CONSTRAINED AND UNCONSTRAINED HARDWARE-SOFTWARE PARTITIONING USING PARTICLE SWARM OPTIMIZATION TECHNIQUE

M. B. Abdelhalim, A. E. Salama and S. E.-D. Habib
Electronics and Communication Department,
Faculty of Engineering,Cairo University, Egypt.

Abstract: In this paper we investigate the application of the Particle Swarm Optimization (PSO) technique for solving the Hardware/Software partitioning problem. The PSO is attractive for the Hardware/Software partitioning problem as it offers reasonable coverage of the design space together with O(n) main loop's execution time, where n is the number of proposed solutions that will evolve to provide the final solution. We carried out several tests on a hypothetical, relatively-large Hardware/Software partitioning problem using the PSO algorithm as well as the Genetic Algorithm (GA), which is another evolutionary technique. We found that PSO outperforms GA in the cost function and the execution time. For the case of unconstrained design problem, we tested several hybrid combinations of PSO and GA algorithms; including PSO then GA, GA then PSO, GA followed by GA, and finally PSO followed by PSO. The PSO algorithm followed by another PSO round gave the best result as it allows another round of domain exploration. The second PSO round assign new randomized velocities to the particles, while keeping best particle positions obtained in the first round. We propose to name this successive PSO algorithm as the Re-excited PSO algorithm. The constrained formulations of the problem are investigated for different tuning or limiting design parameters constraints.

Keywords: Embedded systems, Hardware/Software co-design, Hardware/Software Partitioning, Particle Swarm Optimization, Genetic Algorithm, Evolutionary Algorithms, re-excited PSO.

1. INTRODUCTION

Embedded systems typically consist of application specific hardware parts, i.e. FPGAs or ASICs, and programmable parts, i.e., processors like DSPs or ASIPs. In comparison to the hardware parts, the software parts are much easier and faster to develop and modify. Thus, software is less expensive in terms of cost and development time. Hardware, however, provides better performance. For this reason, a system designer's goal is a system which minimizes the weighted sum of the software delay cost, Hardware area cost, and power consumption cost. The weights are determined by the user according to the design's critical parameters.

Hardware/software co-design deals with the problem of designing embedded systems, where automatic partitioning is one key issue. This paper describes a new approach for hardware/software partitioning for single-processor systems. This approach is based on the Particle Swarm Optimization (PSO) algorithm. To further improve the quality of the result obtained by the PSO algorithm, a successive iterative approach is described in this paper that finds a near global optimum solution in a small time. To the best of our knowledge, there is no work based on PSO for HW/SW partitioning problem, apart from our precursor paper [1], limited to the unconstrained case.

The outline of the paper is as follows: Section 2 gives an overview of related work in the field of Hardware/Software Partitioning. In Section 3, a brief summary of the Particle Swarm Optimization algorithm is presented. In Section 4, the implementation of the algorithm and the results are discussed and compared with the results of the Genetic Algorithm, also the concept of re-excited PSO is presented. Section 5 extends the PSO algorithm for constrained HW/SW partitioning problems. The paper conclusions are given in Section 6.

2. HARDWARE/SOFTWARE PARTITIONING

The most important challenge in the embedded system design is partitioning; i.e. deciding which components of the system should be implemented in hardware and which ones in software. Finding an optimal partition is hard because of the large number and different characteristics of the components that have to be considered.

Traditionally, partitioning was carried out manually [2]. However, because of the increase of complexity of the systems, many research efforts have been undertaken to automate the partitioning as much as possible. The suggested partition approaches differ significantly according to the definition

they used to the problem. One of the main differences is whether to include other tasks (such as scheduling where starting times of the components should be determined) [3-4] or just map components to hardware or software only [5-6]. Some formulations assign communication cost to links between hardware and/or software units [7]. The system to be partitioned is generally given in the form of task graph, the graph nodes determined by the model granularity, i.e. the semantic of a node. The node could represent a single instruction, short sequence of instructions [8], basic block [9] or a function or procedure [10-11]. A flexible granularity may also be followed where a node can represent any of the above [5, 12]. Regarding the suggested algorithms, one can differentiate between exact and heuristic methods. The proposed exact algorithms include branch-and-bound [13], dynamic programming [6], and integer linear programming [10]. Due to the slow performance of the exact algorithms, heuristic-based algorithms are proposed. In particular, Genetic algorithms are widely used [7,14] as well as simulated annealing [12, 15], hierarchical clustering [5], and Kernighan-Lin based algorithms such as in [14]. Less popular heuristics are used such as Tabu search [15] and greedy algorithms [16]. Some researchers used custom heuristics, such as MFMC [14], GCLP [17], process complexity [18], and the expert system presented in [3].

3. PARTICLE SWARM OPTIMIZATION

Particle swarm optimization (PSO) is a population based stochastic optimization technique developed by Eberhart and Kennedy in 1995 [19-20]. The PSO algorithm is inspired by social behavior of bird flocking or fish schooling. In PSO, the potential solutions, called particles, fly through the problem space by following the current optimum particles. PSO has been successfully applied in many areas. A good bibliography of PSO applications could be found in [22].

As stated before, PSO simulates the behavior of bird flocking. Suppose the following scenario: a group of birds are randomly searching for food in an area. There is only one piece of food in the area being searched. All the birds do not know where the food is. During every iteration, they learn via their inter-communications, how far the food is. So the best strategy to find the food is to follow the bird which is nearest to the food [21].

PSO learned from this bird-flocking scenario, and used it to solve the optimization problems. Each single solution (particle) is perceived as a "bird" in the search space. Each particle has a fitness value which is evaluated by the fitness function (the cost function to be optimized), and has

a velocity which directs its flight. The particles fly through the problem space by following the current optimum particles.

PSO is initialized with a group of random particles (solutions) and then searches for optima by updating generations. During every generation (iteration), each particle is updated by following two "best" values. The first one is the position vector of the best solution (fitness) this particle has achieved so far. The fitness value is also stored. This position is called ***pbest***. Another "best" position that is tracked by the particle swarm optimizer is the best position, obtained so far, by any particle in the population. This best position is the current global best and is called ***gbest***.

After finding the two best values, the particle updates its velocity and position according to equations (1) and (2).

$$v_{k+1}^{i} = wv_{k}^{i} + c_1 r_1 (pbest^{i} - x_{k}^{i}) + c_2 r_2 (gbest_{k} - x_{k}^{i}) \quad (1)$$

$$x_{k+1}^{i} = x_{k}^{i} + v_{k+1}^{i} \quad (2)$$

Where v_{k+1}^{i} is the velocity of particle number (i) at the $(k+1)^{th}$ iteration, x_{k}^{i} is the current particle solution (or position). (r_1) and (r_2) are random numbers between 0 and 1. (c_1) is the self confidence (cognitive) factor; (c_2) is the swarm confidence (social) factor. (w) is the inertia factor [21].

The 1st term in equation 1 represents the effect of the inertia of the particle, the 2nd term represents the particle memory influence, and the 3rd term represents the swarm influence.

The velocities of the particles on each dimension may be clamped to a maximum velocity V_{max}, which is a parameter specified by the user. If the sum of accelerations causes the velocity on that dimension to exceed V_{max}, then this velocity is limited to V_{max} [21]. Another type of clamping is to clamp the position of the current solution to a certain range in which the solution has a meaning; otherwise the solution is meaningless [21]. In this paper, position clamping is only applied with no limitation on the velocity values.

In [1] a comparison between PSO and GA showed that PSO is better than GA in that it has small number of tuning parameters, and it is faster than GA due to its main loop's linear complexity.

4. IMPLEMENTATION AND RESULTS

The PSO algorithm is written as a script in the MATLAB program environment [22]. The input to the scripts is a design that consists of the number of nodes. Each node is associated with cost parameters. For experimental purpose, these parameters are randomly generated.

The used cost parameters are:

A Hardware implementation cost (*HWCost);* which is the cost of implementing that node in hardware (e.g. hardware area and delay, but in most cases the hardware delay is ignored with respect to the software delay).

A Software implementation cost (*SWCost);* which is the cost of implementing that node in software (e.g. Processing delay and CPU area, the CPU area is ignored as it is independent of the problem size).

This above cost values are uniformly and randomly generated in the range from 1 to 99 [14].

A Power implementation cost (*POWERcost);* which is the power consumption of the node's implementation. This cost is uniformly and randomly generated in the range from 1 to 9. We use a different range for the Power consumption cost to test the addition of other cost terms with different range characteristics (the cost term will be normalized in the cost function).

The communications cost is included in the hardware and software costs of the nodes, as communications between nodes are already known from the design itself [14].

Consider a design that consists of **m** nodes. A possible solution (particle) is a vector of **m** elements, where each element is associated to a given node. The elements assume a "0" value (if the node is implemented in hardware) or a "1" value (if the node is implemented in software). There are **n** particles (solutions). The particles are initialized randomly.

The velocity of each node is initialized in the range from (-1) to (1), where a negative velocity indicates that the particle is moving towards 0 and a positive velocity indicates that the particle is moving towards 1.

For the algorithm's main loop, equations 1, 2 are evaluated in each loop. If the particle goes out of the permissible region (position from 0 to 1); it is clamped to the nearest limit by the aforementioned clamping technique. The cost function is evaluated for each particle. The used cost function is the normalized sum of the hardware, software, and power cost of each particle according to equation 3.

$$Cost = 100 * \left\{ \frac{HW\,\mathrm{cost}}{allHW\,\mathrm{cost}} + \frac{SW\,\mathrm{cost}}{allSW\,\mathrm{cost}} + \frac{POWER\mathrm{cost}}{allPOWER\mathrm{cost}} \right\} \quad (3)$$

Where ***allHWcost*** (***allSWcost***) is the maximum hardware (software) cost when all nodes are mapped to hardware (software), and ***allPOWERcost*** is the sum of the power cost of the all-hardware solution and the all-software solution.

According to equations 1 and 2; the particle nodes values can take any real value between 0 and 1. But, as a binary problem, the nodes values must be rounded 1 or 0. Therefore we round the position to the nearest integer (i.e. the node is mapped to hardware if the node position is lower than 0.5 and mapped to software if the node value is greater than 0.5)

The algorithm's main loop is terminated if the improvement in the global best solution ***gbest***; remains less than a predefined value (ε) for a predefined number iterations. This predefined number of the iterations and the value of (ε) are user controlled parameters.

The following experiments are performed on a Pentium-4 PC with 3GHz processor speed, 1 GB RAM and WinXP operating system. The experiments are performed using the MATLAB 7 program [22]. The parameters used for PSO experiments are as follows:
No. of particles (Population size) **n** = 60, No. of design size **m** = 512 nodes, ε = 100 * eps, where eps is defined in MATLAB as a very small (numerical resolution) value that equals 2.2204e-016 [22], for each node, the same cost values are used for the two algorithms, for PSO, $c_1 = c_2 = 2$, *w* starts at 1 and decreases linearly until reaching 0 at the end of the run. Those values are suggested in [23-24].

To measure the efficiency of the PSO algorithm, we compared it with the famous evolutionary Genetic Algorithm (GA) [1]. To get the best results for GA, we followed the suggestions in [14] where the algorithm stops after 50 unchanged iterations, but at least 100 iterations must be performed. For the sake of fair comparison between the two algorithms, the stopping criterion is made the same in PSO. Hence, *w* reaches 0 after 100 runs. The GA parameters are set as follows: Selection rate = 0.5 (best 50% of the population are kept unchanged while the others go under the crossover operators [25]). Mutation rate = 0.05 (randomly selected 5% of the population nodes change their values form 0 to 1 and vice-versa at each iteration). The mating is performed using single point crossover.

The results of the GA and PSO algorithms are shown in Figures 1 and 2 respectively.

As shown in the figures, the initialization is the same, but at the end, the best cost of GA is 143.1 while for PSO it is 131.6. This result represents around 8% improvement in the result quality in favor of PSO. Regarding the algorithm processing time, of PSO terminates after 0.609 seconds while GA terminates after 0.984 seconds. This result represents around 38% improvement in performance in favor of PSO.

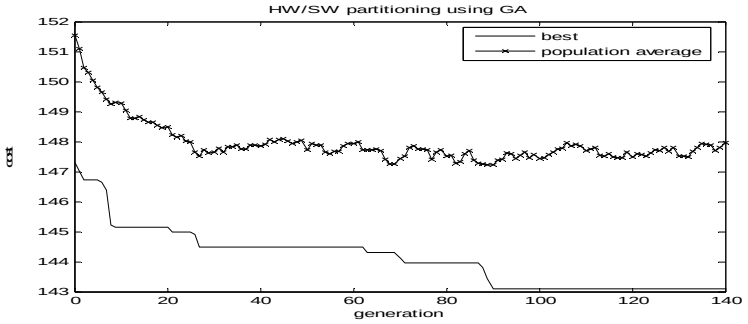

Figure 1. Best cost and Average Population cost of GA

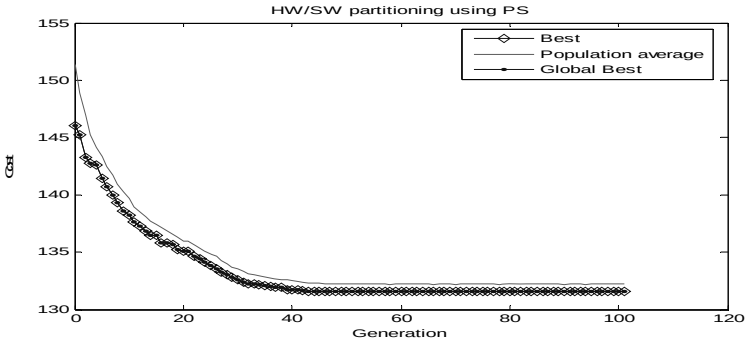

Figure 2. Best cost and Average Population cost of PSO

To further enhance the quality of the results, we tried cascading the two algorithms (GA followed by PSO and PSO followed by GA). This hybrid cascade yielded insignificant improvements [1]. This conclusion motivated us to investigate other methods to improve the quality of the result. The main idea is to cascade the algorithm with itself. Successive GA rounds failed to obtain a better result due to limitations in the algorithm itself [1]. Successive rounds of PSO yielded good results as is discussed in the following subsection.

4.1 Successive PSO (Re-excited PSO) Algorithm

As the PSO proceeds, the effect of the inertia factor (w) is decreased until it reaches 0. Therefore, v_{k+1}^{i} at the end depends only on the particle memory influence and the swarm influence (2^{nd} and 3^{rd} terms in equation 1). Hence, the algorithm may converge to local optimum positions. We thus propose to take the run's final results (particles positions) and start allover again with (w) = 1 and re-initialize the velocity (v) with new random values. We found that the result quality is improved with each new round until it settles around a certain value. Figure 3 plots the best cost in each round. The curve starts

with cost ~133 and settles at round number 30 with cost value ~116.5 which is significantly below the results obtained in the previous subsection. The program performed 100 rounds but it could be modified to stop much earlier if the result remains unchanged for a certain number of rounds.

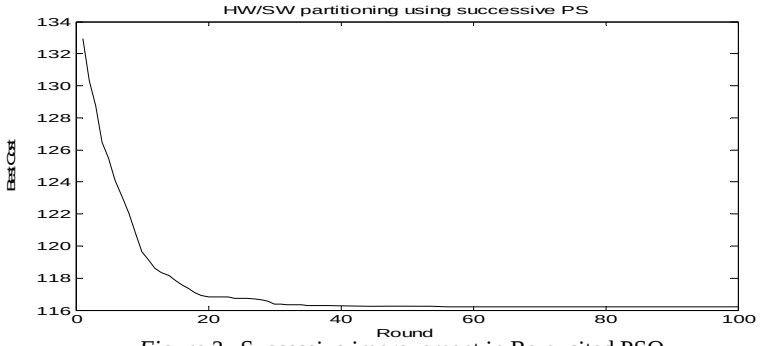

Figure 3. Successive improvement in Re-excited PSO.

As the new algorithm depends on re-exciting new randomized particle velocities at the beginning of each round, while keeping the particle positions obtained so far, it allows another round of domain exploration. We propose to name this successive PSO algorithm as the *Re-excited PSO algorithm.* In nature, this algorithm looks like giving the birds a big push after they have settled in their best position. This push re-initializes the inertia and speed of the birds so they are able to explore new areas, unexplored before. Hence, if the birds find a better place, they will go there, otherwise they will return back to the place from which they were pushed. This re-excited PSO algorithm can be viewed as a variant of the re-start strategies for PSO published elsewhere [26-29]. However, re-excited PSO algorithm is not identical to any of these previously published re-start PSO algorithms.

4.2 Comparison for Different Design Sizes

To make sure that our results are not affected by the design size, we considered different designs with size ranging from 5 nodes to 1020 nodes. We used the same parameters as described in Section 4 and we ran the algorithms on each design size 10 times and took the average results. The stopping criterion of the re-excited PSO is to stop when the best result remains the same for 10 successive rounds. Figure 4 represents the design quality improvement of PSO over GA, re-excited PSO over GA, and re-excited PSO over PSO. It shows that, on the average, PSO outperforms GA by an average of 7.8%. On the other hand, re-excited PSO outperforms GA

by an average of 17.4%, and o utperforms normal PSO by an average of 10.5%.

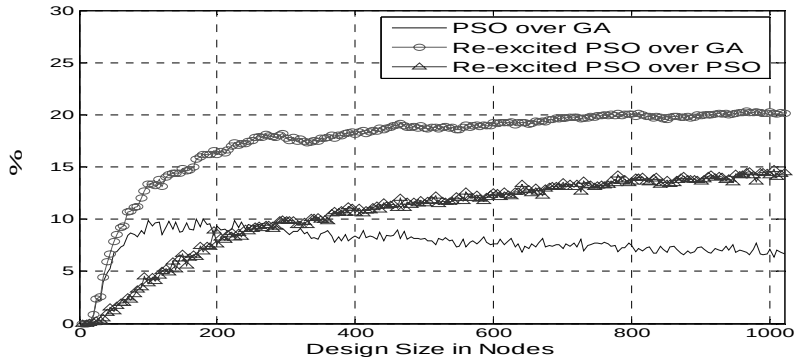

Figure 4. Quality improvement percentage for different design sizes

Figure 5 represents the speed improvement of PSO over GA for different design sizes (original and fitted curve, the curve fitting is done using MATLAB's Basic Fitting tool). Figure 5 shows that, on the average, PSO outperforms GA by a ratio 29.3% improvement in speed.

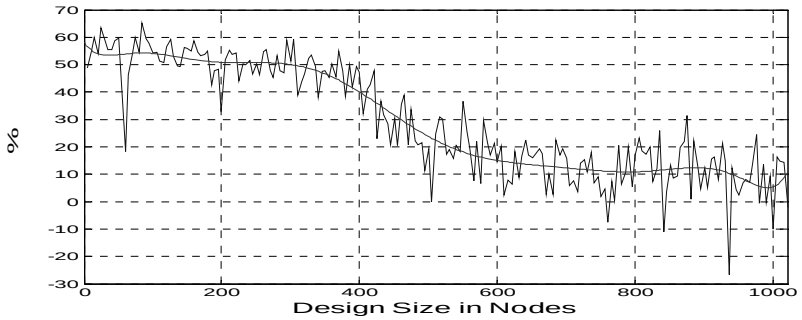

Figure 5. Speed improvement percentage for different design sizes.

From Figure 5, the speed improvement is high for small designs (from 40% to 60%), constant for large designs (around 10%), and decreases linearly from 40% to 10% for medium-size designs.

5. CONSTRAINED PROBLEM FORMULATION

In embedded systems, the constraints play an important role in the success of a design, where hard constraints mean higher design effort and therefore a high need for automated tools to guide the designer in critical design decisions. In most cases, the constraints are mainly the software deadline times (for real-time systems) and the maximum allowable area for

hardware. For simplicity, we will refer to them as software constraint and hardware constraint respectively.

Mann [14] divided the hardware software partitioning problem into 5 sub-problems (P1 – P5). The unconstrained problem (P5) is discussed in Section 4. P1 deals with both Hardware and Software constraints. P2 deals with hardware constrained designs while the software cost is unconstrained. P3, in the other hand, deals with software constrained designs while the hardware cost is unconstrained. Finally, P4 minimizes HW/SW communications while satisfying hardware and software constraints.

The constraints affect directly the cost function. Hence, equation 3 should be modified to account for constraints violations. Lopez-Vallejo[3] suggested three different techniques for the correction function formulation:

Mean Square Error minimization: This technique is useful for forcing the solution to meet certain equality, rather than inequality, constraints.

The general expression for Mean Square Error based cost function is:

$$\mathrm{MSE\_cost_i} = \mathrm{k_i} * \frac{(\mathrm{cost_i} - \mathrm{constraint_i})^2}{\mathrm{constraint_i}^2} \tag{4}$$

Where $\mathrm{constraint_i}$ is the constraint on parameter i, $\mathrm{k_i}$ is a weighting factor, and $\mathrm{cost_i}$ is the parameter cost value.

Penalty Methods: These methods punish the solutions that produce medium or large constraint violations, but allow solutions close to the boundaries defined by the constraints [3]. The penalty methods may produce invalid solutions with better overall cost than valid ones. The cost function in this case is formulated as:

$$Cost(x) = \sum_i k_i * \frac{\mathrm{cost}_i(x)}{Total\mathrm{cost}_i} + \sum_{ci} k_{ci} * viol(ci, x) \tag{5}$$

Where $\mathrm{k_i}$ and $\mathrm{k_{ci}}$ are weighting factors (100 in our case), i denotes the design parameters, ci denotes the constrains-violated parameters, viol(i) is the correction function of the constrains-violated parameters. viol(i) could be MSE_cost (equation 4) [3] or its square root [14], or any other suitable form. Note that viol(i) will be zero if the constrained parameter is not violated.

Barrier Techniques [3]: which forbid the exploration of solutions outside the allowed design space by adding a high cost term to all invalid solutions. There are two forms of the barrier techniques. The first form assigns a constant high cost to all invalid solutions (for example infinity). The second form adds an extra constant base cost term to all invalid solutions. This second form is obviously superior to the first form[14]. The base cost term of the second form can be a constant larger than any cost

value produced by any valid solution. In our case, and since our cost function is normalized, we select the constant base cost to be "one" for each violation term ("one" for hardware violation, "one" for software violation … etc.)

In order to determine the best method to be adopted, a comparison between the penalty methods (MSE_cost or its square root correction function) and the barrier methods (infinity cost vs. constant base cost) is performed.

For double constraints problem (P1), generating valid initial solutions is hard and time consuming, and hence, the barrier methods should be ruled out for such problems.

When dealing with single constraint problems (P2 and P3), one can use the Fast Greedy Algorithm (FGA) [14] to easily generate valid initial solutions. FGA starts by assigning all nodes to the unconstrained side. It then proceeds by randomly moving nodes to the constrained side until the constraint is violated.

Our experiments showed that combining the constant base cost barrier method with any penalty method (MSE_cost square term gives slight improvements over its square root) gives higher quality solutions and guarantees that no invalid solutions beat valid ones as the main disadvantage of the Penalty methods is that they admit invalid solutions.

In all experiments, all HW (SW) constraints are given relative to the area (delay) of an all-hardware (software). In the following experiments, the used cost function takes the combined form shown in equation 6.

$$Cost(x) = \sum_i k_i * \frac{\cos t_i(x)}{Total\cos t_i} + \sum_{ci} k_{ci}(Penalty\_viol(ci,x) + Barrier\_viol(ci)) \quad (6)$$

5.1 Single constraint experiments

As P2 and P3 are treated the same in our formulation, we consider the software constrained problem (P3) only. Two experiments were performed, the first one with relaxed constraint where the deadline (Maximum) delay is 40% of the delay of an all-software solution. The second one is a hard real-time system where the deadline is 15%. The parameters used are the same as in Section 4. FGA is used to generate the initial solutions and re-excited PSO is performed for 10 rounds. In the cases of GA and normal PSO only, all results are based on averaging the results of 100 runs.

For the first experiment; the average quality of the GA solution is ~ 137.6 while for the PSO it is ~ 131.3 and for the re-excited PSO it is ~ 120. All final solutions are valid since the FGA is used as the initialization scheme.

For the second experiment, the average quality of the GA solution is ~ 147 while for the PSO it is ~ 137 and for the re-excited PSO it is ~ 129.

5.2 Double constraints experiments

When testing P1 problems, the same parameters as Section 4 are used except that FGA is not used. Two experiments were performed: balanced constraints where hardware area and software delay constraints are 45%. The other one is an unbalanced-constraints problem where hardware constraint is 60% and the software delay constraint is 20%.

For the first experiment; the average quality of the solution of GA is ~ 158 and invalid solutions are obtained during 22 of the runs. The best valid solution cost was 137. For PSO, the average quality is ~ 131 with valid solutions during all the runs. The best valid solution cost was 128.6. Finally for the re-excited PSO; the final solution quality is 119.5.

For the second experiment, the average quality of the GA solution is ~ 287 and no valid solution is obtained during the runs (100 is added to the cost as a constant base penalty) as there is always violations in delay constraints. For PSO the average quality is ~ 251 with no valid solution is obtained during the runs as there is always violations in delay constraints. Finally for the re-excited PSO, the final solution quality is 125 (a valid solution is found in the seventh round). This indicates the power of re-excited PSO over both PSO and GA for hard constrained problems.

6. CONCLUSIONS

In this paper we investigated the application of Particle Swarm Optimization (PSO) technique for the Hardware/Software partitioning problem. We tested the PSO algorithm against the Genetic Algorithm (GA). Averaged over different design sizes ranging from 5 nodes to 1020 nodes; The PSO outperforms GA by a ratio of 7.8% improvements in the result quality and 29.3% speed improvement. The hybrid algorithms where one algorithm is cascaded with the other were tested. Hybrid cascading of GA and PSO proved fruitless. The best performance is obtained when one PSO round is cascaded by another PSO round with new randomized particle velocities, while keeping best particle positions obtained in the first round. We propose to name this successive PSO algorithm as the re-excited PSO algorithm. The quality of re-excited PSO solutions outperforms both GA and normal PSO design qualities by a ratio of 17.4% and 10.5% respectively.

The constrained problem is also investigated in the cases of tuning constraints and limiting constraints. The cost function is modified in each

case to accommodate the problem under investigation. Our experiments showed that re-excited PSO outperforms both PSO and GA. For hard unbalanced constrained problems, the re-excited PSO succeeded in finding a valid solution, while PSO and GA alone failed to obtain a valid solution.

The PSO features gradual exploration of the design space as the particles fly through the design space guided by the weighted sum of their inertia, memory and the flock communications. No jumps take place. This gradual exploration of the design space enables the PSO to detect steep local minima that are hard to detect by other algorithms. Also, PSO is blessed with a linear complexity of its main loop. Hence, PSO is attractive for large Hardware/software partitioning problems.

REFERENCES

[1] M. B. Abdelhalim, A. E. Salama and S. E.-D. Habib, "Hardware-Software Partitioning Using Particle Swarm Optimization Technique", Proceedings of 6th Int. Workshop on System-On-Chip for Real-Time Applications, Cairo, Egypt, pp. 189 – 194, 2006.

[2] P. L. Marrec, C. A. Valderrama, F. Hessel, A. A. Jerraya, M. Attia, and O. Cayrol, "Hardware, software and mechanical cosimulation for automotive applications", Proceedings of 9th Int. Workshop on Rapid System Prototyping, Leuven, Belgium, pp. 202 – 206, 1998.

[3] M. Lopez-Vallejo and J. C. Lopez, "On the hardware-software partitioning problem: system modeling and partitioning techniques", ACM transactions on design automation for electronic systems, Volume 8, Issue 3, pp. 269-297, July 2003.

[4] B. Mei, P. Schaumont, and S. Vernalde, "A hardware/software partitioning and scheduling algorithm for dynamically reconfigurable embedded systems", Proceedings of 11th ProRISC, Veldhoven, Netherlands, 2000.

[5] F. Vahid, "Partitioning Sequential Programs for CAD using a Three-Step Approach", ACM Transactions on Design Automation of Electronic Systems, Volume 7, Issue 3, pp. 413-429, July 2002.

[6] J. Madsen, J Gorde, P. V. Knudsen, M. E. Petersen, and A. Haxthausen, "LYCOS: The Lyngby co-synthesis system", Design Automation of embedded Systems, Volume 2, Issue 2, pp. 195-236, April 1997.

[7] N. K. Jha and R. P. Dick, "MOGAC: a multiobjective genetic algorithm for hardware-software co-synthesis of distributed embedded systems", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Volume 17, Issue 10, pp. 920 – 935, Oct. 1998.

[8] G. Stitt, F. Vahid, G. McGregor, and B. Einloth, "Hardware/Software Partitioning of Software Binaries: A Case Study of H.264 Decoder", IEEE/ACM CODES+ISSS'05, New York, USA, pp. 285 – 290, 2005.

[9] P. V. Knudsen, and J. Madsen, "PACE: a dynamic programming algorithm for hardware/software partitioning", 4th Int. Symposium on Hardware/Software Co-Design, Pittsburgh, Pennsylvania, USA, pp. 85 – 92, 1996.

[10] M. Ditzel. "Power-aware architecting for data-dominated applications", PhD thesis, Delft University of Technology, 2004.

[11] J.R. Armstrong, P. Adhipathi, and J.M. Baker, Jr. "Model and synthesis directed task assignment for systems on a chip", 15th Int. Conf. on Parallel and Distributed Computing Systems, Cambridge, USA, 2002

[12] J. Henkel, R. Ernst, "An approach to automated hardware/software partitioning using a flexible granularity that is driven by high-level estimation techniques", IEEE Transactions on Very Large Scale Integration Systems, Volume 9, Issue 2, pp. 273 – 289, 2001.

[13] N. N. Binh; M. Imai, A. Shiomi, and N. Hikichi, "A hardware/software partitioning algorithm for designing pipelined ASIPs with least gate counts", Proceedings of 33rd Design Automation Conf., Las Vegas, Nevada, USA, pp. 527 – 532, 1996.

[14] Z. A. Mann, "Partitioning algorithms for Hardware/Software Co-design", PhD thesis, Budapest University of Technology and Economics, Hungary, 2004.

[15] P. Eles, Z. Peng, K. Kuchcinski, and A. Doboli, "System level hardware/software partitioning besed on simulated annealing and tabu search", Design automation for embedded systems, Volume 2, Issue 1, pp. 5 – 32, Jan. 1997.

[16] K. S. Chatha, and R. Vemuri, "MAGELLAN: multiway hardware-software partitioning and scheduling for latency minimization of hierarchical control-dataflow task graphs", Proceedings of the 9th Int. Symposium on Hardware/Software Codesign, Copenhagen, Denmark, pp. 42 – 47, 2001.

[17] A. Kalavade and E. A. Lee. "A global criticality/local phase driven algorithm for the constrained hardware/software partitioning problem", 2nd Int. Symposium on Hardware/Software Codesign, Grenoble, France, pp. 42–48, 1994.

[18] P. Adhipathi, "Model based approach to Hardware/Software Partitioning of SOC Designs", MSc Thesis, Virginia Polytechnic Institute and State University, 2004.

[19] J. Kennedy, and R.C. Eberhart, "Particle swarm optimization", Proceedings of IEEE int. Conf. on Neural Networks, Volume. 4, Perth, Australia, pp. 1942-1948, 1995.

[20] R.C. Eberhart, and J. Kennedy, "A new optimizer using particle swarm theory", Proceedings of the 6th int. symposium on micro-machine and human science, Nagoya, Japan, pp. 39-43. 1995.

[21] www.swarmintelligence.org

[22] www.mathworks.com

[23] Y. Shi, and R. Eberhart, "Parameter selection in particle swarm optimization", Proceedings of 7th Annual Conf. on Evolutionary Computation, New York, USA, pp. .591-601, 1998.

[24] Y. L. Zheng, L. H. Ma, L. Y. Zhang, and J. X. Qian, "On the convergence analysis and parameter selection in particle swarm optimization", Proceedings of the 2nd Int. Conf. on Machine Learning and Cybernetics, Xi-an, China, Volume 3, pp. 1802 – 1807, 2003.

[25] R. L. Haupt, and S. E. Haupt, "Practical Genetic Algorithms", Second Edition, Wiley Interscience, 2004.

[26] M. Settles and T. Soule. "A hybrid GA/PSO to evolve artificial recurrent neural networks" Intelligent Engineering Systems through Artificial Neural Networks, volume 13, pp. 51-56, St. Louis, USA. 2003. ASME Press.

[27] J. Tillett, T.M. Rao, F. Sahin, and R. Rao, "Darwinian particle swarm optimization", Proceedings of the 2nd Indian Int. Conf. on Artificial Intelligence, Pune, India , pp. 1474-1487 , 2005

[28] S. Pasupuleti and R. Battiti, "The Gregarious Particle Swarm Optimizer (GPSO)", Genetic And Evolutionary Computation Conf., Seattle, USA, pp. 67 - 74, 2006.

[29] F. Van den Bergh, " An Analysis of Particle Swarm Optimizer.", PhD thesis, Department of Computer Science, University of Pretoria, Pretoria, South Africa, 2002.