Genetic Programming based Iterative Improvement Algorithm for HW/SW Cosynthesis of Distributted Embedded Systems

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, Genetic Programming.

Abstract: In this work we present a novel genetic programming based iterative improvement approach for hardware/software cosynthesis of distributed embedded systems. The approach starts from a ready solution which is an embryo of a genotype. Other nodes in the genotypes are chromosomes. The chromosomes contain system refinement options. The final solution is obtained after evolution process and mapping genotype to phenotype. Unlike existing genetic programming iterative improvement methodologies our algorithm starts from randomly generated system. Therefore the search space is not constrained by any initial condition. It is also easier for the algorithm to escape local minima of optimizing parameters.

1 INTRODUCTION

Today a lot of devices can be solved as embedded systems or Internet of Things solutions (IoT). Examples of such devices can be: cars (Srovnal, Machacek, Hercik, Slaby and Srovnal, 2010), drones (Yoon, Anwar, Rakshit and Raychowdhury 2019), digital door keys (Beaufour and Bonnet 2004), parking management system (Tsiropoulou et al. 2017) and many others. Nowadays a lot of such systems have distributed structure (Martins, Tavares, Solieri, Bertogna, and Pinto, 2020). The complexity of such systems is increasing. Therefore it is very important to provide appropriate methodologies to design of embedded systems.

According to DeMicheli and Gupta (DeMicheli and Gupta 1997) exist three basic phases of designing of embedded systems: modelling, implementation and validation. Another very important problem appears when system works in defined structure and meets unexpected situation (Górski and Ogorzałek 2016).

Cosynthesis of distributed embedded systems (Yen and Wolf 1997) is a process which concurrently selects architecture of embedded system, assign tasks to the resources and schedule the tasks. Most of the cosynthesis methodologies can be divided on two groups: constructive algorithms and iterative improvement approaches.

Constructive algorithms (Srinivasan and Jha, 1995) build systems step by step making separate assignment decisions for each task. The advantage of those methodologies is low complexity. However such methodologies can stop in local minima of optimizing parameters. In (Dave, Lakshminarayana and Jha 1997) is presented a constructive algorithm that can change previously made decision, but the complexity of the algorithm is increasing.

Iterative improvement methodologies (Oh, Ha, 2002) can escape from local minima. Such an approaches starts from a suboptimal architecture. The target architecture is obtained after making local decisions like allocating or deallocating resources or changing tasks assignment. However obtained solutions can still be suboptimal.

Genetic methodologies (Conner, Xie, Kandemir, Link and Dick, 2005) can also escape from local minima but often results depends on parameters (Dick, and Jha, 1998). Very good results obtained using genetic programming (Deniziak and Górski 2008, Górski and Ogorzałek 2014a) especially genetic programming based adaptive methodologies (Górski and Ogorzałek 2014b, Górski and Ogorzałek 2017) which can adapt to the environment during its work.

In this paper we present a novel iterative improvement methodology for hardware/software cosynthesis of distributed embedded systems. The methodology is based on developmental genetic

Górski, A. and Ogorzałek, M.
Genetic Programming based Iterative Improvement Algorithm for HW/SW Cosynthesis of Distributted Embedded Systems.
DOI: 10.5220/0010391501200125
In *Proceedings of the 10th International Conference on Sensor Networks (SENSORNETS 2021)*, pages 120-125
ISBN: 978-989-758-489-3

programming (Koza, Bennett III, Lohn, Dunlap, Keane and Andre, 1997). Genetic programming evolves tree based genotypes. In the nodes of the trees are system creating functions. The final solution is created after evolution process by making genotype to phenotype mapping.

2 THE NOTATION

PE – Processing Element
CL – Communication Link
b – CL's bandwidth.
PP – Programmable Processors
HC – Hardware Core
T – task
E – edge in a task graph
G={T,E} – task graph
C_f – cost of an embedded system
c – cost of a task's execution
t_{ij} – transmission time between tasks T_i and T_j
dij – amount of data transferred between tasks T_i and T_j
t – time of a task's execution
n – number of tasks
e – number of PEs in given database
Π – number of individuals in each population
α – parameter which controls number of individuals in each population
β – parameter which controls number of individuals obtained by reproduction
γ – parameter which controls number of individuals obtained by crossover
δ – parameter which controls number of individuals obtained by mutation
Φ – number of individuals obtained by reproduction
Ψ – number of individuals obtained by crossover
Ω – number of individuals obtained by mutation
X – number of tasks in a node
P – probability of selection operator
r – position of an individual in a rank list
p – number of communication links in a solution
m – number of programmable processors in a solution
ε – parameter which controls the stop of an algorithm

3 ASSUMPTIONS

Distributed embedded system is built of two kind of resources: Processing Elements (PEs) and Communication Links (CLs). PEs execute the tasks. CLs provides communication between connected PEs. Processing elements can be divided into: Programmable Processors (PPs) and Hardware Cores (HCs). PPs are universal resources and can execute more than one task. HCs are resources dedicated to execute a single task only. The cosynthesis process consists of: resource allocation, task assignment and task scheduling. Resource allocation is a phase that selects number and type of Processing Elements. Task assignment chooses appropriate PE for each task. In this paper we use a graph representation of embedded system G={T, E}. The task graph describes dependencies between tasks. Each node of the graph contains a task (T). Each edge (E) presents amount of data that has to be sent between two connected tasks. The transmission time is described by the following formula:

$$t_{i,j} = \frac{d_{i,j}}{b} \quad (1)$$

Where b represents a bandwidth. If two connected tasks are executed on the same resources, the transmission time is equal to 0.
On figure 1 an example of a task graph was presented.

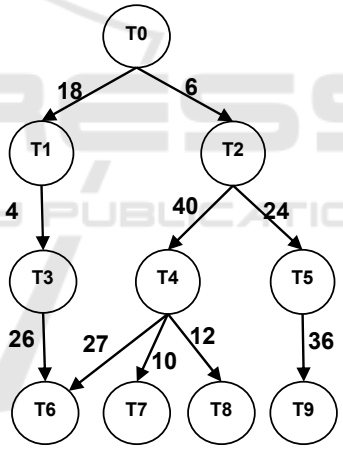

Figure 1: Example of task graph.

The graph consists of ten nodes: T0, T1, T2, T3, T4, T5, T6, T7, T8 and T9. Tasks T1 and T2 can start their execution only after execution of T0 is terminated. T3 starts its execution after T1. T4 and T5 can start the execution after T2 is executed. T7 and T8 can be executed after T4. T9 can start its execution after T9. Task T6 can start its execution after terminating execution of T4 or T3. The following groups of tasks: T1 and T2, T3, T4 and T5, T6, T7, T8 and T9 can be executed at the same time using another resources.

There are four possible kinds of processing elements: two Programmable Processors (PP1 and PP2) and two Hardware Cores (HC1 and HC2). Processing elements can be connected by two

communication links (CL1 and CL2). The costs of all processing elements (C) and connection of PPs to communication links (c) are given in the database. Cost of PP1 is equal to 130. Cost of PP2 is equal to 210. Each task is characterized by the cost (c) and time (t) of its execution. Cost of dedicated resources (HC1 and HC2) is added to the cost of tasks execution. Cost of connecting PP1 to CL1 is equal to 8. Cost of connection of PP1 to CL2 is equal to 18.

Table 1: Example of resource database.

Task	PP1 C=130		PP2 C=210		HC1		HC2	
	t	c	t	c	t	c	t	c
T0	10	9	20	5	5	300	4	380
T1	35	10	14	15	6	200	9	150
T2	18	3	15	4	10	80	2	110
T3	25	5	20	8	9	250	4	300
T4	81	23	72	16	23	180	18	190
T5	13	8	10	9	4	130	5	140
T6	19	25	16	40	9	180	8	210
T7	49	6	23	19	7	200	8	160
T8	14	8	20	10	5	200	4	300
T9	28	12	32	20	8	180	6	250
CL1, b=2	c=8		c=2		c=35			
CL2, b=12	c=18		c=12		c=41			

If n presents a number of tasks to be executed by the system described by the graph from table1, m is a number of programmable processors (PPs) and p describes number of communication links (CLs) the cost of a final system (C_f) is described by the following formula:

$$C_f = \sum_{k=1}^{m} C_{PE_k} + \sum_{l=1}^{n} c_l + \sum_{z=1}^{p} \sum_{y=1}^{P_z} c_{CL_z, PC_k} \quad (2)$$

The methodology presented in next chapter must find a system with the lowest C_f value which does not exceed the time constrains.

4 THE APPROACH

The presented methodology is genetic programming approach. Therefore at the beginning the initial population of genotypes must be created. The number of individuals in the population is described by the formula 3 below:

$$\Pi = \alpha * n * e \quad (3)$$

n is a number of tasks executed by the system, e is a number of PEs in the database, α is given by the designer.

According to genetic programming rules every individual is a tree. The first node in the tree is an embryo. In existing constructive GP based algorithms (Deniziak and Górski, 2008, Górski and Ogorzałek, 2014a) embryo is a random implementation only of the first task. Others iterative improvement GP based solutions (Górski Ogorzałek 2014b) stat with the fastest implementation of all the tasks. Unlike others methodologies in presented approach embryo is random implementation of all the tasks. Every next node include number of tasks taken from the previous node and system building option for the tasks. Such an approach is also unique in GP based co-synthesis algorithms in which so far the number of tasks in the nodes was constrained (equal to 1) or determined by genes. The number of reassignment tasks is chosen randomly from 1 to X -1. The algorithm selects task by maximum value of the following formula:

$$F_i = \max(c * t) \quad (4)$$

X is the number of tasks in predecessor node.

Number of nodes and theirs positions are generated randomly.

Table 2 include the options for reassignment of tasks. The options are divided into two groups: for PEs and for CLs. For PEs the following options are possible: the cheapest implementation of the tasks, the fastest implementation of the tasks, min value of multiply of time and cost, the same as task's predecessor and the rarest used PP. For CLs the algorithm can choose from options: the fastest CL, the cheapest CL and the rarest used CL. Each option, either for PEs and CLs, has a probability of being chosen. Options a and b for PEs have probability 0,15, option c 0,3, and options d and e 0,2. Options a and b for CLs have probability equal to 0,3, option c equal to 0,4.

Table 2: Options for building system.

Step	Option	Probability
PE	a. The cheapest implementation of the tasks	0,15
	b. The fastest implementation of the tasks	0,15
	c. min (t*c)	0,3
	d. The same as task's predecessor	0,2
	e. The rarest used PP	0,2
CL	a. The fastest CL	0,3
	b. The cheapest CL	0,3
	c. The rarest used	0,4
Task scheduling	list scheduling	

If there is more than one task assignment to one PE list scheduling is used to set the tasks' order.

Every next generation is created using genetic operators: mutation, crossover and reproductions. The evolution process is controlled by the parameters β, γ, δ which value is given by the designer. Number of individuals created by the reproduction (Φ), crossover (Ψ) and mutation (Ω) are given by the following formulas below:

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

To have the same number of individuals in every population the following condition must be satisfied:

$$\beta + \gamma + \delta = 1 \quad (5)$$

Crossover randomly selects Ψ genotypes. Then the crossing point is chosen randomly different for every genotypes. Next the subtrees of the genotypes are substituted.

Mutation randomly chooses Ω individuals and a node. Next the option in the node is substitutes on another from table 2.

Reproduction selects Φ genotypes and copies them to a new population. Every individual has a probability of being chosen. The probability depends on a position of the individual in a rank list (r) as follows:

$$P = \frac{\Pi - r}{\Pi} \quad (6)$$

The approach stops calculating if in ε next generations better solution was not found. Parameter ε is given by the designer of the system.

5 THE EXAMPLE

On figure 2 was presented an example of a genotype. The genotype was generated randomly for the graph from figure 1. The genotype consists of 5 nodes. In every node (except the first one) there is an option from table 2 for PE and for CL. The nodes consists also a number of a tasks which are taken from the previous node.

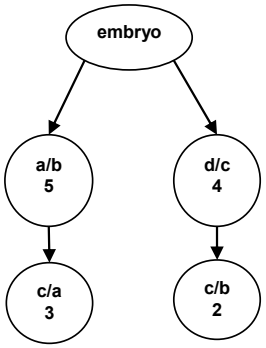

Figure 2: Example of genotype.

The first node in the genotype is an embryo. The embryo is random implementation of all the tasks. The task assignment was made as follows: PP1: T0, T2, T4 and T6, PP2: T1 and T8, HC1: T3 and T9, HC2: T5 and T7. The cost of the solution was 1335. and the time of execution of all the tasks 135. In the next node option a for PEs and option b for CLs for 5 tasks were chosen. In the third node for 4 task option d for PEs and c for CLs was selected. The next node include options c (PEs) and a (CLs) for 3 tasks. In the last one node options c for PEs and b for CLs for 2 tasks were chosen. The final cost of generated solution is 475 and the time of execution of all tasks is equal to 225.

6 FIRSTS RESULTS

Firsts results obtained by the algorithm described in this paper were compared with results obtained using constructive GP based algorithm for cosynthesis proposed by Deniziak and Górski (Deniziak and Górski 2008). The results were made on a benchmark with 10 nodes (Deniziak and Górski 2008). Algorithm proposed by Deniziak and Górski was proved to give better results than algorithm EWA for cosynthesis (Deniziak, 2004) which is not genetic approach. In table 3 the results were presented. The parameters were set as follows: $\alpha = 10$, $\beta = 0.2$, $\delta = 0.1$, $\gamma = 0.7$ $\varepsilon = 5$.

Table 3: Options for building system.

Algorithm	Tmax	t	c	generation
DGP08	1000	956	1063	4
GP21	1000	**956**	**788**	**6**

The maximum time constrained was set to 1000. Algorithm presented in this paper (GP21) generated solution with the same time as algorithm DGP08

(time was equal to 956) but with much lower time (equal to 788). The figure 3 presents graphical comparison of the results.

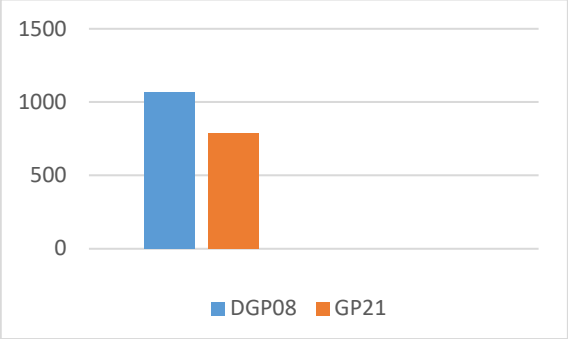

Figure 3: Graphical presentation of obtained results.

As it can be observed for used beenchmark algorithm GP21 generated result about 26% better than DGP08.

7 CONCLUSIONS AND FUTURE WORK

The paper presents a novel iterative improvement, Genetic Programming based approach for hardware/software cosynthesis of distributed embedded systems. Unlike other methodologies the algorithm starts from randomly generated solution. Therefore it is easier for the algorithm to escape from local minima of optimizing parameters. Starting from randomly generated genotype can also increase the search space. First results are promising, however the algorithm needs to be further investigated.

The future work will concentrate on providing more experimental results and test the behaviour of proposed approach. We will also provide more Genetic Programing based algorithms in which we will investigate another system building options and test others genetic operators.

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.

Martins J., Tavares A., Solieri M., Bertogna M., Pinto S., 2020. Bao: A lightweight static partitioning hypervisor for modern Multi-Core Embedded Systems. In *Workshop on Next Generation Real-Time Systems*

L. Sun, L. Zhang, D., Li, B., Guo, B., Li, S., 2010. Activity recognition on an accelerome- ter embedded mobile phone with varying positions and orientations. In Zhi-wen Yu, Ramiro Liscano, Guanling Chen, Daqing Zhang, Xingshe Zhou (Eds.), Ubiquitous Intelligence and Computing, Lecture Notes in Computer Sciences, 6406, pp. 548–562, Springer, Xi'an, China.

Beaufour, A., and Bonnet, P.,2004 Personal Servers as Digital Keys. In *Proceedings of the 2nd IEEE International Conf. of Pervasive Computing and Communications (PerCom)*.

Tsiropoulou, E. E., Baras, J. S., Papavassiliou S. and Sinha, S., 2017. RFID-based smart parking management system. In Cyber-Physical Systems, Vol.3, pp.22-41.

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 Work-shop on Hardware/Software Codesign*, pp.133–138.

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*.

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 Conf. (ASP-DAC)*, pp. 709-712.

Deniziak, S., Górski, A., 2008. Hardware/Software Co-Synthesis of Distributed Embedded Systems Using Genetic programming. In *Proceedings of the 8th International Conf. 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 Conf. on Pervasive and Embedded Computing and Communication Systems, 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 Conf. 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 Conf. on Pervasive and Embedded Computing and Communication Systems*, Madrid, Spain.

Koza, J., R., Bennett III, F., H., Lohn, j., Dunlap, F., Keane, M., A., Andre, D., 1997. Automated synthesis of computational circuits using genetic programming. In Proceedings of the IEEE Conf. on Evolutionary Computation. IEEE.

Deniziak, S., 2004. Cost-efficient synthesis of multiprocessor heterogeneous systems. In Control and Cybernetics, Vol.33, No. 2, pp.341-355.