Adaptive Iterative Improvement GP-based Methodology for HW/SW Co-synthesis of 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, Genetic Algorithm, Adaptive Systems.

Abstract: The paper presents a novel adaptive genetic programming based iterative improvement algorithm for hardware/software co-synthesis of distributed embedded systems. The algorithm builds solutions by starting from suboptimal architecture (the fastest) and using system-building options improves the system's quality. Most known genetic programming algorithms for co-synthesis of embedded systems are built choosing fixed probability. In our approach we decided to change the probability during the work of the program.

1 INTRODUCTION

Embedded systems (Jiang, Eles and Peng, 2012) are hardware systems able to execute some tasks by embedded software. Distributed embedded systems (Garcia, Botella, Ayuso, Prieto, and Tirado, 2013, Konar, Bhattacharyya, Sharma, Sharma, Pradhan, 2017) are built using two kinds of resources: processing elements (PEs) and communication links (CLs). PEs are responsible for execution of the tasks. Communication links provide the communication between PEs. There are two possible groups of PEs: programmable processors (PPs) and hardware cores (HCs). PPs are universal resources able to execute more than one task. HCs are dedicated resources that can execute only one task. In some works (Grzesiak-Kopeć, Oramus and Ogorzałek, 2015) it is proposed to implement PEs in 3D layout.

According to De Micheli and Gupta (De Micheli and Gupta 1997) design of embedded systems can be divided into three phases: modelling, implementation and validation.

Unexpected tasks can appear when the system has been already designed and produced and no changes in its architecture are possible. Thus another part of embedded systems design process could be assignment of unexpected tasks (Górski and Ogorzałek, 2016).

One of very important problems in embedded system design process is the problem of co-synthesis. Co-synthesis (Densmore, Sangiovanni-Vincentelli and Passerone, 2006, Jozwiak, Nedjah and Figueroa, 2010) is a process of generation of an architecture of embedded system based on the specification provided. The process consists of: allocation of resources – the choice of type and number of resources, assignment – the choice of PE for predicted tasks, task scheduling – needed if one resource has to execute more than one task.

Existing co-synthesis methodologies can be divided into two groups: constructive and iterative improvement. Constructive algorithms (Dave, Lakshminarayana, and Jha, 1997) make decisions for every task separately. Therefore they can stop in local minima when searching optimal parameters. Iterative improvement methodologies (Yen and Wolf, 1995) start from suboptimal architecture and by local changes (like allocation or reallocation of resources, moving task from one resource to another, etc.) try to improve system quality. However obtained results are still suboptimal.

In genetic algorithms (Dick and Jha, 1998, Guo, Li, Zou and Yhuang, 2007) solutions are mostly represented by a string or an array. Next, using operators such as mutation, crossover and reproduction, new solutions are created.

Very good results were obtained using genetic programming (Deniziak and Górski 2008, Górski and Ogorzałek 2014b). In such methodologies (Koza, 2010) trees which represent design decisions are evolved.

Adaptive solutions (Górski and Ogorzałek 2014a) are able to adapt to the environment by making changes during their execution.

In the present paper we propose a novel adaptive iterative improvement methodology based on genetic programming. Unlike other genetic programming based adaptive methodologies the embryo, in every individual, is a suboptimal solution – the fastest architecture. Therefore any other node includes function which allows modification of the architecture or task assignment.

2 PRELIMINARIES

The behaviour of distributed embedded system is described by a task graph G = {T, E}. It is an acyclic directed graph which shows the concurrency and other dependencies of the tasks. The nodes of the graph represent the tasks (T). The edges describe amount of data $d_{i,j}$ that need to be transferred between two connected tasks. Transmission time ($t_{i,j}$) is depended on CLs' bandwidth (b). The time can be calculated using the following formula:

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

Figure 1 presents an example of task graph.

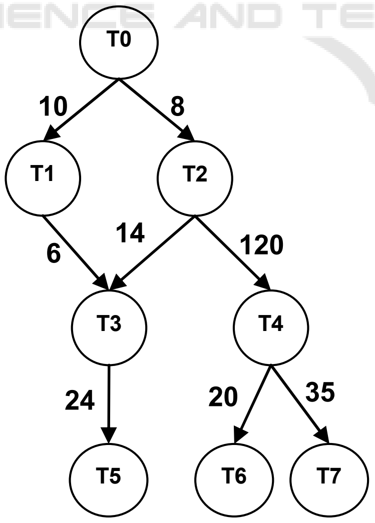

Figure 1: Example of task graph.

The graph consists of eight nodes: T0, T1, T2, T3, T4, T5, T6 and T7. Tasks T1 and T2 can start their execution after T0 is executed. T3 cannot start its execution before T1 or T2 is finished. T4 can be executed only after T2, T5 after T3, T6 and T7 after T4. Tasks T1 and T2, T3 and T4, T5, T6 and T7 are parallel. Table 1 includes example of resource database for the system described by the graph of figure 1. The target system can be built of four kinds of PEs: two kinds of PP (PP1 and PP2) and two kinds of HCs (HC1 and HC2). Cost (C) of used PP must be added to overall cost of the system only ones. Cost of HCs is added to the cost of tasks' execution. Communication can be provided by two kinds of CLs (CL1 and CL2). CLs also have cost (c) of connection to each PE.

Table 1: Example of resource database.

Task	PP1 C=100		PP2 C=250		HC1		HC2	
	t	c	t	c	t	c	t	c
T0	20	6	30	5	10	200	5	360
T1	40	8	35	10	12	150	8	190
T2	23	3	20	4	5	120	1	210
T3	14	10	18	9	2	300	5	220
T4	68	20	80	15	20	150	10	200
T5	34	5	30	7	9	110	8	130
T6	29	9	32	8	4	200	5	180
T7	36	10	28	15	6	190	8	160
CL1, b=5	c=5		c=6		c=25			
CL2, b=10	c=6		c=7		c=30			

The final system specified by the graph of figure 1 executes n processes consists of, m programmable processors and p communication links (CLs). The resources are chosen from the database given in table 1. Overall cost (C_o) of the system is described by the formula below:

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

The system cannot exceed the time constrains. The co-synthesis algorithm needs to find an architecture with the lowest C_o value.

3 THE METHODOLOGY

At the beginning of the algorithm initial population must be created. The population consists of genotypes. Every genotype is a tree. The tree includes system-building options in its nodes. Number of individuals in the population is described by the following formula:

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

where: n – number of tasks in task graph, e – number of possible types of PEs, α – parameter given by the designer.

The first node in every genotype is called an embryo. The embryo is chosen as the fastest possible architecture. In such an architecture every task is executed by a different resource. The rest of the genotype is created randomly. Therefore every genotype can be a different tree containing different number of nodes. Number of nodes in the tree is not greater than number of tasks in the task graph. Table 2 presents options for system construction. Each option has probability to be chosen.

Table 2: Options for building system.

Step	Option
PE	a. The cheapest implementation of the most expensive task
	b. The fastest implementation of the slowest task
	c. min (t*s) for most expensive task
	d. The last task from critical path move on the same processor as task's predecessor
	e. The first task from the most expensive path move on the cheapest implementation
	f. The slowest task from critical path move on the fastest implementation
CL	a. The fastest CL
	b. The cheapest CL
	c. the same as in the previous node
Task scheduling	list scheduling

The options are executed according to the level in the tree. Such options can allow more than one modification of the assignment of any task in investigating genotype. Nodes at the same level are executed from left to right. After generating each population solutions are rank by cost. Next the algorithm counts the percentage/frequency of appearance of each system-building option in the first half of the population. The result is the new value of probability for the options.

The first half of new population is generated the same way as initial population using options included in table 2 but with modified probability.

Second half of the next population is created using genetic operators: reproduction, crossover and mutation. Number of individuals obtained by every genetic operator is presented below.

- Φ = β*Π/2 – number of individuals obtained by reproduction;
- Ψ = γ*Π/2 – number of individuals obtained by crossover;
- Ω = δ*Π/2 – number of individuals obtained by mutation;
- β + γ + δ = 1 – the condition responsible for fixed number of individuals in every population.

Parameters β, γ, δ control the evolution process. Their values are given by the designer.

Mutation randomly chooses Ω genotypes and replaced option from randomly selected node by another option from table 2 but with probability actual for current population.

Reproduction selects Φ individuals and copies them to a new population. Each solution has a probability to be selected during reproduction. The probability depends on position r of the solution in a rank list:

$$P = \frac{\Pi - r}{\Pi} \tag{4}$$

Crossover generates Ψ solutions. The algorithm randomly selects two individuals and randomly set one different crossing point for each solution. Next created subtrees are substituted between the genotypes.

The stop condition is stimulated by parameter ε. If the algorithm does not find better solution in ε next steps, it will be stopped. Parameter ε is set by the designer.

4 THE EXAMPLE

Figure 2 presents an example of genotype for the task graph from figure 1. The example is consisted of five nodes. The nodes contain system-building options from table 2.

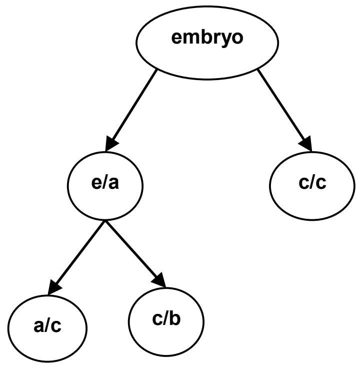

Figure 2: Example of genotype.

In accordance with developmental genetic programming rules the first node in the genotype is an embryo. The embryo is the fastest implementation of all the tasks. For the transmission it was used CL2 which has the highest bandwidth value. The cost of a system is 2020, the time of execution of all tasks is 38,3. Next the second node is executed. Therefore task T0 is moved to PP1. The transmission for T0 is also provided by CL2. The third node moves T3 to PP1. The fourth node assigns T2 to PP1. The last node moves T6 to PP1. The system is contained of one PP (PP1) which executes four tasks, four HCs which execute four tasks and one CL (CL2). The final cost of the system is 963. The time of execution of all the tasks is 93,8.

5 CONCLUSIONS

In this paper a new adaptive genetic programming approach to HW/SW co-synthesis was presented. The approach builds genotypes by starting from suboptimal solution and improves the system quality by local changes. The methodology is able to adapt to the environment. It is achieved by modifying the probability of selecting each system-building options during the work of the algorithm. The main advantage of presented methodology, in comparison with constructive algorithm, is reduced complexity. Therefore the time of calculation can be much less.

In the future we plan to examine another chromosomes and genetic operators.

REFERENCES

Jiang, K., Eles, P., Peng, Z., 2012. Co-design techniques for distributed real-time embedded systems with communication security constrains. *Design Automation and Test in Europe (DATE 2012)*.

Grzesiak-Kopeć, K., Oramus, P., Ogorzałek, M.J., 2015. Using shape grammars and extremal optimization in 3D IC layout design. Microelectronic Engineering, Vol. 148, pp. 80-84, 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.

Garcia, C., Botella, G., Ayuso, F., Prieto, M., Tirado, F., 2013. Multi-GPU based on multicriteria optimization for motion estimation system EURASIP Journal on Advances in Signal Processing, Vol. 23, Springer-Verlag.

Konar, D., Bhattacharyya, S., Sharma, K., Sharma, S., Pradhan, S. R., 2017. An improved Hybrid Quantum-Inspired Genetic Algorithm (HQIGA) for scheduling of real-time task in multiprocessor system. Applied Soft Computing, Vol. 53, pp. 296-307, Elsevier.

Densmore., D., Sangiovanni-Vincentelli, A., Passerone, R. (2006). A platform-based taxonomy for ESL design. IEEE Design & Test of Computers Vol. 23, No 5, pp. 359-374.

Jozwiak L., Nedjah N., Figueroa, M., 2010. Modern development methods and tools for embedded reconfigurable systems – a survey. *Integration, VLSI Journal*, pp.1-33.

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)*.

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

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*.

Guo R., Li, B., Zou, Y., Yhuang, Z., 2007. Hybrid quantum probabilistic coding genetic algorithm for large scale hardware-software co-synthesis of embedded systems. In *Proc. of the IEEE Congres on Evolutionary Computation*, pp. 3454-3458.

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.

John R. Koza. 2010. Human-competitive results produced by genetic programming. In *Genetic programming and evolvable machines, vol. 11, issue 3-4*. Springer-Verlag.