2015 International Symposium on Nonlinear Theory and its Applications
NOLTA2015, Kowloon, Hong Kong, China, December 1-4, 2015

Design of a Processor System for Particle Swarm Optimizers

Hidehiro Nakano† and Arata Miyauchi†

†Department of Computer Science, Tokyo City University
1–28–1, Tamazutsumi, Setagaya-ku, Tokyo, 158–8557 Japan
Email: {nakano, miyauchi}@ic.cs.tcu.ac.jp

Abstract—In this paper, a processor which realizes the parallel processing of Particle Swarm Optimizer (PSO) is proposed. As the main operations in PSO, there are initialization of particles, calculation of evaluation values, update of the best solution information, update of velocity and position vectors, and so on. In our processor, these operations are executed by parallel circuits. Also, in order to adapt various objective functions, the evaluation values are calculated by software program using a RISC-type processor. Performing the experiments, the results for the evaluation of the circuit scale by logic synthesis and for the measurement of the clock cycles by HDL simulator are shown.

1. Introduction

The Particle Swarm Optimizer (PSO) is one of optimization algorithms, which is classified into swarm intelligence [1]. In PSO, particles located in a solution space share the solution information to each other, and find a design variable vector as a solution which minimizes or maximizes an objective function. Each particle has a velocity vector and a position vector (corresponding to a design variable vector), and memorizes the best solution information in its search process. Also, all the particles share the best solution information in their search process to each other. Based on the information, each particle repeats to update the velocity and position vectors and the best solution information. PSO can be executed by simple arithmetic operations, and can efficiently provide good solution candidates.

On the other hand, recently, engineering systems have become large-scale and complex. Since the solution space can be significantly large in the design problems for these systems, it is difficult to search acceptable solutions by using the small number of particles. Therefore, this paper considers effective search algorithms by using the large number of particles. The basic PSO with the large number of particles can easily trap into undesirable local solutions. However, it is possible to overcome such a problem by introducing the following approaches: (1) the PSO algorithms with network topologies [2]-[7], and (2) the PSO algorithms with multi-swarm structure [8]-[11].

In this paper, the parallel processing of PSO is considered. The operations to each particle are independent except for sharing the best solution information. Also, the operations to each component in velocity and position vectors are independent. That is, it can be said that the operations in PSO have significantly high parallelism. Then, a processor which realizes the parallel processing of PSO is proposed. In our processor, the high-speed execution of the PSO algorithm is possible. Performing the experiments, the results for the evaluation of the circuit scale by logic synthesis and for the measurement of the clock cycles by HDL simulator are shown.

2. Basic PSO algorithm

In this section, the basic PSO algorithm is explained. Let us consider that an optimization problem with D design variables is solved by N particles. At the search iteration t, each particle (the ith particle) has a velocity vector $\mathbf{v}_i^t = (v_{i1}^t, v_{i2}^t, \cdots, v_{iD}^t)$ and a position vector $\mathbf{x}_i^t = (x_{i1}^t, x_{i2}^t, \cdots, x_{iD}^t)$, and memorizes the personal best solution (Pbest) $\mathbf{p}_i^t = (p_{i1}^t, p_{i2}^t, \cdots, p_{iD}^t)$ in its search process. Also, all the particles share the global best solution (Gbest) $\mathbf{g}_i^t = (g_{i1}^t, g_{i2}^t, \cdots, g_{iD}^t)$ in their search process to each other.

The basic PSO algorithm for minimizing an objective function $f(\mathbf{x})$ is described by the followings.

Step 0: Preparation

Set the total number of particles N, the parameters of the particles (w, c_1, c_2), and the maximum search iteration t_{max}.

Step 1: Initialization

Let $t = 1$. Initialize velocity vector $\mathbf{v}_i^1$ and position vector $\mathbf{x}_i^1$ for all i by random numbers. Initialize Pbest $\mathbf{p}_i^1$ for all i and initialize Gbest $\mathbf{g}^1$ by the following equations.

$$p_{ij}^1 = x_{ij}^1, \quad i = 1 \sim N,\ j = 1 \sim D \tag{1}$$

$$k = \arg\min_i f(\mathbf{p}_i^1) \tag{2}$$

$$g_j^1 = p_{kj}^1, \quad j = 1 \sim D \tag{3}$$

Step 2: Update velocity and position vectors

Update the velocity vector $\mathbf{v}_i^{t+1}$ and the position vector $\mathbf{x}_i^{t+1}$ by the following equations.

$$v_{ij}^{t+1} = wv_{ij}^t + c_1 r_1 (p_{ij}^t - x_{ij}^t) + c_2 r_2 (g_{ij}^t - x_{ij}^t) \tag{4}$$
$$i = 1 \sim N,\ j = 1 \sim D$$

$$x_{ij}^{t+1} = x_{ij}^t + v_{ij}^{t+1} \tag{5}$$
$$i = 1 \sim N,\ j = 1 \sim D$$

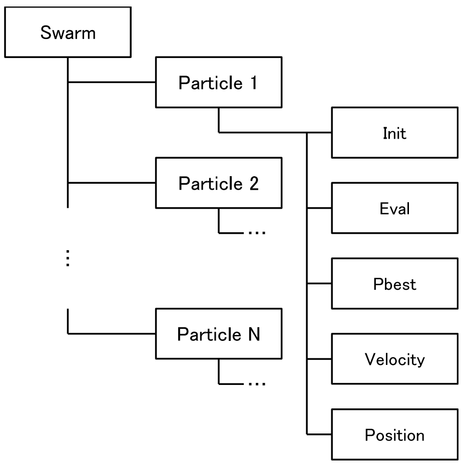

Figure 1: The structure of particle modules

where r_1 and r_2 in Eq. (4) are the uniform random numbers in $[0, 1]$.

Step 3: Update the best solutions

Update Pbest $\mathbf{p}_i^{t+1}$ and Gbest $\mathbf{g}^{t+1}$ by the following equations.

$$p_{ij}^{t+1} = \begin{cases} x_{ij}^{t+1}, & \text{if } f(\mathbf{x}_i^{t+1}) < f(\mathbf{p}_i^t) \\ p_{ij}^t, & \text{otherwise} \end{cases} \quad (6)$$
$$i = 1 \sim N,\ j = 1 \sim D$$
$$k = \arg\min_i f(\mathbf{p}_i^{t+1}) \quad (7)$$
$$g_j^{t+1} = p_{kj}^{t+1} \quad (8)$$
$$j = 1 \sim D$$

Step 4: Judgment of termination

If $t \neq t_{max}$, let $t = t + 1$ and go to Step 2.

PSO can be executed by the simple arithmetic operations, and can efficiently provide good solution candidates for various optimization problems.

It should be noted that the operations in PSO have significantly high parallelism. The operations of $\mathbf{v}_i^{t+1}$ and $\mathbf{x}_i^{t+1}$ in Step 2, and of $\mathbf{p}_i^{t+1}$ in Step 3 are independent for i and j. The operations of $\mathbf{v}_i^1$, $\mathbf{x}_i^1$ and $\mathbf{p}_i^1$ in Step 1 are also independent for i and j. In addition, the operations of $f(\mathbf{x}_i^{t+1})$ and $f(\mathbf{p}_i^t)$ are independent for i, and the operations of $\mathbf{g}^{t+1}$ are independent for j. That is, the operations in PSO have significantly high parallelism. By implementing multiple functional units in parallel, the high-speed execution of the PSO algorithm is possible.

3. Design of a PSO processor

Figure 1 shows the structure of the particle modules. The operations in this processor is based on the single precision floating point number. In the figure, "Particle 1" ~ "Particle N" are the particle modules, and "Swarm" is the hub

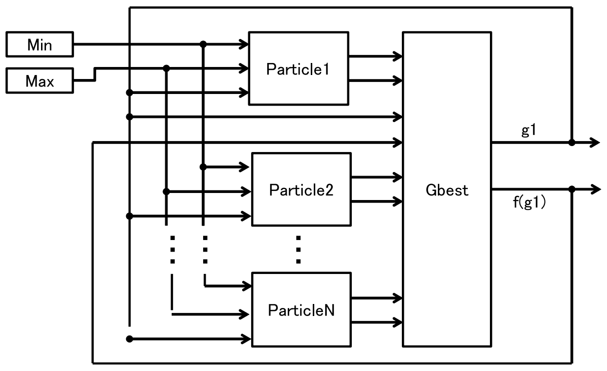

Figure 2: "Swarm" module

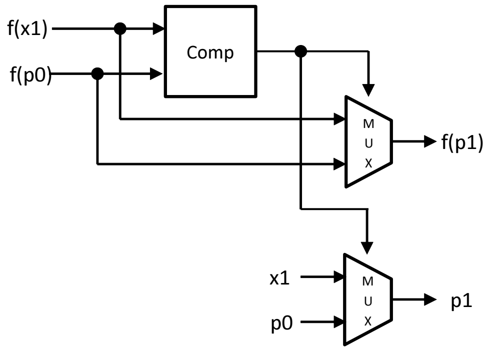

Figure 4: "Pbest" moudule

module to connect all the particle modules. Each particle module has the lower layer modules which correspond to the PSO algorithm explained in the previous section. Each particle module can operate in parallel; the parallel processing can be easily realized.

Figure 2 shows the block diagram of the "Swarm" module. In the figure, "Min" and "Max" are the minimum and maximum value registers, respectively. They are used in the initialization process. "Gbest" is the module which updates Gbest $\mathbf{g}^t$. This module outputs the values g1 (=$\mathbf{g}^t$) and f(g1) (=$f(\mathbf{g}^t)$).

Figure 3 shows the block diagram of the "Init" module. In the figure, "Mseq" is the M-sequence generator to be used as a pseudo-random number generator, "x/v mem" is the memory module for $\mathbf{x}_i^t$ or $\mathbf{v}_i^t$, and "dim" is the counter which holds the design variable index j. "Init" module generates random numbers and stores these values in "x/v mem" as follows:

$$\begin{array}{rcll} r_1 & \leftarrow & [1, 2) & ; +1.rrr\cdots r_{(2)} \times 2^0 \\ r_2 & \leftarrow & [0, 1) & ; r_1 - 1 \\ r_3 & \leftarrow & [0, \text{Max} - \text{Min}) & ; r_2 \times (\text{Max} - \text{Min}) \\ \text{mem} & \leftarrow & [\text{Min}, \text{Max}) & ; r_3 - \text{Min} \end{array}$$

where $rrr\cdots r$ is the 23-bit random significand value. Since this module has the pipeline structure for the design variable index j, the clock cycles of all the operations are proportional to the number of design variables D.

Figure 4 shows the block diagram of the "Pbest" module.

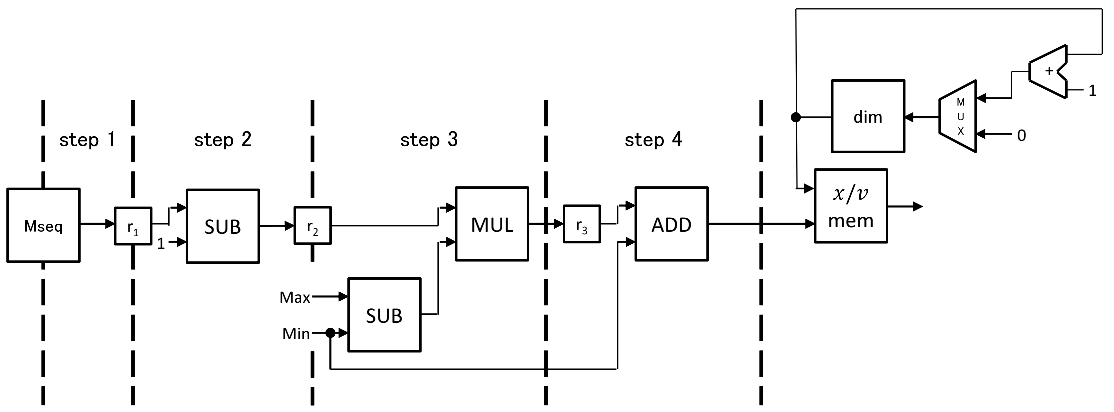

Figure 3: “Init” module

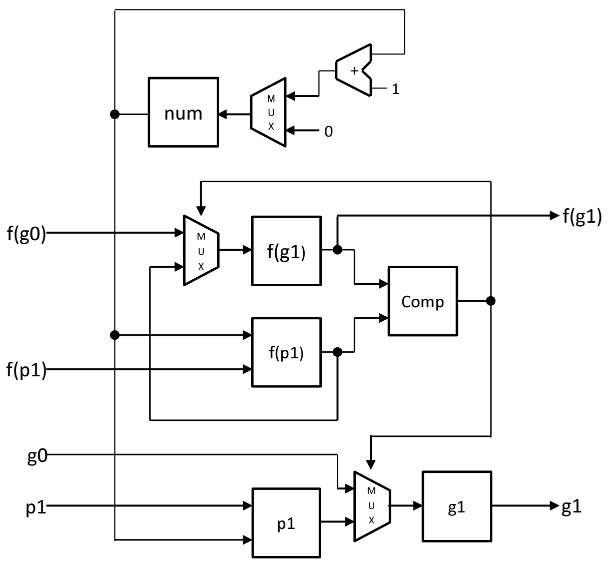

Figure 5: “Gbest” module

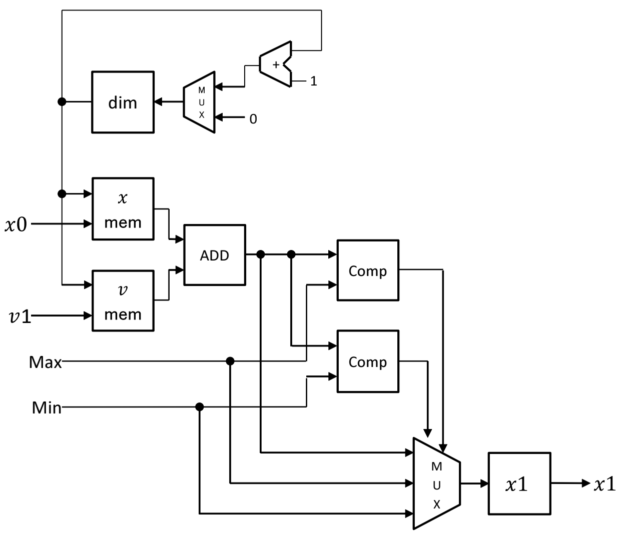

Figure 6: “Position” module

The values f(x1) (=$f(\mathbf{x}_i^{t+1})$) and f(p0) (=$f(\mathbf{p}_i^t)$) are compared. Then, the values f(p1) (=$f(\mathbf{p}_i^{t+1})$) and p1 (=$\mathbf{p}_i^{t+1}$) are output.

Figure 5 shows the block diagram of the “Gbest” module. In the figure, “num” is the counter which holds the particle index i. The values f(g0) (=$f(\mathbf{g}^t)$) and f(p1) (=$f(\mathbf{p}_i^{t+1})$) are compared. Then, the values f(g1) (=$f(\mathbf{g}_i^{t+1})$) and g1 (=$\mathbf{g}_i^{t+1}$) are output.

Figure 6 shows the block diagram of the “Position” module. From the “v mem” and “x mem”, each component of the velocity and position vectors are sequentially read out by using “dim” as the memory address, and these values are added. Then, the added values are sequentially output. Note that the boundary value Min or Max is output if an added value exceeds the boundary.

The “Velocity” module is desined by the 4-stage pipeline structure for the design variable index j, as shown in Table 1. By the above operations, the same results by using Eq. (4) are obtained. Since this module has the pipeline structure for the design variable number j, the clock cycles of all the operations are proportional to the number of design variables D.

Table 1: Pipeline stage in the “Velocity” module

Stage 1:	$k_{11} = w \cdot v_{ij}^t, \quad k_{12} = c_1 \cdot r_1, \quad k_{13} = c_2 \cdot r_2$ $k_{14} = p_{ij}^t - x_{ij}^t, \quad k_{15} = g_j^t - x_{ij}^t$
Stage 2:	$k_{21} = k_{12} \cdot k_{14}, \quad k_{22} = k_{13} \cdot k_{15}$
Stage 3:	$k_3 = k_{21} + k_{22}$
Stage 4:	$k_4 = k_{11} + k_3$

The “Eval” module is constructed by a RISC-type MIPS pipeline processor including the single precision floating point number units. By this design, various benchmark functions can be easily executed by software.

4. Experiments

The PSO processor is described by Verilog-HDL, and the performances are evaluated. The parameters of each particle are fixed to $w = 0.9$, $c_1 = c_2 = 1.0$. Altera Quartus II is used for the logic synthesis, and ModelSim-Altera is used for the logic simulations. Altera Cyclone IV EP4CE30F23I7N is selected as the target device.

In order to perform basic circuit operation verification, only 2 particles are implemented. Table 2 shows the number of the Logic Elements (LEs) for each module. Note that the LEs of higher layer modules include those of lower layer modules. "I/O" includes the control modules of input and output devices. "Top" is the top module which connects the "Swarm" and "I/O" modules.

The "Velocity" module are designed by the pipeline structure with 9 single precision floating point number functional units. Therefore, the LEs of this module are dominant. The "Particle" module operates based on the PSO algorithm and has the exclusive operations between the lower layer modules. Therefore, by sharing their functional units, the total LEs can be reduced further.

Next, the execution time of the PSO processor is considered. As an example, the clock cycles of the "Velocity" module are focused on. From Eq. (4), 9 arithmetic operations are executed in updating a single component of the velocity vector. The clock cycles of the velocity update in serial processing are given by $CC_0 = 9NDL$, where N, D and L are the number of particles, the number of design variables and the maximum latency of the functional units, respectively. On the other hand, the PSO processor has multiple "Particle" modules in parallel and has the pipeline structure for the design variable index j. If D is assumed to be large, the clock cycles of the velocity update in the PSO processor are given by $CC_1 \simeq DL$. From $CC_0/CC_1 \simeq 9N$, the PSO processor is $9N$ times faster than the serial processing.

The PSO processor has an advantage in executing the PSO algorithm by the large number of particles. The proposed prosessor has a development potential espesially for solving larger-scale problems.

5. Conclusions

This paper has proposed the PSO processor suitable for parallel processing. The logical simulations and the logical synthesis on FPGA have been performed, and the correct circuit operations have been confirmed. Future problems include the evaluation of the performances for various benchmark problems, and the implementation of the PSO algorithm with the network topologies[2]-[7] and/or the multi-swarm structures [8]-[11].

Table 2: Logic Elements (LEs) of each module

Layer	Module	LEs
0	Top	28,092
1	I/O	1,132
1	Swarm	26,865
2	Particle	13,342
3	Init	3,933
3	Eval	1,602
3	Pbest	86
3	Gbest	1,027
3	Velocity	4,421
3	Position	2,273

References

[1] J. Kennedy and R. Eberhart, "Particle Swarm Optimization," *Proc. IEEE Int. Conf. Neural Networks*, pp. 1942-1948, 1995.

[2] T. Tsujimoto, T. Shindo, T. Kimura, and K. Jin'no, "A Relationship between Network Topology and Search Performance of PSO," *Proc. IEEE CEC*, pp. 1526-1531, 2012.

[3] E. Miyagawa and T. Saito, "Particle Swarm Optimizers with Growing Tree Topology," *IEICE Trans. Fundamentals*, E92-A, pp. 2275-2282, 2009.

[4] S. B. Akat and V. Gazi, "Particle Swarm Optimization with Dynamic Neighborhood Topology: Three Neighborhood Strategies and Preliminary Results," *Proc. IEEE Swarm Intelligence Symposium*, pp. 1-8, 2008.

[5] J. Lane, A. Engelbrecht and J. Gain, "Particle Swarm Optimization with Spatially Meaningful Neighbors," *Proc. IEEE Swarm Intelligence Symposium*, pp. 1-8, 2008.

[6] R. Sano, T. Shindo, K. Jin'no, and T. Saito, "Particle Swarm Optimization with Switched Topology," *Proc. IEEE SMC*, pp. 530-535, 2012.

[7] H. Matsushita, Y. Nishio and T. Saito, "Particle Swarm Optimization with Novel Concept of Complex Network," *Proc. NOLTA*, pp. 197-200, 2010.

[8] G. G. Yen and M. Daneshyari, "Diversity-based Information Exchange among Multiple Swarm in Particle Swarm Optimization," *Proc. IEEE CEC*, pp. 6150-6157, 2006.

[9] M. Iwamatsu, "Multi-Species Particle Swarm Optimizer for Multimodal Function Optimization," *IEICE Trans. Inf. and Syst.*, vol. E89-D, no. 3, 2006.

[10] H. Nakano, Y. Taguchi, Y. Kanamori, A. Utani, A. Miyauchi, and H. Yamamoto, "A Competitive Particle Swarm Optimizer and its Application to Wireless Sensor Networks," *IEEJ Trans.*, vol. 7, no. S1, pp. 52-58, 2012.

[11] T. Sasaki, H. Nakano, A. Miyauchi, and A.Taguchi, "Solving Performances of PSO Networks with Temporal Couplings," *Proc. IEEE SMC*, pp. 605-609, 2014.