A Novel Discrete Hopfield Neural Network Approach for Hardware-software Partitioning of RTOS in the SoC

Bing Guo[1,*], Yan Shen[2], Yue Huang[3], Zhishu Li[1]

[1] School of Computer Science & Engineering, SiChuan University, ChengDu 610065, China
guobing@sohu.com

[2] School of Mechatronics Engineering, University of Electronic Science & Technology of China, ChengDu 610054, China
shenyan02@163.com

[3] Software College, Kyungwon University, Songnam, Gyeonggi-Do 405-760, South Korea
huangy3903@hotmail.com

Abstract. The hardware-software automated partitioning of a RTOS in the SoC (SoC-RTOS partitioning) is a crucial step in the hardware-software co-design of SoC. First, a new model for SoC-RTOS partitioning is introduced in this paper, which can help in understanding the essence of the SoC-RTOS partitioning. Second, a discrete Hopfield neural network approach for implementing the SoC-RTOS partitioning is proposed, where a novel energy function, operating equation and coefficients of the neural network are redefined. Third, simulations are carried out with comparisons to the genetic algorithm and ant algorithm in the performance and search time used. Experimental results demonstrate the feasibility and effectiveness of the proposed method.

Keywords: Hardware-software partitioning, RTOS, SoC, Hopfield neural network

1 Introduction

As a new type of embedded systems, a SoC (System-on-a-Chip) almost implements the functionality of an overall computer system in a single IC (Integrated Chip). In general, embedded software in the SoC is composed of RTOS (Real-time Operating System) and embedded application software. The RTOS in the SoC is shortly called SoC-RTOS. Recently, the SoC becomes more and more popular in the market, according to its architecture, SoC-RTOS functionality doesn't need to be implemented solely by software, whereas some functions of SoC-RTOS can be implemented by hardware. This can greatly improved the performance of SoC-RTOS. Thus, *hard-*

[*] Corresponding author.

ware-software automated partitioning of the SoC-RTOS (SoC-RTOS partitioning) is significant to the SoC design, *i.e.* determining which components of the SoC-RTOS should be realized in hardware and which ones should be in software. It should be pointed out that the SoC-RTOS partitioning techniques is one of the most crucial design steps in the SoC *hardware-software co-design* (HSCD) methodologies, and have a dramatic effect on the whole cost and performance of the SoC final design. Besides, the SoC-RTOS partitioning is also an important foundation of re-configurable RTOS, application-specific RTOS and RTOS automatic generation research and development [1].

Along with a growth of complexity in embedded system design, traditional methods demonstrate their weakness in automate partitioning. This motivates some studies on solving this problem in the past, e.g., Gupta and De Micheli developed an iterative improvement algorithm to partition real-time embedded systems between a co-processor and a general-purpose processor [2]; Eles et al. proposed a simulated annealing and taboo search hardware-software partitioning algorithm [3]; Saha et al. applied a genetic algorithm for hardware-software partitioning [4]; Filho et al. designed a Petri Nets based approach for hardware-software partitioning of embedded system [5]; Xiong et al. suggested a dynamic combination of genetic algorithm and ant algorithm for hardware-software partitioning of embedded systems [6]; Arató et al. presented an algorithm based on integer linear programming to solve the partitioning problem optimally even for quite big systems [7]; Stitt et al. considered a solution for dynamic hardware-software partitioning [8].

In [9], Mooney III presents a δ hardware-software generation framework for the SoC, which provides with automatic hardware-software configurability of the RTOS between a few pre-designed partitions, e.g., SFR (Special Function Register), RTU (Real-Time Unit), LC (Lock Cache), DDU (Deadlock Detection Unit) and DMMU (Dynamic Memory Management Unit). These hardware facilities, which realize the task management and IPC (Internal Procedure Call) of the RTOS, remarkably improve the performance of the multi-task SoC-RTOS. Meanwhile, a *SystemWeaver* module in the multi-core SoC architecture is designed in [10], which acts as a hardware task controller to implement the task list management and IPC of RTOS. In fact, many modern CISC and RISC microprocessors, such as Intel Corporation’s Pentium and ARM Corporation’s ARM, also provide some special control registers and its corresponding hardware circuits to support the process switch and IPC for RTOSs and other general-purpose OSs (Operating System), e.g., Windows and Linux.

However, these SoC-RTOS partitioning solutions in [9] and [10] are based on experience and difficult to guarantee an optimal partitioning. In order to further facilitate the advance of the RTOS, SoC and microprocessor, it is imperative to explore some theoretical aspects of the SoC-RTOS partitioning.

Due to the characteristics of the SoC-RTOS, the SoC-RTOS partitioning is quite different from the partitioning of embedded systems and SoCs. Usual hardware-software partitioning methods are inadequate for such SoC-RTOS partitioning tasks in many aspects. The composition of hardware and software elements in the SoC-RTOS creates some new problems, such as modeling the SoC-RTOS, refining constraints and multi-object conditions, designing an appropriate optimization algorithm, evaluating the partitioning results, and system architecture issues. In this paper, we focus on the optimization algorithm development and design of SoC-RTOS partitioning [11].

2. Description of the SoC-RTOS Partitioning Problem

The SoC-RTOS partitioning is a NP-complete problem, which main objective is to optimally allocate the functional behavior of the RTOS to the hardware-software system of the SoC under constraints. The SoC-EOS partitioning also can be considered a part of the SoC-EOS hardware-software co-synthesis in some literatures. The functional behavior of the SoC-RTOS can be modeled by a task graph. For software, a task is a set of coarse-grained operations with definite interface, which can be an algorithm procedure, an object or a component; for hardware, a task is a specific IP (Intellectual Property) module with clear functions, interface and constraints [11, 12].

To formulate our problem, the following notations are used in this paper:

G : A *directed acyclic graph* (DAG) and also refers to the task graph of a SoC-RTOS, $G = (V, E)$

V : The task node set that has to be partitioned, $V = \{v_1, v_2, \ldots, v_n\}$

E : The directed edge set that represents the control or data dependency and communication relationship between two nodes, $E = \{e_{ij} | v_i, v_j \in V, i \neq j\}$

N : Total number of task nodes belonging to G , $N = |V|$

P : A hardware-software partitioning of G

V_H : The subset of nodes partitioned into the hardware, $V_H \subseteq V$

V_S : The subset of nodes partitioned into the software, $V_S \subseteq V$

$s(v_i)$ (or s_i) : The software costs of v_i

$h(v_i)$ (or h_i): The hardware costs of v_i

$c(v_i, v_j)$ (or c_{ij}): The communication costs between v_i and v_j if they are in different contexts (Hardware or Software) whereas the communication costs between the nodes in the same context are neglected

c_i : The sum of c_{ji} , $c_i = \sum_{j=1, j\neq i}^{N} c_{ji}$

H_P : The hardware costs of P , $H_P = \sum_{v_i \in V_H} h_i$

S_P : The software costs of P , $S_P = \sum_{v_i \in V_S} s_i$

C_P : The communication costs of P , $C_P = \sum_{v_i \in V_S, v_j \in V_H \text{ or } v_i \in V_H, v_j \in V_S} c_{ij}$

$g_p(V_H, V_S)$: The total costs of P

$f_P(V_H, V_S)$: The total performance of P

Definition 1 (k-way partitioning). For given $G = (V, E)$, it is called k-way partitioning if there exists a cluster set $P = \{p_1, p_2, \ldots, p_k\}$, which satisfies:

$$\begin{cases} p_i \subseteq V, 1 \le i \le k \\ \bigcup_{i=1}^{k} p_i = V \\ p_i \cap p_j = \phi, 1 \le i, j \le k, i \ne j \end{cases} . \tag{1}$$

As $k = 2$, P is called bi-partitioning, which means that only one software context (e.g., one general-purpose processor) and one hardware context (e.g., one ASIC or FPGA) are considered in the target system; as $k > 2$, P is called multi-way partitioning, which means that multiple software contexts and multiple hardware contexts are considered in the target system. According to the architecture of the target system, the SoC-RTOS partitioning can be categorized into bi-partitioning and multi-way partitioning. Bi-partitioning is the foundation of the multi-way partitioning, and is widely applied in domain applications. Hence, the partitioning only refers to the bi-partitioning without any additional declaration in this paper.

Definition 2 (SoC-RTOS partitioning). For given $P = (V_H, V_S)$, $V_H \cup V_S = V$ and $V_H \cap V_S = \phi$, the SoC-RTOS partitioning is formulated as the following constrained optimization problem:

$$\begin{cases} \max \ f_P(V_H, V_S) \\ s.t.\ C_{\min} \le g_p(V_H, V_S) = H_P + S_P + C_P \le C_{\max} \\ v_i \in V, e_{ij} \in E, 1 \le i, j \le n, i \ne j \end{cases} , \tag{2}$$

where $C_{\min} > 0$ and $C_{\max} > 0$ are the minimal and maximum value of given costs of SoC-RTOS, respectively.

3 A Novel Discrete Hopfield Neural Network Approach

The *discrete Hopfield neural network approaches* (DHNNA) have been successfully applied to signal and image processing, pattern recognition and optimization. In this paper, we employ this type of neural network to solve the SoC-RTOS partitioning optimization problem.

3.1 Neuron expression

A neural network with N neurons is used to give a response for each of the N nodes in the graph G . The *i*-th neuron belongs to the subset with node *i*, and has an input U_i and output V_i . The output of the neuron is given by:

$$V_i = f(U_i) = \begin{cases} 0, & if \quad U_i > 0 \\ 1, & if \quad U_i \le 0 \end{cases}, \tag{3}$$

where the neuron output $V_i = 0$ indicates $v_i \in V_H$ and $V_i = 1$ indicates $v_i \in V_S$.

To avoid the local optimum caused by initial conditions, the neuron input value should be restricted within a certain range. The upper limit $U_{\max}$ and the lower limit $U_{\min}$ of the neuron input are set as follows, where the average value U_{avg} of c_{ij} is calculated from all connected task nodes:

$$U_{avg} = \frac{2}{N} C_P , \quad U_{\min} = -\frac{U_{avg}}{2}, \quad U_{\max} = \frac{U_{avg}}{2} , \tag{4}$$

$$U_i = \begin{cases} U_{\min}, & if \quad U_i < U_{\min} \\ U_{\max}, & if \quad U_i > U_{\max} \end{cases}. \tag{5}$$

3.2 Energy function

In response to the constraint and objective condition of the SoC-RTOS partitioning, an energy function consisting of the following two terms is defined by:

$$E = \frac{A}{2} E_1 + \frac{B}{2} E_2 , \tag{6}$$

$$E_1 = \sum_{i=1}^{N} f_i^2 \left(\sum_{j=1}^{N} \left(h_j V_i (1 - V_j) + s_j (1 - V_i) V_j + c_{ij} \left(V_i (1 - V_j) + (1 - V_i) V_j \right) \right) - C_{\min} \right), \tag{7}$$

$$E_2 = -f_P(V_H, V_S) = -\alpha \sum_{i=1}^{N} \left(\sum_{j=1,\, j \ne i}^{N} \left(\beta_i V_i (1 - V_j) + (1 - V_i) V_j \right) \right) , \tag{8}$$

where A and B are two positive coefficients which are specified in Subsection 3.4 below. α is the system architecture speedup ratio, which means a performance compared value between hardware-software partitioned SoC-RTOS and purely software realized SoC-RTOS; β_i is the hardware task speedup ratio and has the different values for different tasks, which means a performance compared value between hardware implementation and software implementation in the same task node.

The function $f_i(x)$ used in Eq. (7) is given by:

$$f_i(x) = \begin{cases} x + C_{\min} - (h_i + s_i + c_i), & if\ x < -C_{\min} + (h_i + s_i + c_i), \\ 0, & if\ 0 \le x \le C_{\max} - C_{\min}, \\ x - C_{\max} + (h_i + s_i + c_i), & if\ x > C_{\max} - (h_i + s_i + c_i). \end{cases} \quad (9)$$

E_1 is an energy function associated to the constraint $g_p(V_H, V_S) = \sum_{i=1}^{N} \left(\sum_{j=1, j \ne i}^{N} \left(h_j V_i (1 - V_j) + s_j (1 - V_i) V_j + c_{ij} \left(V_i (1 - V_j) + (1 - V_i) V_j \right) \right) \right)$, while E_2 is associated to the objective function $f_P(V_H, V_S)$, which indicates the relative value of SoC-RTOS performance. Furthermore, E_2 takes the minimum value when the performance of hardware-software partitioned SoC-RTOS attains the maximum [12, 13, 15].

3.3 Operating equation

The operating equation for the *i*-th neuron is governed by:

$$\begin{aligned} \frac{dU_i}{dt} &= -\frac{\partial E}{\partial V_i} \\ &= -Af_i \left(\sum_{j=1}^{N} \left(h_j V_i (1 - V_j) + s_j (1 - V_i) V_j + c_{ij} (V_i + V_j - 2 V_i V_j) \right) - C_{\min} \right) \times \\ &\left(\sum_{j=1}^{N} \left(h_j (1 - V_j) - s_j V_j + c_{ij} (1 - 2 V_j) \right) \right) - \frac{B}{2} \alpha \sum_{j=1, j \ne i}^{N} \left((\beta_i + 1) V_j - \beta_i \right). \end{aligned} \quad (10)$$

In order to avoid the local optimum and obtain a high quality solution within a limited computation time, a noise term D given by Eq. (11) is added to the operating equation (10); that is,

$$D = \eta (1 - 2V_j) = \begin{cases} +\eta, & if\ V_j = 0 \\ -\eta, & if\ V_j = 1 \end{cases}. \quad (11)$$

If the noise term D is kept adding in the updating rule, the state changes excessively and even a local optimum solution may not be reachable. Hence, the term D will be discarded in the operating equation as $[t/T_0] \ge \lambda$, where $[\bullet]$ is a round-off operator, i.e., it gives an integer most close to the entity, $\lambda = T_0 - (t \times T_0)/T_{\max} - 1$, T_0 is a positive coefficient and $T_{\max}$ is a maximal step of iterations.

3.4 Setting of coefficients for the operating equation

The coefficient A depends on the average value $\overline{\varpi}$ of the task node costs; that is,

$$\overline{\varpi} = \frac{H_P + S_P}{N} . \qquad (12)$$

The coefficient B depends on the U_{avg}. In this study, we set $A\overline{\varpi} = KBU_{avg}$, where K is a regularizing constant. In our simulations, we take $K = 1/3$, $B = 1$, $T_0 = 10$, $T_{\max} = 200$ and $\eta = U_{avg} / (3 + 10 \times \rho)$. Here, ρ is the edge generation ratio of the graph G ($0 < \rho \le 1$) [14].

The values of α and β_i were obtained empirically, and are set as random numbers in the intervals $[1.5 , 2]$ and $[2 , 4]$, respectively [8].

Note that $C_{\min} = \sum_{i=1}^{N} s_i$ is a minimal cost if the SoC-RTOS is totally realized by the software, and $C_{\max} = \sum_{i=1}^{N} h_i$ is a maximal cost if the SoC-RTOS is totally realized by the hardware. To achieve a more practical and favorable solution, we amend the maximal cost to be $C_{\max} = \frac{1}{2}\sum_{i=1}^{N} h_i$ in this study [12, 13, 14].

4 Performance Evaluation by Simulation

To verify the feasibility and effectiveness of the proposed method in this paper, we employed the similar simulation methods used in [6], [14] and [15]. Also, a comparative study was carried out with the *genetic algorithm* (GA) and *ant algorithm* (AA).

4.1 Target system architecture

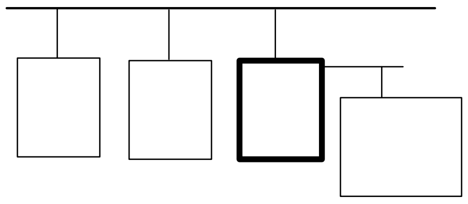

Fig. 1. Abstract model of target system architecture

This study targets the bi-partitioning problem, so there is one processor and one programming hardware component (e.g., FPGA) in the target system. We use the Spartan-3 S1000 chip manufactured by Xilinx Corporation as our FPGA model, which could contain 4 processor cores and 17,280 *programming logic blocks* (PLBs) at most.

In our experiments, we only use one processor and 15,452 PLBs. The target system architecture is shown in Fig. 1. Since software is stored in memory and executed by MPU, ARM core and memory represent the software. FPGA PLBs represent the hardware indicated by bold line box in Fig. 1.

4.2 Simulation conditions

To date, no standard benchmark and test cases for this topic is available. The methods commonly adopted in the literature are to generate the random DAG, and to assign some attributes to the nodes and edges.

To simplify our simulations, the following assumptions are made:

(1) The costs (e.g., running time and occupied hardware area) of task implementation on the processor and PLBs are static and can be calculated in advance.

(2) The costs of communication between the nodes in different contexts are constant during the execution time.

(3) To compare with the software implementation under equal conditions, the parallel of hardware implementation is neglected.

In this simulation, we constrain the settings as follows:

(1) Use the GVF (Graph Visualization Framework) software package to generate 5 groups of random DAG as our task graphs. To achieve the exact results in a limited time, the number of task nodes (N) in each group is set as 50, 100, 300, 800 and 1200, respectively. Each group has 30 sample graphs, in which each graph has the different edge generation ratios (ρ). The average value of 30 samples in each group is taken as the final performance result of this group [6].

(2) The costs of task nodes and communication costs of edges, each task node is related with two functions for one is a hardware function and another is a software function, while each edge is associated with one function. The output of function is taken as the cost of task node and edge. The appropriate cost function for each task node and edge are chosen from the MediaBench benchmark program package [6].

(3) As the initial partitioning, $N/2$ task nodes are assigned to each subset.

(4) The simulation environment used the Intel Celeron 2.6GHz processor, 512MB SDRAM, Linux 9.0 operating system and KDevelop 3.2 IDE.

4.3 Simulation results and analysis

Table 1 shows the experimental results of search time and $f_P(V_H, V_S)$ produced by the DHNNA, GA and AA on the different node number. Fig. 2 shows the relationship between the search time used and count of task nodes in these three algorithms.

It is observed that the search time from the DHNNA is shorter than that obtained by GA, and slightly worse than that obtained by the AA. However, the overall performances obtained from the DHNNA are better than the others. In particular, with the increase of node number, the DHNNA remarkably outperforms the other algorithms.

In the target system architecture shown in Fig. 1, the aim of this experiment is to optimize the running time of SoC-RTOS under the occupied hardware area constraint. In fact, the DHNNA can be also applied to the hardware-software partitioning of embedded system and SoC after some modification, while taking into account other constraints and optimization performances, such as energy consumption, hard real-time and multi-processor.

Total DAG Nodes	DHNNA		GA		AA	
	Time (s)	$f_P(V_H,V_S)$	Time (s)	$f_P(V_H,V_S)$	Time (s)	$f_P(V_H,V_S)$
50	17.91	89.32	21.53	84.56	14.67	83.2
100	134.47	311.63	252.21	294.16	109.28	256.43
300	659.02	886.57	948.24	743.83	570.86	694.37
800	2623.16	2306.62	3581.64	1875.97	2367.29	1562.68
1200	6582.74	3608.48	9157.36	2846.06	6136.82	2593.68

Table 1. Comparison of DHNNA, GA and AA

Along with an increase of the node number, the value of $f_P(V_H,V_S)$ is increasing continuously. This is a main disadvantage of the DHNNA. Practically it is expected to give an exact and more stable solution for the $f_P(V_H,V_S)$. The reason resulting in such a case may be of the initial condition and network parameters.

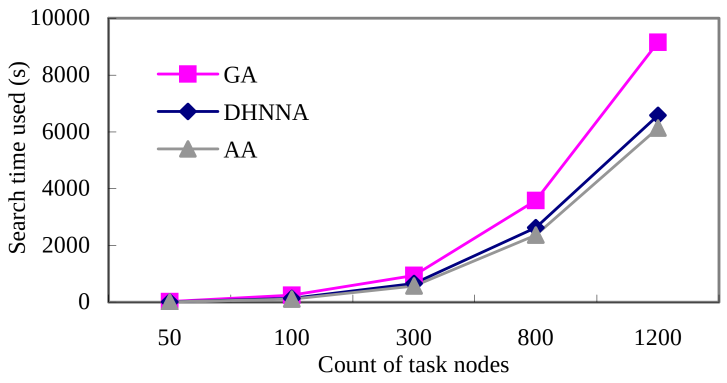

Fig. 2. Running-time/Nodes curve

5 Conclusions

In this paper, we developed a discrete Hopfield neural network approach for solving a problem of SoC-RTOS partitioning. According to the characteristics of SoC-RTOS partitioning, a new energy function for a Hopfield neural network is defined, and some practical considerations on the state updating rule are given. Simulation results demonstrate that our method is superior to some conventional methods, such as genetic algorithm and ant algorithm. A further investigation on the robustness of the solution with respect to the initial state and model parameters is being expected.

References

1. Wolf, W.: A Decade of Hardware/Software Co-design. IEEE Computer, 36(4) (2003) 38-43
2. Gupta, R.K., De Micheli, G.: Hardware-Software Co-synthesis for Digital Systems. IEEE Design and Test of Computers, Vol. 10 (1993) 29-41
3. Eles, P., Peng, Z., Kuchcinski, K., Doboli, A.: System Level Hardware/Software Partitioning Based on Simulated Annealing And Tabu Search. Design Automation for Embedded Systems, vol. 2 (1997) 5-32
4. Saha, D., Mitra, R.S., Basu, A.: Hardware/Software Partitioning Using Genetic Algorithm. Proc. of Int. Conf. on VLSI Design (1998) 155-159
5. Filho, F.C., Maciel, P., Barros, E.: A Petri Nets Based Approach for Hardware/Software Partitioning. Integrated Circuits and system design, 8(6) (2001) 72-77
6. Xiong, Z.H., Li, S.K., Chen, J.H.: Hardware/Software Partitioning Based on Dynamic Combination of Genetic Algorithm and Ant Algorithm. Journal of software, 16(4) (2005) 503-512
7. Arató, P., Juhász, S., Mann, Z.Á., Papp, D.: Hardware-Software Partitioning in Embedded System Design. Proceedings of the IEEE International Symposium on Intelligent Signal Processing (2003) 63-69
8. Stitt, G., Lysecky, R., Vahid, F.: Dynamic Hardware/Software Partitioning: A First Approach. Design Automation Conference (DAC) (2003) 74-81
9. Mooney III, V.J.: Hardware/software Partitioning of Operating System. Proceedings of the International Conference on Engineering of Reconfigurable System and Algorithm (ERSA'03) (2003) 31-37
10. Ignios Corporation: SystemWeaver Technology White Paper-MultiCore Enabler. Http://www.ignios.com (2005)
11. Jerraya, A.A., Yoo, S., Verest, D., When, N.(eds): Embedded Software for SoC. Kluwer Academic Publishers, Netherlands , ISBN 1-4020-7528-6 (2003)3-11
12. Yu, H., Gajski, D.: RTOS Modeling in System Level Synthesis. CECS Technical Report 02-25, University of California, Irvine (2002)
13. Tan, T.K., Raghunathan, A., Jha, N.K.: Energy Macromodeling of Embedded Operating Systems. ACM Transactions on Embedded Computing Systems (TECS), 4(1) (2005) 231-254
14. Wang, G., Gong, W.R., Kastner, R.: A New Approach for Task Level Computational Resource Bi-partitionging. Proc. of the IASTED Int'l Conf. on Parallel and Distributed Computing and Systems (PDCS), ACTA Press (2003) 434-444
15. Tamaki, Y., Funabiki, N., Nishikawa, S.: A Binary Neural Network Algorithm for the Graph Partitioning Problem. Electronics and Communications in Japan, Part 3, Vol. 82, No. 12 (1999) 34-42