DOI: 10.14529/jsfi260204

Rapid Training of Neural Networks Using the Arctur Reconfigurable Computer System

Ilya I. Levin[1], *Dmitriy A. Sorokin*[1], *Vasiliy B. Kovalenko*[1]

Linear scaling of hardware resources in widely used systems based on graphics processing units (GPUs) and central processing units (CPUs) for neural network training problems provides only logarithmic scaling of real performance. This is due to the presence of dataflow discontinuities in the computational graphs of neural network training problems, which lead to a significant decrease in computational intensity and may even lead to a complete stop between execution stages. These limitations are unacceptable if neural network training must be performed within a limited time regardless of the amount of processed data. Reconfigurable computer systems (RCS) based on field-programmable gate arrays (FPGAs), unlike traditional systems, demonstrate the potential to address these limitations through the structural organization of calculations, as well as by adapting to the specifics of solving the problem. Methods for minimizing the impact of dataflow discontinuities have been formulated and theoretically justified for RCS. Based on these methods, a methodology has been developed to construct efficient computing structures to solve neural network training problems, ensuring real RCS performance that exceeds 70% of peak performance. For GPU-based systems, achieving this level of computational efficiency is generally impossible. For the first time, neural network-based data processing problems have been implemented using a parallel-pipeline approach on the Arctur RCS, which contains 96 XCVU37P FPGAs. Experimental results have shown that for these problems, the energy efficiency of a single Arctur RCS block is 56% higher than the energy efficiency of 16 NVIDIA DGX A100 blocks. Increasing the number of Arctur RCS nodes enables near-linear scaling of real performance. For example, the RCS rack consisting of 16 Arctur nodes is capable of providing the real performance of about 7 PFLOPS. For neural network problems such as ResNet-50v1.5 training, the performance of this rack is comparable to that of 12 NVIDIA DGX A100 racks with 2.2× lower power consumption.

Keywords: reconfigurable computer systems, FPGA, neural network training, dataflow discontinuities.

Introduction

The progressive development of national economies is impossible without the widespread application of artificial intelligence technologies for analysis of visual,audio and textual information. Neural network technologies are increasingly used to solve such weakly formalized problems, characterized by high computational complexity [1]. Neural networks are becoming a universal computational tool, and further expansion of their application domains is expected in the coming years.

The practical value of neural network technologies is determined not only by the achieved accuracy in solving the problem but also by the ability to adapt them to changing data environments. However, the increasing diversity of analyzed data is not only accompanied by an increase in the amount of processed data and computational complexity, but also by the degradation of previously trained models, thereby reducing their effectiveness. It is necessary to fine-tune or retrain them.

In real-world applications, when new objects of interest appear, retraining of modern neural networks under conventional conditions can take from several days to several weeks, and in some cases months. However, in rapidly changing environments, it is often necessary to allow for fast

[1] "Scientific Research Center of Supercomputers and Neurocomputers" Co Ltd ("SRC SC & NC" Co Ltd), Taganrog, Russian Federation

training (within several hours), regardless of the amount of processed data. In these conditions, rapid adaptation of neural network systems becomes a key requirement for maintaining the quality of information analysis.

To solve the problem promptly, it is required to linearly increase the computing power of training systems with corresponding scaling of hardware resources. However, most existing systems used for high-accuracy neural network training are based on tensor processing systems: NVIDIA GPUs [2], as well as similar TPU systems by Google [3] and Ascend systems by Huawei [4]. The architecture of tensor processors is adapted for matrix-vector multiplication and data-parallel execution. While they provide efficient pipelined execution within individual neural network layers, inter-layer execution remains constrained by strong data dependencies inherent in the computational graph of training problem.

As a result, execution across layers is segmented into synchronization-bound stages defined by the runtime execution model. Inter-layer data exchange is handled through host-managed runtime scheduling and synchronization in CPU-accelerator coordinated execution systems, introducing additional overhead that reduces continuity of pipeline execution across the full computational graph. These effects directly introduce dataflow discontinuities and reduce computational intensity, as execution phases become bounded by synchronization and resource idling occurs between dependent stages.

Another limitation arises from normalization operations, including post-normalization [5] and pre-normalization [6] of activations, which are required to stabilize training dynamics. These methods improve numerical stability and reduce the risk of gradient fading or explosion. However, commonly used normalization schemes (batch, layer, etc.) require computation of statistical properties over the entire set of normalized elements. Before these statistics are calculated, full buffering of intermediate activations is required, which introduces dataflow discontinuities. These combined effects, under synchronization-dominated execution regimes, result in a logarithmic performance scaling trend for representative neural network training workloads. This behavior is a direct consequence of the inability to sustain continuous pipeline execution across the full computational graph under inter-layer synchronization constraints. Therefore, achieving fast neural network training independent of data volume requires a corresponding exponential increase in hardware resources, resulting in increased system cost and energy consumption.

In contrast, the RCS based on FPGAs demonstrate the potential for near-linear performance scaling in time-consuming, strongly coupled problems across domains such as mathematical physics, geophysics, and evolutionary modeling [7–9]. Several attempts have been made to apply FPGA systems to solve machine-learning problems. However, existing approaches typically use FPGAs as coprocessors within GPU-like execution models [10]. More efficient problem solving on the RCS requires the use of a structural organization of calculations [11]. In this paradigm, the input data, target output data, and the chosen solution method define a computational graph that is mapped to the RCS architecture without a semantic gap. In CPU- and GPU-based systems, this mapping is realized through control-flow-driven or data-parallel execution models with runtime-managed scheduling and synchronization. In RCS, the computational graph vertices corresponding to computational operations and edges representing data dependencies are implemented directly in hardware, enabling pipeline-oriented execution across multiple stages of the graph. This approach enables parallel-pipeline processing of multiple stages of the computational graph and mitigates dataflow discontinuities, defined as breaks in data propagation between dependent operations that lead to reduced computational intensity

and underutilization of computing resources. Calculations are parallelized across both data and iterations using linear or nested pipeline structures, thereby also reducing memory overhead.

The idea of applying structural organization of calculations to neural network training is not new. In [12], Academician A.V. Kalyaev proposed a structural implementation of deep neural network training using static neuroprocessors and inter-layer hardware switching networks. The proposed approach explicitly considers simultaneous execution of forward pass, backpropagation, and weight update operations within a unified hardware structure. However, such a tight coupling of training phases introduces fundamental execution constraints in modern backpropagation-based training systems. Weight update operations depend on globally accumulated gradient information and therefore require completion of preceding computational stages. As a result, update phases introduce synchronization points that interrupt continuous execution flow. Given that the number of weight parameters in modern neural networks is comparable to the amount of intermediate data processed between updates, these synchronization points lead to significant dataflow discontinuities and reduced utilization of computing resources.

It should be noted that the earlier RCS (from the early 2000s until recently) were unable to efficiently solve neural network training problems. Structural implementation of real neural network problems on RCS, especially learning problems, requires integration of multiple FPGAs into a unified computing field, with inter-device communication implemented via high-speed hardware interconnects. Given the data dependency patterns in neural network workloads, such interconnects can currently be implemented only using multi-gigabit transmission technologies [13]. In addition, structural training implementations require multiple independent memory access channels per FPGA, which can be addressed using stacked multi-channel DRAM technologies [14].

The architectural design of the Arctur RCS, based on Xilinx XCVU37P FPGAs, satisfies these requirements. The system provides the high computational density (7.11 TFLOPS/L), a high-bandwidth inter-FPGA interconnect subsystem based on MGT transceivers with aggregate throughput up to 200 Tbps, and distributed HBM memory with a total capacity of 768 GB. This paper covers the methods for minimizing dataflow discontinuities in structural implementation of neural network training problems on the Arctur RCS and evaluates their performance and energy efficiency in comparison with NVIDIA tensor processing systems. The remainder of the paper is organized as follows. Section I describes the Arctur RCS used in the experiments. Section II analyzes dataflow discontinuities in neural network training and proposes mitigation methods. Section III presents hardware implementations of computing structures for neural network training, including a representative convolutional layer in training mode. Section IV presents performance and energy efficiency results for the ResNet-50v1.5 training problem, compared to the NVIDIA DGX A100 system. The conclusion summarizes the main results of the research, highlights the most significant findings, and discusses their practical applicability and potential for future use.

1. Arctur Reconfigurable Computer System

The Arctur computer system implements a set of architectural solutions that realize the proposed computing paradigm. The computing and communication subsystems of the Arctur RCS enable the implementation of large-scale computational graphs within a unified computing field. The system provides the required power delivery and cooling characteristics [15]. The Arctur RCS block is shown in Fig. 1.

Figure 1. The Arctur RCS block

The Arctur RCS node is implemented in a 3U 19" form factor and contains 16 vertically mounted computing modules interconnected via a backplane. Each module integrates six XCVU37P FPGAs from the Xilinx UltraScale+ family. A photograph of the computing module is shown in Fig. 2.

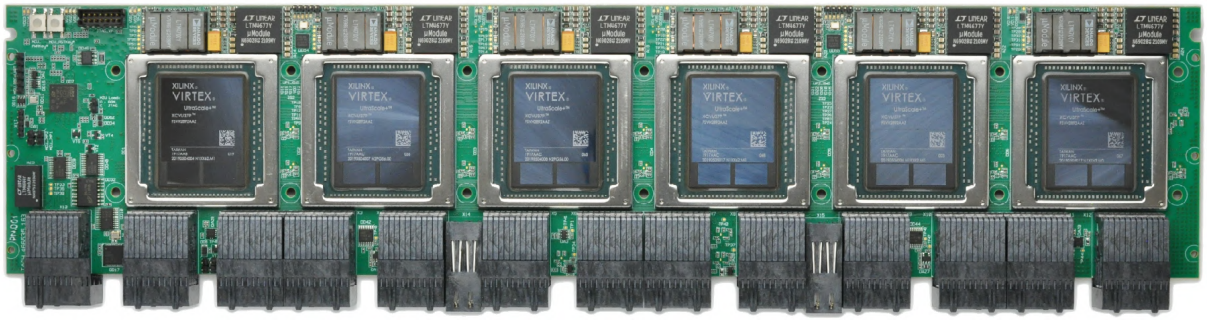

Figure 2. The Arctur RCS computing module

Therefore, a single Arctur RCS block provides high-density integration of computing resources, comprising 96 high-capacity FPGAs with a total of more than 270 million logic cells. The cooling of all components is implemented using immersion technology based on a dielectric fluid with high electrical insulation strength, thermal conductivity, and heat capacity at low viscosity.

The Arctur RCS features an extensive communication subsystem. FPGAs are interconnected via both horizontal and vertical communication links. Horizontal interconnects within a computing module are implemented using 24 MGT transceivers with data rates up to 24 Gbps per link, providing a full-duplex bandwidth of up to 576 Gbps between adjacent FPGAs. In addition, a ring interconnect between the first and last FPGA in the module is implemented using 16 MGT transceivers with a bandwidth of up to 384 Gbps. An auxiliary full-duplex communication channel based on 24 LVDS transceivers provides additional bandwidth of up to 28.8 Gbps. The structure of horizontal interconnects is shown in Fig. 3.

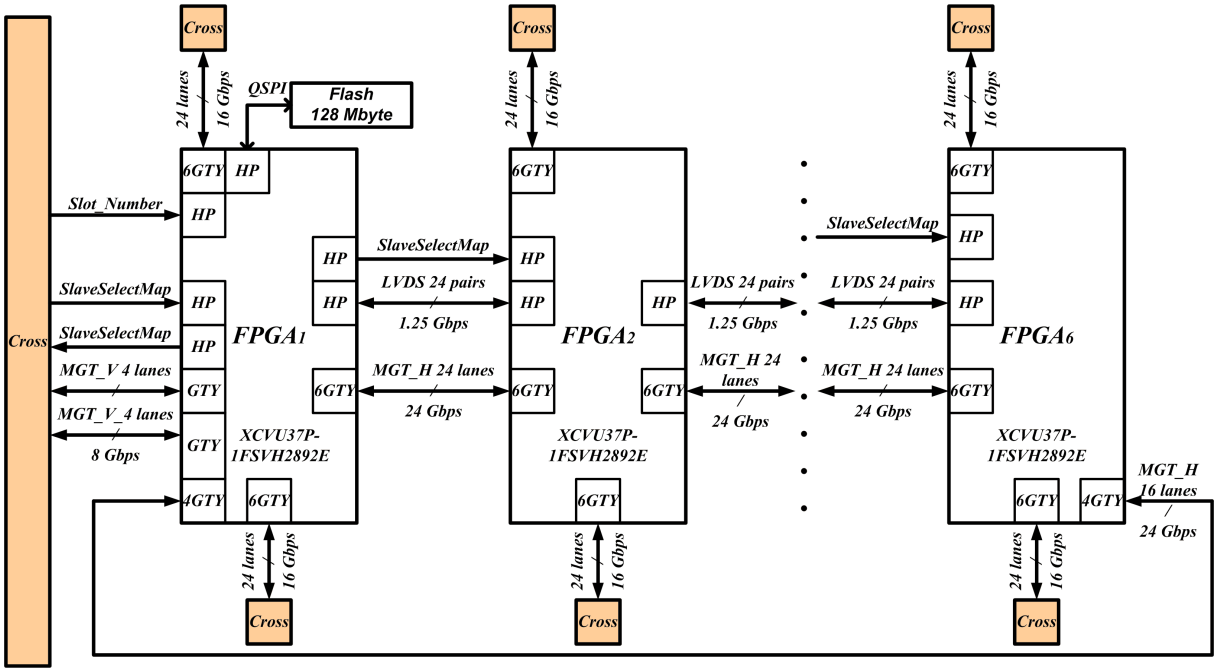

Figure 3. Horizontal interconnect structure of the Arctur RCS computing module

Vertical interconnects between computing modules are implemented using 144 MGT transceivers with data rates up to 16 Gbps per link, providing a full-duplex bandwidth of up to 2.3 Tbps between modules. In addition, dedicated service channels are provided for FPGA configuration, command transmission, and data exchange between the host processor and any computing module, with a bandwidth of 32 Gbps. The Arctur RCS block can also be interconnected with other blocks via optical transceivers, forming full-duplex links with bandwidth up to 1.8 Tbps. The structure of vertical interconnects is shown in Fig. 4.

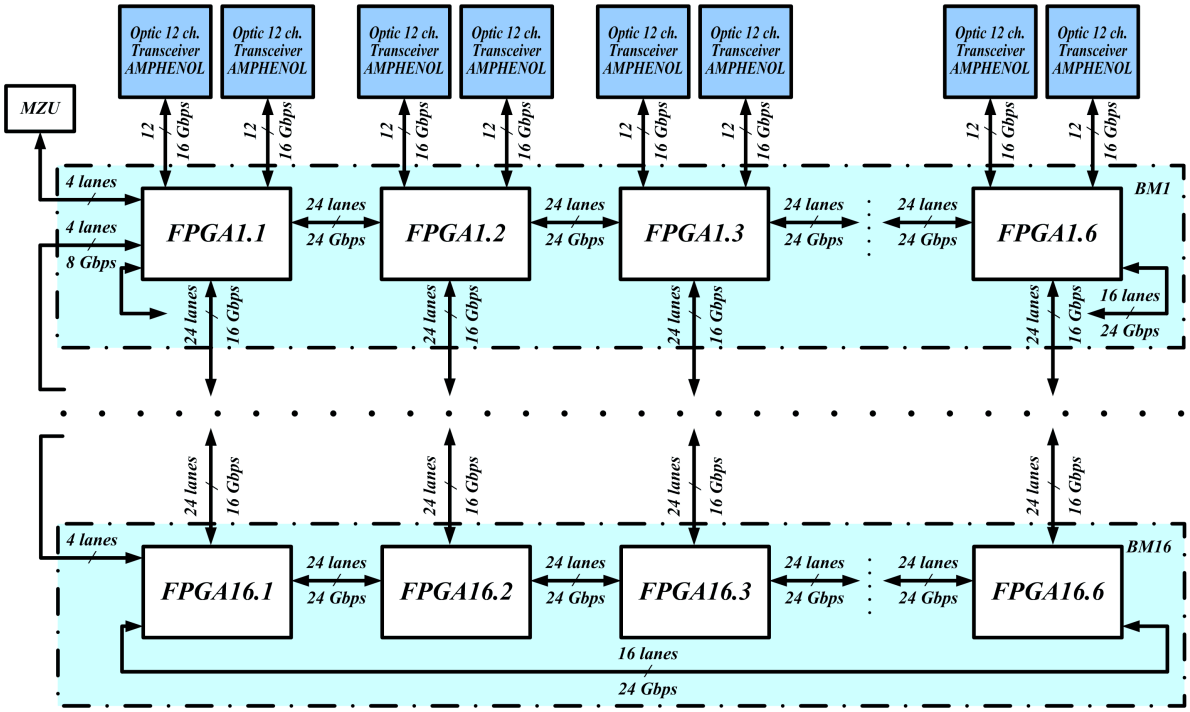

Figure 4. Vertical interconnect structure of the Arctur RCS block

The main characteristics of the Arctur RCS block are presented in Tab. 1.

Table 1. Technical characteristics of the Arctur RCS

Parameter	Value
Number of computing modules	16
Number of XCVU37P FPGAs	96
Peak performance, TFLOPS	610
HBM memory capacity, GB	768
Optical transceivers (12-channel, 12 Gbps)	12
Interfaces	2×1Gb Eth, USB, DisplayPort
Constant power supply voltage, V	380 ± 20
Dimensions, mm	$485 \times 131.5 \times 1344$
Maximum power consumption, kW	25
Limit power consumption (floating-point workloads), kW	16

The theoretical energy efficiency of the Arctur RCS block, defined as the ratio of peak performance to maximum power consumption, is 38 GFLOPS/W. For comparison, a single NVIDIA DGX A100 provides a peak performance of 2496 TFLOPS with the power consumption of 6.5 kW [16], corresponding to a theoretical energy efficiency of 384 GFLOPS/W. Therefore, under peak theoretical estimates, the Arctur RCS block demonstrates lower energy efficiency compared to the DGX A100 block. However, experimental research shows that the computational efficiency of a single Arctur RCS block is approximately five times higher than that of the DGX A100 block for real neural network training problems. Such loss of computational efficiency in tensor systems is caused by low specific performance in solving practical problems. The primary reasons for this include high inter-layer data dependencies and the presence of dataflow discontinuities.

Next, we consider the types of dataflow discontinuities in detail, as well as the methods for their mitigation, enabling significantly higher real performance on the Arctur RCS block compared to DGX-based systems.

2. Methods for Minimizing Dataflow Discontinuities

The efficiency of solving problems both on RCS and tensor (and any other) data flow processing systems is significantly reduced if there are such g-th and $(g+1)$-th fragments of the information graph between which the intensity of data flows Ig of processed data [17] drops to zero. Such a situation is defined as a dataflow discontinuity. In RCS, dataflow discontinuities also affect the entire computational graph, similarly to tensor processing systems, since the structure of the main data dependencies in implementations of problems of neural network training remains unchanged. Therefore, linear scaling of hardware resources does not lead to a proportional increase in achieved performance. These discontinuities must be eliminated or at least minimized. Suppose that for solving a problem F, containing subtasks f_{g-1},f_g, and f_{g+1}, it is necessary to organize pipeline processing of the input vector X_i.

Without loss of generality, we assume that $X_i = \{x_1, x_2, \ldots, x_J\}$ with intensity I_1 is fed to subtask f_{g-1}, and at its output a vector $X'_i = \{x'_1, x'_2, \ldots, x'_J\}$ is formed with intensity I_1 and is passed to f_g and f_{g+1}. At the output of f_g, a variable m_i is formed with intensity I_2 and

is also passed to subtask f_{g+1}, so that the output vector $Y_i = \{y_1, y_2, \ldots, y_J\}$ is formed with intensity I_3. A simplified computing structure of problem F is shown in Fig. 5.

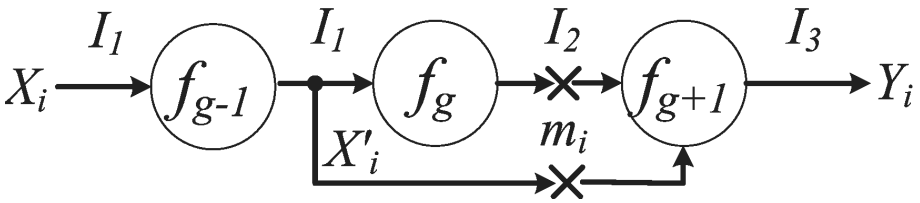

Figure 5. Simplified structure of the problem F

Within the structural organization of calculation paradigm, the computational graph of any problem F is directly mapped onto the RCS hardware resources. In this context, representations of the problem F, its subproblem f_g, or even an individual function, as well as their corresponding computing structures implemented at the system level in the RCS, are considered as mutually consistent levels of abstraction.

Let subtask f_g implement a function of the form $m_i = \frac{1}{J}\sum_{j=1}^{J} x'_j$ and subtask f_{g+1} implement a function of the form $Y_i = \frac{X_i}{m_i}$. Obviously, until all J elements of vector X'_i are processed in f_g, the result m_i cannot be formed; therefore, at this time $I_2 = I_3 = 0$. A dataflow discontinuity arises, and subtask f_{g+1} cannot process vector X'_i during formation of m_i.

The processing time T_{proc} of a set of vectors X_i, for $i = 1, ..., N$, in problem F can be approximated by the following empirical model capturing the effect of dataflow discontinuities:

$$T_{proc} = \sum_{i=1}^{N} \left[(1 + \frac{I_{in} - I_{out}}{I_{in}}) \cdot t(X_i) \right], \tag{1}$$

where $t(X_i)$ is the time for reading X_i,

I_{in} is the intensity of input data entering the pipeline,

I_{out} is the intensity of output data formation from the pipeline.

Since subtask f_{g+1} starts execution only after completion of f_g, and both process vectors of equal dimensions $\|X'\| = \|X_i\| = J$, then for problem F at the moment of reading X_i, the output intensity is equal to $I_{out} = I_3 = 0$. Then, the expression (1) transforms into:

$$T_{proc}(I_3) = 2 \cdot \sum_{i=1}^{N} t(X_i). \tag{2}$$

Therefore, in the presence of a dataflow discontinuity, the processing time of problem F is doubled. Moreover, during calculation of X'_i and m_i, the hardware resources assigned to f_{g+1} are idle, and during calculation of Y_i, the resources assigned to f_{g-1} and f_g are idle. To overcome the dataflow discontinuity between the g-th and $(g+1)$-th fragments of the information graph of the problem F, the transformation shown in Fig. 6 can be performed.

If there exists a function f_g^*, such that the domain of its acceptable values $D(f_g^*)$ is close to the domain $D(f_g)$, then for $Y_i^* = f_{g+1}(f_g^*(f_{g-1}(X_i)))$ and $Y_i = f_{g+1}(f_g(f_{g-1}(X_i)))$ there exists $D(f_g^*) \cap D(f_g)$, where $Y_i^* \approx Y_i$. If $f_{g+1}(f_g^*)$ produces output vector Y_i^* with intensity $I_3^* \approx I_2^* > 0$, the dataflow discontinuity is eliminated. Such cases are referred to as first-type

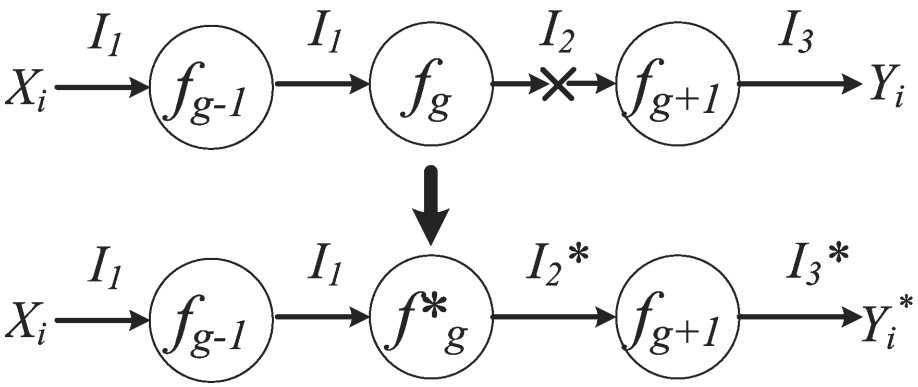

Figure 6. Overcoming dataflow discontinuity in the problem F

dataflow discontinuities. In this case, the pipeline execution with output intensity $I_{out} = I_3^*$ can be implemented.

The transformation $f_g \xrightarrow[D(f_g^*) \cap D(f_g)]{} f_g^*$ in this case enables pipeline data processing in the problem F, reducing the solution time proportional to $\frac{T_{proc}(I_3)}{T_{proc}(I_3^*)}$.

In practice, such substitution is not always possible, since the existence of $f_g^* \approx f_g$ depends on the algorithm of the problem, required accuracy, and data processing velocity. Therefore, in the general case, the influence of dataflow discontinuities can only be reduced.

Suppose that for solving a problem Z, containing subtasks z_{g-1}, z_g, and z_{g+1}, it is necessary to organize the pipeline processing of the input vector X_i. Without loss of generality, assume that X_i is processed sequentially through z_{g-1}, z_g and z_{g+1}, producing intermediate vectors X_i', X_i'', and output vector Y_i with intensities I_1 and I_2, respectively. $X_i' = z_{g-1}(X_i, Y_{i-1})$, $X_i'' = z_g(X_i', Y_{i-1})$, and $Y_i = z_{g+1}(z_g(z_{g-1}(X_i, Y_{i-1}), Y_{i-1}), Y_{i-1})$.

A simplified structure of the problem Z is shown in Fig. 7.

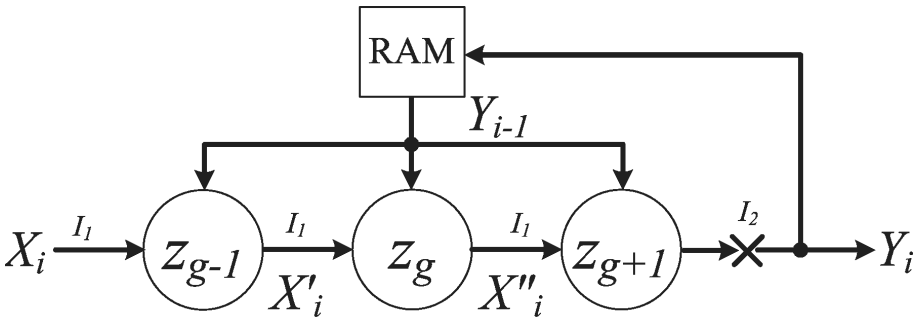

Figure 7. Simplified structure of the problem Z

A dataflow discontinuity arises ($I_2 = 0$), since z_{g+1} cannot update the vector Y_{i-1} until the entire vector X_i has been processed. The processing time T_{proc} for a set of vectors X_i, where $i = 1, \ldots, N$, in the problem Z can be estimated by expression (1).

At $I_2 = 0$, the updating of vector Y_i will begin only after the completion of processing of X_i. In the case where $\|X_i\| \gg \|Y_i\|$, the update time can be neglected and pipeline execution remains efficient, $T_{proc} \to t(X_i)$.

However, when $\|X_i\| \approx \|Y_i\|$, $T_{proc} \to 2 \cdot N \cdot t(X_i)$, meaning that half of the computing resources remain idle during updates of Y_{i-1}.

For time-consuming neural network training problems, the substitution $Z \xrightarrow[D(Z^*)\cap D(Z)]{} Z^*$ leads in practice to a change in architecture and training algorithm, which is unacceptable. Therefore, the effect of this dataflow discontinuity can only be reduced by partitioning RAM into p independent pages and organizing independent p-channel access to write Y_i, as shown in Fig. 8. Such cases are referred to as second-type dataflow discontinuities.

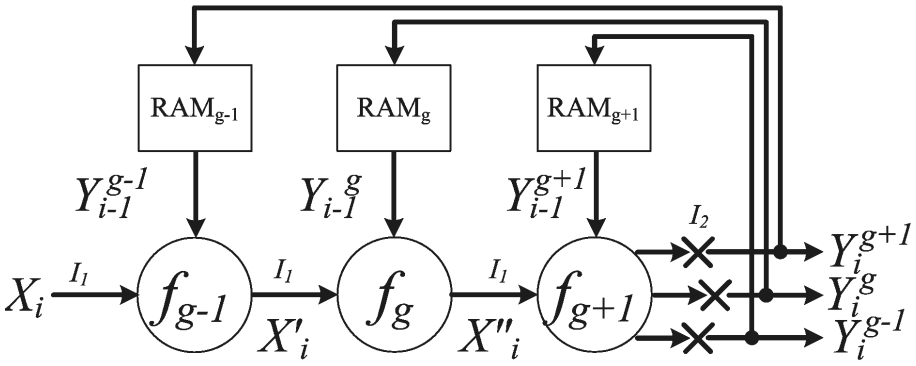

Figure 8. Structure of the problem Z with p-channel RAM access

The presence of a large number of high-speed horizontal and vertical links in the Arctur RCS switching subsystem enables independent multi-channel access to distributed memory in each FPGA. This reduces the write time of Y_i by a factor of p. In this case $T_{proc} \to N \cdot \left[t(X_i) + \frac{t(Y_i)}{p}\right]$. If p is such that $\frac{t(Y_i)}{p} \ll t(X_i)$, then $T_{proc} \to N \cdot t(X_i)$.

The described methods for minimizing first- and second-type dataflow discontinuities under structural organization of calculations improve the RCS performance by reducing the overhead costs of organizing the computing process. However, the efficient implementation of neural data processing on RCS is not possible using existing synthesis frameworks for neural network functional blocks. Existing FPGA frameworks such as FPGA AI Suite (Altera) [18] and Vitis AI (AMD Xilinx) [19] are intended for adaptation and deployment of pretrained neural network models developed in TensorFlow, PyTorch, and ONNX. FINN (AMD Xilinx) is oriented toward automated generation of specialized computing structures for execution of quantized neural networks on FPGAs in low-bit integer representations [20]. Universal frameworks are significantly less developed for the full training cycle of neural network models on FPGAs. Implementation of training, including forward pass, backpropagation, gradient calculation, and parameter updates, requires dedicated RTL-level hardware architecture. Existing solutions such as F-CNN are research-oriented and typically limited to small convolutional neural networks (up to five layers) in integer representation [21]. To improve the efficiency of structural implementation of neural calculations, the authors developed a library of neural functional blocks and a prototype framework for synthesis on RCS of convolutional neural network computing structures, including training structures. Using these developments, the authors synthesized the computing structure for ResNet-50v1.5 neural network for the first time on multi-chip RCS with support for various floating-point formats.

3. CNN Framework for the RCS

Convolutional neural networks (CNNs) are intended for data processing with spatial or locally structured organization, in which feature extraction is performed through convolution

operations using a set of local filters. Due to their ability to form hierarchical feature representations, this class of neural networks is widely used in image analysis problems, including classification, object detection, and segmentation. In addition to image processing, CNNs are used to detect anomalies, identify characteristic patterns, and predict text, sound, and time series processing. The developed library of neural network functional blocks contains all necessary components for designing CNNs on RCS, including: a linear convolution layer, normalization layers (batch, layer, and group normalization), activation functions (ReLU, sigmoid, Softmax), max-pooling layer, global average pooling layer, and fully connected layer. For neural network training on RCS, the library implements the components, required for calculating errors based on cross-entropy, calculating gradients of layers of all types, calculating gradients of normalization and activation, and correcting the weighting coefficients. In addition, the library includes synchronization and dataflow management blocks required for neural network construction on RCS.

All library components are universal for RCS systems with different hardware configurations and resource capacities. Therefore, to ensure the required RCS performance level, the prototype framework enables synthesis of computing structures using different methods for organization of calculations: procedural, pipeline, parallel-pipeline, nested pipeline, and macro-pipeline. Parallelization of calculations or structural reduction can be performed across different dimensions, including number of neurons, convolution kernels, data channels, operations, and other parameters, defined in the reduction transformation methodology [22].

Calculations in the developed functional blocks can be implemented using different numeric formats: FP32 (IEEE754 standard), TF32 (NVIDIA format), NicFP (18-bit format developed by SRC SC & NC), and BF16 (Google format).

Figure 9 shows a typical computing structure of a convolutional layer in training mode, which can be synthesized in the prototype framework using the neural network functional block library.

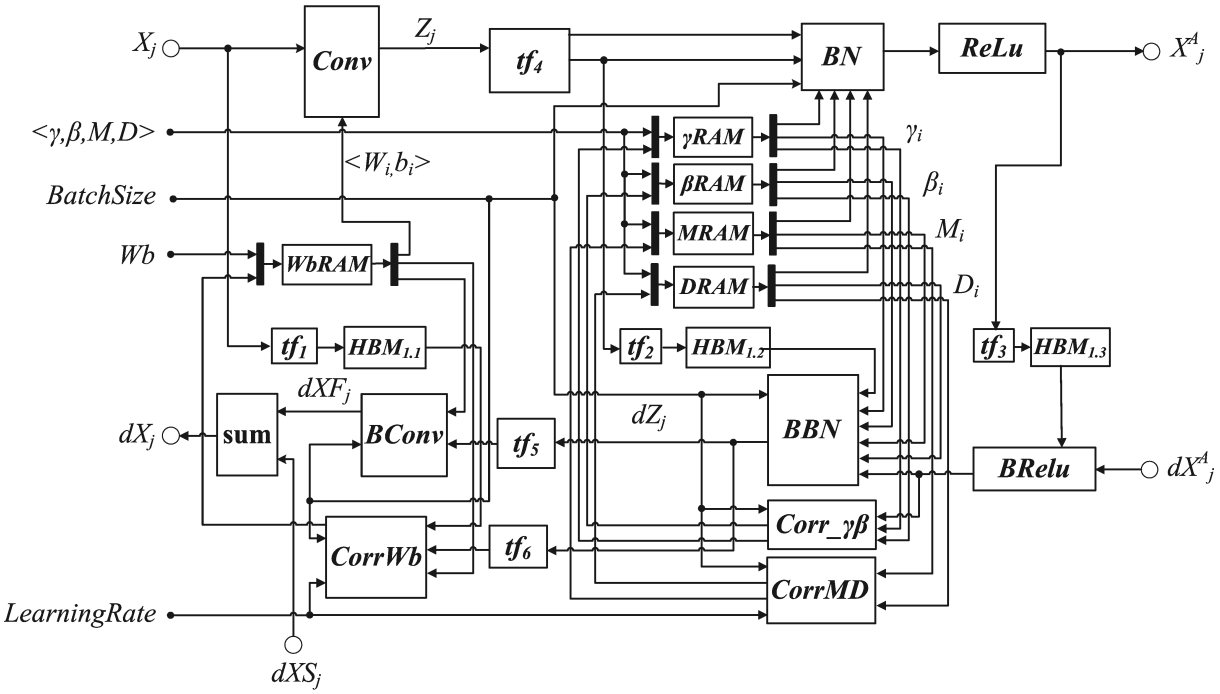

Figure 9. Computing structure of a convolutional layer in training mode

The presented training computing structure includes the following functional blocks for the forward pass: convolution layer ($Conv$), batch normalization block (BN), activation block ($ReLU$), weight memory block W_i and bias memory block b_i ($WbRAM$), scaling coefficient memory blocks γ_i and β_i ($\gamma RAM, \beta RAM$), and mean and variance memory blocks M_i and D_i ($MRAM, DRAM$).

For backpropagation, the computing structure includes: input operand memory block X_i ($HBM_{1.1}$); pre-activation memory block Z_i ($HBM_{1.2}$); activation memory block X_i^A ($HBM_{1.3}$); dataflow transformation blocks tf_1–tf_6 to reorder data flow between connected functional blocks according to the execution order defined by the computing structure; gradient calculation block for activations ($BReLU$); gradient calculation block for pre-activations dZ_i (BBN); scaling coefficient update block ($Corr\_\gamma\beta$); mean/variance update block ($CorrMD$); weight and bias update block ($CorrWb$); input gradient calculation block dXF_j ($BConv$); dXF_j accumulation block with gradients of the dXS_j traversal branch (sum).

Similarly, convolutional layer computing structures for both inference and training on RCS can be implemented with different kernel types (1×1, 1×1/2, 3×3, 3×3/2, 7×7/2, etc.), different numbers of neurons per layer, kernel counts per neuron, and memory access channel configurations. Based on this, CNNs of different architectures can be synthesized, including Sequential networks, Inception networks (parallel concatenation of Sequential branches), Residual networks (accumulation of Sequential branches with shortcut connections), and Dense networks (concatenation of outputs of all previous layers).

The authors' research is established a strict functional relationship between available RCS hardware resources, parallelization and reduction coefficients, the chosen calculation method, data representation format, and the parameters with which convolutional layers and networks can be implemented. For the developed framework prototype, this enabled the creation of a methodology for automated calculation of neural network computing structure parameters.

This enabled a transition from native RTL design of FPGA computing structures to structural-parametric RTL design [23], which significantly accelerates the development of multi-FPGA neural network architectures and reduces synthesis time for a 12-FPGA RCS "Arctur" implementation of ResNet-50v1.5 training from one and a half years to three months.

The developed library and prototype framework enabled synthesis of a computing structure for ResNet-50v1.5 training with pipelined data processing and support for multiple floating-point formats for the first time on a multi-chip RCS.

4. Analysis of the Arctur RCS Efficiency in Neural Network Training Problems

The selection of the ResNet-50v1.5 neural network from the MLPerf benchmark for implementation the training problem on the Arctur RCS hardware platform is based on the fact that ResNet-50v1.5 is a widely used reference model in image classification problems. Its training results are used to compare different computing platforms in terms of performance and energy consumption during the CNN training. In addition, a large number of scientific publications are available that provide experimental data obtained during ResNet-50v1.5 training on tensor computing systems. This makes it possible to perform a reliable analysis of the efficiency of the proposed solutions. The ResNet-50v1.5 training task consists of two subtasks: *Forward pass* and *Backward propagation*. Its computing structure is shown in Fig. 10.

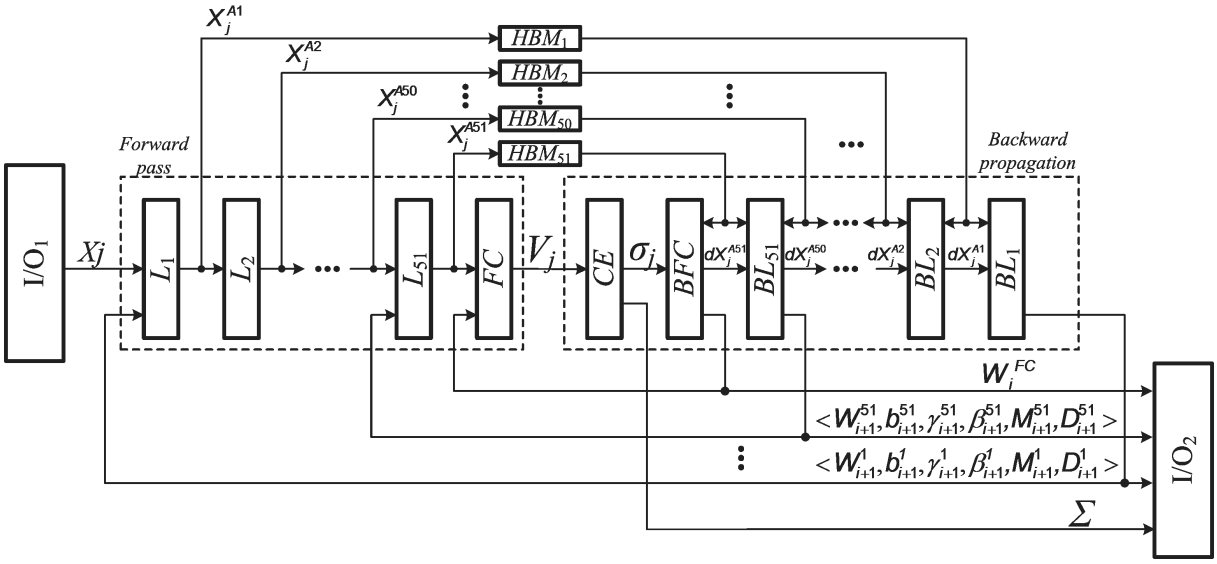

Figure 10. Computing structure of the ResNet-50v1.5 training task

The *Forward Pass* computing structure consists of the functional blocks L_1–L_{51} and FC, implementing the linear convolution layer $conv7\times7/2$, local maximum pooling layer $max\_pool3\times3/2$, linear convolution layers $conv1\times1$, $conv1\times1/2$, $conv3\times3$,$conv3\times3/2$, global average pooling layer $avg_pool7\times7$, as well as the fully connected classification layer $full\_connect1000$. The L_1, L_3–L_{50} linear layer blocks also implement the $BatchNormalization$ and $ReLu$ activation functions. The FC block also implements $soft\_max$ activation function.

The *Backward Propagation* computing structure consists of the functional blocks CE, BFC, BL_{51}–BL_1, implementing the calculation error based on cross-entropy $calc\_\sigma$, gradient calculation with respect to $soft\_max$ input, and gradient calculation with respect to the input and parameters of $full\_connect1000$ layer, as well as gradient calculation for $avg\_pool7\times7$ and $max\_pool3\times3/2$ layers, and gradient calculation for convolution layers.

Before the training process of ResNet-50v1.5, initial values of the packet counter $i = 0$, layer parameters $W_i, b_i, \gamma_i, \beta_i, M_i, D_i$ are initialized in the computing structure. The image counter $j = 0$ is also initialized for images within a batch, and the $BatchSize$ and $LearningRate$ are set.

In *Forward Pass* computing structure, pixel values of image X_j from the batch $Batch_i = \{X_0, X_1, \ldots, X_{BatchSize}\}$ are fed into block L_1. At the output of L_1, an activation stream X_j^{A1} is formed, which is passed to L_2 and simultaneously stored in HBM_1. From L_2, the activation stream X_j^{A2} is passed to L_3 and stored in HBM_2. For all other blocks L_3–L_{51} and FC, data streams are formed similarly and written to HBM_3–HBM_{51}.

In *Backward Propagation* computing structure, the vector of class probability predictions V_j from FC is passed to block CE, where the error vector σ_j is formed. The input of BFC receives the activation stream X_j^{A51} together with vector σ_j. At the output of BFC, the gradient stream dX_j^{A51} is formed, which is passed to BL_{50} together with X_j^{A51} and X_j^{A50}. From BL_{50}, the gradient stream dX_j^{A50} together with X_j^{A50} and X_j^{A49} is passed to BL_{49} and so on for all BL_{48}–BL_1. Simultaneously, during gradient calculation for inputs dX_j^{A51}–dX_j^{A1}, in blocks BFC, BL_{50}–BL_3 and BL_1 for all images in the i batch ($j = 1 \ldots BatchSize$), gradients with respect to parameters are computed: weight coefficients dW_i^{FC}, dW_i^{50}–dW_i^3, dW_i^1, bias coeffi-

cients db_i^{FC}, db_i^{50}–db_i^3, db_i^1, scaling coefficients $d\gamma_i^{50}$–$d\gamma_i^3$, $d\gamma_i^1$, shift coefficients $d\beta_i^{50}$–$d\beta_i^3$, $d\beta_i^1$, as well as moving averages dM_i^{50}–dM_i^3, dM_i^1 and variances dD_i^{50}–dD_i^3, dD_i^1.

After processing $Batch_i$, new values W_{i+1}, B_{i+1}, γ_{i+1}, β_{i+1}, M_{i+1}, D_{i+1} are calculated and overwritten in the parameter memory $WbRAM$, γRAM, βRAM, $MRAM$, $DRAM$ for each layer L_1, L_3–L_{50}. At this stage, $LearningRate$ may also be updated. After that, the images from $Batch_{i+1}$ are fed into L_1, and the process repeats. In CE block the total loss Σ is also calculated, based on which training termination can be decided. Once training is complete, the parameters obtained in *Backward propagation* can be used for image classification.

Experimental research studies of the proposed approach were performed on two Arctur RCS computing modules (12 Xilinx XCVU37P FPGAs). The computing structure of ResNet-50v1.5 training was synthesized using Vivado 2022.1 in NicFP processing format. The clock frequency of the computing structure was 500 MHz. Training of ResNet-50v1.5 was performed on the ImageNet dataset with RGB-image resolution 224×224, $BatchSize = 256$, $LearningRate = 0.01$. The processing velocity of the computing structure was 2200 img/s. This corresponds to an achieved performance of 55.44 TFLOPS on two computing modules or 27.72 TFLOPS per computing module. The peak performance of a computing module in NicFP format is 38.16 TFLOPS. Therefore, the computational efficiency of the computing module in the ResNet-50v1.5 training task is 72.6%.

For comparison, the processing velocity of ResNet-50v1.5 training on a single NVIDIA Tesla A100 80GB GPU in TF32+XLA processing mode is 938 img/s, with the real performance of 23.63 TFLOPS [24]. At the same time, the peak performance of A100 in TF32 mode is 156 TFLOPS [25], which corresponds to a computational efficiency of 15.14% during ResNet-50v1.5 training.

The NicFP (18-bit) and TF32 (19-bit) data representation formats differ only in mantissa length – 9 bits and 10 bits, respectively. The numerical analysis of the accuracy shows an error of about 0.07%. Therefore, it can be argued that the computational efficiency of the Arctur RCS exceeds the computational efficiency of the NVIDIA A100 GPU by almost 4.8 times.

Further scaling of the computing structure of the ResNet-50v1.5 training task on the Arctur RCS block (16×computing module) enabled, due to the structural organization of calculations, a linear increase at image processing velocity to 17600 img/s. The real performance of about 443.5 TFLOPS was obtained with the computational efficiency of 72%.

Based on experimental energy consumption data, obtained during the research, an evaluation of the energy efficiency of the Arctur RCS block was performed in comparison with a single NVIDIA DGX A100 block (8×Tesla A100 80GB GPUs + 2 AMD EPYC 7742 CPUs). The DGX A100 block achieves a performance of 182.6 TFLOPS [24] with the estimated power consumption of about 4.9 kW based on NVIDIA experimental data for the DGX A100 (4×Tesla A100 40GB) node [26]. The energy efficiency of DGX A100 node in this case is estimated at 37.27 GFLOPS/W. The Arctur RCS block achieves a performance of 443.5 TFLOPS with a power consumption of about 11 kW. Therefore, its energy efficiency is 40.3 GFLOPS/W, which is about 8% higher than that of DGX A100 block.

At the same time, NVIDIA A100 GPUs, focused on solving AI problems, do not have linear scalability. Published research results, conducted by NVIDIA [26] and the results of the MLPerf benchmark [27], show that at training the ResNet-50v1.5 neural network an eight-fold increase in the number of A100 GPUs in the computing system leads to a 7.7-fold increase in system performance, while a 128-fold increase results in increase of only 79 times, and a 4000-fold

increase results an increase of only 600 times. This is due to the fact that with an increase in the scaling factor, the impact of the time required to exchange calculation results between system nodes on the overall time to solve the problem increases. Unlike tensor systems, information links between blocks of the computing structure (Fig. 9) on RCS are implemented in hardware, which helps minimize the overhead costs of data exchange.

Theoretical researches show that the computing structure in NicFP format scaled to four The Arctur RCS blocks provides the real performance of about 1.77 PFLOPS. Such performance levels are only achievable on 16 DGX A100 nodes. At the same time, power consumption of four The Arctur RCS nodes is approximately 50 kW, while 16 DGX A100 nodes consume about 78.4 kW.

A single RCS rack, consisting of 16 Arctur RCS blocks, can potentially provide the real performance of 7 PFLOPS (NicFP). For neural network problems such as image classification or training of ResNet-50v1.5, the performance of this rack is approximately equivalent to 12 NVIDIA DGX A100 racks. The power consumption of the Arctur RCS rack does not exceed 0.21 MW, which is 2.2 times lower than that of 12 DGX A100 racks (approximately 0.46 MW).

Conclusion

The main advantage of the structural organization of calculations over the procedural approach is the ability to adapt the RCS architecture to the information graph of the solving problem. To exploit this advantage on the Arctur RCS for neural network data analysis and training problems, methods for minimizing dataflow discontinuities have been developed and experimentally validated. These methods enable the synthesis of pipelined neural network computing structures on RCS with parallelization across both data and iterations.

A library of functional blocks has been developed to create and train multilayer CNNs of various architectures (Sequential, Inception, Residual, and Dense) on RCS. Pipelined data processing is supported for widely used floating-point formats (FP32, TF32, BF16), as well as for the proprietary NicFP format.

A prototype framework has also been developed, enabling the transition from native to structural-parametric RTL design, accelerating the development process of neural network computing structures for RCS by up to six times.

Therefore, a hardware-software foundation has been formed, which can be the basis for the development of domestic reconfigurable high-performance computing complexes that provide high quality and operational solutions to neural network problems. With linear scaling of computing resources, these systems provide near-linear performance scaling. At the same time, the energy efficiency of computing systems based on the Arctur RCS, consisting of dozens of racks, is expected to be significantly higher than that of systems based on tensor accelerators comprising hundreds of racks.

References

1. Lavrentyev, A.S., Solovyev, V.D.: Artificial intelligence: theory and practice. Nauka Publ., Moscow (2020).

2. NVIDIA DGX A100. `https://www.nvidia.com/ru-ru/data-center/dgx-a100/`, accessed: 2026-04-06

3. Cloud TPU system architecture. https://cloud.google.com/tpu/docs/system-architecture-tpu-vm, accessed: 2026-04-06.

4. Huawei Ascend 910 (512 TFLOPS). https://www.ixbt.com/news/2019/08/23/huawei-ascend-910-512-tflops-310.html, accessed: 2026-04-06

5. Xiong, R., Yang, Y., He, D., *et al.*: On layer normalization in the transformer architecture. Proceedings of the 37th International Conference on Machine Learning, 2020, vol. 119, pp. 10524–10533. https://proceedings.mlr.press/v119/xiong20b.html, accessed: 2026-04-08

6. He, K., Zhang, X., Ren, S., Sun, J.: Identity mappings in deep residual networks. Computer Vision – ECCV 2016, vol. 9908, pp. 630–645. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46493-0-8

7. Kasarkin, A.V., Levin, I.I., Sorokin, D.A.: New iteration parallel-based method for solving graph NP-complete problems with reconfigurable computer systems. IOP Conference Series: Materials Science and Engineering 919(1), 052007 (2020). https://doi.org/10.1088/1757-899X/919/5/052007

8. Sorokin, D.A., Matrosov, A.Y., *et al.*: Real-time implementation of the problem of surface-related multiple prediction on RCS. Parallel Computational Technologies, PCT 2019, Kaliningrad, Russia, 2019, pp. 91–98.

9. Sorokin, D.A., Dordopulo, A.I., Levin, I.I.: Implementation of docking for molecular modeling on reconfigurable computing systems. Izvestiya SFedU. Engineering Sciences 7, 217–224. (2011).

10. APEgate project. https://apegate.roma1.infn.it/?page_id=656, accessed: 2026-04-08

11. Kalyaev, A.V., Levin, I.I.: Modular scalable multiprocessor systems with structural-procedural organization of computations. Janus-K Publ., Moscow (2003).

12. Kalyaev, A.V.: Training procedure for a multilayer neuroprocessor network using feedback. Information Technologies 6 (2002).

13. AMD. UltraScale Architecture GTH Transceivers User Guide (UG576). Version 1.7.1. San Jose, 2021. https://www.amd.com/content/dam/xilinx/support/documents/user_guides/ug576-ultrascale-gth-transceivers.pdf, accessed: 2026-04-08

14. Lee, J.-C., Kim, J., Kim, K.W., *et al.*: High bandwidth memory (HBM) with TSV technique. 2016 International SoC Design Conference, ISOCC, pp. 181–182 (2016). https://doi.org/10.1109/ISOCC.2016.7799847

15. Levin, I.I., Doronchenko, Yu.I.: Advanced reconfigurable supercomputer with immersion cooling. Proceedings of the XIV All-Russian Conference on Control Problems, VSPU-2024, pp. 2256–2260 (2024). https://doi.org/10.25699/vspu2024.2256

16. NVIDIA DGX A100 user guide. https://docs.nvidia.com/dgx/dgxa100-user-guide/introduction-to-dgxa100.html, accessed: 2026-04-08

17. Lohrmann, B., Warneke, D., Kao, O.: Nephele streaming: stream processing under QoS constraints at scale. https://doi.org/10.1007/s10586-013-0281-8 (2013), accessed: 2026-04-08

18. Intel Corporation. FPGA AI Suite – AI inference development tools for FPGAs. https://www.intel.com/content/www/us/en/products/details/fpga/development-tools/fpga-ai-suite.html, accessed: 2026-04-08

19. AMD. Vitis AI software for adaptive SoCs and FPGAs. https://www.amd.com/en/products/software/vitis-ai.html, accessed: 2026-04-08

20. AMD Research I& Advanced Development. FINN: dataflow compiler for QNN inference on FPGAs. https://xilinx.github.io/finn/, accessed: 2026-04-09

21. Zhao, W., Fu, H., Luk, W., *et al.*: F-CNN: an FPGA-based framework for training convolutional neural networks. Proc. 27th IEEE Int. Conf. Application-Specific Systems, Architectures and Processors, ASAP 2016, London, UK, 2016. pp. 107–114. IEEE (2016). https://doi.org/10.1109/ASAP.2016.7760779

22. Levin, I.I., Dordopulo, A.I.: Reconfigurable computing systems: resource-independent programming. Southern Federal University Publ., Rostov-on-Don (2025).

23. Chu, P.P.: RTL hardware design using VHDL: coding for efficiency, portability, and scalability. Wiley, Hoboken, NJ (2006).

24. NVIDIA DeepLearningExamples: ResNet-50 v1.5. https://github.com/NVIDIA/DeepLearningExamples/blob/master/PyTorch/Classification/ConvNets/resnet50v1.5/README.md, accessed: 2026-04-09

25. NVIDIA A100 datasheet. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-nvidia-us-2188504-web.pdf, accessed: 2026-04-09

26. NVIDIA AI Enterprise sizing guide (archived). https://web.archive.org/web/20251110193817/https://docs.nvidia.com/ai-enterprise/sizing-guide/latest/appendix.html, accessed: 2026-04-09

27. MLCommons training benchmarks. https://mlcommons.org/benchmarks/training/, accessed: 2026-04-09