HARDWARE/SOFTWARE PARTITIONING USING PARTICLE SWARM OPTIMIZATION AND ARTIFICIAL NEURAL NETWORKS

PARTICIONAMENTO HARDWARE/SOFTWARE COM OTIMIZAÇÃO POR ENXAME DE PARTÍCULAS E REDES NEURAIS ARTIFICIAIS

Maurício Acconcia Dias[*]
Fernando Santos Osório[**]

ABSTRACT

Embedded systems became an important part of computational systems. Deciding which part of the system is going to be implemented in hardware or software is an important step of the design process with direct impact on system's cost and performance defined as hardware/software partitioning. System design methods based almost entirely on designer's experience are not able to provide optimization levels needed for current embedded systems. The problem is that designers are not able to deal with high number of components and complexity without the aid of highly automated hardware design methods. Considering embedded systems scenario this work presents a training method for hardware/software partitioning artificial neural networks. The database used to train the artificial neural network is generated by a particle swarm optimization algorithm. This work's method achieved faster database generation and more accurate network results compared to previous works results. Considering the fact that real system information that could be used to create a database is considered industrial secret by systems design companies, this method can be used as an interesting alternative for embedded systems design.

Keywords: Hardware/Software co-design. Particle Swarm Optimization. Artificial Neural Networks. Hardware/Software Partitioning.

RESUMO

Sistemas embarcados se tornaram partes essenciais de sistemas computacionais. Decidir qual parte de um sistema será implementada em hardware e qual parte do sistema será implementada em software é uma parte importante do projeto e impacta diretamente no resultado final de custo e desempenho do sistema. Devido à complexidade dos sistemas e ao alto número de componentes por placa, os designs baseados apenas na experiência dos engenheiros não são capazes de utilizar toda a capacidade do hardware fazendo com que sejam necessárias ferramentas de desenvolvimento de hardware para auxiliar nos projetos. Considerando este cenário este trabalho apresenta um método de treinamento para redes neurais artificiais capazes de realizar o particionamento hardware/software. O banco de dados utilizado no treinamento da rede é gerado por um algoritmo de otimização por enxame de partículas. O resultado deste trabalho apresenta uma melhora na geração do banco de

[*] Universidade de São Paulo (USP) – Laboratório de Robótica Móvel (LRM). macdias@icmc.usp.br
[**] Universidade de São Paulo (USP) – Laboratório de Robótica Móvel (LRM). fosorio@icmc.usp.br

dados em comparação a trabalhos anteriores. Considerando os resultados obtidos e o fato de que o banco de dados pode ser composto também por dados de sistemas reais demonstra que o sistema é uma alternativa interessante para solucionar o problema do particionamento hardware/software.

Palavras-chave: Hardware/Software co-design. Otimização por Enxame de Partículas. Redes Neurais Artificiais. Particionamento Hardware/Software.

Introduction

Embedded systems nowadays have become more complex as a consequence of the increasing number of functionalities that are requested on projects. Designed systems are not able to be implemented using fully hardware or fully software solutions. These challenges motivated the development of hardware/software co-design method that generates hybrid solutions exploring the best features of hardware and software implementations in the same design. According to Wolf (2003) these systems can be described as a joint operation component group developed for some task resolution. As the number of required functionalities increase also the complexity of the system increases adding some technical challenges to the co-design method (WOLF, 2003).

When an embedded system is designed important information can be retrieved from prototype analysis. Field Programmable Gate Arrays (FPGAs) are hardware chips that can be programmed to accurately simulate a hardware logic allowing hardware designs to be tested at low costs. After the introduction of reconfigurable devices as FPGAs and their design automated tools, design space became possible to be better explored on acceptable time (BOBDA, 2007). Considering the decreasing size of transistors and other components, exploring the design space for an embedded system design is becoming a high complexity task.

The increasing design complexity created a demand for design automated tools. This work presents a method to train an Artificial Neural Network (ANN) that receives an embedded system represented as a Directly Acyclic Graph (DAG) and classifies each module of the system as a hardware or software module. Network database in this work's method is generated by a Particle Swarm Optimization (PSO) algorithm. After training the network is able to classify different types of hardware designs instantly. Results showed that proposed

method achieved acceptable results compared to previous works and that this classifier is an interesting alternative to be incorporated to embedded systems design tools.

Next section presents the basics of hardware/software co-design, PSO and ANNs. Section three describes the state of the art related to this work and proposed method on section four. Results and analysis are presented on section five and conclusions on section six followed by references.

1 Hardware Design and Computational Intelligence

Hardware/Software Co-design is a method that explores interaction between hardware and software components during development (STRAUNSTRUP; WOLF, 1997). This method tries to increase the predictability of embedded system design by providing tools that allow designers to know if proposed system meets its requirements together with synthesis methods that let researchers and designers rapidly evaluate many potential design methodologies (WOLF, 2003).

Co-design of complex electronic systems is a hard task and computational tools are used to make design faster and increases system's degree of optimization. The introduction of FPGA's on circuit co-design enabled flexibility and made prototyping easier. Also the advent of FPGAs has made hardware/software partitioning even more relevant because they have begun to co-exist with microprocessors for computing and optimization purposes (VAHID, 2009). This fact is really important because the circuit can be developed and tested before being manufactured reducing design costs and time. This flexibility opens new digital circuit applications and the rise of new Hardware /Software Co-design problems (MICHELI; ERNST; WOLF, 2002).

1.1 Hardware/Software Co-Design

Hardware/Software Co-design method is a method usually applied on embedded systems design. The main goal of the method is to design the system using hardware and software modules considering their costs separately and the communication costs between the modules.

Method starts with a list of requirements that is presented to a group of designers. The system is then analyzed by them and a project is proposed with all necessary components and modules. Usually these modules can be implemented in hardware or in software. After defining what module will be implemented in each way the system is designed, implemented, tested and, if the system addresses all the requirements, manufactured.

The definition of which part of the system will be implemented in hardware and which part will be implemented in software is a famous problem on co-design method called Hardware/Software Partitioning. According to Vahid (2009) Partitioning is the problem of defining what module of the system will be executed as a series of instructions (software) and what module will run in parallel circuits on some chip as FPGA (hardware), such as to achieve design goals like performance, cost, size and power. The circuit usually is used as a co-processor. Designers had to perform partitioning since early days of computing, but in the 1990's the field of automated hardware/software partitioning became possible with the first design automated tools (VAHID, 2009).

Partitioning problems generally are classified as NP-Complete (non-deterministic polynomial complete) problems. Hardware/Software Partitioning problem is considered NP-Hard, that means, NP-Complete problems with optimization features (ARATÓ et al, 2003). An algorithm that can provide an exact solution for these problems is not able to do it in acceptable time. In this case using a heuristic method that can be modified to handle quickly complex problems can be a good choice.

Heuristics are methods that use strategies as guided search or bioinspired algorithms to find solutions that are sufficiently close to the exact solution in an acceptable execution time. These methods are commonly used to solve NP-Complete or NP-Hard problems because exact methods for this problems have a high computational cost related to execution time. Proposed method is composed by two widely applied heuristics methods ANNs and PSO, both described in next subsection.

1.2 Computational Intelligence

Computational intelligence is a field of study interested in how to design hardware and software capable of intelligent behavior. A considerable number of computational

intelligence algorithms are heuristics. This work method uses two well-known algorithms: PSO and ANNs.

1.2.1 Particle Swarm Optimization

Particle swarm optimization (EBERHART; KENNEDY, 2001; KENNEDY; EBERHART, 1995) is a stochastic optimization algorithm inspired by social behavior of bird flocking and fish schooling (ENGELBRECHT, 2005). PSO shares many concepts with evolutionary computation techniques such as Genetic Algorithms (GA), where there's an initial population (where each individual represents a possible solution) and a fitness function (whose value represents how far an individual is to an expected problem's solution). However, unlike GA, PSO has no explicit concepts of evolution operators such crossover, mutation or selection. There is an important conceptual contrast between PSO and Evolutionary Algorithms (EA). According to Engelbrecht (2005) EA's driving force is the best individual's survival and procreation while in PSO the driving force is social interaction among particles and knowledge exchange about search space.

The main idea of the PSO algorithm is that a group of individuals searching for something will follow the individual that is more close to find it. Each individual in PSO algorithm is called a *particle.* All particles are evaluated by a fitness function and have velocities assigned to them that are responsible to guide other particles. The search space is covered by the particles that follow the current optimum ones.

Algorithm’s first step is the generation of a *population* that is a set of particles, each one representing a possible solution for the problem. After the population is evaluated and their positions are updated the algorithm is evaluated again and the positions are updated again until a stopping criteria is achieved. Each one of these cycles is called a *generation.* Stopping criteria is usually a pre-determined number of generations or a convergence to a solution that satisfies the initial requirements. *Convergence* can be verified if a significant number of individuals have the same fitness compared to the current optimum solution or are considerably close to its fitness value.

Algorithm works with two different best values. The p_{best}value is the best value achieved by the particles and the g_{best} value is the best fitness found in all previous generations. There are two main particle update equations:

$$v[\,] = v\,[\,] + c_1 * rand(\,) * (p_{best}[\,] - present[\,]) + c_2 * rand(\,) * (g_{best}[\,] - present[\,]) \quad (1)$$

$$present[\,] = present[\,] + v\,[\,] \quad (2)$$

Equation 1 describes the update of the particle velocity vector $v[\,]$. *Present* [ ] is the current particle (solution), *rand ( )* is a number between *(0,1)*, c_1 and c_2 are learning factors. Usually $c_1 = c_2 = 2$. Equation 2 describes the update of particle's position. Basically a PSO generation evaluates the population using a fitness function, finds the g_{best} and p_{best} and for all individuals uses Equation 1 to calculate the new velocity vector and Equation 2 to update particles position. Instead using operators as Gas the PSO algorithm evolves using the velocity. The optimization process occurs in two different ways simultaneously: through cooperation (group learning) and competition (individual learning) among individuals (particles) from a population (ENGELBRECHT, 2005).

There are two key steps hen PSO is applied to optimization: the representation of the solution and the fitness function. That means, the individuals have to be designed to represent each important feature of the problem and the fitness function has to return the correct value for each individual in order to correct guide the search through the search space. Algorithms parameters are: number of particles, range of particles, v_{max} that is the maximum value that *v*[ ] can achieve, learning factors c_1 and c_2 and the stopping criteria.

In this work PSO algorithm is used to create a database used to train the ANN used as a classifier.

1.2.2 Artificial Neural Networks

Artificial neural networks are computational structures designed based on human's brain behavior. These structures are composed by units called artificial neurons (Figure 1) organized in a way that the structure can be trained and used after training for different

purposes as pattern recognition, classification, function approximation. The main feature of ANNs is the ability to generalize, that means, to be able to correctly classify an input pattern without been presented to it during training.

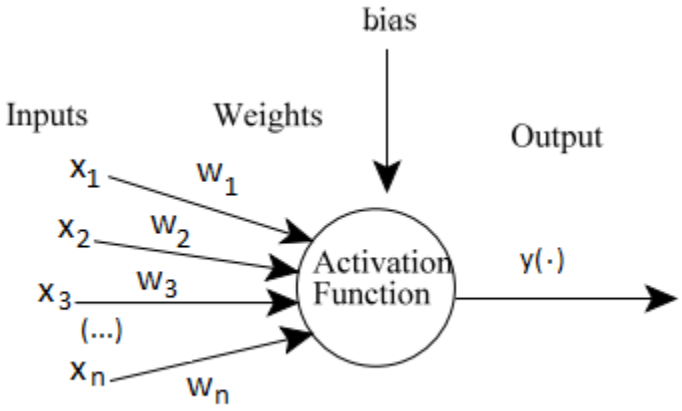


Figure 1 – Artificial Neuron. This structure is based on natural neuron's structure. It is composed by a set of inputs with weights associated to each one. These values added to bias are the input of an activation function that will be responsible for the artificial neuron's output. Output calculus is presented in Equation 3.

$$y = f\left(\left(\sum_{i=1}^{n} x_i * w_i\right) + bias\right) \quad (3)$$

Equation 3 presents the equation of the artificial neuron. The output of the neuron *y* can be defined as the output of the activation function *f* for the inputs of the neuron (x_1 *to* x_n) multiplied by the weights ($w_1 to\ w_n$) added to a *bias*. If a linear activation function is used the neuron can be defined as a linear classifier. This fact can be considered a limitation because only simple problems can be solved by linear classifiers. Rosenblatt (1957) proposed the *perceptron*, that is the same artificial neuron described by Equation 3 including a learning algorithm.

ANNs are structures that learn from a training dataset and are able to use this knowledge over new data. The knowledge is represented by the values of the weights. The training algorithm proposed by Rosenblatt (1957) for the perceptron neuron is based on weight modification considering the value of the output error of each neuron. Equation 4

presents how the new weight value w_{ij}^{t} from the input signal *i* connected to the neuron *j* is calculated considering the value of the input x_i, a learning rate η, the previous value of the weight w_{ij}^{t-1} and the output error e_j. Equation 5 describes how the error is calculated based on the difference between the desired output d_j and the current output o_j.

$$w_{ij}^{t} = w_{ij}^{t-1} + \eta * x_i * e_j \quad (4)$$

$$e_j = d_j - o_j \quad (5)$$

This algorithm can be applied to a single neuron or to a neural network composed by one layer of perceptrons, that means, the inputs are connected to a number of perceptrons and the outputs of each perceptron is one of the outputs. The problem is that even with the learning algorithm and the perceptron networks the structure stills a linear classifier. Minsky and Papert (1987) suggested that this structure will not be useful without improvements.

The problem was solved only with the design of a training algorithm that was able to train multi-layer networks that use non-linear activation functions. This algorithm is called error backpropagation and was proposed by Werbos in 1974 (WERBOS, 1994). Multi-layer perceptron networks trained using the backpropagation algorithm are able to solve non-linear problems. An example of a multi-layer perceptron network is presented in Figure 2.

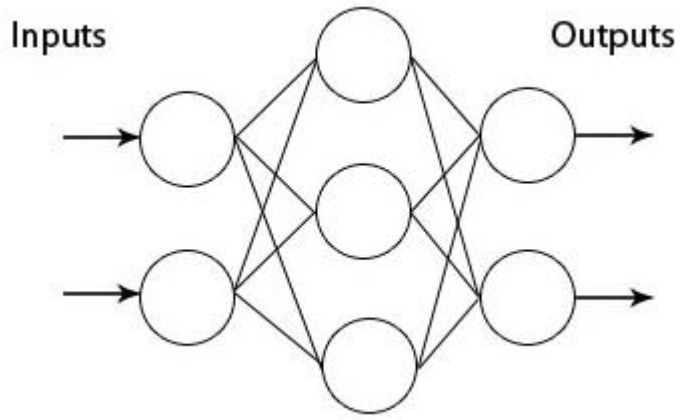


Figure 2. Multi-layer perceptron network. This network has 2 input neurons, one hidden layer with 3 neurons and an output layer with 2 neurons.

Multi-layer perceptron networks are trained by supervised learning algorithms, that means, there is a desired output for each input presented to the network and the error can be calculated in order to train the network. This work database used to train the ANN is generated by the PSO algorithm. The method is described in details in section 4.

After training, ANN turns into a black box that gives an output for a determinate input. Some advantages of ANN's are: (i) data independency caused by the fact that neurons interpret inputs only as numbers without meaning, (ii) response time that is almost instantaneous for trained networks, and (iii) after training the ANN can be converted to tables and this fact, despite causing a loss in networks generalization capabilities, allows simple implementation of ANN's in hardware.

Hardware/Software partitioning is a classification problem that can be solved by a trained ANN. The problem in this case is that is difficult to find circuit databases to train the network. In this cases is possible to generate a database from some real circuit examples (DIAS; LACERDA, 2009). This work presents a ANN trained by a database generated using a PSO algorithm.

2 Related Works

Hardware/Software partitioning is an important step of embedded systems design and the early works in this area, considering the use of computational intelligence methods to solve the problem, are dated from more than 20 years ago. A considerable number of computational intelligence algorithms and other techniques were applied to solve the problem, but hardware/software partitioning stills a challenge (BHUVANESWARI; JAGADEESWARI, 2015).

Heuristics are interesting options to solve NP-HARD problems as hardware/software partitioning, but some of the algorithms that can be used as classifiers need a database to be trained. In this particular problem is difficult to create or find a database available because the hardware/software codesign of embedded systems is a key step directly related to product's final costs and performance.

This problem can be solved using at least two different approaches. The first and most intuitive one is to use algorithms that don't need a database. A considerable number of works

used different techniques to solve the hardware software partitioning problem as exact algorithms including banch-and-boud and dynamic or lienar prograaming approaches, genetic algorithms, simulated annealing, tabu search, expert systems, binary constraint search algorithms. Examples of research works using of each one of the enumerated techniques are presented by Bhuvaneswari and Jagadeeswari (2015).

Second approach is to use a classifier that has to be trained using a database and generate the database. Neural networks can be used as classifiers and usually achieve good results after a well designed training step (HAYKIN, 1998). Neural networks can be an interesting choice for hardware/software partitioning but still not well explored. Guo et. al. (2006) proposed a partitioning for real-time operating systems using a discrete hopfied neural network, but the training part is omitted and the results are porly explained. Despite of that it is difficult to see why using a hopfield network, that works as a content-adressable memory, is a good choice if no information about the patterns used to train the network is provided. The solution for a partitioning problem of a real-time operating system that has a considerable number of specific constraints cannot be applied to embedded systems in general. Pan et. al. (2011) also present a solution using hopfield networks with no result comparison or detailed information about training.

Ma, Wang and Li (2006) proposed a tabu search neural network to solve the partitioning problem. Tabu search neural network is a new approach proposed by the authors poorly tested and considered a good choice to solve the problem. The structure needs to be more tested. Chang and Xiong (2007) also proposed a neural partitioning based on Pulse Coupled Neural Networks. Results showed that this structre can be a good choice, but no comparisons were presented.

An alternative for database problem was proposed by Dias and Lacerda (2009). The idea was ot use a genetic algorithm to generate the partitioning database and than train a Multi-Layer Perceptron (MLP) network. The problem is genetic algorithm's execution time and the results of the trained netork that didn't achieved the expected precision. Based on Dias and Lacerda's database generation idea this work proposes a method that uses a PSO algorithm that is faster and more accurate to generate a database that is used to train a MLP network.

3 Method

Proposed method (figure 1) initiates with the generation of the input file containing graphs for partitioning. The result of generation is a text file with valid graphs (DIAS; LACERDA, 2009), that means, graphs generated with small modifications from real circuit graph presented by Hidalgo and Lanchares (1997) without partitioning results.

After that, PSO algorithm give the partitions for t_r (limit execution time) from 35 to 130 (seconds for example) for each graph, considering five second intervals (35, 40, 45 ... 130). The result of this process is a database, on text file format, with input/output pairs that can be used on ANN training. ANN is trained using this database and a supervised learning algorithm. The results given by trained ANN are compared to the results of GA and PSO algorithm for the real graph. We consider good results when ANN results respect limit execution time constraints and optimum results when the execution time of the solution is equal to solution of Hidalgo and Lanchares for the same graph.

Particle Swarm Optimization algorithm preserved proposed methodology's important characteristics like flexibility, achieved due to the simple fitness function proposed and fast database generation.

The mains idea is to compare the results of PSO database generation with Dias and Lacerda (2009) database generation and change ANN architecture to optimize training time and mean square error (MSE). After comparing the results, it will be possible to decide which evolutionary technique is better for database generation and what ANN topology is more convenient. Figure 3 illustrates proposed method.

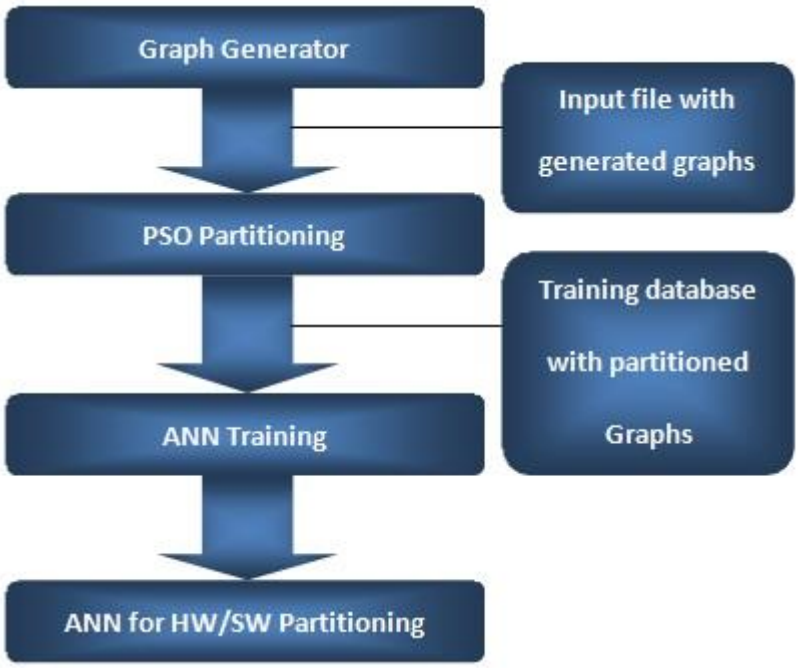


Figure 3. Proposed method's steps. Initially graphs are generated using preciously described technique. After the generation a PSO algorithm partitions generated graphs. After the partitioning a database is created with each generated graph and their respective partitioning as input/output pairs. The result of the proposed method is an ANN trained using this database.

3.1 Graph Generation

PSO input graphs for partitioning are represented as Direct Acyclic Graphs (DAGs), a common representation for hardware/software partitioning algorithms (BHUVANESWARI; JAGADEESWARI, 2015). As showed on Figure 4, a DAG has a group of nodes that represents system tasks and a group of directed edges that denotes the relationships between the nodes.

Formally, a task graph consists on a set of nodes, a set of directed edges, the configuration of nodes and a set of node ports. This work's system representation is a task graph (that is also a DAG) with nodes configured with respective information. Input file with randomly generated graphs consider only communications forward, that means, instead of all possible communications for each node now the possible communications are calculated as: *[(number of nodes) - (node number+1)]*. On a seven node graph, for node 0 there are 6 possible communication connections, for node 1 there are 5 possible connections and so on.

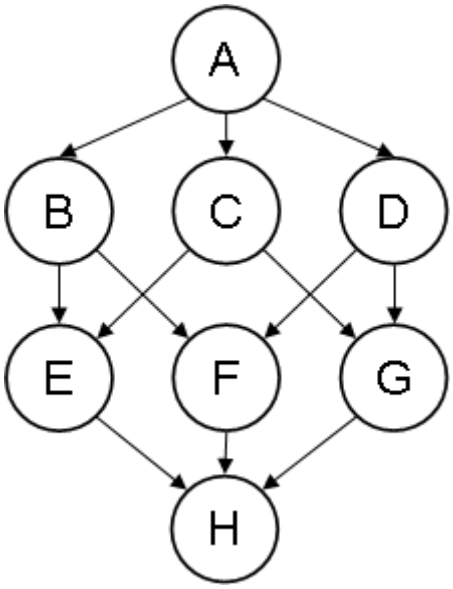


Figure 4. Example of a Directed Acyclic Graph (DAG). A Task Graph (TG) is a DAG where every node represents a system task and the values associated to the edges represents the relations between the tasks, for example, the transition time and resources needed.

Every graph in the input file is represented by a 50 value line where the first value is the execution time limit (t_r), and for each one, of the seven nodes of each graph, values are arranged as follows: hardware execution time (t_{hw}), hardware costs (c_{hw}), software execution time (t_{sw}), software costs (c_{sw}) followed by one value of communication for each node that the current node connects. Graph edge values represents communication costs. These seven nodes represent seven system components that can be implemented in both hardware and software.

Database graphs can be generated in many ways. Most efficient way to create a database for partitioning problem is to get together a large amount of Hardware/Software Co-design information from real systems of hardware development companies. Due to previously presented difficulties the database was generated according to Dias and Lacerda (2009) work.

3.2 Generated Graphs Partitioning

A discrete version of particle swarm optimization algorithm was implemented. The fitness function equations for this algorithm are detailed in (1), (2) and (3). Equation (2) presents total execution time calculus (t) where t_{hw} is hardware execution time and t_{sw} is

software execution time. In equation (3), total cost is calculated T_c with hardware costs c_{hw}, software costs c_{sw} and communication costs c_c only considered between nodes with different implementations (one in hardware other in software). In all equations t_r represents the execution time limits for each task.

All parameters are multiplied by sub-particle (s_p) in accord to its value. If the node is implemented in hardware s_p value is set to 1 and only hardware information is considered and if node is implemented in software s_p is 0 and only software information is considered. These two results are part of equation (1) that represents fitness function. The result for a graph is penalized for k_1 and k_2 where $k_1 = 1000$ and $k_2 = 1$. These values were empirical results of previous works. If total execution time is higher than maximum constraint time t_r the first part of the equation achieves a high value, otherwise it gets a value that practically considers only the costs of the implementation. In this case, PSO algorithm is used to minimize fitness function value.

$$f = \begin{cases} k_1 * (t - t_r) + k_2 * T_c & if\ t \geq t_r \\ T_c & if\ t < t_r \end{cases} \qquad (1)$$

$$t = \sum t_{hw} * s_P + \sum t_{sw} * \neg s_P \qquad (2)$$

$$T_c = \sum c_{hw} * s_b + \sum c_{sw} * \neg s_b + c_c \qquad (3)$$

Maximum speed in the range *[-4,4]* to avoid saturation of the sigmoid function. PSO confidence is used to define individual (c1) or social (c2) tendency importance. Default PSO works with static and equal trust values (c1=c2), that means, the group experience and the individual experience are equally important (called Full Model). In this case c1 and c2 were set to 2.0.

Social structures (or topologies) de ne ways on gbest obtaining, that can be best global or neighbor. Star-type neighborhood (where particle knows the best position found between all particles) is proved to be one of the best options for PSO (ENGELBRETCH, 2005). Algorithm implements star-type social structure where all particles have connections to each

other, in practical terms, that means there is a unique gbest for all particles. Other structures allow *gbest* to be generated on particle subgroups. Star-type model has faster convergence compared with other structures.

3.3 Artificial Neural Network Design and Training

Database generation, on this method, results on a file that contains input-output pairs consisting in graphs and their respective partitioning. Database file was used on network supervised training.

Fast Artificial Neural Network (FANN[1]) implementation of artificial neural networks was chosen for this work. FANN is a library that implements multi-layer perceptron ANN's and some training algorithms. Compared to previous network proposed Dias and Lacerda (2009), the architecture of neural network designed in this work stills feedforward multi-layer perceptron but now with 50 inputs. The same representation of DAG's adopted to database generation algorithm.

Architecture changes impact directly on the number of neurons on each hidden layer of the ANN. Network with better output values is composed by 3 hidden layers of 4 perceptron neurons. All layers steepness were of *0.4*. Training algorithm used was resilient back-propagation and all layer's activation function was sigmoidal symmetric. Network outputs are 7 corresponding to each one of the seven nodes of input DAGs that will be implemented in hardware (0) or in software (1). Figure 3 illustrates final ANN.

The algorithm was also congured with default error function and maximum of *1000* epochs. Network training is interrupted by maximum number of epochs or desired MSE (medium square error).

4 Results

Database generated by a Genetic Algorithm execution time, with the same fitness function used for particle swarm optimization algorithm, is three orders of magnitude slower

[1] http://leenissen.dk/fann/wp/

than PSO algorithm execution time. The improvement is significant because this algorithm will be executed thousands of times.

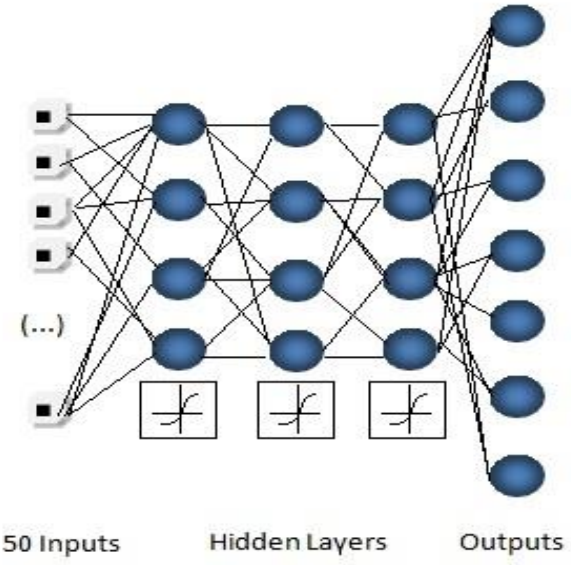


Figure 5. Artificial Neural Network architecture. The final architecture was composed by 50 input neurons, three hiddens layer with four neurons each and seven outputs. All neurons were configured to use sigmoid simmetryc as activation function.

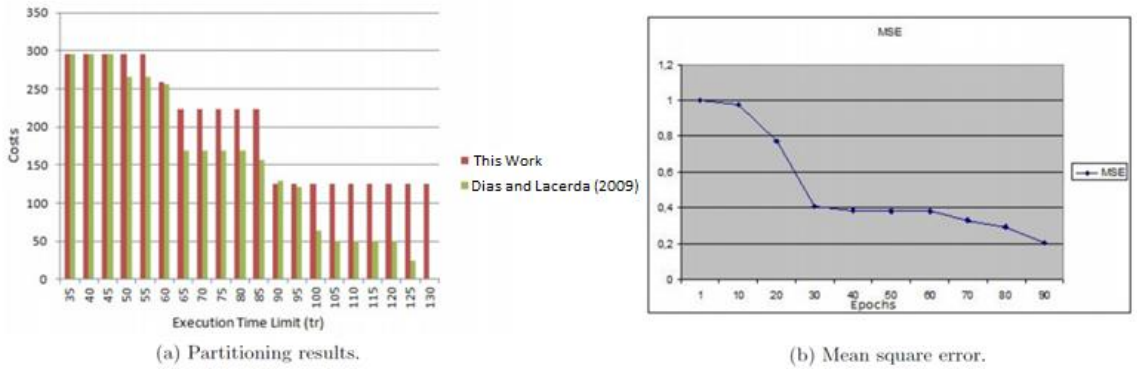
After several simulations to evaluate fitness decreasing (in accord to inertial parameters, swarm size and number of generations), population of 15 individuals and 15 generations achieved satisfactory results with *100%* of convergence. For 10 individuals and 10 generations only *20%* of simulations converged. These results had inertia parameter of *0.9*.

Neural network's parameters were changed during exhaustive testing part of development. Networks with more than four neurons per layer obtained higher MSE and training time while networks with less than four neurons weren't able to solve the problem obtaining constant error with no training evolution and resulting on invalid partitioning. Activation function changes from sigmoid symmetric (range -1 to 1) to sigmoid (range 0 to 1) worsened the precision of network results.

With new architecture for neural network, with 4 perceptron neurons on each one of the three layers, MSE of training achieved 0.28, resulting on significant reduction from Dias and Lacerda's (2009) network 1,5 error. From generated database, *80%* of the graphs were

selected for training and 20% for validation. Higher learning steepness parameters and lower learning steepness parameters also a ected network results increasing MSE resulting on *0.4* and *0.5* as the best parameters, with no difference between them. Low number of layers and neurons per layers reached better training time.

During implementation database balancing was tested and the best output results were for sequentially organized data. Sequentially means, in this case specifially, that the sequence of partitioning should be the same DAG with crescent t_r on limited range. Data arranged this way improved ANN partitioning results. Output network values, for this work, can't be normalized because there's information loss caused by the lack of pattern related to the value of the changes.



(a) Partitioning results. (b) Mean square error.

Figure 6. Results of this work's method. (a) Comparison between partitioning results of Dias and Lacerda (2009) and this work results. (b) Mean square error for this work's ANN.

For the same test graph used by Hidalgo and Lanchares (1997), partitioning results are compared to GA results in Figure 6(a). GA results are exactly the same as PSO results. Training MSE for the best ANN can be seen in Figure 6(b). The results of Artificial Neural Network are better then results achieved by Dias and Lacerda's method, because all network outputs are valid (with *25%* of optimum results) and in this case the numbers that network tried to change (form 0 to 1 or form hardware to software comparing to previous partitioning) decreased or increased more significantly. For example, for 1 to 0 previous network modifies 0.1 on double results (*0.97 - 0.87*), new network modifies *0.3* or *0.4* (*0.97 - 0.67*). This fact is caused by better ability of generalization of the new ANN.

Conclusions

As expected, due to previous comparisons and related works, PSO generated the same database but in lower magnitude of execution time. This result is very important because databases are usually very large and it reduces the total time for method application. Embedded systems design today uses Hardware/Software Co-design and this technique can reduce significantly design time.

The advantage of Artificial Neural Networks usage in this case are response time and generalization feature. Fast response time is an inherent characteristic of chosen ANN architecture because after training these networks can give and output almost instantaneously. Design space exploration is a time consuming task and a fast response time is an imperative feature for an automatic method.

Generalization feature is also important on design space exploration because some unpredicted case can be well partitioned by ANN based on knowledge acquired in the training process. Techniques without this feature couldn't be able to handle these unknown cases in an acceptable way.

A database with real partitioning graphs will result on ahigher generalization capacity. Considering results on Artificial Neural Networks, most of all implementation possibilities for multi-layer perceptron networks using supervised learning with resilient back propagation algorithm were implemented and tested and the results are better but not as expected.

Obtained results, for similar applications, can be applied. This work, as other works using artificial neural networks, proposed a network for an specific circuit, so each different case will have to follow the method in order to use artficial neural networks or will have to adequate system modeling.

PSO is faster than GA for database generation and new architecture achieved better partitioning results and this fact indicates the use of this work’s methodology for database generation.

For future works some new architectures can be tested and other types of learning algorithms either. One possible way to solve problems is changing this database fitness function to another one because this function may be experiencing some dificulty for

network's learning process. This technique can be a part of a Co-design CAD tool in cases that fast results are needed but the network should support larger input graphs. Another improvement for this methodology is to find a better way to build database input graphs using softwares that generates graphs automatically as Task Graphs For Free (TGFF[2]) or searching for real project graphs. These partitioning results can also be implemented on FPGA's to be validated on real hardware environments.

References

ARATÓ, P.; JUHÁSZ, S.; MANN, Z.; ORBÁN, A.; PAAP, D. Hardware-software partitioning in embedded system design. **Proceedings of the IEEE International Symposium on Intelligent Signal Processing**. p. 197-202, 2003.

BOBDA, C. **Introduction to Reconfigurable Computing:** Architectures, Algorithms, and Applications. Houten: Springer Publishing Company, Incorporated, 2007.

BHUVANESWARI, M. C., JAGADEESWARI, M. Hardware/Software Partitioning for Embedded Systems. In BHUVANESWARI, M. C. Ed. **Application of Evolutionary Algorithms for Multi-objective Optimization in VLSI and Embedded Systems**. India, Springer, 2015.

CHANG, Z.; XIONG, G. Hardware/Software Partitioning of Core-Based Systems Using Pulse Coupled Neural Networks. **Advances in Neural networks. Lecture Notes in Computer Science**. v. 4493, p. 1015-1023. Berlin, Springer-Verlag, 2007.
DE MICHELI, G.; ERNST, R.; WOLF, W. **Readings in Hardware/software Co-design**. Massachusetts: Morgan Kaufmann Publishers, 2002

DIAS, M.; LACERDA, W. Hardware/Software co-design using artificial neural network and evolutionaryy computing. **Proceedings of the 5th Southern Conference on Programmable Logic (SPL)**. São Carlos, p 153-157, 2009.

EBERHART, R.; KENNEDY, J. ; SHI, Y. **Swarm Intelligence**. Massachusetts: Morgan Kaufmann, 2001.

ENGELBRECHT, A. **Fundamentals of Computational Swarm Intelligence**. Hoboken: John Wiley & Sons, 2005.

GUO, B., WANG, D., SHEN, Y., LIU, Z. Hardware–software partitioning of real-time operating systems using hopfield neural networks. **Neurocomputing**. v. 69, n. 16-18. Elsevier, 2006.

[2] http://ziyang.eecs.umich.edu/~dickrp/tgff/

HAYKIN, S. **Neural Networks:** A Comprehensive Foundation. Prentice Hall. 2nd ed. Upper Saddle River, NJ, USA, Prentice Hall 1998.

HIDALGO, J.; LANCHARES, J. Functional paritioning for hardware-software codesign using genetic algorithms. In: EUROMICRO, 23., **Proceedings**... Budapeste, Hungary, 1997. p. 631-638.

KENNEDY, J.; EBERHART, R. Particle swarm optimization. In: IEEE INTERNATIONAL CONFERENCE ON NEURAL NETWORKS. Perth, WA. v. 4, p. 1942-1948, 1995.

MA, T., WANG, X., LI, Z. Neural Network Optimization for Hardware-Software Partitioning. In: INTERNATIONAL CONFERENCE ON INNOVATIVE COMPUTING, INFORMATION AND CONTROL, 1., **Proceedings**…, Beijin, 2006. p. 423-426.

MINSKY, M; PAPERT, S. **An Introduction to Computational Geometry**. Cambridge, MA: The Mit Press, 1987.

PAN, Z. et al. Hardware-software partitioning for the design of system on chip by neural network optimization method. In: INTERNATIONAL SYMPOSIUM ON PRECISION ENGINEERING MEASUREMENTS AND INSTRUMENTATION (SPIE), 7., **Proceedings…,** Lijiang, 2011. v. 8321. , p. 1T1- 1T6.

ROSENBLATT, F. **The Perceptron**-a perceiving and recognizing automaton. Report 85-460-1, Cornell Aeronautical Laboratory, 1957.

STRAUNSTRUP, J.; WOLF, W. **Hardware Software Co-Design:** Principles and Practice. Norwell: MA Springer, 1997.

VAHID, F. What is hardware/software partitioning? **ACM SIGDA newsletter**, New York, NY. v. 39, n. 6, p. 4-7, 2009.

WERBOS, P. **The Roots of Backpropagation:** From Ordered Derivatives to Neural Networks and Political Forecasting. New York, NY: Wiley-Interscience, 1994.

WOLF, W. A decade of hardware/software codesign. **Computer**, IEEE Society Press, Los Alamitos, CA, USA v. 36, n. 4, April, 2003.