Hardware/Software Co-design for Particle Swarm Optimization Algorithm

Shih-An Li, Ching-Chang Wong, and Chia-Jun Yu
Dept. of Department of Electrical Engineering
Tamkang University
Taipei, Taiwan.
lishyhan@gmail.com

Chen-Chien Hsu
Dept. of Applied Electronics Technology,
National Taiwan Normal University
Taipei, Taiwan.
jhsu@ntnu.edu.tw

***Abstract*—This paper presents a hardware/software (HW/SW) co-design approach using SOPC technique and pipeline design method to improve the performance of particle swarm optimization (PSO) for embedded applications. Based on modular design architecture, a particle updating accelerator module via hardware implementation for updating velocity and position of particles and a fitness evaluation module implemented on a soft-cored processor for evaluating the objective functions are respectively designed and work closely together to accelerate the evolution process. Thanks to a flexible design, the proposed approach can tackle various optimization problems of embedded applications without the need for hardware redesign. To compensate the deficiency in generating truly random numbers by hardware implementation, a particle re-initialization scheme is also presented in this paper to further improve the execution performance of the PSO. Experiment results have demonstrated that the proposed HW/SW co-design approach to realize PSO is capable of achieving a high-quality solution effectively.**

***Keywords*—HW/SW Co-design, Particle swarm optimization (PSO), system on a programmable chip (SOPC), Field Programmable Gate Array (FPGA)**

I. INTRODUCTION

Particle swarm optimization (PSO) is a population-based, self-adaptive search optimization technique first introduced by Kennedy and Eberhart [1] in 1995. Based on simulation of simplified animal social behaviors such as fish schooling, bird flocking, etc [2], PSO has advantages of simplicity for implementation and ability to quickly converge to a reasonably good solution [3]. Because of its robustness, PSO has been widely adopted in various engineering applications [4, 5, 6].

As a population-based method, however, PSO still suffers from the problem of time-consuming evolution process to derive an answer, given the fact that PSO has proved itself as an effective and efficient method in comparison to other evolutionary approaches. When it comes to embedded and industrial applications, for example, optimization of memory usage [7], navigation of mobile sensors [8], and evolutionary mobile robots [9], [10], etc., the situation will be even worse because these applications generally uses low-performance microprocessors with limited computational resources, rather than high-performance desktop personal computers, as the computational platform [11, 12]. As a result, poor execution performance might occur because of the evolutionary nature of PSO. To solve this problem, several hardware-based implementations of PSO have been proposed in recent years to accelerate the execution performance for embedded applications. Among them, a parallel processing method implemented on FPGA [12] was proposed to calculate the flying of particles. Serial communication was adopted to transfer data among functional modules. Although this method can speed up the execution of PSO algorithms, larger FPGA area is required because of the parallel processing scheme adopted, which is undesired for practical industrial applications if cost is a primary consideration. Also, hardware redesign of the fitness evaluation module might be inevitable for different problems under consideration. In [11], a modular architecture of a hardware PSO engine was proposed for accelerating the algorithm's performance. Unfortunately, fitness evaluation of this approach is problem dependent and needs to be designed case by case. If the problem changes, then the hardware of the fitness evaluation module must be redesigned, which creates extra efforts in the design process. In [13, 14], a more desired architecture for PSO algorithm was proposed, in which an embedded processor via SOPC design technique and a hardware accelerator were used to calculate the fitness function and update velocity and position of particles, respectively. However, this architecture has less design flexibility if users wish to change the evolution parameters, for example, population size or dimension of the particle, etc. Under these circumstances, the PSO hardware must be redesigned. Furthermore, there is no mention of successful rate by using the proposed SOPC-based PSO. As a result, difficulties arise in evaluating the performance of the proposed approach.

As mentioned earlier, there is still room for further improving the performance of hardware-implemented PSO, as far as computational efficiency, usage flexibility, and resources utilization, etc., are concerned. As an attempt to solve this problem, this paper presents a hardware/software (HW/SW) co-design approach using SOPC technique and pipeline design method to improve the execution performance of PSOs while maintaining usage flexibility for embedded applications, in which a particle updating accelerator module via hardware implementation for updating velocity and position of particles and a fitness evaluation module implemented on a soft-cored processor for evaluating the objective functions are respectively designed and work closely together to accelerate the evolution process. Because of modularity design of this approach, fitness function can be flexibly modified for evaluation via the Fitness Evaluation module. As a result, various problems of embedded applications can be tackled simply by changing the objective functions for the problems under consideration. Considering the empirical nature where evolution parameters are subject to modification, the hardware design of the proposed approach has taken this requirement into consideration, where the population size, number of bit strings for encoding, inertia weight can be modified by users to suit the needs of a specific application by simply modifying an evolution parameters file. No hardware redesign is required if these evolution parameters are altered. To compensate the deficiency in generating truly random numbers by hardware implementation during the evolution process, a particle re-initialization scheme is also presented in this paper to further improve the execution performance of the PSO.

II. Particle Swarm Optimization Algorithms

Particle swarm adaptation has been shown to successfully optimize a wide range of continuous functions [1, 2]. The algorithm, which is based on a metaphor of social interaction, searches a space by adjusting the trajectories of individual vectors, called "particles" as they are conceptualized as moving points in multidimensional space [15, 16, 17]. As an evolutionary technique, the PSO is a population-based algorithm, formed by a set of particles representing potential solutions for a given problem. Each particle moves through a n-dimensional search space, with an associated position vector $x_i(t) = \{x_{i1}(t), x_{i2}(t), \ldots, x_{in}(t)\}$ and velocity vector $v_i(t) = \{v_{i1}(t), v_{i2}(t), \ldots, v_{in}(t)\}$ for the current evolutionary iteration t. The individual particle in PSO flies in the search space with velocity which is dynamically adjusted according to its own flying experience and its companions' flying experience [4]. The former was termed cognition-only model and the latter was termed social-only model [18]. By integrating these two types of knowledge, the particle behavior in a PSO can be modeled by using the following equations:

$$v_i(t+1) = w \times v_i(t) + c_1 \times rand \times (pbest_i - x_i(t)) + c_2 \times rand \times (Gbest - x_i(t)) \quad (1)$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (2)$$

, where

c_1, c_2 : acceleration constants;

$rand$: random number between 0 and 1;

$x_i(t)$: the position of particle i at iteration t ;

$v_i(t)$: the velocity of particle i at iteration t ;

w : inertia weight factor;

$Gbest$: the personal best position among all the particles;

$Pbest_i$: the personal best position of particle i.

Note that the first term on the right-hand side of the velocity-updating rule in Eq. (1) represents the previous velocity, which provides the necessary momentum for particles to roam across the search space. The second term, known as the "cognitive" component, represents the personal thinking of each particle, which encourages the particles to move toward their own best positions found so far. The third term is known as the "social" component, which represents the collaborative effect of the particles in finding the global optima.

Figure 1 shows the flow chart of a typical PSO algorithm, in which a population of particles is initialized with random position x_i and velocity v_i. Fitness of particles is evaluated by calculating the objective function $f(x_i)$. The current position of each particle is set as $Pbest_i$. The $Pbest_i$ with best value in the swarm is set as $Gbest$. As evolution continues, next position for each particle is repeatedly generated by using Eqs. (1) and (2). If a better position is achieved by an agent, the $Pbest_i$ value is replaced by the current value. If a new $Gbest$ value is better than the previous $Gbest$ value, the $Gbest$ value is replaced by the current $Gbest$ value. Iterations repeat until a predetermined iteration number is reached.

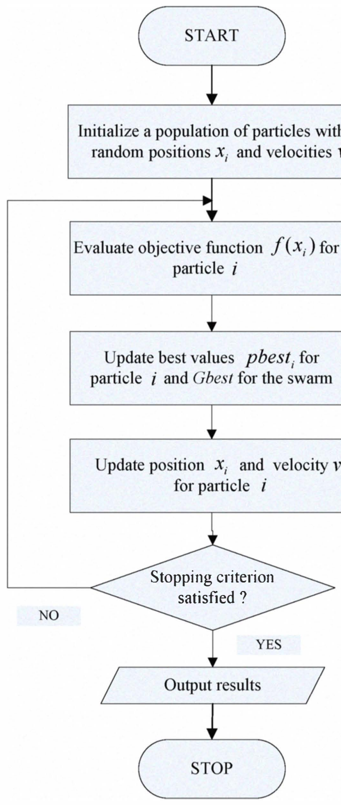
START
Initialize a population of particles with random positions x_i and velocities v_i
Evaluate objective function $f(x_i)$ for particle i
Update best values $pbest_i$ for particle i and $Gbest$ for the swarm
Update position x_i and velocity v_i for particle i
Stopping criterion satisfied ?
NO
YES
Output results
STOP

Fig. 1 Evolution processes of a typical PSO algorithm [19].

III. System Architecture of HW/SW Co-design of PSO Algorithms

Field Programmable Gate Array (FPGA) is a flexible chip for easy reconfiguration. There have been many prototype designs successfully implemented using FPGA as their development platform. In recent years, HW/SW co-design methods are getting more and more popular in the design of system on a programmable chip (SOPC). With the availability of Electronic Design Automation (EDA) tools by major vendors, for example, Altera Corporation [20], SOPC design has become much easier by compiling the hardware description language to synthesize a digital circuit for downloading to a FPGA chip.

Figure 2 shows the architecture of HW/SW Co-design of PSO algorithms proposed in this paper, where modular design is performed to construct the major components, including an Evolution Parameters File, a Fitness Evaluation module, an Avalon Slave Component module, and a Particle Updating Accelerator. The Evolution Parameters File is used to store control parameters for evolution. The Fitness Evaluation module is a software module for evaluating objective functions to tackle different applications under consideration, which can be implemented on any available processors, for example the embedded Nios II CPU used in this paper. The Avalon Slave Component module is to generate the particle registers and fitness registers required according to the evolution parameters specified. The Particle Updating Accelerator module is used to update the position and velocity of particles in a population. In what follows, we will describe these functional modules in more details.

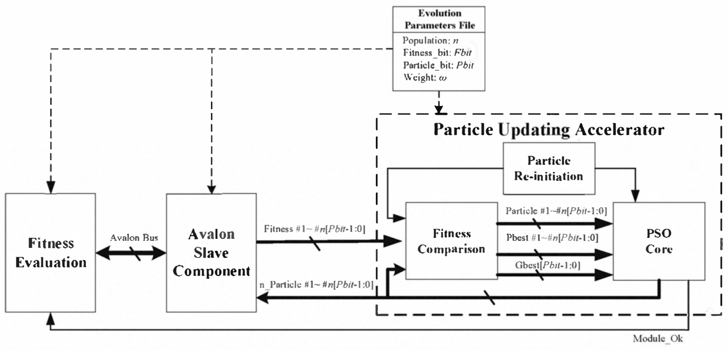

Fig. 2 HW/SW Co-design architecture of PSO algorithms.

A. Evolution Parameters File

The Evolution Parameters File is used to store control parameters for evolution, which can be arbitrarily altered by the user to facilitate the optimization process. Four parameters are contained in the Evolution Parameters File, including population size n, bit number of fitness *Fbit* representing the resolution of fitness value, bit number of a particle *Pbit* representing the resolution of variables, and inertial weight ω. When the evolution parameters are changed, an electronic design automation (EDA) tool can be used to recompile the design and all of the wire connections on the FPGA can be automatically generated to complete the design. This has rendered maximal HW/SW design flexibility without the need for hardware redesign.

B. Fitness Evaluation module

Because various applications need to be dealt with, the Fitness Evaluation module should accommodate the diversities of problems, and is therefore preferably implemented by a software module for evaluating objective functions under consideration.

On the other hand, it is easy to integrate various kinds of hardware functions into a system module on FPGA by software via SOPC builder developed by Altera. To render maximal flexibility without using extra external circuits, Fig. 3 shows the proposed soft-cored system for implementing the Fitness Evaluation module, including a soft-cored embedded processor [21] (Nios II CPU), a system timer, a timestamp timer, a SDRAM controller, an Avalon bus interface, and a particle flying finished acknowledgement module. The Nios II CPU accesses the objective function stored in SDRAM via the SDRAM controller for calculating its fitness values for particles transferred from the Particle Updating Accelerator module. To enhance the floating-point computing ability for speeding up the calculation of exponential and trigonometric functions, a floating-point hardware is added into the Nios II CPU module.

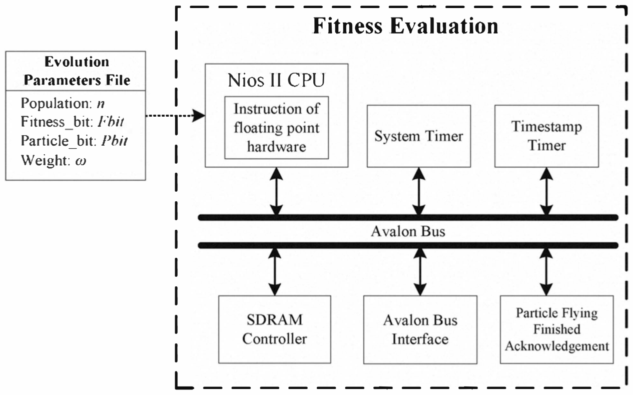

Fig. 3 Fitness Evaluation module implemented on a soft-cored system.

C. Avalon Slave Component module

The soft-cored system module on which the Fitness Evaluation module is implemented can't be modified automatically — it needs to be manually generated via SOPC builder by the user. If a flexible hardware architecture of PSO algorithm is desired, we need to design an interface to connect the soft-cored system module and Particle Updating Accelerator module. Based on the above concept, we design a flexible Avalon Slave Component module shown in Fig. 4 which can be synthesized and generated by Quartus II software by Altera. When the evolution parameters are changed, we just use the EDA tool to recompile the design and the required particle registers and fitness registers can be generated according to the evolution parameters specified. An Avalon Bus Interface [22] in the Avalon Slave Component module is responsible for sending particle information to and receiving fitness values from the Nios II CPU.

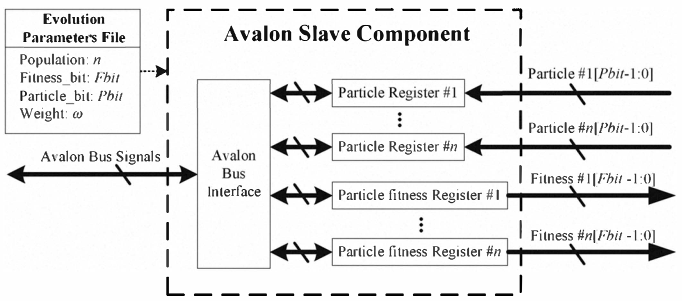

Fig. 4 Hardware architecture of the Avalon Slave Component module.

D. *Particle Updating Accelerator module*

Figure 5 shows the functional blocks of the Particle Updating Accelerator module in Fig 1. There are two hardware circuits implemented in this module: Fitness Comparison module and PSO Core module. The Fitness Comparison module is used to calculate the personal best position *Pbest #i* for particle *i*, $i \in [1, n]$, and the global best position *Gbest*, for updating the position and velocity of particles in a population by the PSO Core module.

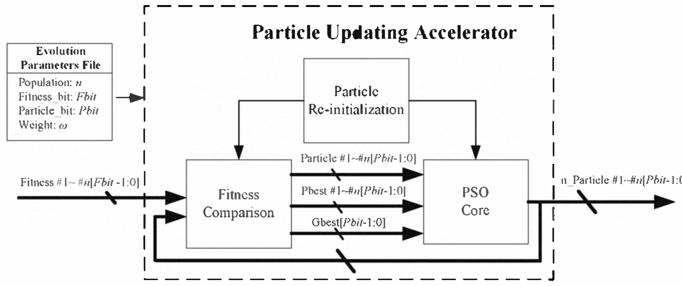

Fig. 5 Hardware architecture of the Particle Updating Accelerator module.

IV. EXPERIMENT RESULTS

In this paper, a DE2-70 development board by Altera is used as an experiment platform for evaluating the performance of the proposed approach, where a Cyclone II FPGA with chip number EP2C70F896C6N having a total of 68416 logic elements (LE) is used to realize the system architecture shown in Fig. 2 by using the Verilog language.

A. *Comparison of computational costs via the Particle Updating Accelerator*

Table 1 shows a comparison of computational cost in terms of clock cycles by the soft-cored processor (Nios II CPU) and the proposed hardware implementation realize the Particle Updating Accelerator to complete a full cycle of position update for all particles in a single generation. As shown in Table 1, hardware implementation the proposed Particle Updating Accelerator has a processing speed approximately 10 times faster than that of the soft-cored Nios II CPU. To demonstrate the resource usage of the proposed approach, Table 2 shows the logical elements (Les) required on FPGA in realizing the Particle Updating Accelerator. Note that no memory bits are used in the design as shown in Table 2. If memory bits are used in the design, LE usage can be further reduced.

Table 1 Computational cost by Nios II CPU and the proposed Particle Updating Accelerator

Population size (*n*)	Clock cycles required	
	Nios II CPU	Particle Updating Accelerator
8	8724	892
16	16962	1448
32	33572	2415

Table 2 Resource usage on FPGA

Particle Updating Accelerator		
Population size	LEs	Memory bits
8	4265	0
16	9698	0
32	18585	0

B. *Basis for evaluating the HW/SW co-design approach*

A 32-bit soft-cored embedded Nios II CPU running at a maximal clock frequency of 50 MHz is adopted for implementing the Fitness Evaluation module to calculate the fitness values of the benchmark functions. Hardware circuit of the Avalon Slave Component and Particle Updating Accelerator also run at a clock frequency of 50 MHz. Particles of the PSO algorithm are coded in binary string. For verification purpose, eight benchmark functions [23] listed in Table 3 are used to test the performance of the PSO algorithm via the proposed HW/SW co-design realization. Detailed descriptions of the benchmark functions are listed as an appendix at the end of this paper. Control parameters for implementing the PSO algorithm using the HW/SW co-design architecture are listed in Table 4.

Table 3 Benchmark functions used for performance analysis

No.	Function Name	Variable no.	Optimal OBJ value
1	B2 function	2	0
2	Branin RCOC function	2	0.397887
3	Goldstein and function	2	3
4	Rosenbrock function	2	0
5	Zakharov function	2	0
6	De Joung function	3	0
7	Hartmann function	3	-3.86343
8	Variably dimensioned function	4	0

Table 4 Control parameters of the HW/SW co-design realization of PSO algorithm

Control Parameters	Value
Particle coding	Binary
Bit number of variables	8
Population size	8, 16, 32
Iteration number	100
Maximal function evaluations (N)	10000
$c_1 r_1$	0~2
$c_2 r_2$	0~2
Inertia weight ω	0.25
Termination Error Threshold	$< 10^{-4}$

C. *Experiment results of the HW/SW co-design approach*

Experiment results of the PSO algorithm shown in Eqs. (1) and (2) implemented via the proposed HW/SW co-design approach to test the eight benchmark functions are shown in Table 5. Different population sizes of n=8, 16, and 32 are adopted to evaluate the performance. The evolutionary program terminates if the criteria on termination error threshold of the objective function values below:

$$\left| f(x_{best}) - f(x_{opt}) \right| < \varepsilon, \tag{3}$$

is satisfied or a maximal number of function evaluations (FE) of 10000 is reached, where $f(x_{opt})$ is the objective function value of the known analytic optimal solution x_{opt}.

To evaluate the performance of the evolutionary algorithms, 100 runs are executed for each benchmark function to ensure credible simulation results have been obtained. As long as one of the terminating criteria is met, the algorithm stops and returns the best solution x_{best} ever searched for calculating the objective function value $f(x_{best})$. A particular optimization process is considered "*successful*" if the best solution x_{best} derived via the evolutionary algorithm satisfies the condition in Eq. (3). Depending on performance requirement, the termination error threshold ε can be arbitrarily designated, as long as a fair comparison can be guaranteed. In this paper, a termination error threshold of $\varepsilon = 10^{-4}$ is selected for all the benchmark functions. It is clear that larger population size results in a better successful rate as expected.

By averaging the function evaluations required for all the successful runs, we obtain the averaged function evaluation number required for optimizing the benchmark functions via the proposed method. Averaged time elapsed for successful optimization runs can be obtained in a similar way. Care must be taken that the Averaged function evaluations and Elapsed time shown in Table 5 show successful runs only. If all of the optimization results, including successful and non-successful runs, are taken into account, the performance in terms of Averaged function evaluations and Elapsed time might not be as good as presented in the table.

Although the successful rates in Table 5 somehow reach 90% or higher for 6 out of 8 benchmark functions with a population of 32, the performance, however, is not good enough. During the experiment, we discover that the RNG might not work satisfactorily in providing truly uniformly-distributed random numbers for particles in searching the optimal solution during the evolution process. Therefore we use the particle re-initialization scheme presented in Section 3 to randomly generate particles for replacing the particles in the current population except the personal best solution and global best solution based on an elitism preservation concept every 100 generations in this paper. Much satisfactory results as shown in Table 6 can be obtained via the particle re-initialization scheme, where successful rates have reached 100% for the majority of the cases. Also revealed in this table is that population size of n=16 has a desired performance in terms of function evaluations required while maintaining the same successful rate of 100% for these benchmark functions in comparison to n=32.

Table 5 Experiment results of PSO algorithm via HW/SW co-design approach

Function	Successful rate (%)			Averaged function evaluations			Elapsed Time (sec.)		
	n=8	n =16	n =32	n=8	n =16	n =32	n=8	n =16	n =32
Fun1	86	89	94	116	183	362	0.1914	0.3003	0.6204
Fun2	82	89	94	176	190	311	0.1570	0.1646	0.2700
Fun3	66	85	87	173	214	310	0.1468	0.1747	0.2533
Fun4	53	84	94	725	1079	1114	0.2283	0.3162	0.3313
Fun5	80	90	93	110	216	276	0.4385	0.8745	1.1381
Fun6	70	92	93	246	352	460	0.0701	0.0946	0.1268
Fun7	46	80	84	217	320	492	0.8815	1.2639	1.9368
Fun8	26	61	91	848	1378	1932	2.0773	3.2955	4.7520

Table 6 Experiment results of PSO algorithm via HW/SW co-design approach and Particle Re-initialization Scheme

Function	Successful rate (%)			Averaged function evaluations			Elapsed Time (sec.)		
	n=8	n =16	n =32	n=8	n =16	n =32	n=8	n =16	n =32
Fun1	100	100	100	271	299	640	0.4171	0.5143	1.0236
Fun2	100	100	100	324	278	415	0.2865	0.2353	0.3565
Fun3	99	100	100	731	500	770	0.6109	0.4110	0.6232
Fun4	98	100	100	1991	1203	1314	0.6041	0.3485	0.3882
Fun5	100	100	100	275	205	443	1.0267	0.7961	1.7344
Fun6	100	100	100	617	490	731	0.1682	0.1293	0.1993
Fun7	100	100	100	732	680	819	2.9664	2.7379	3.2473
Fun8	95	100	100	2570	2036	2042	5.4898	4.7192	5.0632

V. Conclusions

In this paper, we have presented a HW/SW co-design architecture to implement PSO on a FPGA chip, where the Particle Updating Accelerator module responsible for updating position and velocity of particles in a population is implemented by hardware. Any available processors or circuits capable of performing fitness evaluation can be used to work with the Particle Updating Accelerator to speed up the execution performance of PSO. Because of flexible design of the proposed approach where the Fitness Evaluation module is implemented by a software module for evaluating objective functions under consideration, various applications can be dealt with. To meet optimization needs for different applications, evolution parameters can be arbitrarily adjusted by the user without the need for hardware redesign. This has rendered maximal HW/SW design flexibility to complete a design by the proposed approach. Experiment results have shown that significant improvement in terms of clock cycles required of the proposed Particle Updating Accelerator has been achieved, where processing speed is approximately 10 times faster than that of the soft-cored Nios II CPU. To circumvent the difficulties in generating truly uniformly-distributed random numbers by hardware implementation, the particle re-initialization scheme is incorporated into the Particle Updating module to improve the evolutionary efficiency and effectiveness in searching an optimal solution. Experiment results have proved that successful rates in optimizing the 8 benchmark functions have reached 100% for all cases. As is apparent from the experiments, the bottleneck for accelerating the execution of PSO algorithms lies in the calculation of fitness function. Although fitness evaluation module can be realized by hardware circuits, the flexibility to deal with different problems will inevitably reduces. It is therefore the objective of this paper to use HW/SW co-design method to realize PSO in a hope to balance the computational efficiency and usage flexibility of PSO.

Acknowledgment

This work was partially supported by the National Science Council, Taiwan, under Grants NSC 97-2221-E-032-032 and NSC 94-2213-E-032-002. The authors wish to thank Jui-Pin Wang for his efforts to carry out additional experiments in a very restricted timeframe during the preparation of this paper.

References

[1] J. Kennedy and R. Eberhart, "Particle swarm optimization," IEEE International Conference on Neural Networks, Perth, Australia, 1995, pp. 1942-1948.

[2] K.E. Parsopoulos and M.N. Vrahatis, "Recent approaches to global optimization problems through particle swarm optimization," Natural Computing, vol. 1, 2002, pp. 235-306.

[3] A. Ratnaweera, S.K. Halgamuge, and H.C. Watson, "Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients," IEEE Transaction on System, Man and Cybernetics, vol. 8, no. 3, 2004, pp. 240-255.

[4] Z.L. Gaing, "A particle swarm optimization approach for optimum design of PID controller in AVR system," IEEE Transactions on Energy Conversion, vol. 19, no. 2, 2004, pp. 384-391.

[5] J.F. Schutte and A.A. Groenwold, "A study of global optimization using particle swarms," Journal of Global Optimization, January 2005, vol. 31, no. 1, pp. 93-108.

[6] J.F. Schutte, B. Koh, J.A. Reinbolt, R.T. Haftka, A.D. George, and B.J. Fregly, "Evaluation of a particle swarm algorithm for biomechanical optimization," Journal of Biomechanical Engineering, vol. 127, no. 3, 2005, pp. 465-474.

[7] J.L. Risco-Martín, J. I. Hidalgo, O. Garnica, J. Lanchares, D. Atienza, "Particle swarm optimization of memory usage in embedded systems," Inderscience Journal of High Performance Systems Architecture (IJHPSA), 2009, pp. 1-11.

[8] G.K. Venayagamoorthy and S. Doctor, "Navigation of mobile sensors using PSO and embedded PSO in a fuzzy logic controller," 39th IEEE IAS Annual Meeting on Industry Applications, Seattle, WA, USA, October 2004, pp. 1200-1206.

[9] N. Nedjah, L. Coelho, L. Mourelle, "Mobile Robots: The Evolutionary Approach," Springer, Berlin Heidelberg, 2007.

[10] L. Wang, K.C. Tan, and C.M. Chew, "Evolutionary Robotics: From Algorithms to Implementations," World Scientific Publishing Company, 2006.

[11] S.T. Girma, M.H. Darrin, R.E. Haskell, "Accelerating the performance of particle swarm optimization for embedded applications," IEEE Congress on Evolutionary Computation, Trondheim, Norway, 2009, pp.2294-2300.

[12] G.S. Tewolde, D.M. Hanna, and R.E. Haskell, "Multi-swarm parallel PSO: hardware implementation," IEEE on Swarm Intelligence Symposium, April 2009, pp.60-66.

[13] A. Farmahini-Farahani, S.M. Fakhraie, and S. Safari, "SOPC-Based architecture for discrete particle swarm optimization," IEEE International Conference on Electronics, Circuits and Systems, ICECS), Marrakech, Morocco, Dec. 2007, pp. 1003-1006.

[14] A. Farmahini-Farahani, S.M. Fakhraie, and S. Safari, "Scalable architecture for on-chip neural network training using swarm intelligence," Proceedings of the Conference on Design, Automation and Test, Munich, Germany, Europe, March 10-14, 2008, pp. 1340-1345.

[15] F. van den Bergh and A.P. Engelbrecht, "A study of particle swarm optimization particle trajectories," Information Sciences, vol. 176, 2006, pp. 937-971.

[16] M. Clerc and J. Kennedy, "The particle swarm explosion, stability, and convergence in a multidimensional complex space," IEEE Transaction on Evolutionary Computation, vol. 6, no.1, 2002, pp. 58-73.

[17] P.K. Tripathi, S. Bandyopadhyay and K.S. Pal, "Multi-objective particle swarm optimization with time variant inertia and acceleration coefficients," Information Sciences, vol. 177, Iss. 22, 2007, pp. 5033-5049.

[18] J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm," Proceedings IEEE International Conference on Systems, Man, and Cybernetics, Piscataway, New Jersey, USA, vol. 5, 1997, pp. 4104-4108.

[19] J.F. Schutte, J.A. Reinbolt, B.J. Fregly, R.T. Haftka, and A.D. George, "Parallel global optimization with the particle swarm algorithm," Numerical Methods in Engineering, vol. 61, no.13, 2004, pp. 2296-2315.

[20] Altera Corporation homepage. URL://www.altera.com

[21] Altera Corporation, Nios embedded processor software development reference manual, 2003.

[22] Altera Corporation, Avalon interface specifications, 2008.

[23] S.K.S. Fan and E. Zahara, "A hybrid simplex search and particle swarm optimization for unconstrained optimization," European Journal of Operational Research, vol. 181, no. 2, 2007, pp. 527-548.