	402 407.110	12.944	776.628
Diabetes dataset				
1k	0.051	0.293	5.728	343.655
2.5k	0.217	1.515	6.988	419.273
5k	0.926	6.293	6.798	407.895
10k	3.483	24.401	7.005	420.287
25k	20.584	150.958	7.334	440.021
50k	79.121	617.931	7.810	468.595
100k	365.181	2 487.348	6.811	408.677
250k	2 199.903	15 207.929	6.913	414.780
500k	8 504.588	61 598.728	7.243	434.580
1M	31 088.792	223 559.505	7.191	431.460

TABLE II: Execution time for both types of implementation for REDUCT-PHIDM algorithm (two $subRED$ blocks).

Objects	t_H	t_S	$C = \frac{t_S}{t_H}$	$C_{clk} = 60\frac{t_S}{t_H}$
—	[s]	[s]	—	—
Poker Hand dataset				
1k	0.030	0.527	17.320	1039.212
2.5k	0.129	2.727	21.131	1267.881
5k	0.551	11.327	20.558	1233.474
10k	2.074	43.922	21.182	1270.947
25k	12.253	271.724	22.177	1330.623
50k	47.096	1 112.276	23.617	1417.032
100k	217.370	4 477.226	20.597	1235.838
250k	1 309.466	27 374.271	20.905	1254.295
500k	5 062.255	110 877.710	21.903	1314.170
1M	18 505.234	402 407.110	21.746	1304.735
Diabetes dataset				
1k	0.030	0.293	9.622	577.340
2.5k	0.129	1.515	11.740	704.378
5k	0.551	6.293	11.421	685.263
10k	2.074	24.401	11.768	706.082
25k	12.253	150.958	12.321	739.235
50k	47.096	617.931	13.121	787.240
100k	217.370	2487.348	11.443	686.577
250k	1 309.466	15 207.929	11.614	696.830
500k	5 062.255	61 598.728	12.168	730.094
1M	18 505.234	223 559.505	12.081	724.853

The presented results demonstrate a substantial improvement in data-processing performance across all evaluated configurations. When compared with the software implementation of the row-by-row discernibility matrix computation (the REDUCT-PHIDM algorithm), the hardware execution achieves a speed-up of approximately 5 times when using

TABLE III: Execution time for both types of implementation for REDUCT-PHIDM algorithm (four $subRED$ blocks).

Objects	t_H	t_S	$C = \frac{t_S}{t_H}$	$C_{clk} = 60\frac{t_S}{t_H}$
—	$[s]$	$[s]$	—	—
Poker Hand dataset				
1k	0.019	0.527	27.193	1 631.563
2.5k	0.082	2.727	33.176	1 990.573
5k	0.351	11.327	32.276	1 936.554
10k	1.321	43.922	33.256	1 995.386
25k	7.804	271.724	34.818	2 089.079
50k	29.997	1 112.276	37.079	2 224.741
100k	138.452	4 477.226	32.338	1 940.266
250k	834.055	27 374.271	32.821	1 969.243
500k	3 224.366	110 877.710	34.387	2 063.247
1M	11 786.773	402 407.110	34.141	2 048.434
Diabetes dataset				
1k	0.019	0.293	15.107	906.424
2.5k	0.082	1.515	18.431	1 105.874
5k	0.351	6.293	17.931	1 075.863
10k	1.321	24.401	18.476	1 108.548
25k	7.804	150.958	19.343	1 160.599
50k	29.997	617.931	20.599	1 235.967
100k	138.452	2 487.348	17.965	1 077.926
250k	834.055	15 207.929	18.234	1 094.024
500k	3 224.366	61 598.728	19.104	1 146.248
1M	11 786.773	223 559.505	18.967	1 138.019

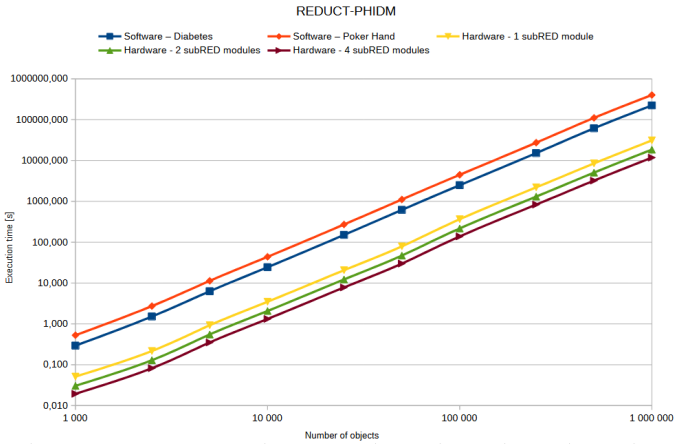

Fig. 3: Processing time in relation to input data size.

a single $subRED$ module and up to 37 times when four $subRED$ modules are employed. After accounting for the clock-frequency difference between the PC and FPGA, the effective speed-up increases significantly, yielding average factors of 418 times (1 instance) and 1 988 times (4 instances). For a fixed hardware configuration, the speed-up remains nearly constant and is consistent across all evaluated dataset sizes.

The hardware processing times for both datasets are identical. The bit-width of individual objects does not influence execution time as long as it fits within the defined memory boundaries. Each hardware processing unit always operates on fixed-size 64-bit words and executes the same sequence of operations regardless of the dataset's original attribute width. This holds for all configurations of the proposed reduct-computation hardware modules.

Expanding the horizontal size of the dataset (i.e., increasing the number of condition attributes) is possible; however, it requires enlarging the comparator structures in the DMGM module and increasing the width of the ones counters in the RGM and ARM modules. This is necessary because the system must process a wider binary word within the same clock cycle. Such modifications substantially increase FPGA resource requirements. In these cases, a larger FPGA device would be necessary, although modern high-capacity chips, such as the Intel Agilex I-Series 040 FPGA, which offers millions of logical elements, can satisfy these demands.

It should also be noted that the average speed-up obtained by increasing the number of $subRED$ modules is not linear. The measured improvements are:

- 1.680 times for a solution with two $subRED$ blocks,
- 2.638 times for a solution with four $subRED$ blocks.

The reduction in scaling efficiency is primarily due to the overhead introduced by the NIOS II processor, which must copy binary data into the RAM_n memories. As the number of $subRED$ modules increases, this data-transfer overhead becomes more significant, thereby reducing the overall speed-up.

V. Conclusions and Plans For Research

Presented reduct calculation approach based on programmable logic demonstrates significant performance advantages over traditional software solutions. By exploiting the inherent parallelism and configurability of FPGA devices, the proposed architecture substantially reduces computation time, even for very large datasets. These results highlight the potential of hardware acceleration as a foundational component of scalable decision-support systems, particularly in domains where rapid or real-time processing is essential, such as medical diagnostics, embedded classification engines, cybersecurity systems, and large-scale data analytics pipelines.

Although the current implementation already achieves notable speed-ups, the hardware reduct-calculation units have not yet been exhaustively optimized. Several promising avenues exist for further performance enhancement. Increasing the FPGA clock frequency or introducing dual-edge triggering would directly reduce cycle counts for sequential operations. Additionally, pipelining internal processing stages and refining routing strategies within the FPGA could minimize latency and improve throughput. Since the scaling results indicate that computation accelerates with replication of the reduct modules, deeper exploration of architectural parallelism is also justified. At present, four parallel units occupy roughly half of a medium-size Stratix III device, suggesting that significantly larger and more powerful modern FPGA platforms could support dozens of parallel reduct units, enabling near real-time processing even for extremely large datasets.

Future work will therefore concentrate on several complementary directions. One line of research involves evaluating different granularities of the $subRED$ modules, potentially

leading to more efficient resource utilization or higher operating frequencies. Another promising direction is to integrate multiple softcore processors or lightweight hardware controllers to manage data transfers for large collections of parallel reduct units. Additionally, optimizing the communication pathways between external memory, on-chip RAM blocks, and the processing modules may further reduce bottlenecks caused by data movement.

An important consideration for future development is the nature and quality of the processed data. The current hardware architecture is optimized for consistent decision tables that contain no missing or uncertain attribute values. However, many modern datasets, especially those collected from medical, sensor-based, or social systems, are incomplete or noisy. As rough set theory inherently supports reasoning under uncertainty, extending the hardware framework to accommodate incomplete information systems represents a natural and necessary next step. Achieving this capability would substantially broaden the range of practical applications and reinforce the role of hardware-accelerated rough set methods in future intelligent systems.

It should also be noted that the presented experimental methodology compares a parallel FPGA-based architecture with a sequential software implementation. Consequently, an important direction for future research will be the evaluation of the proposed hardware approach against contemporary parallel software solutions, including implementations based on multicore CPUs, SIMD vectorization and GPU computing.

REFERENCES

[1] Borowik, G.; Jankowski, J.; Kowalski K. Fast algorithm for feature extraction. Proceedings of SPIE 9662, In Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments, Proceedings of SPIE 2015, pp. 1110-1117.

[2] Cattral, R.; Oppacher, F.; Deugo, D. Evolutionary Data Mining With Automatic Rule Generalization, 2002.

[3] Dua, D.; Graff, C. UCI Machine Learning Repository, University of California, Irvine, School of Information and Computer Sciences, 2017.

[4] Geng T. et al., O3BNN-R: An Out-of-Order Architecture for High-Performance and Regularized BNN Inference," in IEEE Transactions on Parallel and Distributed Systems, vol. 32, no. 1, 1 Jan. 2021, doi: 10.1109/TPDS.2020.3013637, pp. 199-213.

[5] Grześ T.; Kopczyński M. Hardware Implementation on Field Programmable Gate Array of Two-Stage Algorithm for Rough Set Reduct Generation. In *Lecture Notes in Computer Science*, Publisher: Springer, 2019, Vol. 11499, pp. 495-506.

[6] Liang, Y.; Lu, L.; Xie, J. OMNI: A Framework for Integrating Hardware and Software Optimizations for Sparse CNNs in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, doi: 10.1109/TCAD.2020.3023903.

[7] Li, Z. N.; Zhu, C.; Gao, Y. L.; Wang Z. K.; Wang J. "AlphaGo Policy Network: A DCNN Accelerator on FPGA" in IEEE Access, doi: 10.1109/ACCESS.2020.3023739.

[8] Kanasugi, A.; Yokoyama, A. A basic design for rough set processor. In Proceedings of the 15th Annual Conference of Japanese Society for Artificial Intelligence, 2001.

[9] Kopczyński, M.; Grześ, T.; Stepaniuk, J. FPGA in Rough-Granular Computing: Reduct Generation. In proceedings of the 2014 IEEE/WCI/ACM International Joint Conferences on Web Intelligence Vol.2, Warsaw, IEEE Computer Society, 2014, pp. 364-370.

[10] Kopczyński, M.; Grześ, T.; Stepaniuk, J. Computation of Cores in Big Datasets: An FPGA Approach. In *Lecture Notes in Computer Science* Vol.9436, Berlin, Springer-Verlag, 2015, pp. 153-163.

[11] Kopczyński, M.; Grześ, T. FPGA supported rough set reduct calculation for big datasets, Journal of Intelligent Information Systems, vol. 59, 2022, pp. 779-799.

[12] Lewis, T.; Perkowski, M.; Jozwiak, L. Learning in Hardware: Architecture and Implementation of an FPGA-Based Rough Set Machine. In proceedings of the Euromicro, 25th Euromicro Conference (EUROMICRO '99), Volume 1; 1999, pp. 13-26.

[13] Muraszkiewicz, M.; Rybiński, H. Towards a Parallel Rough Sets Computer In: *Rough Sets, Fuzzy Sets and Knowledge Discovery*. Springer-Verlag; 1994, pp. 434-443.

[14] Marz, N.; Warren, J.H. *Big Data: Principles and best practices of scalable realtime data systems*. Manning Publications Co. Greenwich; 2015.

[15] Narsale N.; Agarwal V. Implementation Of LEM2 algorithm On FPGA, 2019 3rd International Conference on Electronics, Communication and Aerospace Technology (ICECA), Coimbatore, India, 2019, doi: 10.1109/ICECA.2019.8822143, pp. 1-5.

[16] Nguyen, S. H.; Nguyen H. S. Some Efficient Algorithms for Rough Set Methods. In Sixth International Conference on Information Processing and Management of Uncertainty on Knowledge Based Systems IPMU'1996, volume III, Granada, Spain, July 1-5 1996, pp. 1451–1456.

[17] Pawlak, Z. Elementary rough set granules: Toward a rough set processor. In *Rough-Neurocomputing: Techniques for Computing with Words, Cognitive Technologies*, Springer-Verlag, Berlin, Germany; 2004, pp. 5-14.

[18] Polkowski, L. Rough Sets, Rough Mereology and Uncertainty, Thriving Rough Sets: 10th Anniversary-Honoring Professor Zdzislaw Pawlak's Life And Legacy & 35 Years Of Rough Sets, Studies in Computational Intelligence, vol. 708, 2017, pp. 49-85.

[19] Stepaniuk, J. Knowledge discovery by application of rough set models. In *Rough Set Methods and Applications. New Developments in Knowledge Discovery, Information Systems*, Physica-Verlag, Heidelberg; 2000, pp. 137-233.

[20] Stepaniuk, J. *Rough-Granular Computing in Knowledge Discovery and Data Mining*. Springer, 2008.

[21] Stepaniuk, J.; Kopczyński, M.; Grześ, T. The First Step Toward Processor for Rough Set Methods. *Fundamenta Informaticae* 2013; *127*, pp. 429-443.

[22] Tiwari K. S.; Dattasamje U. D.; Kadam R. S.; Dudhedia M. A.; Andhale A. A.; Pansare J. R.; Marlapalle B. G. Design of exact reduct rough set hardware accelerator for early-stage diabetes risk prediction, Results in Control and Optimization, Volume 14, 2024.

[23] Tonde, S.; Agarwal, V. Implementation of Lower and Upper Approximation features of rough set theory on FPGA, 2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT), Kharagpur, India, 2021, pp. 01-05.

[24] Zhang, J.; Wong, J.; Pan, Y.; Li, T. A parallel matrix-based method for computing approximations in incomplete information systems. In *IEEE Transactions on Knowledge Data Engineering*, 2015, *27*, pp. 326-339.