already calculated parameter increments are used to generate a new solution already at the next iteration, which reduces the number of sequentially performed calculations.

Procedure A2. For the subsequent optimization of the training algorithm, let us consider in more detail the stage of calculating the functional value.

Let current solution be equal to x, and new solution equal to y. During the calculation of the functional value on the video card for the solution y, the processor simultaneously generates two new solutions $x_1 \in N(\mathrm{x})$ and $y_1 \in N(\mathrm{y})$ where N — set of possible generated solutions from the current. After calculating the value of the functional, the necessity of transition to a new solution is checked. If a new decision is accepted, the next solution to be tested would be y_1 otherwise x_1. This procedure allows you to mask the stage of generating a new solution. Thus, when moving to the next iteration, the functional value for the new solution will be immediately calculated without its explicit generation. However, with this approach, the amount of computation on the processor almost doubles, which can be critical for a small network.

In some cases, procedure 1 will be optimal in other cases, procedure 2, which was discussed in detail in [8].

Procedure A3. Its main idea is to most accurately estimate the speed of the procedures A1 and A2 when solving a specific applied problem. Since the execution of one iteration of the algorithm requires, as a rule, less than one thousandth of a second of time, and the processor cache appears after some time and the timer has a nonzero error, a large number of iterations should be used for an accurate estimate.

Step 1. Running the procedure A1 on ten thousand iterations with the measurement of the running time.

Step 2. Running the procedure A2 on ten thousand iterations with the measurement of the running time.

Step 3. Based on the results of the execution time, a training procedure is selected for the remaining iterations with a shorter running time.

When training neural networks using the annealing method using connected devices, the framework automat- ically evaluates the power of computing devices using the A3 procedure and selects the most efficient parallelization option.

Matrix multiplication on the video card is implemented using two-level parallelization. The first level cyclically distributes the calculations of individual blocks of the resulting matrix between the working groups of the video card. The second level cyclically distributes the calculations of individual elements of the block of the resulting matrix between the cores of a separate group. Since the resulting matrix contains a large number of blocks, there is no problem of irregular loading of workgroups. The number of elements in the block is a multiple of the number of cores in the group, so there is no irregular cores loading. Computing the value of the objective function requires assembling the result on the processor. The calculation of the objective function value is carried out in several stages. At the first stage, shared video memory is created for individual workgroups. At the second stage, each core of the video card calculates a fragment of the objective function value. At the third stage, one core in each working group summarizes the results of the calculations of the cores of the group into memory shared between the groups. Exactly one shared memory cell is allocated for each group. At the fourth stage, the video card transfers the contents of the shared memory to the processor upon completion of work. At the fifth stage, the processor summarizes the results of the groups, because on a video card, synchronization of calculations is possible only within the same working group. The number of working groups is chosen in such a way that there is not too much data transfer and long assembly, and at the same time there is no too long calculation of objective function fragments on the video card.

VIII. EXPERIMENTS

Let's check the developed software package efficiency using the example of solving the color image compression problem.

For the experiments, the STL-10 dataset from Stanford University was used [9].

The dataset contains 100,000 color images with a resolution of 96x96 pixels. The images can show an arbitrary object [10], which makes compressing image data quite a challenge.

For experiments, 8-fold, 16-fold and 32-fold com- pressions were chosen. Lower compression ratios are more efficiently produced using classical compression algorithms, and higher ones are meaningless due to too large losses.

For all degrees of compression, the images were divided into fragments of 4 by 4 pixels. Splitting into smaller fragments leads to a decrease in the compression quality, an increase, in turn, leads to an oversized neural network architecture and requires too much data and computing resources for training. Each individual fragment is compressed by a separate restricted Gauss- Bernoulli Boltzmann machine. For 8-fold compression, the number of neurons in the hidden layer of each machine was 48, for 16-fold - 24, for 32-fold - 24, but to achieve the required degree of compression, another layer of restricted Bernoulli-Bernoulli type Boltzmann machines was added with 48 neurons in the input layer and 24 in the hidden layer.

To train restricted Boltzmann machines with the gradient method, the CD-1 algorithm will be used. The PCD algorithm is more efficient on a small number of iterations [11], however, it is based on the assumption