Features of hardware implementation of Particle Swarm Optimization (PSO) on FPGA

Danila Nikiforovskii, Ivan Deyneka, and Daniil Smirnov

Research Institute of Light-Guided Photonics,
ITMO University, Saint-Petersburg, Russia
{danikiforovskii, igdeyneka, dsmirnov}@corp.ifmo.ru
http://sf.ifmo.ru

Abstract. Swarm Intelligence algorithm family is a widespread tool to solve optimization problems occurring in various areas of computer science and data processing. Nevertheless, the majority of existing solutions are software-based. On the other hand, the very essence of independent agents predisposes to hardware implementation employing parallel calculations. Field Programmable Gate Array (FPGA) is the perfect platform to implement such system. Particle Swarm Optimization hardware implementation using FPGA made by Intel FPGA (formerly Altera) is described in this paper. Both genuine parallel and sequential schemes are considered, as well as different approaches regarding arithmetic operations.

Keywords: FPGA, PSO, swarm, Intel, optimization, hardware, robustness, parallel

1 Introduction

Modern computer science and IT face various optimization problems in different areas, such as curve fitting, data mining, artificial intelligence learning, etc. To solve the problems, where the analytical evaluation of gradients is impossible or have a high computational cost, several meta-heuristic algorithms have been proposed.

The swarm algorithm family is one of such groups. It consists of population-based algorithms with group behavior emerging from individual behavior of independent agents moving in search space to find a solution. As with any other population-based algorithms, the different movement and communication rules and topologies can change the overall behavior of the particles and algorithm as a whole. Even within the group of algorithms with identical ruleset, convergence and the ability to find correct results depends strongly on the choice of algorithm parameters. Because of the probabilistic nature of the agent behavior, there is no universally suitable parameters. The selection of appropriate problem-specific parameters is a problem itself, which requireS additional effort to solve by various methods, including meta-optimization.

One of the swarm intelligence algorithms advantages is the relative ease of the software implementation, which ensues from the basic principle of simple movement rules of individual agents. Therefore, most of the swarm algorithms implementations are software based. It is often sufficient, as it provides a result in a finite time, which is acceptable in lab environment. On the other hand, optimization problems can occur in the area of device control and management. In this case, the timing constraints are presented, which leads to software implementation being unable to find the result in a suitable time and without additional customization, it, generally, damaging the precision.

On the other hand, the multi-agent nature of swarm algorithms allows construction of the hardware system based on the independent agent blocks. Because the agents are, in general, identical, the development of only one block is required, which can be then reused or re-instantiated. There are different approaches and platforms to implement such systems, such as development of ASICs with dedicated computation of the agent mechanics. It is possible to use GPUs instead. However,the use of GPUs implies the conventional desktop computer or at least the CPU. The use of the FPGAs, on the other hand, allows integration of swarm intelligence implementation into another system implemented on FPGAs, which eliminates the need for construction of more complex interfaces between swarm intelligence and other devices. The ability to perform genuine parallel computations with the consideration of affordable cost makes them the perfect computational platform to perform mathematical tasks that imply the repeating operations in independent entities, such as calculating the invert Hessian matrix for every data point in curve fitting problems or cost function calculation for every agent in swarm intelligence algorithms. FPGA based hardware implementation of Particle Swarm Optimization Algorithm is described in this paper.

2 Particle Swarm Optimization

Particle Swarm Optimization is one of the swarm intelligence algorithms. It is the algorithm to solve the multi-dimensional global optimization problem i.e. to find such vector $\boldsymbol{x} : f(\boldsymbol{x}) < f(\boldsymbol{x}'); \forall \boldsymbol{x}' \neq \boldsymbol{x}$, given m-variable function $f(x_1, x_2, \ldots, x_m) : \mathbb{R}^m \to \mathbb{R}$. There are different methods to define the m-variable cost function f. The most obvious one is to define it by an explicit analytical expression. However, in this case, it is possible to calculate the derivatives and the gradient, as well as second derivatives to construct Hessian matrix. Therefore, it is possible to use other methods to solve optimization problems, such as well-known gradient descent method or LevenbergMarquardt algorithm; although Particle Swarm Algorithm can find the appropriate solution in this case, it is less effective.

There are situations, however, where calculation of derivatives is very computationally difficult; in this scenario, swarm intelligence is much more suitable. If the analytical representation of the function is unknown, cost function can be defined as a series of observations or measurements. In this case the coordinatevector $\boldsymbol{x}$ corresponds to the parameters of such measurement; these parameters can reflect both the physical values (e.g. the position of the measurement, voltage levels etc.) as well as specific instructions for the external black box performing the measurement. In this case, there can be no information about gradient in general sense, which renders traditional methods relying on it virtually useless. Particle Swarm Optimization algorithm performs optimization by moving the agents, each representing the candidate solution in search space. Aside from the position vector, the velocity vector is assigned to each agent. In addition, each agent possesses the knowledge of the best previously found personal position and cost function value at that position. The only way of interaction between agents is the best previously found global position, available to every agent. On each iteration of the algorithm, the new value of the velocity vector is calculated for every agent using the expression

$$\boldsymbol{v}^{j+1} = \omega * \boldsymbol{v}^{j} + \phi_p * r_p * (\boldsymbol{p} - \boldsymbol{x}^{j}) + \phi_g * r_g * (\boldsymbol{g} - \boldsymbol{x}^{j}) \tag{1}$$

where ω, ϕ_p and ϕ_g are predefined constant values. r_p and r_g are random values in range of [0;1] with the uniform distribution[1].

After the update of the velocity, the agent moves to a new location according to the simple motion law:

$$\boldsymbol{x}^{j+1} = \boldsymbol{v}^{j+1} + \boldsymbol{x}^{j} \tag{2}$$

Each step concludes with an evaluation of cost function at the new point. If $f(\boldsymbol{x}^{j+1}) < f(\boldsymbol{p}))$, which means that current position is better then previously found personal best, then the value of $\boldsymbol{p}$ is updated, i.e. its old value is replaced with $\boldsymbol{x}^{j+1}$. The final step of the iteration is the evaluation of $\boldsymbol{g}$. If $f(\boldsymbol{p}_{min}) < f(\boldsymbol{g})$, where $\boldsymbol{p}_{min}$ is the overall best of personal values of a swarm , then the value of $\boldsymbol{g}$ is replaced with $\boldsymbol{p}_{min}$.

There are many different variations of the algorithm. The degree of interaction can be changed, there are possibilities to add another terms to eq. (1) and so on. Still, if basic principles of PSO remain unchanged, the conclusions of this article can be applied to any variation of PSO. The basic version of PSO, based strictly on equations (1) and (2) is described in this article.

3 FPGA-based hardware implementation

There are several key points in constructing the hardware-based implementation of Particle Swarm Algorithm. The main advantage is, as aforementioned, the ability to use parallel computing[3]. Since each agent is independent, there are possibilities to implement the algorithm using both genuine parallel and sequential approach.

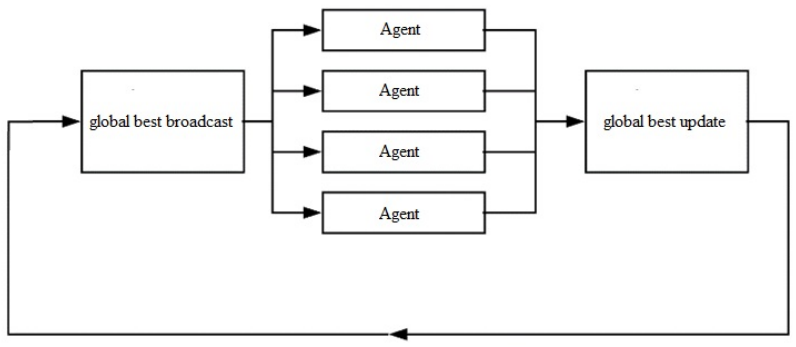

Fig. 1: Genuine parallel system scheme.

The genuine parallel system is depicted on Figure 1. Since the only way of interaction is broadcast value g-vector, each of the agent blocks can run independently, which can vastly improve performance. In addition, since these blocks are identical as well, single developed block can be re-instantiated. However, each block has to include the evaluation of optimization function, which can be computationally costly, as well as a random number generator and an arithmetic system to perform multiplication and addition. Moreover, as all agents are independent, the values of x, v and p should be stored inside each block to allow simultaneous access. Therefore, memory blocks cannot be employed to store these values. Tightly coupled memory should be used instead.

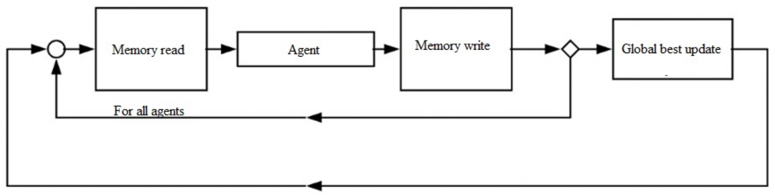

Fig. 2: Sequential system scheme.

The system built with the sequential approach is depicted on Figure 2. Parameters of agent (x, v, p) are stored in memory. After the processing in the agent block and evaluating the cost function, the updated parameters are written back, and the next agent can be processed. This is possible because agents are independent, but instead of implementing the copies of agent blocks, the system is reduced to one block. Both embedded FPGA blocks and any external memory device can be used, as there are no need for simultaneous access. In mdimensional optimization problem that employ n agents, there is need to store three values (x, v, p) per dimension per agent, one value of $f(\boldsymbol{p})$ per agent, one value of g per dimension, and one value of $f(\boldsymbol{g})$.

Therefore, total number of bits needed to implement PSO is

$$((3m + 1)n + m + 1)b, \tag{3}$$

where b is number of bits per number. For instance, 128 agents in a 6-dimensional optimization problem would demand 76.2 Kbit in single precision. As with other components, there are only one instance of cost function evaluation block. If the evaluation of cost function involves any external device, there is no possibility to implement simultaneous evaluation of cost function, so sequential approach presents the only viable option.

As shown in equations (1) and (2), the only operations employed are addition and multiplication. Therefore, there are two different strategies considering the implementations of arithmetic blocks. The first one involves the use of fixed-point or integer arithmetic, which demands the quantization of search space. In this case, fixed-point coordinates correspond to address of a single cell. However, if it is impossible to perform evaluation of the cost function with just integer arithmetic operations, the integer values of candidate solution point coordinates should be converted back to floating-point values every time the cost function is evaluated. The transformation of the cost function is also necessary to adjust to changed scale of variables, e.g. function $f(x_1, x_2) = x_1^2 + 2x_2; x_1, x_2 \in [-1, 1]$ should be transformed into $f'(x_1', x_2') = (\frac{x_1'}{32767})^2 + 2\frac{x_2'}{32767}; x_1', x_2' \in [-32767, 32767]$.

To perform the initial seeding as well as the calculations according to equation (1), several stochastic values should be generated. In fixed-point strategy, the initial seeding, i.e. generating the random values in search space is still integer and can be implemented by any widespread method, e.g. such as linear feedback shift registers (LFSRs)[4]. The values of r_p and r_g however, are distributed in range of $[0; 1]$, which goes against the integer approach. On the other hand, coordinates of the products $r_p(\boldsymbol{p} - \boldsymbol{x})$ and $r_g(\boldsymbol{g} - \boldsymbol{x})$ are distributed in range of $[0; (\boldsymbol{p} - \boldsymbol{x})]$ and $[0; (\boldsymbol{g} - \boldsymbol{x})]$ correspondingly, which can be implemented using LFSRs as well. Parameters ω, ϕ_p and ϕ_g which, in general, might be real values, are still constants, and implementation of the multiplication could be reduced to the set of integer addition, bit shifting (for division by the power of 2) and integer multiplication. Fixed-point approach definitely leads to precision loss, as solution can not be improved after the cell containing optimal value is found. Still, many optimization problems do not require precise solution, as good enough solution acquired in reasonable time is considered better than precise solution that could be acquired only in infinite time. If the precision is insufficient, the second round of optimization can be performed, with new search space being the neighborhood of the preliminary solution point.

Despite fixed-point approach is more effective considering intrinsic PSO operations, there can be no overall advantage, because there is need to convert values. Cost function itself may contain computational-costly operations such as exponentiation, logarithms and so on, which cannot be reduced to fixed-point calculations without the significant loss of precision.

The other way is to use floating-point arithmetic in particle swarm algorithm intrinsic mechanic. In this scenario, there is no need to convert values. If there is any CPU present in system, the use of floating-point approach allows native interchange of data between CPU and PSO block. However, this strategy implies the use of specialized arithmetic blocks for addition and multiplication; therefore, area demands increase.

In context of random number generation, the only challenge floating-point strategy poses is the generation of uniformly distributed value. The LFSR bus output is uniformly distributed when treated as integer, but distribution of floating-point number obtained by straightforward reading of LFSR bus output is far from uniform. Several ways of generating the floating-point random numbers are available, considering the possibility of fixing the output range of generator. For example, certain LSFR bits can be used as a significand, and the exponent value being adjusted to make the resulting value just below the range upper threshold.

To prevent precision-related issues linked to representation of the values from affecting the behavior of the swarm, normalized search space approach can be utilized. It is possible to scale the variables in the cost function in such way that search space is transformed to $[-1;1]^m$; in this case all agents are moving with velocities of same order across all dimensions.

As PSO is a method to solve multi-dimensional problems, to build the system in genuine parallel fashion, there is need to implement independent calculation subsystem not only for every agent, but for every dimension within agent block as well. Equations (1) and (2) consist of 5 multiplications and 5 additions per agent per dimension in it, which vastly increase the area demands even in fixed-point approach. As it seems impractical, a finite state machine (FSM) based scheduling can reduce this to 1 multiplication and 1 addition per agent. In case of sequential strategy, i.e. when only one agent block is implemented, it is possible to construct parallel scheme regarding dimensions only.

The cost function evaluation can be a very computationally costly task. For example, the evaluation of the sum of the squares of residuals

$$\textstyle\sum_{i=0}^{n}(\Phi(x(i),\beta_0,\beta_1,\ldots,\beta_m)-y(i))^2 \tag{4}$$

where $x(i)$ and $y(i)$ are the coordinates of data points, β_i - independent parameters of target function, demand calculations of $\Phi(x)$ n times for every candidate parameter set. $\Phi(x)$ itself can be quite complex and can contain multiple entries of resource intensive functions as exponentiation, trigonometric functions etc. For example, it takes 21 clock cycles to perform logarithm function calculation, commonly used in statistic-related tasks, in Intel FPGA dedicated IP-core in single precision, versus 7 clock cycles to perform addition. In the majority of applications, the evaluation of cost function demands more time than the operations of PSO. Therefore, the performance increase achieved by construction of parallel PSO system is relatively small especially considering the FPGA area demands. Still, the absence of dependencies between candidate solutions allows the implementation of multiple blocks of cost function evaluation. The major setback is, again, the FPGA area requirements, as genuine parallel dataflow demands n independent cost function evaluation blocks. It is possible to construct a system with fewer cost function blocks, however, dividing the agents into subsets assigned to separate blocks and employing the scheduling system inside the subset.

4 Validation Examples

Using the considerations described above, the hardware version of Particle Swarm Optimization was implemented on the Intel FPGA EP4CE115F29C7N (Cyclone IV) platform, which is a part of Terasic DE2-115 development board. Consecutive FSM-based scheduling and genuine parallel schemes have been implemented, as well as mixed scheme with partial parallelism.

4.1 Parallel strategy, fixed-point approach

The multi-dimensional sphere function

$$\sum_{i=0}^{m}(x_i - a_i)^2 \tag{5}$$

have been chosen as a cost function in the implementation of parallel PSO system with fixed-point approach. As there is no need for additional arithmetic blocks neither for PSO nor for cost function evaluation, math operations were implemented using standard VHDL operators. The results of the experimental run are shown on Figure 3.

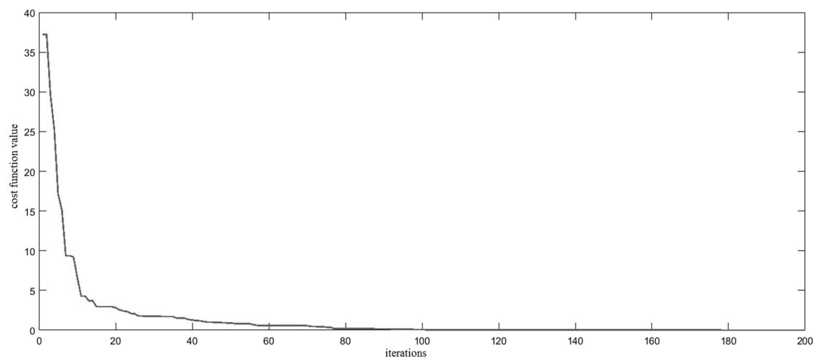

Fig. 3: Best achieved cost function value on selected iteration in genuine parallel fixed-point PSO system

As could be seen on the Figure 4 , the large portion of FPGA is used by the system, although sphere function is relatively easy to implement. Least squares

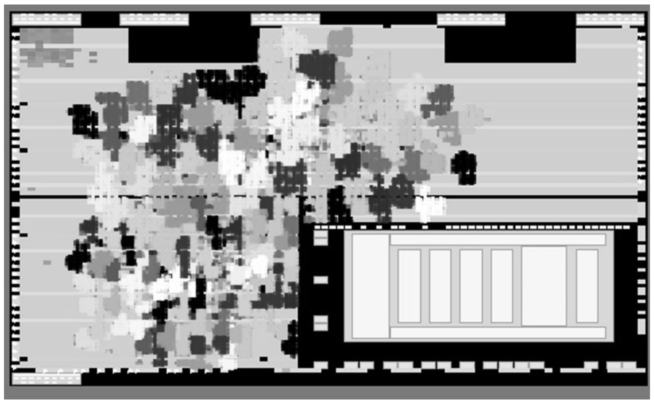

Fig. 4: FPGA resources utilized by hardware implementation of genuine parallel fixed-point PSO system. Different shades correspond to different instances of agent block.

method relies on equations that can be considered multidimensional sphere equations. This allows the use of the genuine parallel fixed-point PSO system to solve curve fitting problems.

4.2 Sequential strategy, floating-point approach

The multi-dimensional Rosenbrock function

$$\textstyle\sum_{i=0}^{m/2}(100(x_{2i-1}^2 - x_{2i})^2 + (x_{2i-1} - 1)^2 \tag{6}$$

have been chosen as a cost function in the implementation sequential PSO system with floating-point approach[2]. The arithmetical operations, both for PSO and for evaluation of cost function, were implemented using Intel FPGA floating-point arithmetic IP-cores. Results of hardware test are shown on the Figure 5. Algorithm converges after 223 iterations. Behavior of the hardware system does not diverge from software modeling, and all of the launches ended with an optimal value founded. As seen on the Figure 6, the area of the FPGA employed by sequential system is considerably smaller than area employed by parallel system, as expected, at expense of calculation speed. To estimate the calculation time needed to evaluate cost function, behavioral simulation was carried out in the ModelSim environment.

5 Conclusion

Different aspects of FPGA hardware implementation of particle swarm algorithm are described in this paper. Both genuine parallel and sequential schemes of PSO are viable and can be used to solve optimization problems. Intel FPGA EP4CE115F29C7N (Cyclone IV) platform was used for implementation.

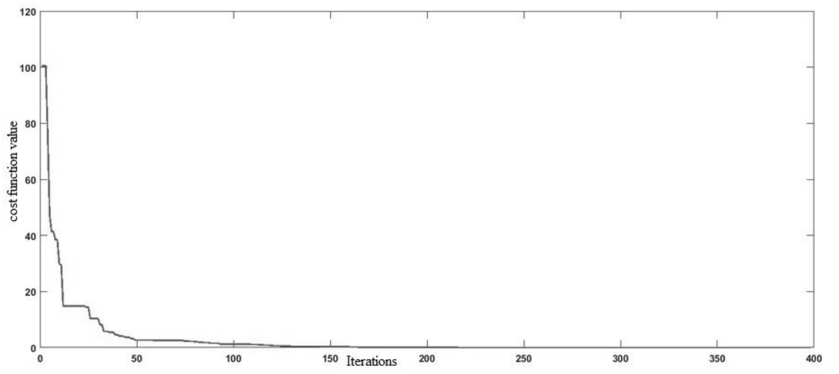

Fig. 5: Best achieved cost function value on selected iteration in sequential floating-point PSO system.

Fig. 6: FPGA resources utilized by hardware implementation of sequential floating-point PSO system.

Both simulation and hardware testing have verified the efficiency of the hardware implementation. Results achieved by testing correspond to those achieved by preliminary modelling. Achieved values are the global minimums of the benchmark functions used in testing.

The acceleration of the algorithm may be achieved by employing the parallel calculations of vectors v and x as well as cost function value. There is expected trade-off between FPGA area demands and performance boost achieved by parallel computing. If the problem-specific cost function is computationally costly, e.g. curve fitting problems or data mining problems, it takes the largest portion of computation time to evaluate the value of cost function. Therefore, there is no need to employ parallel scheme to boost the performance of agent computing blocks. Instead, parallel blocks of cost function evaluation should be implemented, if possible, to achieve performance increase. It takes 964 clock cycles to perform one iteration of PSO in sequential hardware scheme. Algorithm converges after 223 iterations (data averaged based on 50 runs), or 2.15 ms. Thus, even the sequential hardware implementation of Particle Swarm Algorithm is faster than software implementation due to the use of dedicated arithmetic blocks. Hardware implementation of the PSO on FPGA allows the use of the system with other devices, namely CPU portions of SoCs (systems on chip) or soft-processors, which in turn allow construction of complex control and management devices with built-in PSO system.

Acknowledgments. This work was supported by the Ministry of Science and Higher Education of the Russian Federation (The unique identifier of the project: goszadanie No. 8.3134.2017/4.6).

References

1. Calazan, R.M., Nedjah, N., Mourelle, L.M.: Parallel coprocessor for PSO. International Journal of High Performance Systems Architecture (4) (2011) 233240.
2. Tang, K., Yao, X., Suganthan, P.N., MacNish, C., Chen, Y.P., Chen, C.M., Yang, Z.: Benchmark functions for the CEC 2008 special session and competition on large scale global optimization. Tech. rep. (2007)
3. Calazan, R.M., Nedjah, N., Mourelle, L.M.: A hardware accelerator for Particle Swarm Optimization Applied Soft Computing, Elsevier (2013)
4. Press, W., Teukolsky, S., Vetterling, W.; Flannery, B.: Numerical Recipes: The Art of Scientific Computing, Third Edition. Cambridge University Press. (2007).