Indian Journal of Science and Technology, Vol 8(36), DOI: 10.17485/ijst/2015/v8i36/90316, December 2015
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Design of Parallel Architecture Co-Processor for Particle Swarm Optimization Algorithm

S. Aravind Babu, S. Babu Ramki and S. Sivanantham*

ASIC Design Laboratory, School of Electronics Engineering, VIT University, Vellore – 632014, Tamil Nadu, India;
ssivanantham@vit.ac.in

Abstract

The direct implementation of parallel particle swarm optimization algorithm on field programmable gate array (FPGA) is presented in this paper. In the proposed design, the particle unit architecture is independent of fitness unit and hence the particle unit is reusable and flexible for different fitness function. The parallel co-processor implementation of each particle accelerates the execution speed and reduces the operating power as compared to the software execution of the design on a general purpose processor. The proposed implementation reduces the number of registers by 2.76% and the number of look-up-tables by 0.62% on average.

Keywords: Co-processor, FPGA Implementation, Particle Swarm Optimization, Parallel Architecture

1. Introduction

Particle Swarm Optimization (PSO) algorithm is basically a heuristic based evolutionary algorithm proposed by Kennedy et al.,[1], after studying behavior of flock of birds. Each particle in PSO corresponds to a bird in flock. The particle updates its location in a multidimensional space by evaluating its fitness of current position and updating its velocity and position. PSO can be used to solve optimization problems. PSO is computationally complex, so the PSO algorithm on hardware can provide solution faster compared to running it on a compiler in general purpose processor. Different architectures have been tried to implement PSO on hardware[2,3]. In massively parallel architecture[2], co-processor architecture is proposed which creates as many instances of particles as required. The drawback in the design is that fitness function module is designed into the particle, which requires redesign of particle for every function. In this paper a modified architecture for particle design is proposed, which can operate faster than and is independent of fitness function[2]. Special architectures are required to implement floating-point multipliers for low-power integrated circuits applications[3,4]. The field programmable gate arrays (FPGA) implementation of the PSO algorithm along with fitness unit and Floating point unit shows better performance than the software execution on a general purpose processor[5].

2. PSO Algorithm

The PSO algorithm is basically inspired by the social behavior of organisms for finding rich sources of food using an 'information sharing' approach. Each swarm consists of number of particles, and these fly around multidimensional search space. Each particle consists of three memory units namely, position (p_i^k - position of ith particle in kth iteration), velocity (vel_k^i) and its previous best experience ($pbest_k^i$). Position and velocity are constantly based on its memory and the global best position ($gbest$) information available in the swarm as shown in Equations (1) and (2).

*Author for correspondence

$$vel_{k+1}^{i} = w \times vel_{k+1}^{i} + c1 \times r1 \times (pbest_{k}^{i} - p_{k}^{i}) + c2 \times r2 \times (gbest - p_{k}^{i})$$

$$p_{k+1}^{i} = p_{k}^{i} + vel_{k+1}^{i}$$

where c1 and c2 represent the acceleration coefficients and the terms r1 and r2 represent uniformly distributed random numbers in [0,0.5]. The gbest is the best position information available in the swarm in the current iteration.Each particle is stochastically accelerated towards either its own best position or towards the global best position[6]. The PSO algorithm quickly converges to reasonably acceptable solution. But, due to its searching behavior of the particles, it may be trapped in the local optimum especially while solving complex problems[1]. During PSO algorithm execution, particles are initialized with initial position and velocity value in all dimensions. Each particle evaluates its fitness and if the fitness of current position is better than fitness of previous best (pbest) position, then current position is updated. After calculating the pbest value for all particles, these pbest are compared with each other. The fittest of these positions is compared with gbest and it is updated, if the fittest position is better than gbest position in the swarm's memory. Once the gbest value is updated, the position and velocity values are also updated based on Equations (1) and (2). The iteration is repeated until the terminating condition is achieved. The termination condition can be either number of iteration or a particular fitness value for gbest. Figure 1 describes the flowchart of PSO algorithm.

3. Proposed Architecture

The proposed architecture for PSO on hardware consists of three modules namely, Swarm Unit (SU), Particle Processing Element Unit (PPE) and Fitness Evaluation Unit (FEU). These units together perform the initialization of algorithm and evaluation of algorithm till required value is reached or till the iterations are completed. These processes are given as pseudo code in Figure 2.

3.1 Swarm Unit

The Swarm Unit (SU) is a collection of PPE and FE units. In addition, it contains a comparator and registers to store gbest position and fitness value. The comparator compares the gbest value with the pbest fitness value from all PPEs. If better position exists the position and its fitness value are stored in the gbest register. The architecture of Swarm Unit is shown in Figure 4.

3.1.1 Swarm Control Unit

The swarm control unit is the core of the entire proposed hardware. All particle units and fitness unit are controlled by swarm unit in addition to the initialization block and gbest unit. The PE_ctrl and FE_ctrl are the signal used for controlling the particle processing Element and fitness evaluation unit. The swarm control unit performs the entire PSO operation by feeding the control signals inn synchronization with clock until stopping criteria is achieved.

3.1.2 Initialization Block

This block initializes the position and velocity of each particle. It also initializes the value of different variables used in design operation. The initialization block is controlled by the Swarm unit and it works in synchronization with clock. The number of particles and dimension of each particle decided by the swarm unit is given as input to initialization block. Based on the received data the initialization operation is performed.

3.1.3 The gbest Comparator Unit

The gbest comparator unit is designed to compare the IEEE-754 floating point inputs. The gbest value stored in the gbest register is compared with the pbest value stored in pbest register. If the current pbest fitness is better than gbest, En signal is asserted and the current position along with the corresponding fitness are stored in the gbest registers, else the gbest register values are not updated. The updated gbest is given as input to the particle unit.

3.2 Particle Processing Element Unit

A Particle Processing Element (PPE) unit performs the core PSO algorithm and input/output are controlled by SU. The proposed architecture of PPE is shown in Figure 3. The PPE has the following units: Particle control unit (PCU), floating point unit (FLPU), pbest comparator unit, pbest registers and a linear feedback shift register (LFSR). A PCU in turn contains a Particle core unit and register bank to store the current position and velocity. The SU initializes the PPE and its register to reset values through initialization unit of SU.

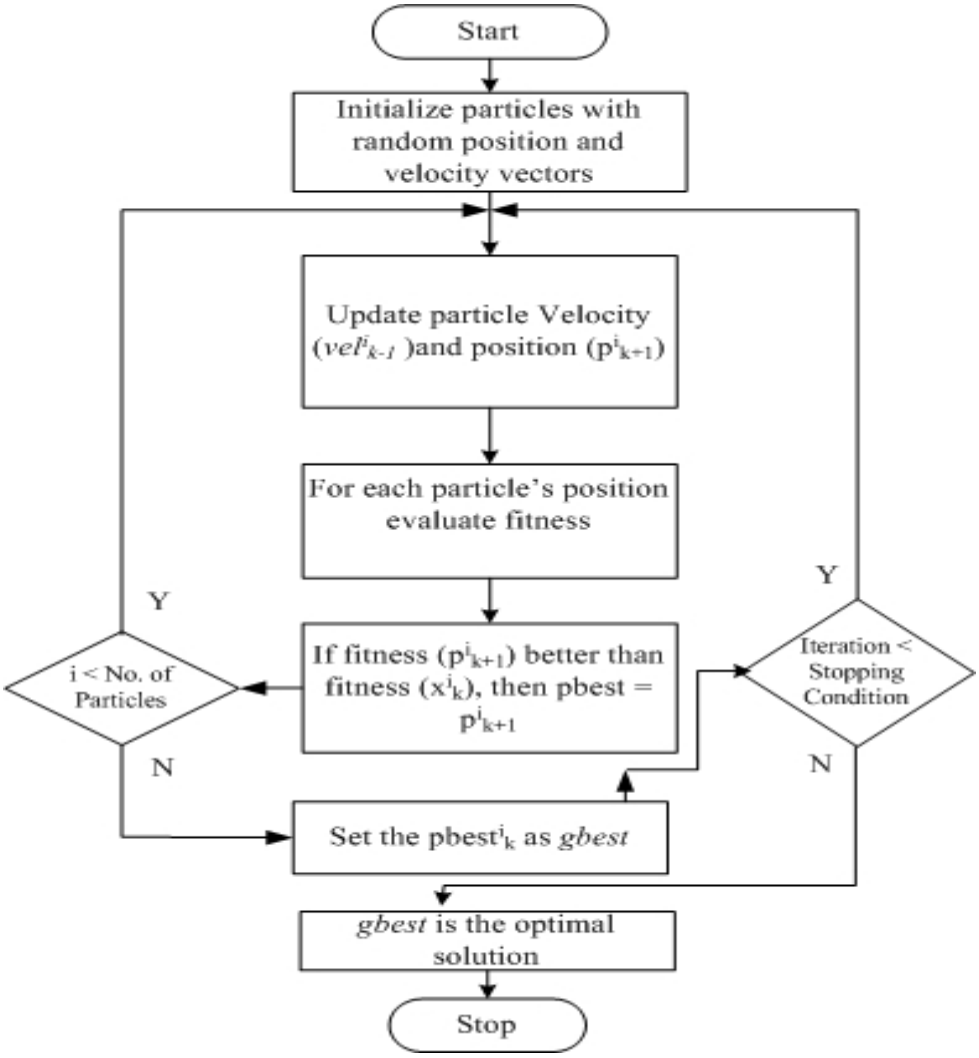

Figure 1. Flow-chart for PSO.

After initialization, the coefficients w, c1 and c2 are provided by SU to all PPE. Also the gbest register position is given as input. Based on these inputs and pbest register the PCU sends appropriate values to the FLPU for calculations. This task is done by Particle core unit present in PCU. Both velocity and position are computed for the current iteration and stored in the memory bank inside PPE. The position value is given to FE, present outside the PPE. The PPE evaluates the fitness of the current position and returns a fitness value.

3.2.1 Floating Point Unit (FLPU)

The Floating point unit is designed to perform arithmetic operation on normalized IEEE-754 based floating point inputs (A and B) with single precision format[7].

The FLPU performs desired arithmetic operation based on the control signal (op). IEEE-754 standard is a 32 bit format and has three parts namely signed (1- bit), exponent (8-bits) and mantissa (23-bits) and exponent part is represented as E-Bias (where bias value is 127). Initially, normalized floating point inputs are aligned before performing floating point operation. The mantissa part of the smallest exponent input is right shifted by the difference value of the two exponents. After aligning, the desired arithmetic operation is performed on the mantissa part of the inputs based on the signed bit. Finally, the mantissa part is rounded-off and then normalized. The unit is designed to work for normalized floating point inputs and special flags are asserted when any overflow occurs or when an out of bound inputs are detected [8].

Step 1-Initialization:
repeat particle=1 to N in Space do
repeat dimension=1 to D do
p^i_k = *random();*
vel^i_k = *random();*
end repeat
end repeat

repeat particle=1 to N in Space do
$pbest^i_k$= *pi*
if fitness($pbest^i_k$) < *fitness(gbest) then*
gbest = $pbest^i_k$
end if
end repeat

Step 2-Particle Swarm Optimization (Global Best)
Initialize
repeat
for particle=1 to N in Space do
if fitness(p^i_k) < *fitness*($pbest^i_k$) *then*
$pbest^i_k = p^i_k$
end if
if fitness($pbest^i_k$) < *fitness(gbest) then*
gbest = $pbest^i_k$
end if
end for

Step -3 PSO Core Operations – Position and Velocity Value Update
for Particle=1 to N in Space do
for Dimension=1 to D do
Update Position and Velocity using equation (1) and (2)
end for
end for

iteration= iteration + 1
Continue Process until iteration < Maximum number of Iterations
end repeat
return gbest and its position

Figure 2. Particle swarm optimization algorithm.

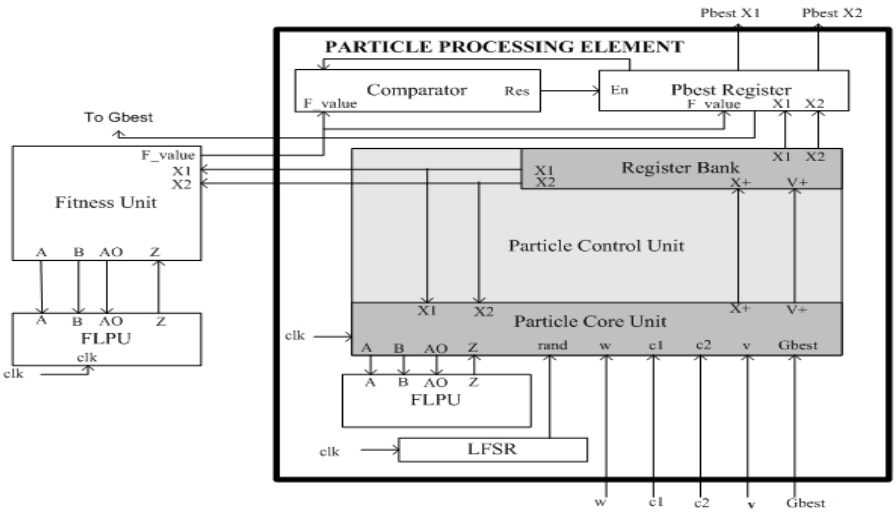

Figure 3. Proposed architecture for PSO algorithm implementation.

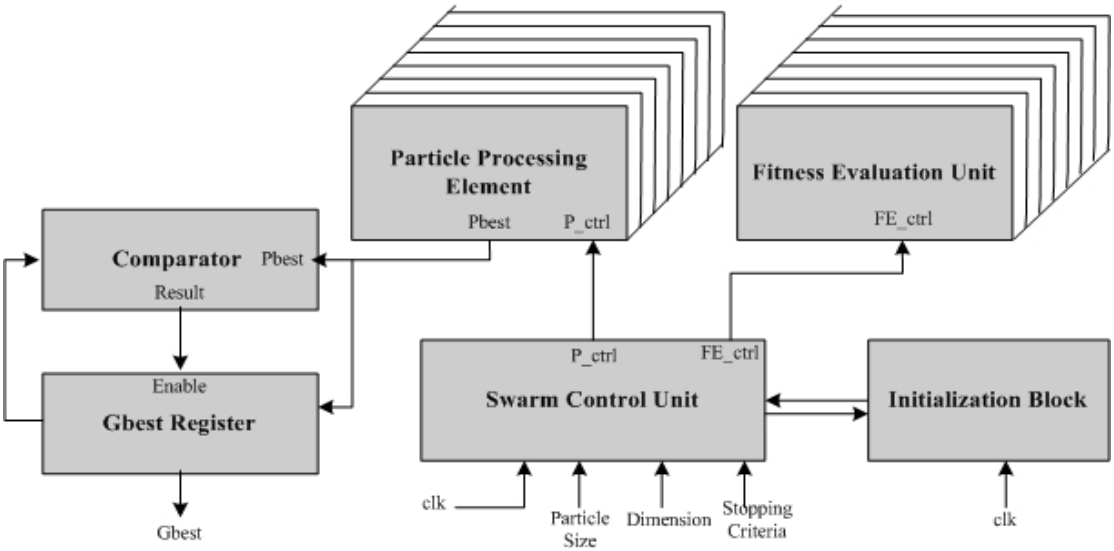

Figure 4. Swarm unit of particle swarm optimization.

3.2.2 *Particle Control Unit (PCU)*

The PCU updates the position and velocity of each particle in the multi-dimensional space based on equation (1) and (2). A PCU consists of Particle core unit and register bank. Register bank stores the position and velocity information of the current iteration. These values are updated on each iteration. Value from these register, random values from LFSR and the coefficient inputs from SU are sent to the FLPU serially by Particle core unit and the outputs are directed to appropriate registers by it. It decides when to send the input to FE for evaluation. It also activates the pbest comparator to update pbest position. The computation happens iteratively until the terminating criterion is achieved.

3.2.3 *Linear Feedback Shift Register (LFSR)*

A LFSR acts as a random number generator to generate the random values r1 and r2 required in Equation (1). The LFSR is a series of 32 successive registers with XOR taps at specified position to obtain random values required for the algorithm. LFSR generates pseudo-random values in the range [0, 0.5] and is implemented as given in Equation (3).

$$p = x^7 + x^3 + 1 \tag{3}$$

3.2.4 *pbest Comparator Unit*

The comparator unit compares the IEEE 754 floating point inputs. The comparator unit compares the pbest fitness value from the pbest register with the fitness value from the FE register. The En signal is asserted once current fitness becomes better as compared with pbest, and the current position and corresponding fitness are stored in the pbest registers, else the pbest register values are not updated.

3.3 Fitness Evaluation Unit (FE)

In the proposed architecture fitness unit is designed outside the particle unit so that the architecture of the particle unit is constant for different fitness function. The fitness functions implemented in the FE is logic implementation of a function or a microcode block that produces output based on the inputs. Consequently, the FE can have its own FLPU if it is a hardware implementation. In this paper, hardware FE is implemented. Also the fitness function implemented is one of the following: Rosenbrock function, Sphere function, F6 function or DeJong F2 function (7) as described in Equations (4) through (7) respectively. These functions are two dimensional functions with objective to find position that provides minimum fitness value.

$$f1(x, y) = 100 \times (y - x^2)^2 + (x - 1) \tag{4}$$

$$f2(x, y) = x^2 + y^2 \tag{5}$$

$$f3(x, y) = 0.5 - \frac{(\sin\sqrt{x^2 + y^2})^2 - 0.5}{(1 + 0.001 \times (x^2 + y^2))^2} \tag{6}$$

$$f4(x, y) = 100 * (x^2 - y^2 + (1 - x)^2 \tag{8}$$

Table 1. Range of position and criteria to terminate algorithm for the functions

Function	Range	Itrations
Sphere	[-100, 100]	200
DeJong	[-5, 5]	200
Rosenbrock	[-16, 16]	500
F6	[-100, 100]	500

Table 2. FPGA synthesis result of PSO algorithm

Fitness Function	Registers Max		LuT's Max	
	Our Work	Reference [2]	Our work	Reference [2]
Sphere	14780	15100	38200	38498
DeJong	14920	15572	38480	38670
Rosenbrock	14810	15168	38640	38873
F6	14810	15168	38640	38873

The independent particle unit designed outside the Fitness Evaluation unit makes the design reusable for different fitness function. The process is terminated when the stopping criteria is reached. After the process termination the gbest along with its position is returned. The returned gbest is the optimum solution for designed fitness function.

4. Results and Discussion

In proposed architecture, by designing FLPU unit to complete an operation in one clock cycle, we can reduce the sphere evaluation and particle updates to 23 clock cycles. One clock cycle is required to generate pbest and one to check the number of iterations remaining. Hence within 25 clock cycles a particle can update its position, compared to existing architecture [2].The 1st column of the Table 1 list the fitness functions that are implemented in the Fitness Unit. The range for the position of particle and the maximum number of iterations of execution is given in 2nd and 3rd column of the Table 1. The algorithm was implemented on Altera Development and Educational–II (DE-II) board which has Cyclone-II FPGA (EP2C20F484C7). The comparison of synthesis result for the existing and proposed architecture is provided in Table 2. The 1st column of Table 2 lists the fitness functions. The 2nd and 3rd column shows maximum register consumed by our proposeddesign and exiting work. The 4th and 5th column shows the LUT's synthesized for our proposed design and the previous existing work.

5. Conclusion

The independent FE placed outside the PPE makes it as a reusable design entity, thereby reducing the design cycle time. The FPGA implementation of parallel PSO algorithm has increased the execution speed compared to existing architecture.

The operating speed is increased and operating power of the proposed design is reduced due to parallelization of the PSO algorithm, thus making the design more flexible compared to the existing architecture.

6. References

1. Kennedy J, Eberhart R. Particle swarm optimization. IEEE International Conference on Neural Networks; 1995; 4: p. 1942–8.
2. de Moraes Calazan R, Nedjah N, de Macedo Mourelle L. A massively parallel reconfigurable co-processor for computationally demanding Particle Swarm Optimization. IEEE Third Latin American Symposium on Circuits and Systems (LASCAS); 2012; p. 1–4.
3. Sivanantham S. Design of low power floating point multiplier with reduced switching activity in deep submicron technology. Int J Appl Eng Res. 2013; 8(7):851–59.
4. Sivanantham S, Jagannadha Naidu K, Balamurugan S, Bhuvana Phaneendra D. Low power floating point computation sharing multiplier for signal processing applications. (IJET) Int J Eng Tech. 2013; 5(2):979–85
5. Munoz DM, Llanos CH, Coelho LD S, Ayala-Rincon M. Hardware Architecture for Particle Swarm Optimization Using Floating-Point Arithmetic. Ninth International Conference on Intelligent Systems Design and Applications, ISDA '09; 2009; p. 243–48.

6. Ratnaweera A, Halgamuge S, Watson HC. Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. IEEE Trans Evol Comput; 2004; 8(3):240–55.
7. IEEE Standards Board; IEEE-754, IEEE Standard for Binary Floating- Point Arithmetic. New York: IEEE; 1985.
8. Al-Ashrafy M, Salem A, Anis W. An efficient implementation of floating point multiplier. Saudi International Electronics, Communications and Photonics Conference (SIECPC); 2011; p. 1–5.