Genetic Programming based Algorithm for HW/SW Cosynthesis of Distributed Embedded Systems Specified using Conditional Task Graph

Adam Górski and Maciej Ogorzałek
Department of Information Technologies, Jagiellonian University in Cracow,
Prof. Stanisława Łojasiewicza 11, Cracow, Poland

Keywords: Embedded Systems, Architecture, Hardware/Software Co-Synthesis, Conditional Task Graph, Genetic Programming.

Abstract: In this paper we propose a novel genetic programming based iterative improvement approach for hardware/software cosynthesis of distributed embedded systems. Unlike other genetic programming solutions for distributed embedded systems in this work the system is specified using conditional task graph. In such a graph every node represents a single task. The edge represents amount of data needed to be transferred between connected tasks, however some of the edges can be conditional. The data is transferred using those edges only if condition is satisfied. Proposed methodology is based on genetic programming. Therefore the genotype is a system construction tree. In each nodes of the tree are system building options. The next generations are obtained using standard genetic operators: mutation, crossover, cloning and selection.

1 INTRODUCTION

Embedded systems can be found everywhere. For example: in modern cars (Srovnal, Machacek, Hercik, Slaby and Srovnal, 2010), drones (Yoon, Anwar, Rakshit and Raychowdhury 2019), autonomous robots (Vaidyanathan, Sharma and Trahan, 2021) and many others. Many of such systems have distributed architecture.

De Micheli and Gupta (De Micheli and Gupta 1997) separated embedded system design process on three phases: modelling, implementation and validation. Górski and Ogorzałek added another phase to this process: assignment of unexpected tasks (Górski and Ogorzałek 2016).

Co-synthesis process (Yen and Wolf 1995) generates the architecture of embedded system. The process include hardware allocation, task assignment and task scheduling.

Co-synthesis methods can be divided on constructive and iterative improvement solutions. Constructive solutions (Srinivasan and Jha, 1995) build system step by step by making separate decisions for each task. They usually have low complexity. However such methods can easily stop in local minima of optimizing parameters. Allowing the algorithms to changed previous decisions (Dave, Lakshminarayana, and Jha, 1997) increases the complexity and the time of computation. Iterative improvement algorithms (Oh, Ha, 2002) build system by starting form suboptimal solution. Usually it is the fastest architecture. Next, by making local decisions, like allocating and deallocating resources or reassignment of tasks, they try to improve the quality of the system. Therefore those types of algorithms can escape from local minima but obtained results are still suboptimal.

Genetic algorithms (Conner, Xie, Kandemir, Link and Dick, 2005) were also used in cosynthesis process. They can escape from local minima but very often provide only acceptable results in acceptable time. Moreover the results can strongly depend on the values of the parameters (Dick, and Jha, 1998). Genetic programming solutions build system by evolving the genotype which is a decision tree (Deniziak and Górski 2008, Górski and Ogorzałek 2014a). The nodes of the tree contain system construction options. The probabilities of choosing the options were constant. Therefore the designer had to establish the value of the probability for every option. Genetic programming based adaptive methodologies (Górski and Ogorzałek 2014b, Górski and Ogorzałek 2017) can adapt to the environment

Górski, A. and Ogorzałek, M.
Genetic Programming based Algorithm for HW/SW Cosynthesis of Distributed Embedded Systems Specified using Conditional Task Graph.
DOI: 10.5220/0011011700003118
In *Proceedings of the 11th International Conference on Sensor Networks (SENSORNETS 2022)*, pages 239-243
ISBN: 978-989-758-551-7; ISSN: 2184-4380

during its work by dynamically changing the probability of using each system designing option. However the time of computation in such methods was increasing. Good results were obtained by using a penalty functions for such algorithms (Górski and Ogorzałek 2021a Górski and Ogorzałek 2021b).

One of the most popular way of representation of embedded systems is task graph. In such a graph all of the tasks need to be executed. However sometimes some of the tasks needed to be executed conditionally – only if the condition is satisfied. To solve those situations conditional task graph can be used to specify the behaviour of embedded systems (Eles, Kuchciński, Peng, Doboli, Pop, 1998).

In this paper we propose a genetic programming based method for cosynthesis of distributed embedded systems specified using conditional task graph. In our method the phenotype is a decision tree consisted of system constructing options. Next populations are created using genetic operators: mutation, crossover, cloning and selection.

2 SPECIFICATION OF EMBEDDED SYSTEM

Embedded system can be consisted of two kinds of resources: Processing Elements (PEs) and Communication Links ((CLs). PEs are responsible for tasks execution. CLs provide communication between connected tasks. PEs can be divided into two groups: Programmable Processors (PPs) and Hardware Cores (HCs). PPs can execute more than one tasks thus they are cheaper, but slower. HCs are dedicated to execute one task thus they are faster but more expensive. Let's assume that n is a number of tasks needed to be executed, m is number of PPs, and l is a number of CLs. The overall cost of designed system (C_o) can be described by the formula below:

$$C_o = \sum_{k=1}^{m} C_{PE_k} + \sum_{h=1}^{n} c_l + \sum_{z=1}^{l} \sum_{y=1}^{L_z} c_{CL_z,PC_k} \qquad (1)$$

The algorithm searches for the system with lowest value of C_o that does not exceed the time constrains.

The behaviour of embedded system in our method is described using conditional task graph G={T, E}. In the nodes of the graph the are Tasks T_i. The edges E_{ij} describe the amount of data that need to be sent between two connected tasks Ti and Tj. Except an usual edges in the graph there are conditional edges. The data flows through the conditional edges only if a condition is satisfied. Transmission time $t_{i,j}$ between two connected tasks T_i and T_j can be calculated as follows:

$$t_{i,j} = \frac{d_{i,j}}{b} \qquad (2)$$

In the formula above $d_{i,j}$ is an amount of data transmitted between the tasks, and b is a bandwidth of a CL which was used to connect the PEs. If tasks are executed by the same resource transmission time is equal to 0.

The figure 1 presents an example of a conditional task graph.

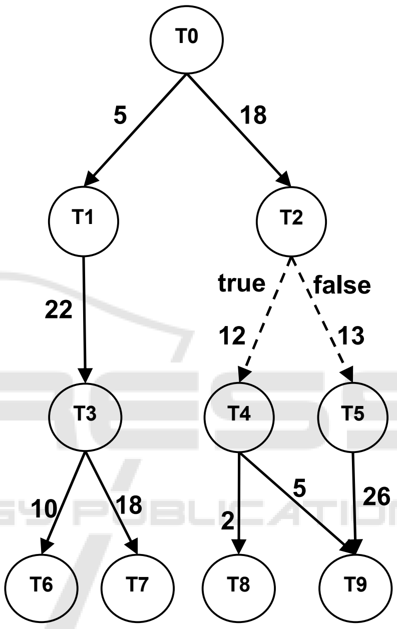

Figure 1: Example of conditional task graph.

As it is presented on figure 1 the system can execute 10 tasks: T0, T1, T2, T3, T4, T5, T6, T7, T8, T9. The edges contain amounts of data transmitted between tasks. For example the amount of data transmitted between tasks T4 and T9 is equal to 5, meanwhile amount of data transmitted between T3 and T7 is equal to 18. In the graph there are two conditional edges: before T4 and T5. In the example the condition of edge before T4 is equal to true and the condition of edge before T5 is equal to false. That means the task T5 will not be executed in the example.

Table 1 includes example of a database for the system described by the graph from figure 1. It contains the values of time (t) and cost (c) of execution of each task on every PE, and the costs of connecting PEs to CLs.

Table 1: Example of resource database.

Task	PP1 C=280		PP2 C=350		HC1		HC2	
	t	c	t	c	t	c	t	c
T0	60	6	50	5	10	250	7	350
T1	56	5	44	5	8	150	6	165
T2	100	2	90	1	23	155	15	190
T3	43	8	40	6	12	98	2	160
T4	23	12	21	10	5	86	1	110
T5	120	3	100	1	15	100	11	180
T6	49	10	39	7	1	123	6	154
T7	38	15	30	14	5	106	4	176
T8	133	2	110	2	12	180	10	240
T9	34	21	42	19	4	205	3	250
CL1, b=8	c=3		c=6		c=42			
CL2, b=9	c=5		c=8		c=56			

The target system can be consisted of four types of PEs: two types of PPs (PP1 and PP2), and two kinds of HCs (HC1 and HC2). The cost of PP1 is equal to 280. The cost of PP2 is equal to 350. The processors can be connected using two kinds of CLs: CL1 and CL2. The bandwidth of CL1 is equal to 8. The bandwidth of CL2 is equal to 9. The cost of connecting HCs to CL1 is equal to 42, to CL2 is equal to 56. Cost of connecting PP1 and PP2 to CL1 is equal to 3 and 6. The cost of connection PP1 and PP2 to CL2 is equal to 5 and 8.

3 THE ALGORITHM

The approach we presented in this paper is a constructive algorithm based on genetic programming. Therefore every genotype of each individual is a tree. Because of constructive nature of the algorithm, structure of the genotype must be the same as structure of conditional task graph. The first node is an embryo. It contains the random implementation of the first task. Every next node of the tree contains system construction option for single task. The options are given in table 2. The table contains options for PEs and CLs. For each option there is given a probability of being chosen. The values of probability are different for each option. For greedy options (the cheapest and the fastest implementation) the value of probability is the lowest. For another options the value is bigger. The values of probability is not modified during the work of the algorithm.

Table 2: Options for building system.

Step	Option	Probability
PE	a. The fastest implementation	0,1
	b. The cheapest implementation	0,1
	c. The same as task's predecessor	0,2
	d. The rarest used PP	0,2
	e. min (t*c)	0,2
	f. Idle for the longest time	0,2
CL	a. The cheapest CL	0,3
	b. The fastest CL	0,3
	c. The rarest used	0,4
Task scheduling	list scheduling	

In the first generation Π individuals are created. Π is described by the following formula:

$$\Pi = \alpha * n * e \tag{3}$$

where n is a number of tasks in conditional task graph and α is a parameter which allows the designer to control the size of populations.

If any PP executes more than one task then those tasks need to be scheduled. We decided to use list scheduling to establish the order of tasks' execution. New individuals are created by cloning (Φ), crossover (Ψ) and mutation (Ω) operators. The number of generated individuals are described below by the following formulas:

- $\Phi = \beta * \Pi$
- $\Psi = \gamma * \Pi$
- $\Omega = \delta * \Pi$

In every generation there is constant number of individuals. To make it sure the following condition must be satisfied:

$$\beta + \gamma + \delta = 1 \tag{4}$$

To select individuals for each genetic operators we use rank selection. After each population all of the individuals are ranked by cost. The selection operator selects appropriate number of genotypes for each operator from the rank list but with different probability (P). The probability is depended on a position of individuals (r) in the rank list. It is described by the following formula below:

$$P = \frac{\Pi - r}{\Pi} \tag{5}$$

As it can be observed the genotypes from the top of the list have the biggest valued of the probability.

The worst individuals have the lowest value of probability but not equal to 0.

Crossover chooses Ψ individuals using selection operator. The individuals are randomly connected in pairs. Than the crossing point needs to be chosen. The crossing point must be the same for both individuals in each pair. Next subtrees are copied between chosen solutions to create two new individuals for each pair.

Mutation selects Ω genotypes using selection operator. Then for each solution one node is chosen randomly. Mutation substitutes the option in the node on another using options included in table 2.

Cloning copies Φ individuals from current population to the next one. To not to lose the best individual we assume that the best one (the first in the rank list) is always copied to the next population.

If in next ε populations better solution is not found, the algorithm will stop. All of the parameters: α, ß, ɣ, δ and ε are given by the designer.

4 FIRST RESULTS

To check the efficiency of presented approach we made some experiments using randomly generated graphs with 6, 8 and 10 nodes. The results were compared with results obtained by greedy time algorithm. They are presented in table 3 below. The parameters were set as follows: α=100, β=0,2, γ=0,7, δ=0,1, ε=5.

Table 3: Results of the experiments.

graph	**GPC**			greedy	
	t	**c**	**gen**	t	c
6 T_{max}= 1200	**772**	**783**	**5**	547	1425
8 T_{max}= 120	**119**	**1566**	**5**	85	1592
10 T_{max}= 190	**174**	**619**	**7**	184	1815

In the table 3 there are values of times (t) and costs (c) of generated systems. For an algorithm presented in this paper it is also given a number of generation in which the result was obtained. The time constrains were as follows: for graph with 6 nodes – 1200, for graph with 8 nodes – 120, and for graph with 10 nodes – 190. As it can be observed for every graph better results were generated by the algorithm presented in this paper. Costs of the system described by graphs with 6, 8 and 10 nodes were as follows: 783, 1566, 619 for algorithm GPC and 1425, 1592 and 1815 for greedy solution.

5 CONCLUSIONS AND FUTURE WORK

In this work a novel GP-based algorithm for cosynthesis of embedded systems specified by conditional task graph was presented. Unlike other GP-based algorithms for HW/SW cosynthesis in this paper we investigate the situation when in task graph exist some conditional edges.

The results presented in this paper are first obtained results by described method. To establish the quality of the results well they need to be compared with other known algorithms for HW/SW cosynthesis of distributed embedded systems specified by conditional task graphs. It is also important to compare the algorithms using bigger graphs.

In the future we plan to modify the algorithm by using another system construction options or another genetic operators. We also plan to modify the probability of choose of each options. Especially we would like to provide a version of the algorithm which will be able to change the probability dynamically during the work of the algorithm. We would like to develop an iterative improvement GP-based solution for cosynthesis of embedded systems specified by conditional task graph too. It is also important to check the influence of penalty function for described algorithm on a quality of the results.

REFERENCES

Srovnal V. Jr, Machacek, Z. Hercik, R., Slaby, R., Srovnal, V., 2010. Intelligent car control and recognition embedded system. In *Proceedings of the International Multi- conference on Computer Science and Information Technology*, pp. 831–836.

Yoon I., Anwar A., Rakshit T., Raychowdhury A., 2019. Transfer and online reinforcement learning in STT-Mram based embedded systems for autonomous drones. In *2019 Design, Automation & Test in Europe Conference & exhibition (DATE).,* pp.1489-1494, IEEE.

Vaidyanathan, R., Sharma,, G and Trahan, J., 2021. On fast pattern formation by autonomous robots. Information and Computation, 104699, Elsevier.

De Micheli, G., Gupta, R., 1997. Hardware/software co-design. In *Proceedings IEEE 95*.3 (Mar). IEEE.

Górski, A., Ogorzałek, M.J., 2016. Assignment of unexpected tasks in embedded system design process. Microprocessors and Microsystems, Vol. 44, pp. 17-21, Elsevier.

Yen, T., Wolf, W., 1995. Sensivity-Driven Co-Synthesis of Distributed Embedded Systems. In *Proceedings of the International Symposium on System Synthesis*.

Srinivasan, S., Jha, N.K., 1995. "Hardware-Software Co-Synthesis of Fault-Tolerant Real-Time Distributed Embedded Systems", In *Proceedings European Design Automation Conference*. pp. 334-339.

Dave, B., Lakshminarayana, G., Jha, N., 1997. COSYN: Hardware/software Co-synthesis of Embedded Systems. In *Proceedings of the34th annual Design Automation Conference (DAC'97)*.

Oh, H., Ha, S., 2002. Hardware-software cosynthesis of multi-mode multi-task embedded systems with real-time constraints. In *Proceedings of the International Workshop on Hardware/Software Codesign*, pp. 133–138 .

Conner, J., Xie, Y., Kandemir, R., Link, G., Dick, R., 2005. FD-HGAC: AHybrid Heuristic/Genetic Algorithm Hardware/Software Co-synthesis Framework with Fault Detection. In *Proceedings of Asia South Pacific Design Automation Conference (ASP-DAC)*, pp. 709-712.

Dick, R., P., Jha, N., K., 1998. MOGAC: a multiobjective Genetic algorithm for the Co-Synthesis of Hardware-Software Embedded Systems. In *IEEE Trans. on Computer Aided Design of Integrated Circiuts and systems, vol. 17, No. 10*.

Deniziak, S., Górski, A., 2008. Hardware/Software Co-Synthesis of Distributed Embedded Systems Using Genetic programming. In *Proceedings of the 8th International Conference Evolvable Systems: From Biology to Hardware, ICES 2008*. Lecture Notes in Computer Science, Vol. 5216. SPRINGER-VERLAG.

Górski, A., Ogorzałek, M.J., 2014a. Adaptive GP-based algorithm for hardware/software co-design of distributed embedded systems. In *Proceedings of the 4th International Conference on Pervasive and Embedded Computing and Communication Systems*, Lisbon, Portugal.

Górski, A., Ogorzałek, M.J., 2014b. Iterative improvement methodology for hardware/software co-synthesis of embedded systems using genetic programming. In *Proceedings of the 11th Conference on Embedded Software and Systems (Work in Progress Session),* Paris, France.

Górski, A., Ogorzałek, M.J., 2017. Adaptive iterative improvement GP-based methodology for HW/SW co-synthesis of embedded systems. In *Proceedings of the 7th International Joint Conference on Pervasive and Embedded Computing and Communication Systems*, Madrid, Spain.

Górski, A., Ogorzałek, M.J., 2021a. Genetic Programming based Constructive Algorithm with Penalty Function for Hardware/Software Cosynthesis of Embedded Systems. In *Proceedings of the 16th International Conference on Software Technologies (ICSOFT 2021)*, pp. 583-588.

Górski, A., Ogorzałek, M.J., 2021b. Genetic Programming based Iterative Improvement Algorithm for HW/SW Cosynthesis of Distributted Embedded Systems. In *Proceedings of the 10th International Conference on Sensor Networks (SENSORNETS 2021)*, pp. 120-125.

Eles P., Kuchciński K., Peng Z., Doboli A., Pop P., 1998. Scheduling of conditional process graphs for the synthesis of embedded systems. In *Proceedings of Design Automation and Test in Europe)*, pp. 23-26.