Control and Cybernetics

vol. **33** (2004) No. 2

Cost-efficient synthesis of multiprocessor heterogeneous systems

by

Stanisław Deniziak

Department of Computer Engineering,
Cracow University of Technology
Warszawska 24, 31-155 Cracow, Poland
e-mail: pedenizi@cyf-kr.edu.pl

Abstract: In this paper an algorithm for co-synthesis of distributed embedded systems is presented. The algorithm is based on iterative improvement heuristics, taking into consideration sophisticated modifications and possibilities of further improvements. Starting from the solution with the highest performance, architecture of the system is modified until it achieves the lowest cost. It has been observed that the algorithm presented has the capacity of getting out of the local minima. Experimental results showed high efficiency of the algorithm. Almost all results obtained with the help of the algorithm were significantly better than the results obtained with the help of Yen-Wolf algorithm presented in the literature.

Keywords: HW/SW co-synthesis, distributed systems, SOC.

1. Introduction

In recent years Hardware/Software (HW/SW) co-synthesis has become an almost standard procedure for designing various types of embedded systems. The problem of HW/SW co-synthesis, taking into consideration the cost and performance objectives is illustrated in Fig. 1. The area S comprises all possible solutions. Maximal cost (line Cx) and minimal performance (line Cy) constraints reduce the search space to the area S'. Solutions nearest to the point C are the best, as far as both factors (cost and performance) are considered. Point $B'(A')$ is the co-synthesis goal when only system cost (performance) is minimised (maximised).

Distributed embedded systems are usually specified in terms of communicating tasks. HW/SW co-synthesis (Gupta and De Micheli, 1993) is the process of partitioning system specification into hardware and software processing elements connected by busses. The goal of co-synthesis is to find the best target architecture satisfying given constraints e.g. maximal cost or minimal speed. For many practical embedded systems, multiprocessor heterogeneous architectures are the most efficient ones. Today, it is possible to implement such a system on one chip (System On a Chip - SOC). Design reuse is widely used to reduce time to market for SOCs. The number of available hardware and software reusable IP (Intellectual Property) modules increases significantly every year. The IP-based design becomes the dominating technique for SOCs, and should be taken into account in the hardware/software co-synthesis methods, too.

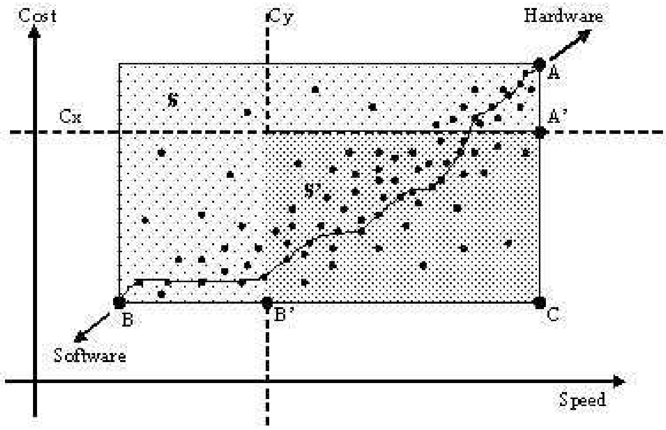

Figure 1. Search space in co-synthesis

The synthesis of multiprocessor heterogeneous systems consists of the following tasks:

- allocation; determines the quality and quantity of resources (processing elements - PEs and communication links - CLs), to be used,
- assignment; determines tasks to be executed on each PE, and a CL for each transmission,
- task scheduling; determines time of execution for each task and each transmission.

Allocation, assignment and scheduling are each NP-complete and so co-synthesis is computationally a very hard problem.

In this paper an algorithm for co-synthesis of distributed embedded systems is presented. In this algorithm all of the co-synthesis tasks are executed simultaneously. Dependencies between allocation, assignment and scheduling are taken into account. The algorithm is based on iterative improvement heuristics, taking into consideration sophisticated modifications and possibilities of further improvements. Starting from the solution with the highest performance, architecture of the system is iteratively modified until it achieves the lowest cost. It has been observed that the algorithm presented has the capacity of getting out of the local minima as far as the system cost is concerned.

The paper is organized as follows. Next section reviews related previous work. In Section 3 basic concepts and definitions are presented. The co-synthesis algorithm is presented in Section 4. In Section 5 experimental results are given. Section 6 presents the conclusions.

2. Previous work

Related work often considers one-CPU-one-ASIC (Application Specific Integrated Circuit) target architectures (Gupta and De Micheli, 1993; Henkel and Ernst,1997; Kalavade and Lee, 1995). In such approach co-synthesis is formulated as a hardware/software partitioning. However, most real-life embedded systems are distributed and heterogeneous i.e. composed of multiple general-purpose processors, microcontrollers, digital signal processors, protocol controllers, etc. Therefore, practical co-synthesis method can not be limited to the mono-processor based systems.

Due to the complexity of co-synthesis, the algorithms giving best solutions (e.g. mixed integer linear programming, Prakash and Parker, 1992, or exhaustive exploration, D'Ambrosio and Hu, 1994) are limited to small systems, only. Other approaches are based on constructive or iterative refinement heuristics. Some probabilistic optimisation methods e.g. simulating annealing (Eles, Peng, Kuchcinski, and Doboli, 1997) or genetic algorithms (Dick and Jha, 1997) have been applied to the co-synthesis problem, as well.

Constructive algorithms (Dave, Lakshminarayana and Jha, 1997; Bianco, Auguin and Pegatoquet, 1998; Dave and Jha, 1998) build a system allocating incrementally new components. Since such approach is capable of inspecting only local effects of changes, different performance estimation methods were used to predict the global impact of these changes. The methods, usually based on the best- and worst-case analysis, prefer PEs with the highest speed or with the lowest cost and disregard the remaining PEs. Although constructive algorithms are fast and are capable of producing high quality results (Dave, Lakshminarayana and Jha, 1997), they are prone to becoming trapped in local minima.

Iterative improvement algorithms (Yen and Wolf, 1995A,B; Hou and Wolf, 1996) start with a sub-optimal solution and try to improve the system quality by making local changes to the system. Existing iterative algorithms also tend to be trapped in local minima. The main reason is that iterative improvement methods consider only local changes driven by immediate gain. In sensitivity-driven co-synthesis algorithm (Yen and Wolf, 1995A) the movements of one process from one PE to another PE are only considered. Allocation of a more expensive PE that will reduce total system cost due to accommodation of more tasks is not possible. In such cases the algorithm will be trapped in local minima.

Probabilistic optimisation algorithms are capable of escaping from local minima. However, performance of these methods strongly depends on selected parameter values. For example, in the MOGAC genetic algorithm (Dick and Jha, 1997) each task graph has a different random seed for which the algorithm finds the best solution most rapidly. On the other hand, the hardware/software partitioning algorithms based on simulated annealing turned out to be less efficient than the iterative improvement algorithms like tabu search (Eles, Peng, Kuchcinski and Doboli, 1997).

Recently, most of research has addressed specific problems of co-synthesis, like multi-mode embedded systems (Oh and Ha, 2002), energy optimisation and utilisation of dynamic voltage scalable processors (Schmitz, Al-Hashimi and Eles, 2002), partitioning and scheduling of hierarchical specification models (Chatha and Vemuri, 2001; Haubelt, Teich, Richter and Ernst, 2002) or conditional task graphs (Eles, Kuchcinski, Peng, Doboli and Pop, 1998; Xie and Wolf, 2001), and co-synthesis for system-on-a-chip architectures (Dick and Jha, 1999). Finding an efficient co-synthesis algorithm for distributed embedded systems is still an open problem. First, most of existing approaches are not suitable for large systems because of time requirements. Second, quality of results obtained using different methods indicates that there is a lot of work to do in order to improve the efficiency of co-synthesis algorithms.

3. Basic concepts and definitions

A task graph $G = (V, E)$ will be used as an abstract model of system specification. The task graph is a directed acyclic graph. Each node v_i corresponds to one task and each edge e_{ij} is associated with communication between tasks corresponding to nodes v_i and v_j. Weights d_{ij} associated with edges describe the amount of data (in bytes) that must be transmitted between the two connected tasks. An example of a task graph is presented on Fig. 2.

Two types of processing elements (PE) are considered: universal programmable processors (PPs) and dedicated hardware cores (HCs). A PP executes all the assigned tasks sequentially. Each HC executes exactly one task. Hardware units which can execute more than one task are defined as PP (not HC). In this way hardware sharing is possible in the presented algorithm. With each PE_i the following parameters are associated:

- cost of given tasks $C_i\,(v_j)$,
- time of execution of given tasks $T_i\,(v_j)$.

Values of $C_i\,(v_j)$ and $T_i(v_j)$ are known for IP modules. For other tasks they can be computed using performance and hardware effort estimation methods (Yen and Wolf, 1998; Henkel and Ernst, 1998).

With each PE a resource type (RT) is associated. PEs with the same RT may be located in the same integrated circuit (IC). With each IC_i the following parameters are associated:

- unit cost CU_i,

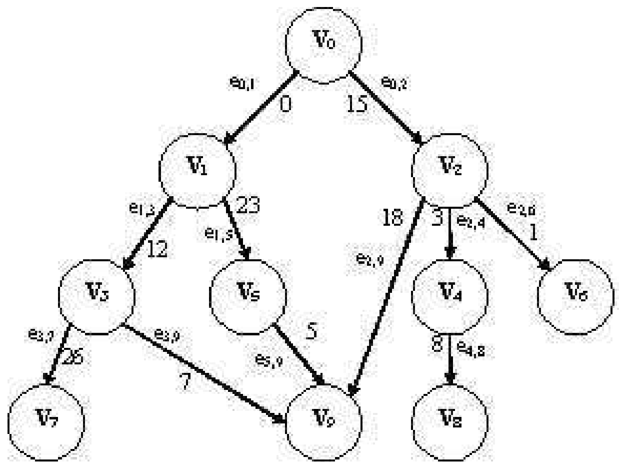

Figure 2. An example of a task graph

- maximal cost CM_i, which defines maximal cost of all tasks mapped to IC_i.

CU_i is independent of the number of tasks allocated to *PE*s located on the IC_i (e.g. it is a cost of *PP*s or a cost of *PP* cores). Maximal cost defines the maximal size of the IC_i.

Communication between processing elements is established using communication links (*CLs*). Sharing of communications links is allowed. Communication links are treated similarly as *PP*. During synthesis link allocation and scheduling of transmissions are performed. Each type of communication link CL_i has the following parameters:

- cost of the link CC_i for each available *PE* type
- bandwidth b_i (Bytes/s).

The time $T_k\,(v_i, v_j)$ required for data transfers between tasks v_i and v_j using communication link CL_k is evaluated using the following rule:

$$T_k(v_i, v_j) = \begin{cases} \left\lceil \dfrac{d_{ij}}{b_k} \right\rceil & - \text{ if tasks are assigned to different } PEs, \\ 0 & - \text{ otherwise.} \end{cases}$$

It is assumed that transmissions do not interfere with computations. Such model of communication is most commonly used, and may be implemented using dual-port buffers between PEs and buses or with communication using shared memory.

Assuming that a cost is defined by the total ASIC area, the total cost of a system may be specified using the following equation:

$$C = \sum_{i=1}^{r} \left(CU_i + \sum_{M(i)} C_i \left(v_{M(i)}\right)\right) + \sum_{i=1}^{c} CC_i \tag{1}$$

where r is the number of IC_s, $M(i)$ is the list of tasks mapped to PEs located on IC_i, and c is the number of communications links.

Illustrative values of resource parameters for task graph from Fig. 2 are presented in Tables 1 and 2. It is assumed that technology library contains 4 types of resources (2 programmable processors and 2 ASIC technologies) and 2 types of communication links (*CL2* is not available for PEs of type RT_l).

Table 1. Resource parameters

PE	$PP_1(RT_1)$ CU_1 =100 CM_1 =30		$PP_2(RT_2)$ CU_2 =200 CM_2 =50		$HC_j(RT_3)$ CU_3 =500 CM_3 =500		$HC_j(RT_4)$ CU_4 =300 CM_1 =100	
v_i	$T_1(v_1)$	$C_1(v_1)$	$T_2(v_1)$	$C_2(v_1)$	$T_3(v_1)$	$C_3(v_1)$	$T_4(v_1)$	$C_4(v_1)$
v_0	30	3	10	2	**3**	**50**	4	10
v_1	50	5	20	4	6	80	**5**	**20**
v_2	20	3	10	3	**3**	**60**	5	20
v_3	10	3	8	1	**1**	**20**	2	5
v_4	30	3	15	2	**4**	**70**	10	30
v_5	50	5	30	3	5	80	**5**	**15**
v_6	40	3	15	2	**10**	**70**	12	15
v_7	30	3	15	2	**5**	**50**	8	18
v_8	20	3	5	1	**2**	**30**	4	10
v_9	10	3	5	1	**3**	**40**	4	12

Table 2. Communication link parameters

CL_j	**Cost**				b_i
	RT_1	RT_2	RT_3	RT_4	
CL1	2	0	10	0	8
CL2	-	0	15	8	16

4. Co-synthesis algorithm

The goal of co-synthesis is to find the cheapest system architecture satisfying given time constraints. The algorithm is based on iterative improvements of sub-optimal solutions. It starts with an initial solution, at each step some changes to the actual solution are considered and then the solution giving the best gain is selected. The main components of the algorithm are:

- the initial solution,
- the metric of the gain,
- system refinement methods

The above components were defined in such a way that the algorithm is capable of escaping from local minima.

4.1. Initial solution

The fastest architecture of the system is always selected as an initial solution. In this solution, the PE with fastest execution time is allocated to each task. If any time constraint is not satisfied then the algorithm stops (there is no solution), otherwise the algorithm continues with refinements reducing the cost of the system. For the example from Fig. 2 the initial solution consists of 10 PEs (8 * RT_3 and 2 * RT_4). The cost is 5025 and execution time equals 16.

4.2. Gain

The value of gain defines the quality of an improvement. Since the goal of refinement is to reduce cost of the system, so this cost should be the main factor influencing the gain. However, greedy algorithms, taking into consideration only cost, are quickly trapped into local minima. Hence, usually more sophisticated gain metrics are used. For example, in Yen and Wolf (1995A) the cost of least utilized PEs is increased in order to force the idle PE elimination. In constructive algorithms more sophisticated metrics are used, too (Bianco, Auguin and Pegatoquet, 1998).

The main idea of the algorithm presented here is to define the gain in such a way that it accounts for the global impact of the considered improvement. Usually, execution time is longer for PEs with lower cost, and moving a task to a less expensive PE may decrease system performance. Obviously, not all tasks have the same influence on system performance. For the example from Fig. 2 the longer the execution time of task v_0 the longer the execution times for all paths in the graph, and finally for the whole system. In the same example the execution time of task v_8 influences only the path containing tasks v_0, v_2 and v_4. From this we may deduce that moving task v_8 to a slower PE has less impact on the possibilities of refinements in the next steps of the algorithm than the same change for task v_0.

In the approach presented the possibilities of modifying system architecture in the subsequent steps of the algorithm are defined using the following parameter:

$$\Omega = \sum_{i=1}^{n} (L_i - S_i)$$

where:

S_i- is the earliest time to start the execution of the i-th task,

L_i- is the latest time to start the execution of the i-th task, ensuring satisfaction of all time constraints.

S_i and L_i are evaluated using ASAP (As Soon As Possible) and ALAP (As Late As Possible) algorithms for the current architecture. If for any of the tasks we have $L_i < S_i$ then the current solution does not satisfy time requirements. This condition is verified for each solution. Bigger $L_i - S_i$ usually means more possibilities of allocating the i-th task. During system refinement task assignments and scheduling are changed, and so L_i and S_i should be computed after each step. For example, assume that we want to find the best architecture executing the graph from Fig. 2 in time $T_{max} = 50$. Then values of parameters L_i and S_i for the initial solution are presented in Table 3.

Table 3. Parameters L_i and S_i

	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9
S_i	0	3	3	8	6	8	6	9	10	13
L_i	34	37	37	44	44	42	40	45	48	47

The global impact of any modification is defined as the increase of Ω caused by the modification:

$$\Delta\Omega = \Omega_{new} - \Omega_{old}$$

Finally, the gain ΔE taking into consideration cost reduction and the global impact of the system refinement is defined as follows:

$$\Delta E = \begin{cases} \dfrac{-\Delta K_s}{-\Delta\Omega}, & \text{for} \quad \Delta\Omega < 0 \\ -\Delta K_s, & for \quad \Delta\Omega = 0 \\ -\Delta K_s \cdot \Delta\Omega & \text{for} \quad \Delta\Omega > 0 \end{cases}$$

where ΔK_s denotes the cost increase.

Gain is defined only for modifications decreasing the cost of a system (otherwise the modification is not taken into consideration). From the three cases of the above formula the first one ($\Delta\Omega < 0$) corresponds to modifications that could decrease the system cost as well as the system performance. Thus, the architecture with the best cost to performance ratio is selected. In the second case ($\Delta\Omega = 0$) modification might not change the performance, so the system with the lowest cost is selected. The last case ($\Delta\Omega > 0$) corresponds to modifications that could increase system performance.

4.3. System refinement methods

In each step of an iterative improvement algorithm different modifications of the current solution are considered and the modification giving the best gain is selected. However, since the number of possible changes in the system is very large then only few of these changes should be taken into consideration. Otherwise, the algorithm would be not suitable for large systems due to time requirements. For this reason the existing approaches apply only simple modifications like moving one task to another PE, removing or allocating one PE etc. (Yen and Wolf, 1995A). But such local changes have no possibility of getting the algorithms out of the local minima during cost vs. speed optimisation.

In the algorithm presented more complex modifications are considered. The main goal of such approach is to increase the possibility of getting out of local minima. The following system changes are considered during refinement:

1. Allocation of one PE and assigning to it as many tasks as possible, so as to achieve the highest gain. After the allocation and assignment is terminated, all PEs which have no task allocated to them are removed. If the graph contains n tasks and the technology library contains r resource types then in the worst case there are $r \cdot n$ possible modifications.
2. Removing one PE, with all tasks, which were allocated to it, being moved to other PEs. All transfers are done according to the highest gain principle. In the worst case there are n^2 such modifications.

The same modifications are considered for communication links and transmissions. It is possible to perform both kinds of changes in the same step, in this way task transfer from some PEs to other ones can be done. Hence, in the worst case $r \cdot n^3$ system modifications are considered. In practice, this number is significantly lower because solutions with higher cost and solutions not satisfying time constraints are not considered.

It should be noticed that such complex system modifications allow for global changes of system architecture. Moreover, simple modifications are still possible. For example, allocation of one PE, assigning of one task to it and then removal of this PE and transfer of the task to some other PE, corresponds to task movement from one PE to another. Observations showed that such an approach has greater capacity of escaping from local minima than other algorithms based on iterative improvements.

4.4. Algorithm description

The scheme of the co-synthesis algorithm is the following:

Create an initial architecture A
Compute cost K_s*;*
repeat
    Gain $= 0$*;*
    for *each available resource type* RT_i ***do***
        $A' = A \cup PE(RT_i)$;
        repeat
            Find task v_k *giving highest* δE *after moving it to* $PE(RT_i)$*;*
            if $\delta E > 0$ ***then*** *Assign task* v_k *to* $PE(RT_i)$

until *there exists no task giving* $\delta E > 0$ *;*
$A' = A' - PE_s$ *with no task assigned;*
if $\Delta E > Gain$ ***then***
$Gain = \Delta E$; $A^{best} = A'$;
endif;
for *each processing element* $PE_j \in A'$ ***do***
$A'' = A' - PE_j$;
for *each task* $v_k \in PE_j$ ***do***
Find $PE_l \in A''$ *giving highest* δE *after moving task* v_k *to it;*
Assign task v_i *to* PE_l
endfor;
if $\Delta E > Gain$ ***then***
$Gain = \Delta E$; $A^{best} = A''$;
endif;
endfor;
endfor;
if $Gain > 0$ ***then*** $A = A^{best}$;
until $Gain = 0$;

where δE means the gain achieved by moving one task to another PE (taking into consideration task costs, only), while ΔE means total gain (including all costs).

The outer ***for*** loop examines all possible allocations of a new PE. Task transfers to a new PE are performed in the inner ***repeat*** loop. After allocating a new PE, a possibility of removing one PE is examined in the second ***for*** loop. If removing one PE increases gain, then such modification is accepted. In each step the modification giving the best gain is accepted and becomes the current solution for the next step of the algorithm. Only solutions with positive gain and satisfying all time constraints are considered. This reduces the search space and assures that the algorithm is convergent.

When a task is moved to PP, task scheduling should be performed. In such case all possible schedules of a new task are examined (the schedule of the previously assigned tasks is not changed) and the schedule giving the best gain is selected. Because scheduling has no influence on cost, then the best gain means lower global impact ($\Delta\Omega$) and higher performance.

5. Experimental results

The results of co-synthesis of the task graph from Fig. 2 are presented in Fig. 3. These results have been obtained for time constraint $T_{max} = 50$, while communication times were neglected. The system consists of 2 PEs: one programmable processor and one dedicated hardware core. It should be noticed that very high utilisation of R_2 was obtained. The cost of the system equals 582, and it is the most efficient architecture for this system (assuming time requirement and task characteristics given in Table 1).

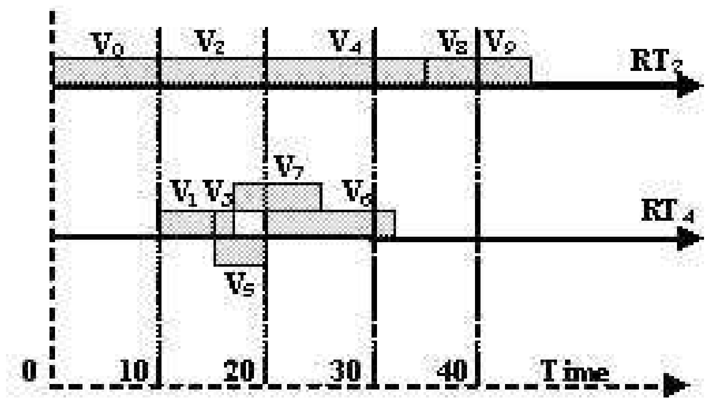

Figure 3. Results of synthesis (a Gantt chart) for the task graph from Fig. 2

To estimate the efficiency of the presented algorithm a modified version of the Yen-Wolf algorithm (Yen and Wolf, 1995a) was implemented. In this algorithm the cost function (1) was used, and then the algorithm was modified to minimise only system cost. For the graph from Fig. 2 the system consisting of 3 PEs was obtained with the Yen-Wolf method. The cost of such system is equal to 757 and the time of execution is 47. The results obtained for other examples are presented in Table 4.

Table 4. Experimental results for example task graphs

Graph	**N**	**Time min.**	**Tmax**	**Yen - Wolf**			**EWA**		
				Cost	**Time**	**CPU**	**Cost**	**Time**	**CPU**
P&P2	9	3	7	10	7	0.01	6	7	0.01
P&P2	9	3	15	8	15	0.01	5	4	0.01
Hou1&2	20	97	150	250	149	0.05	200	147	0.14
Hou3&4	20	82	150	360	150	0.05	250	142	0.14

Graph: name of the graph; N: number of tasks; Time min.: minimum time needed for executing all tasks; Tmax: time constraint; Yen-Wolf: results for the Yen-Wolf algorithm; EWA: results for the author's algorithm; Cost: cost of the obtained architecture; Time: execution time for the obtained architecture; CPU: time of algorithm execution.

Both considered algorithms were implemented in C and run on PC Celeron 1.8GHz. *P&P2* is the Prakash and Parker's task graph (Prakash and Parker, 1992). *Hou1&2* and *Hou3&4* are Hou's task graphs (Hou and Wolf, 1996). All graphs have zero communication delay, zero communication link cost and costs of all tasks are equal to 0 (the total cost of a system is the sum of unit costs of all PEs).

The efficiency of the presented algorithm was also estimated using ten randomly generated task graphs. The results are given in Table 5. Four PE types were available. The table compares results obtained using Yen-Wolf cost function ($C_i(v_j) = 0$ for all tasks) with the results obtained using the cost function presented in this work ($C_i(v_j)$ are randomly generated for each task). In both cases the efficiency of the presented algorithm is significantly higher than that of the Yen-Wolf method. Moreover, in the second case EWA obtains better results with a much shorter CPU time than the Yen-Wolf method.

Table 5. Experimental results for random task graphs

Graph	N	Time min.	Tmax	$C_i(v_j) = 0$						$C_i(v_j) \neq 0$					
				Yen-Wolf			EWA			Yen-Wolf			EWA		
				Cost	Time	CPU	Cost	Time	CPU	Cost	Time	CPU	Cost	Time	CPU
G1	10	183	200	2111	188	0.00	2111	188	0.01	3959	183	0.00	3959	183	0.00
G2	30	259	600	2264	589	0.11	1077	319	0.49	7490	598	0.25	5770	566	0.39
G3	50	248	1000	3451	1000	1.03	1034	422	2.58	11075	971	2.47	8027	988	2.33
G4	70	300	1400	3451	1400	4.08	1034	455	10.14	11963	1371	12.53	9310	1396	9.26
G5	90	437	1800	3451	1800	14.04	1077	555	30.03	14617	1784	40.14	10363	1781	28.23
G6	110	377	2200	3452	2199	29.34	1077	490	65.92	46357	2157	83.04	11791	2193	57.09
G7	130	349	2600	3910	2592	75.55	1034	562	125.71	11930	2506	270.86	10550	2599	144.51
G8	150	441	3000	3451	2996	130.98	1034	792	225.42	14003	2501	437.50	12337	2992	238.08
G9	170	410	3400	3654	3308	252.48	1077	564	382.88	58270	3398	593.52	12211	3399	468.58
G10	200	532	4000	3697	3903	412.85	1077	802	731.65	65638	3966	1205.16	13014	3971	916.73

6. Conclusions

In this work an algorithm for cost-efficient synthesis of distributed heterogeneous systems was presented. The algorithm optimises the cost of the target system taking into consideration time requirements. Experimental results showed high efficiency of the algorithm. Almost all results obtained with the help of the algorithm were significantly better than results obtained with the help of the Yen-Wolf algorithm. The proposed approach is especially suitable for the IP-based SOC designs, when cost is associated with each task. For such systems performance is also significantly better than of the Yen-Wolf method.

Future work will concentrate on expanding the system model so as to include conditional graphs and loops in it.

References

BIANCO, L., AUGUIN, M. and PEGATOQUET, A. (1998) A Path Analysis Based Partitioning for Time Constrained Embedded Systems. *Proceedings of the 6th International Workshop on Hardware/Software Codesign*. IEEE Computer Society Press, Los Alamitos, 85–89.

Chatha, K.S. and Vemuri, R. (2001) MAGELLAN: Multiway Hardware-Software Partitioning and Scheduling for Latency Minimization of Hierarchical Control-Dataflow Task Graphs. *Proceedings of the 9th International Workshop on Hardware/Software Codesign.* ACM Press, New York, 42–47.

D'Ambrosio, J. and Hu, X. (1994) Configuration-Level Hardware/Software Partitioning for Real-Time Systems. *Proceedings of the 3^{rd} International Workshop on Hardware/Software Codesign.* IEEE Computer Society Press, Los Alamitos, 34–41.

Dave, B.P., Lakshminarayana, G. and Jha, N.K. (1997) COSYN: Hardware-Software Co-Synthesis of Embedded Systems. *Proceedings of the 34^{th} Design Automation Conference.* ACM Press, New York, 703–708.

Dave, B.P. and Jha, N.K. (1998) CASPER: Concurrent Hardware-Software Co-Synthesis of Hard Real-Time Aperiodic and Periodic Specifications of Embedded Systems. *Proceedings of the Conference on Design Automation and Test in Europe.* IEEE Computer Society Press, Los Alamitos, 118–124.

Deniziak, S. and Sapiecha, K. (2001) Kosynteza rozproszonych systemów heterogenicznych. III Krajowa Konferencja: *Metody i systemy komputerowe w badaniach naukowych i projektowaniu inżynierskim.* AGH, Kraków, 437–442, in Polish.

Dick, R.P. and Jha, N.K. (1997) MOGAC: A multiobjective Genetic Algorithm for the Co-Synthesis of Hardware-Software Embedded Systems. *Proceedings of the International Conference on Computer Aided Design.* IEEE Computer Society Press, Los Alamitos, 522–529.

Dick, R.P. Jha, N.K. (1999) MOCSYN: Multiobjective Core-Based Single-Chip System Synthesis. *Proceedings of the Conference on Design Automation and Test in Europe.* IEEE Computer Society Press, Los Alamitos, 263–270.

Eles, P., Peng, Z., Kuchcinski, K. and Doboli, A. (1997) System Level Hardware/Software Partitioning Based on Simulated Annealing and Tabu Search. *Design Automation for Embedded Systems* **2** (1), 5–32.

Eles, P., Peng, Z., Kuchcinski, K., Doboli, A. and Pop, P. (1998) Scheduling of Conditional Process Graphs for the Synthesis of Embedded Systems. *Proceedings of the Conference on Design Automation and Test in Europe.* IEEE Computer Society Press, Los Alamitos, 132–138.

Gupta, R.J. and De Micheli, G. (1993) Hardware-Software Co-synthesis for Digital Systems. *IEEE Design & Test* **10** (3), 29–41.

Haubelt, C., Terich, J., Richter, K. and Ernst, R. (2002) System Design for Flexibility. *Proceedings of the Conference on Design Automation and Test in Europe.* IEEE Computer Society Press, Los Alamitos, 854–861.

Henkel, J. and Ernst, R. (1997) A Hardware/Software Partitioner using a dynamically determined Granularity. *Proceedings of the 34th Design Automation Conference*. ACM Press, New York, 691–696.

Henkel, J. and Ernst, R. (1998) High-Level Estimation Techniques for Usage in Hardware/Software Co-Design. *Proceedings of the Asia and South Pacific Automation Conference*. IEEE Computer Society Press, Los Alamitos, 353–360.

Hou, J. and Wolf, W. (1996) Process partitioning for distributed embedded systems. *Proceedings of the 4th International Workshop on Hardware/Software Codesign*. IEEE Computer Society Press, Los Alamitos, 70–76.

Kalavade, A. Lee, E.A. (1995) The Extended Partitioning Problem: Hardware/Software Mapping and Implementation-Bin Selection. *Proceedings of the 6th International Workshop on Rapid Systems Prototyping*, IEEE Computer Society Press, Los Alamitos, 12–18.

Lee, T.Y., Hsiung, P.A. and Chen, S.J. (2001) Hardware-Software Multi-Level Partitioning for Distributed Embedded Multiprocessor Systems. *IEICE Trans. Fundamentals* **E84-A** (2), 614–626.

Oh, H. and Ha, S. (1999) A Hardware-Software Cosynthesis Technique Based on Heterogeneous Multiprocessor Scheduling. *Proceedings of the 7th International Workshop on Hardware/Software Codesign*. ACM Press, New York, 183–187.

Oh, H. and Ha, S. (1999) Hardware-Software Cosynthesis of Multi-Mode Multi-Task Embedded Systems with Real-Time Constraints. *Proceedings of the 10th International Workshop on Hardware/Software Codesign*. ACM Press, New York, 133–138.

Prakash, S. and Parker, A. (1992) SOS: Synthesis of Aplication-Specific Heterogeneous Multiprocessor Systems. *Journal of Parallel and Distrib. Comp.* **16**, 338–351.

Saha, D., Mitra, R.S. and Basu, A. (1997) Hardware Software Partitioning using Genetic Algorithm. *Proceedings of the International Conference on VLSI Design*. IEEE Computer Society Press, Los Alamitos, 155–160.

Schmitz, M.T., Al-Hashimi, B.M. and Eles, P. (2002) Energy-Efficient Mapping and Scheduling for DVS Enabled Distributed Embedded Systems. *Proceedings of the Conference on Design Automation and Test in Europe*. IEEE Computer Society Press, Los Alamitos, 514–521.

Suzuki, K. and Sangiovanni-Vincentelli, A. (1996) Efficient Software Performance Estimation Methods for Hardware/Software Codesign. *Proceedings of the 33rd Design Automation Conference*. ACM Press, New York, 605–610.

Xie, Y. and Wolf, W. (2001) Allocation and scheduling of conditional task graph in harware/software co-synthesis. *Proceedings of the Conference on Design Automation and Test in Europe*. IEEE Computer Society Press, Los Alamitos, 620–625.

Yen, T.Y. and Wolf, W.H. (1995*A*) Sensitivity-Driven Co-Synthesis of Distributed Embedded Systems. *Proceedings of the International Symposium on System Synthesis.* ACM Press, New York, 4–9.

Yen, T.Y. and Wolf, W.H. (1995*A*) Communication synthesis for distributed embedded systems. *Proceedings of the International Conference on Computer Aided Design.* IEEE Computer Society Press, Los Alamitos, 288–294.

Yen, T.Y. and Wolf, W.H. (1998) Performance Estimation for Real-Time Distributed Embedded Systems. *IEEE Transactions on Parallel and Distributed Systems* **9** (11), 1125–1136.