Synthesis of Distributed Embedded Systems

Krzysztof Kuchcinski
Dept. of Computer and Information Science
Linköping University, Sweden
e-mail: kku@ida.liu.se

Abstract

This paper presents a new approach for modeling and synthesis of distributed embedded. These systems are represented as task graphs and modeled using finite domain constraints. The synthesis of such models into a distributed architecture consisting of microprocessors, ASICs and communication devices, is then defined as an optimization problem and it is solved using constraint solving techniques. The presented approach offers a powerful modeling technique of advanced features, such as conditional subtasks and functional pipelining. The experimental results prove the feasibility of this approach and runtimes which are of several orders of magnitude faster than the runtimes of respective MILP formulations.

1. Introduction

Current trends in embedded system design are toward distributed systems which are built of heterogeneous components, such as CPUs, DSPs, ASIPs, specialized hardware devices (ASICs) and memories. The integration of these components on a single chip leads to a system-on-chip (SoC) concept. The specification of SoCs, implementing usually complex functionality, is provided in a form of communicating tasks where each task is specified as a sequential program. During system design, decisions on system partitioning, allocation of basic components, assignment of tasks to allocated components, scheduling of tasks as well as interface synthesis has to be made. Different components with various performance and cost characteristics can be selected to implement the same program. The number of possible partitionings, component allocations and assignments as well as scheduling opportunities is also very large. Thus the design of embedded heterogeneous systems is a difficult design space exploration problem [8].

In this paper, we assume that a distributed embedded system is represented as a task data-flow graph which specifies data dependencies between subtasks. The DSP, image processing or multimedia applications, for example, are well suited for the approach based on the task data-flow graphs.The subtasks of the task data-flow graph are executed iteratively. When the last subtasks finish their execution the first subtasks are started again. Each subtask has specified execution times on selected system components.

The goal of the synthesis is to find an assignment of subtasks to processors and ASICs, and communication subtasks to communication devices (e.g., buses or links) as well as their related static schedule. The schedule, in this case, is given as an assignment of starting times to every subtask and communication activity. Different heuristic and Mixed Integer Linear Programing (MILP) formulations have been presented for this problem [7, 21, 4, 17, 18, 19]. Recently, Constraint Logic Programing (CLP) formulation has been suggested by the author [14].

This paper proposes to use finite domain constraints to model heterogeneous designs in one consistent and elegant formalism. The finite domain constraints are used to model both task data-flow graphs and the features of distributed heterogeneous architectures. An extension which allows to specify control dependencies between subtasks as well as functional pipelining is also introduced. Based on this model, a constraint solving technique and a number of optimization methods are proposed. Together they provide a solution to the synthesis problem of distributed embedded systems.

Constraint logic programming and finite domain constraints have already been used in hardware design automation area. In [1, 10] the problem of timing verification using constraint logic programing has been studied. In [5], ECLiPSe system has been used for a retargetable self-test program generation.

The rest of the paper is organized as follows. Section 2 introduces the system model based on the finite domain constraints. Modeling of advanced features, such as pipelining and conditional tasks execution is discussed in section 3. System synthesis methods, which include both optimal and heuristic methods, are presented in section 4. Finally,

Supported by the Wallenberg Foundation project “Information Technology for Autonomous Aircraft”

the experimental results are presented in section 5 and the conclusions in section 6.

2. Finite Domain Constraints System Model

Finite domain constraints are used in our approach to specify different properties and restrictions imposed on the specified system. We first briefly introduce the concept of finite domain constraints and then present our formulation of the system in terms of these constraints [16].

A Constraints Satisfaction Problem (CSP*)* is a 3-tuple $P=(I, D, C)$ where

$I = \{x_0, x_1, ..., x_{n-1}\}$ is a finite set of variables,
$D = \{D_0, D_1, ..., D_{n-1}\}$ is a finite set of domains, and
C is a set of constraints.

The variable x_i takes the values from the domain D_i.

A *constraint* $c(x_0, x_1, ..., x_{n-1}) \in C$ between variables of I is a subset of the Cartesian product $D_0 \times D_1 \times ... \times D_{n-1}$ which specifies which values of the variables are compatible with each other. In practice, equations, inequalities, global constraints or programs define the constraints. In our formulation, we will use this convenient method to define constraints.

A *solution s* to a CSP *P* is an assignment to all variables that satisfies all the constraints. In most design problems, we are interested to find an optimal solution, i.e., a solution that minimizes or maximizes a cost function. The *optimal solution* to a CSP *P* is a solution that minimizes or maximizes a value *a* assigned to a selected domain variable x_i.

We assume that a distributed embedded system is defined by a task data-flow graph (e.g., [4, 14, 18, 19]). The task data-flow graph is an acyclic graph with nodes denoting subtasks and arcs data precedence relations between them. During the synthesis of the task data-flow graph components are selected, task and communications are assigned to these components and the whole graph is scheduled. In particular, subtasks, represented as nodes in the task data-flow graph, are assigned to selected processors and ASICs. Communications, represented as arcs, can be assigned to interconnection components, such as buses and point-to-pint links. Both subtasks and communications are then scheduled on the assigned components. In a special case when two subtasks are assigned to the same resource the related communication between these subtasks is handled locally using a memory and neither the communication assignment nor its scheduling is required.

The subtask of a task data-flow graph is modeled as a 3-tuple of finite domain variables:

$$T=(\tau, \delta, \rho) \tag{1}$$

where τ denotes the start time of the subtask, δ its duration and ρ the resource number which is assigned for its execution. The data flow precedence relation, represented as an arc in the task data-flow graph, is expressed as an inequality relation. For example, if the subtask T_i precedes subtask T_j, the following inequality constraint is imposed:

$$\tau_i+\delta_i \leq \tau_j \tag{2}$$

This constraint is imposed on all pairs of subtasks which are in the precedence relation. The presented formulation assumes the communication between subtasks at the end of the execution of the subtask T_i and beginning of the execution of the subtask T_j. This very restrictive formulation can be made weaker if we allow starting another subtask at different phases of the execution of the subtask T_i. For example, an execution of a subtasks T_i can be followed by the execution of two subtasks T_j and T_k, where the subtask T_j can start after δ_{ij} execution time of the subtask T_i and the subtask T_k after δ_{ik} execution time of the subtask T_i. It is specified as the following conjunction of two inequalities:

$$\tau_i+\delta_{ij} \leq \tau_j,\ \tau_i+\delta_{ik} \leq \tau_k. \tag{3}$$

To synthesize the previously specified task data-flow graph, it is necessary to introduce additional constraints on resources sharing. These constraints forbid the simultaneous use of shared resources, such as processors and buses. They can be specified using disjunctive constraints defined: as follows

$$\tau_i+\delta_i \leq \tau_j \ \vee\ \tau_j+\delta_j \leq \tau_i \ \vee\ \rho_i \neq \rho_j \tag{4}$$

The constraint imposes a requirement on two subtasks T_i and T_j that they can not be assigned to the same resource if at least a part of their executions overlaps.

In the prototype system, defined in CHIP 5 constrained logic programming system, we make use of a global constraint `diffn/1` [6] to represent these constraints. The `diffn/1` makes use of a rectangle interpretation of a subtask. It takes as an argument a list of n-dimensional rectangles and assures that for each pair of *i*, *j* ($i \neq j$) of n-dimensional rectangles, there exist at least one dimension *k* where *i* is after *j* or *j* is after *i*. The n-dimensional rectangle is defined by a tuple $[O_1, ..., O_n, L_1, ..., L_n]$, where O_i and L_i are respectively called the origin and the length of the n-dimensional rectangle in *i-th* dimension.

The rectangle based resource constraint assures that the 2-dimensional rectangles $R_i=[\tau_i, \rho_i, \delta_i, 1]$ and $R_j=[\tau_j, \rho_j, \delta_j, 1]$ representing the subtasks T_i and T_j do not overlap which is defined using the following `diffn/1` constraint:

$$\texttt{diffn}([[\tau_i,\ \rho_i,\ \delta i,\ 1],\ [\tau_j,\ \rho_j,\ \delta_j,\ 1]]), \tag{5}$$

The graphical representation of this constraint is depicted in Figure 1.

Subtasks connected by an arc in a task data-flow graph communicate by sending messages. This communication uses a communication device and is modeled as another

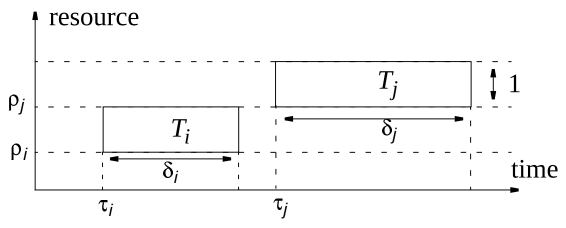

Figure 1. A graphical representation of the rectangle constraint.

communication subtask which is assigned to a communication resource. The related precedence relation (2) and the resource constraint (5) are extended accordingly. The extension of the resource constraint is only required for shared communication resources, such as buses.

The local communication between subtasks assigned to the same resource has to be handled carefully because `diffn/1` constraint does not allow rectangles with the length zero. This situation is modeled using three dimensional `diffn/1` and conditional constraints. The communication subtasks get the value zero in the third dimension when they use communication resources. Local communications get values greater then zero and therefore they do not interfere with the communications assigned to buses.

A subtask execution time depends on a selected resource, such as microprocessors or ASICs. The finite domain constraints make it possible to define this time as a domain variable capturing several possible execution times. The mapping which binds a given execution time to a given resource is defined by `element/3` constraint. This constraint enforces a finite relation between the first and the third variable. The finite relation is given by the vector of values passed as the second argument. For example, the subtask T_i will have an execution time, δ_i, either 3, 7 or 8 depending on the selected resource, ρ_i, taking values 1, 2 or 3. The constraint is defined as follows:

$$\texttt{element}(\rho_i,\ [3,\ 7,\ 8],\ \delta_i). \qquad (5)$$

It can be noted that the `element/3` constraint works in both directions and thus any constraints on the execution time will constraint its resources and constraints on resources will constraint its execution time. This feature is very useful in pruning the search space during synthesis.

The finite domain model of the task data-flow graph presented above fully describes the system and can be directly used for synthesis. However, redundant constraints can be used in the problem formulation to improve the constraint propagation. The most important redundant constraint used in this research is `cumulative/8`.

The `cumulative/8` constraint has been defined in CHIP and other constraint programming systems [6, 16] to specify requirements on the tasks which need to be scheduled on a limited number of resources. It expresses the fact that, at any time instant, the corresponding total of the resources for the subtasks does not exceed a given limit. The following four parameters are used: a list of the start times of tasks O_i, a list of durations D_i of tasks, a list of the amount of resources R_i required by the task and the upper limit of the amount of resources UL. All parameters can be either domain variables or integers. Formally, `cumulative/8` enforces the following constraint:

$$\forall i \in \left[\min_{1 \le j \le n}(O_j),\ \max_{1 \le j \le n}(O_j + D_j)\right] : \sum_{k : O_k \le i < O_k + D_k} R_k \le UL$$

where n is the number of tasks, while *min* and *max* are the minimum and maximum values in the domain of the variable respectively.

The cumulative constraint can be used to describe two types of constraints. In the first formulation O_i is replaced by τ_i, D_i by δ_i and finally R_i by 1. This models the task allocation and scheduling on the limited number of resources represented by UL. The second constraint represents the bin packing problem: O_i is replaced by ρ_i, D_i is always 1 and finally R_i is replaced by δ_i. The variable UL is constrained to the value lower or equal the execution time of the graph. The presented constraints are able to offer different types of propagation since the first formulation uses only τ_i and δ_i while the second one only ρ_i and δ_i.

3. Modeling of Advanced Features

A number of useful extensions to the previously defined model, such as pipelining and conditional task execution, can be defined. They will be discusses in this section. Other extensions can be found in [14, 15].

Pipelining a task data-flow graph is an efficient way of accelerating a design [2, 14]. It introduces, in fact, new constraints on location of rectangles. This is a method well known from computer architecture, where two dimensional reservation tables are used for pipeline analysis [13]. This approach is compatible with our methodology. Introducing an n stage pipeline of the initiation rate of k time units is equivalent to a placement of n copies of existing rectangles, starting at positions k, $2{\cdot}k$, $3{\cdot}k$, etc. This prevents to place subtasks in forbidden locations, which are to be used by subsequent pipeline computations. Since the subtask parameters are defined by domain variables, the copies of the current rectangles do not define final subtask positions but these positions will be adjusted during an assignment of values to domain variables.

The following constraint defines, for example, two stage pipeline for the two subtasks T_i and T_j, depicted in Figure 1, with initiation rate k:

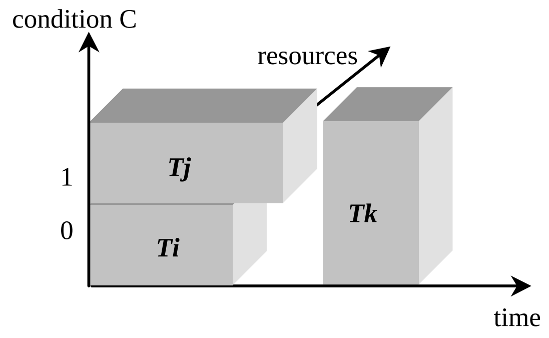

Figure 2. A graphical representation of the conditional subtasks.

$$\texttt{diffn}([[\tau_i, \rho_i, \delta i, 1], [\tau_j, \rho_j, \delta_j, 1], [\tau_i+k, \rho_i, \delta i, 1], [\tau_j+k, \rho_j, \delta_j, 1]]),$$

The rectangle based resource constraints can be easily extended to handle *conditional subtasks*. The formulation of the resource constraints, introduced in this paper, uses 2-dimensional rectangles in the time/resource space. Two subtasks can not use the same resource at the same time in this formulation. However, the conditional subtasks are never executed simultaneously since their execution is controlled by a value of an assigned condition. The main idea of representing conditional subtasks is to use n-dimensional rectangles instead of 2-dimensional. Additional dimensions are used to represent the values of conditions. For example, the Figure 2 represents three subtasks T_i, T_j and T_k. The subtasks T_i and T_j are conditional. They are executed depending on the value of the condition C (either 0 or 1) and can use the same resource at the same time. The subtask T_k can not be executed with these tasks and therefore has the height 2 in the dimension of the condition C.

4. System Synthesis

System synthesis is defined, in our approach, as a process of finding an assignment to all domain variables which satisfies all constraints and minimizes a given cost function defined as a domain variable. The cost function can be defined, for example, as a maximum value among all $\tau_i+\delta_i$. Minimizing this domain variable yields the fastest implementation satisfying all constraints.

The solution to the optimization problem defined above can be found using different methods. In this paper, we use methods which can guarantee optimal solutions as well as heuristic optimization methods.

4. 1. Optimal Solutions

The optimal solutions can be obtained using branch-and-bound algorithm (B&B). This algorithm searches for possible solutions by organizing the search space as a search tree. In every node of this tree a value is assigned to a domain variable and a decision whether the node will be extended or the search will be cut in this node is made. The search is cut if the assignment to the selected domain variable does not fulfill constraints or the estimated value of the cost function is worse than already achieved. Since assignment of a value to a domain variable triggers the constraint propagation and possible adjustment of the domain variable representing the cost function the decision can easily be made to continue or to cut the search at this node of the search tree.

CLP systems offer minimization and enumeration (labeling) procedures. In the CHIP system, there are `min_max/2` and `labeling/4` procedures. The first one offers B&B algorithm implementation while the second one defines basic procedure for assigning values to domain variables. In our case, several specialized labeling methods have been implemented and tested. The simplest one assigns, at each node of the search tree, new values to the pair of domain variables, (τ_i, ρ_i). Our criteria for selecting the pair of domain variables (τ_i, ρ_i) is based on the first variable τ_i. We usually select the most constrained variable to support better constraint propagation at the beginning of the search. In some cases, the input order of variables is used. The ordering forces the subtasks to be assigned before communications.

Other labeling methods are based on a strategy which first assigns an interval of possible values to a domain variable instead of a final value. In many cases, this assignment starts constraint propagation and makes it possible to select the most promising intervals for further examination. The *domain splitting* strategy [16], for example, repeatedly splits the domains of variables in half until they are singletons. More advanced domain splitting divides a domain of possible start times for each subtask into a number of *intervals* q_i. It is based on the formula q_i`*duration + rest` = τ_i, where `duration` is a constant (for example, the longest duration of the task i), `rest::0..duration-1`, q_i`::0..max` and τ_i is defined as before. The related labeling strategy assigns first values to the pairs (q_i, ρ_i) and than to variables τ_i.

4. 2. Heuristics Methods

The above described methods are supposed to examine the whole search space and find the optimal solution as well as prove, by examining the remaining assignments, that it is optimal. In practice, we need very often a good solution which can be found quickly. The big advantage of CLP is possibility to use heuristic search algorithms. In this paper, we have examined two meta-heuristics, *limited discrepancy search* (LDS) [12] and *credit search* [3], which can be used together with B&B algorithm.

LDS is based on a simple intuition that a first solution or

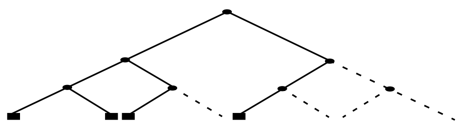

Figure 3. Limited discrepancy search example.

failure obtained with an assignment of values to all domain variables in a selected order can be improved if the assignment will be changed in a limited number of decision points, called discrepancies. For example, Figure 3 presents a simple binary search tree and visited nodes with LDS heuristic with one possible discrepancy. The edges in the tree represented by solid lines indicate visited parts of the search tree.

Credit search combines credit based exhaustive search at the beginning of the tree with the local search in the rest of the tree. The search is controlled by three parameters: number of credits, credit distribution and number of backtracks during local search. In the Figure 4 there is an example of the credit search tree [3]. The search has initially 8 credits and the distribution is specified by `part(1,2)` indicating that half of the credits are distributed to the selected choice. Number of possible backtracks is three. The first part of the search is based on the credits and makes it possible to investigate many possible assignments to domain variables while the other part is supposed to lead to a solution. Since we control the search it is possible to partially explore the whole tree and avoid situations when the search is stuck at one part of the tree which is a common problem of B&B algorithm when depth first search strategy is used.

5. Experimental Results

For experiments we have used random task graphs [9] and selected examples from [18, 19, 4]. The experiments have been carried out using a prototype implementation of the synthesis system implemented in CHIP 5, the constrained logic programming system [6], on the Pentium

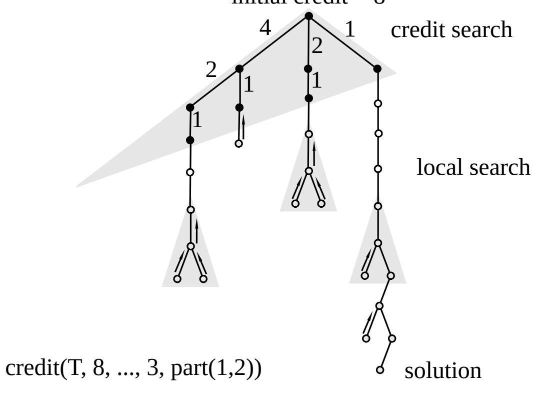

Figure 4. Credit search example.

200MHz computer. If not explicitly indicated, the runtimes presented in this paper are the total optimization execution times for finding a solution and proving that it is the optimal one. In practice, the solution is usually found much faster but the branch-and-bound algorithm needs to visit additional nodes of the search tree to prove that the better solution does not exist.

We have used 125 random task graphs divided into 5 groups of 25 task graphs. Each group had at least 20, 40, 75, 130 and 200 subtasks and communication subtasks (usually this number is ~10% higher since the graphs have been originally generated for fixed architectures and some communication subtasks were not counted when communicating subtasks have been assigned to the same processor). In each group there are 15 task graphs with uniform distribution and 10 with exponential distribution of the subtask execution time. The implementation architecture consists of a number of processors interconnected by buses. The actual number of processors and buses is presented in the tables. The assignment of subtasks to processors and communication subtasks to buses and their scheduling is done in a single optimization step. To evaluate the quality of the results we have also used a conservative lower bound (LB) calculation for the length of the schedule for a task graph given by the following formula:

$$LB = \left\lceil \frac{\sum_i \delta i}{NumberOfProcessors} \right\rceil$$

Table 1 presents results obtained with optimal methods with the execution time-out of 10 minutes. All three methods are able to produce good results but in same cases the search will not provide results for large graphs. The found solutions are either optimal or several percent worse than the lower bound. The method based on domain intervals is very robust and always generates solutions but it has problem to generate optimal solutions. A simple B&B algorithm with input order variable selection in one case can not generate a solution but it is able to compute optimal solutions in many

Table 1: Synthesis results for random task graphs using algorithms providing optimal solutions.

		tasks/processors/buses				
		20/3/1	40/4/2	75/4/2	130/6/3	200/6/3
Input order	optimal/solutions	25/25	25/22	25/12	24/8	25/11
	worse than LB	-	17.74%	2.80%	3.44%	2.80%
	Runtime (s)	8.91	21.14	15.11	17.25	37.78
Domain split	optimal/solutions	25/25	23/21	25/12	23/7	24/11
	worse than LB	-	4.38%	2.86%	3.54%	2.12%
	Runtime (s)	7.79	5.50	49.89	14.93	31.20
Domain intervals	optimal/solutions	25/25	25/19	25/6	25/4	25/4
	worse than LB	-	4.83%	4.03%	6.51%	2.79%
	Runtime (s)	0.57	17.43	1.78	26.95	9.13
Summary	optimal/solutions	25/25	25/24	25/15	25/9	25/13
	worse than LB	-	4.00%	2.96%	3.16%	1.84%
	Runtime (s)	0.38	11.23	39.78	11.19	23.4

Table 2: Synthesis results for random task graphs using heuristic optimization algorithms.

		tasks/processors/buses				
		20/3/1	40/4/2	75/4/2	130/6/3	200/6/3
LDS	δ from best	6.36%	3.09%	-0.60%	-1.33%	0.18%
	Runtime (s)	0.49	2.03	8.10	50.65	142.77
Credit (L/2)	Δ from best	6.49%	2.74%	-0.80%	-2.66%	1.23%
	Runtime (s)	0.63	1.29	6.92	37.23	204.62
Credit (sqrt(L))	Δ from best	6.59%	3.23%	1.27%	2.31%	1.94%
	Runtime (s)	0.57	1.13	3.64	12.62	43.06
Simple heuristic	Δ from best	11.18%	12.59%	6.70%	7.75%	4.65%
	Runtime (s)	0.05	0.13	0.42	1.31	2.90

cases.

In Table 2, we present results obtained with heuristic optimization algorithms (LDS, credit search and a simple heuristic) and compare them with the best obtained results presented in the summary of Table 1. The credit search heuristic has been tried with the credit which is equal the half of the number of all subtasks (L/2) and with the credit value equal to the square root of this number (sqrt(L)). All heuristics always produced solutions with a good quality which is, for larger graphs below 3%. In many cases the heuristics were able to improve obtained results. Even with a very simple greedy heuristic which assigns starting times and processors/buses for the subtasks one after the other we are able to get reasonably good results very quickly (less than 3 s).

The other part of experiments is based on the system synthesis examples presented in [18, 19]. They are based on the task data-flow graph containing nine subtasks which can be implemented on 3 different kinds of processors having different cost and execution characteristics [18]. The original results have been obtained using MILP formulation and solved using commercial tools. The results for the examples presented in [18] has also been presented in [21, 7]. These results are very similar to the original ones but they can not be proved to be optimal since the heuristic algorithms have been used. Our results are presented in the columns labeled CLP (Constrained Logic Programming) while the results reported in [18, 19] are presented in columns labeled MILP. The execution time is the total time for obtaining the results and proving that it is the optimal one.

Table 3 presents synthesis results for this specification using bus and point-to-point communication. The generated designs have the same cost and performance as those presented in [18] since the B&B algorithm assures that the optimal solution is generated.

Table 4 presents synthesis results of the same task dataflow graph but with the cost function extended with the local memory cost as defined in [19]. Our approach can handle different heterogeneous constraints, such as subtask execution times on different processors, communication times constraints and finally local memory cost, and still provide optimal solutions in a reasonable time.

Table 3: Synthesis results for the nine subtask system.

Design style	Cost	Performance (time units)	Performance optimization runtime			Cost optimization runtime	
			MILP (s)	CLP (s)	B&B Nodes	CLP (s)	B&B Nodes
bus	10	6	6438.00	0.43	84	0.55	92
	6	7	5371.80	0.53	114	0.68	144
	5	15	3691.20	0.43	68	0.70	103
point-to-point link	15	5	3732.00	0.43	20	1.67	125
	12	6	26710.20	1.42	98	2.18	169
	8	7	32320.20	1.00	58	2.59	198
	7	8	4510.80	1.64	75	2.02	119
	5	15	385012.20	1.50	32	1.48	77

The last example is the video coding algorithm H.261 derived from [4]. The task graph contains 12 subtasks and 14 interconnections between them. We have made three experiments. The first one is the non-pipeline implementation which is the same as presented in the original paper. The pipelined designs are different. The design which uses 3 stage pipeline and two buses has the stage latency 1154 and the total execution time of 3373 while the results reported in [4] obtained 1320 latency time and 3027 total execution time. The difference in the results comes from the additional constraint introduced in [4]. They do not allow to start a new computation on a given resource before all previous computations did not finish their executions. Our approach does not need this simplifying assumption and thus can produce better results. Finally, we have generated the pipelined designs with one and three buses instead of two. All pipeline designs improve the performance.

Table 4: Synthesis results for the nine subtask system with local memory.

Design style	Cost	Performance (time units)	Performance optimization runtime			Cost optimization runtime	
			MILP (s)	CLP (s)	B&B Nodes	CLP (s)	B&B Nodes
bus	28	6	6592.20	0.61	76	2.58	252
	23	7	5371.80	1.07	193	1.94	266
	22	8	123252.60	0.95	124	14.85	856
	21	10	316860.60	114.92	4 534	119.55	8 799
	18	11	236724.00	88.23	7 015	2.37	477
	17	12	138004.20	0.93	268	10.39	3 076
	14	15	3581.40	0.54	22	9.89	1 896
point-to-point link	38	5	—	0.56	24	2.08	107
	30	6	—	0.99	59	3.75	155
	25	7	—	1.60	79	5.58	314
	23	8	—	1.82	57	3.21	184
	22	10	—	4.50	84	59.25	855
	19	11	—	27.34	794	101.03	2 851
	18	12	—	97.72	2 686	8.66	1047
	14	15	—	1.18	14	4.95	328

Table 5: Synthesis results for H.261 example.

Design	Performance (time units)	Stage latency (time units)	Runtime (s)	B&B nodes
non-pipeline, 2 buses	2963	—	0.3	26
3 stage pipeline, 1 bus	3996	2351	19.34	27
3 stage pipeline, 2 buses	3373	1154	12.07	261
3 stage pipeline, 3 buses	3329	1110	5.91	261

6. Conclusions

In this paper, we have presented a new approach to modeling and synthesis of distributed heterogeneous embedded systems using finite domain constraints and constraints solving techniques. The synthesis has been defined as an optimization of a given cost function while fulfilling specified constraints. The presented methods are based on the B&B algorithm and provide ways to obtain optimal as well as sub-optimal solutions. An advantage of this approach is that for the same problem specification it is possible to use heuristic methods, such as used LDS or credit search.

Optimization using the B&B algorithm is feasible, in our case, because of possible mixture of constraint propagation techniques which have polynomial complexity and branch-and-bound algorithm which, in the worst case, performs an exhaustive search with exponential complexity.

The experimental results indicate that the presented method can be used for synthesis of distributed heterogeneous systems having various constraints which is usually difficult to capture in a single design framework. The performance of this method is superior in comparison with MILP formulations [18, 19]. In all experiments, the runtime of the B&B algorithm was several orders of magnitude faster than the respective MILP runtime. Moreover the same model can be used by heuristics methods to build specialized design environments.

The method presented in this paper makes it possible to include many heterogeneous constraints in a single model and therefore perform synthesis in presence of realistic requirements. The inclusion of power consumption as well as memory constraints has been, for example, studied in [11, 20].

References

[1] T. Amon and G. Boriello, An Approach to Symbolic Timing Verification, In *Proc. 29th ACM/IEEE Design Automation Conference,* pp. 410-413, 1992.

[2] S. Bakshi and D. D. Gajski, A Scheduling and Pipelining for Hardware/Software Systems, In *Proc. of the 10th International Symposium on System Synthesis,* Sept. 17-19, 1997, Antwerp, Belgium.

[3] N. Beldiceanu, E. Bourreau, H. Simonis and P. Chan, Partial search strategy in CHIP, *Presented at 2nd Metaheuristic International Conference MIC97*, Sophia Antipolis, France, 21-24 July 1997.

[4] A. Bender, Design an Optimal Loosely Coupled Heterogeneous Multiprocessor System, In *Proc. of the European Design and Test Conference,* March 11-14, 1996, Paris, France, pp. 275-281.

[5] U. Bierker and P. Marwedel, Retargetable Self-Test Program Generation Using Constraint Logic Programming, In *Proc. 32nd ACM/IEEE Design Automation Conference*, 1995.

[6] *CHIP, System Do*cumentation, COSYTEC, 1996.

[7] B. P. Dave, G. Lakshminarayana and N. K. Jha, COSYN: Hardware-Software Co-Synthesis of Embedded Systems, In *Proc. Design Automation Conference*, 1997.

[8] P. Eles, K. Kuchcinski and Z. Peng, *System Synthesis with VHDL,* Kluwer Academic Publisher, 1997.

[9] P. Eles, K. Kuchcinski, Z. Peng, A. Doboli and P. Pop, Scheduling of Conditional Process Graphs for the Synthesis of Embedded Systems, *Proc. Design, Automation and Test in Europe Conference,* Feb. 23-26, 1998, Paris.

[10] P. Girodias and E. Cerny, Interface Timing Verification with Delay Correlation Using Constraint Logic Programming, In *Proc. European Design and Test Conference*, March 17-20, 1997, Paris, France.

[11] F. Gruian and K. Kuchcinski, Low-Energy Architecture Selection and Task Scheduling for System-Level Design, In *Proc. 25th Euromicro Conference*, Milan, Italy, September 8-10, 1999.

[12] W. D. Harvey and M. L. Ginsberg, Limited Discrepancy Search, *Proc. IJCAI 1995*.

[13] K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability,* McGraw-Hill, Inc., 1993.

[14] K. Kuchcinski, Embedded System Synthesis by Timing Constraints Solving, In *Proc. of the 10th International Symposium on System Synthesis*, Sep. 17-19, 1997, Antwerp, Belgium.

[15] Kuchcinski, K., An Approach to High-Level Synthesis Using Constraint Logic Programming, *Proc. 24th Euromicro Conference, Workshop on Digital System Design,* Västerås, Sweden, August 25-27, 1998.

[16] K. Mariot and P. J. Stuckey, *Programming with Constraints: An Introduction*, The MIT Press 1998.

[17] R. Niemann and P. Marwedel, Hardware/Software Partitioning using Integer Programming, In *Proc. of the European Design and Test Conference,* 1996.

[18] S. Prakash and A. Parker, SOS: Synthesis of Application-Specific Heterogeneous Multiprocessor Systems, *Journal of Parallel and Distributed Computing,* vol. 16, 1992.

[19] S. Prakash and A. Parker, Synthesis of Application-Specific Multiprocessor Systems Including Memory Components, *Journal of VLSI Signal processing,* vol. 8, 1994.

[20] R. Szymanek and K. Kuchcinski, Design Space Exploration in System Level Synthesis under Memory Constraints, In *Proc. 25th Euromicro Conference*, Milan, Italy, September 8-10, 1999.

[21] T.-Y. Yen and W. Wolf, Communication Synthesis for Distributed Embedded Systems, In *Proc. Int. Conference on Computer-Aided Design,* Nov. 1995.