A MULTIOBJECTIVE TABU SEARCH ALGORITHM FOR THE DESIGN SPACE EXPLORATION OF EMBEDDED SYSTEMS

Frank Slomka[1], Karsten Albers[1], Richard Hofmann[2]
[1]*Department of Computer Science, University of Oldenburg, Ammerländer Heerstraße 114-118, 26111 Oldenburg, Germany, {slomka,albers}@informatik.uni-oldenburg.de;*
[2]*Department of Computer Science 7, University of Erlangen-Nürnberg, Martensstr. 3, 91058 Erlangen, Germany, richard.hofmann@informatik.uni-erlangen.de*

Abstract: An important step during the design of embedded systems is to allocate suitable architectural components and to optimally bind functions (tasks) to these components. This design step is called system synthesis. The automation of system synthesis is limited in recent research by developing models only for standard optimization algorithms. This paper describes the first approach to improve a standard optimization technique itself for the use in embedded system design. Our solution extends the heuristic optimization algorithm tabu search by multiobjective optimization. Using the multiobjective approach, domain specific heuristics could easily be included into the algorithm. By performing experiments with the new algorithm, a new effect was discovered: In contrast to known results from literature, the quality of optimization was depending on the size of the neighborhood if the moves in the neighborhood were sorted by domain specific estimation.

Key words: Tabu-Search, Multiobjective, Optimization, System Synthesis.

1. INTRODUCTION

A number of approaches for system synthesis have been proposed in the relevant literature. The common goal of such approaches is to build an optimization model and to use heuristic optimization techniques to solve the synthesis problem. The optimization problem itself is tackled using simulated annealing [1], [6], genetic algorithms [1], [2], [5] and tabu search [1], [6], [10]. In some papers, self-made heuristics are used [4], [7], [11].

Most of the papers considering the analysis of real-time systems, which means different system tasks with different priorities running on one processor, must hold given deadlines. Only in [2] just the latency of the problem is considered.

Embedded system design is a multiobjective optimization challenge. The most important objectives are time, area and power. Many papers in system synthesis do not consider this aspect. The authors describe optimization as a cost or area optimization problem with time constraints. Only [2] and [5] describe multiobjective algorithms using a Pareto approach. Both papers deal with genetic algorithms, but [2] do not consider real-time systems. The algorithm described in [5] deals with real-time systems, but it disregards communication synthesis.

However, no paper deals with aspects for improving the quality of the optimization heuristic by information coming from the application domain, which in our case is embedded system design. The results of [1],[5], and [11] are based on an effective real-time analysis algorithm. In this paper, we present a technique that uses the multiobjective nature of the problem as a chance for improving the optimization technique itself. In contrast to recent literature on tabu search, which uses randomly generated neighborhoods [1], [6], we revealed that a sorted multiobjective neighborhood can improve the optimization algorithm.

2. EXPLORATION MODEL

Our approach to the system synthesis is based on two input models, (1) the problem graph to specify the application, and (2) the architecture graph to describe the maximal available hardware [2]. To synthesize a system architecture, the problem graph is mapped onto the architecture graph, along with the determination of additional parameters. A major difference of our modeling approach to related work is the use of different types of program nodes for modeling semantic aspects, e.g. to describe asynchronous communication between system-level tasks (processes) and other semantic peculiarities of languages based on the model of finite state machines.

2.1 Problem Graph

The problem model consists of two types of graphs: a control-flow graph (CFG) to model the behavior of the system (the control-flow of the specification, given in a formal description technique, e.g. SDL [12]) and a set of data-flow graphs (DFG). The two types of graphs are combined to a control

data flow graph (CDFG), or *problem graph* for short. Each DFG refines a node of the CFG to model computational tasks.

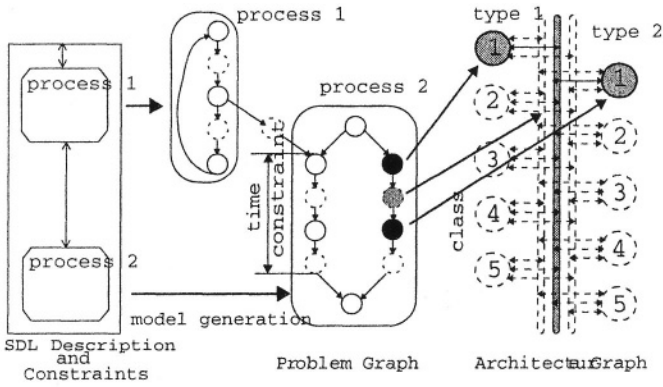

Figure 1. Optimization Modell

In order to derive the problem graph, the behavioral description of the system, as given by e.g. an SDL description, is transformed to the problem graph. The problem graph is a directed graph, where the nodes may represent computation or communication tasks. The edges of the problem graph specify the control flow. In addition to the nodes of the CDFG, we define super nodes that allow to model system-level tasks. A system-level task represents an instance of code with its own memory, i.e. a process running on an architecture component. In our model, a super node is defined as a set of nodes, where each node has the same process identifier and the same priority. The priorities of system-level tasks influence scheduling decisions where two or more super nodes are mapped onto the same architecture component, e.g. a processor.

In order to define time constraints, labels are assigned to the nodes. All nodes of the graph marked with the same label are associated to the same timing constraint.

2.2 Architecture Graph

The architecture graph defines the maximal available configuration of the hardware. It contains different types of nodes to model different types of components. The node types are used to model different scheduling strategies of processing elements: preemptive scheduling, nonpreemptive scheduling, no scheduling and communication. In this terminology, no scheduling means that a resource can only be used by one task exclusively.

All architecture nodes contain a class attribute that specifies different classes of components (e.g. different processor types and technologies). Fig. 1 outlines how the behavioral description — in our case an SDL description — is transformed to a problem graph, which in turn is mapped onto an architecture graph. In the problem graph, dotted nodes represent communication nodes. In the architecture graph, dotted nodes represent architecture nodes not yet allocated.

2.3 Allocation, Binding and Scheduling

As outlined above, our optimization algorithm for system synthesis defines the allocation, the binding and the schedule.

The **allocation** defines the selection of the architecture components from the available architecture components as defined by the architecture graph. The decisions are implicitly defined by the binding, i.e. each architecture node to which a program node is bound to is allocated.

Binding: Each node of the problem graph is bound to exactly one architecture node. The binding of the nodes of the problem graph to the architecture graph is achieved by edges between the two graphs.

Scheduling: We assume that the derived implementation employs runtime scheduling. Thus, decisions made by the system synthesis implicitly define the schedule. The factors influencing the schedule comprise 1) the scheduling strategies employed by the architecture nodes, 2) the binding of the program nodes on architecture nodes, 3) the priorities of the program nodes (super nodes).

3. MULTIOBJECTIVE OPTIMIZATION WITH TABU-SEARCH

3.1 Multiobjective Optimization

Optimization of embedded systems is a multiobjective search problem. Different design parameters like time, area and power need consideration. Using a weighted cost function in a multiobjective optimization problem is questionable [5]. The weights are depending on the problem and finding the right set of weights could be as expensive as the optimization problem itself.

In multiobjective optimization, an effective method for ranking solutions is used: the Pareto approach. Consider a two dimensional objective space, e.g. area and power. For each solution, a value for each objective, power and area, can be calculated. This results in a point in the two dimensional objec-

tive space. A point in the objective space is *dominating* when it is in all objectives at least equal or better then the dominated point. Pareto points are points which are not dominated by other points. Thus, all system implementations represented by Pareto points are equal in terms of their design quality. To further discriminate them requires additional constraints. An optimization tool can find the Pareto points by using an algorithm called Pareto ranking [5]. Pareto ranking sorts all solutions according to the number of solutions that dominate them. The Pareto points are dominated by no other points, so they are on the top of the list.

3.2 Tabu Search

Tabu search is a heuristic optimization algorithm. In contrast to simulated annealing and genetic algorithms, tabu search represents a purely deterministic approach. Similar to simulated annealing, tabu search is based on a neighborhood search. Thus, any new solution is derived from the previous solution. In order to support this, the definition of the neighborhood of a solution and the definition of the moves to transform a previous solution to a new solution is of importance.

Different from greedy algorithms, e.g. as gradient search, tabu search also allows moves to solutions with higher cost. This is important for escaping from local minima. However, allowing non-improving steps may result in a cyclic search. To avoid cycles, tabu search employs a memory, typically called tabu list. The purpose is to prevent moves, which can lead to cycles. This list could have very different implementations. One possible way is to store a fixed number of previous moves, whose recurrence is inhibited.

3.3 Structure of Multiobjective Tabu Search

The idea of Pareto ranking can be used to construct a multiobjective tabu search algorithm. As tabu search defines moves for constructing new solutions and all moves are put to a list, called neighborhood, it is easy to construct a single neighborhood for each objective. The moves are evaluated only by the single objective of the neighborhood. Additionally, it is possible to use estimation techniques to find fast evaluation results. Such a fast estimation technique may be the use of Liu and Laylands real-time analysis instead of a computation expensive worst-case response time analysis or simulation. By using this technique, it is possible to explore a lot of possible system implementations or solutions. All neighborhoods will then be sorted separately. These lists are than merged with Pareto ranking.

However, constructing an optimal solution requires an extended analysis of complex objectives, which is in our approach carried out by selecting the

N best moves given by the Pareto ranking. For the chosen solutions, an extended and precise evaluation is performed.

3.4 Moves and Neighborhood

3.4.1 Definition of Moves

Priority Changing of Processes: To each process, a priority is attached. The priority defines the scheduling priority of a process. This means that all processes bound to the same component will be scheduled with respect to this priority. The actual priority of a process can be increased or decreased within a move.

Partitioning of Processes: Each process can be partitioned. A process partitioning is supported by the super node concept. Each process can be split into a number of super nodes. If a process is split into different super nodes, it is possible to bind the super nodes to different components of the component graph. This is used to support fine grain hardware/software partitioning in a hierarchical environment.

Binding of Super Nodes: This move changes the binding of the super nodes to hardware components. Only the binding of super nodes can be changed.

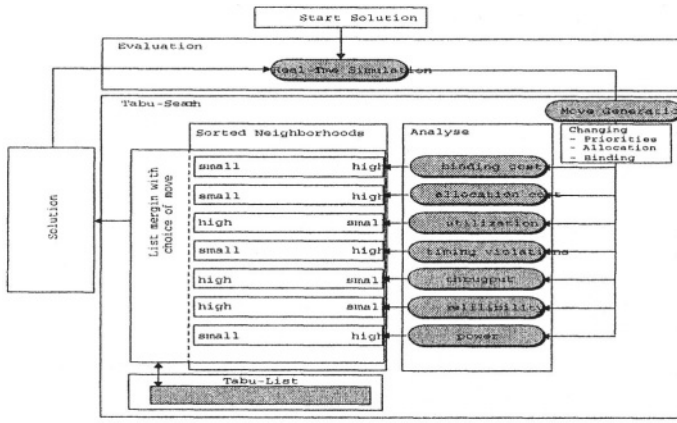

Figure 2. Multiobjective Tabu-Search

Allocation of Components: Moves to allocate and deallocate components are very important. If allocation were implicitly performed by binding moves, in many cases it would take a few moves to deallocate a component.

These intermediate steps are problematical because the not yet deallocated component still needs area, while its workload is moved to other components, which reduces their quality in other objectives like the performance. Deallocating moves prevent this overall stepwise degradation.

3.4.2 Neighborhood

The result of these moves is the neighborhood of a solution. In many cases, the resulting number of different moves is too high for a reasonable performance of the optimization process. Every move in a neighborhood is evaluated separately for each objective. Therefore, a separate list of moves is built for each objective.

Allocation Cost: The total system cost consist of the allocation cost, which represent the fixed cost of architecture components and the binding cost that is caused by binding the program nodes on architecture nodes. Each allocated node of the architecture graph results in a fixed cost. The fixed cost of an architecture component depends on its type and its class.

Binding Cost is caused by the need for memory (software) or cost for registers, ALUs, etc., on ASICs. Similar to allocation cost, binding cost depends on the type and class of the architecture component to which the program node is bound.

Timing Constraints: In order to evaluate meeting of time constraints, the actual execution schedule has to be derived. This schedule not only depends on the allocation and the binding of the processes but also on the distribution of the priorities. In this work, a simple event-driven simulation analyzes the temporal behavior of a given system (for more information see [10]). The problem to verify a given system and scheduling is NP-complete [5]. For that reason, it is problematic to verify a large neighborhood in an exact manner. As a rough estimation about the real-time behavior, the utilization formula in Liu und Layland [8] is used. The calculated utilization is then used as a metric to sort the solutions into the neighborhood.

3.4.3 The Tabu List

A new approach combines a problem independent sizing of the tabu list with a reduction of program memory size: It uses the multiobjective nature of the problem to implement an effective tabu list. The evaluation parameter of each neighborhood is stored separately. Although the used data structure is very compact, it describes a solution very exactly: Let us assume e.g., a system implementation needs **100 mm²** allocation area, **50 mm²** program memory (binding cost) at 70% processor utilization. In such a case, the set

{100, 50, 70} completely describes the system. As equivalent solutions have the same evaluation parameter, they can be mapped to the same description.

4. EXPERIMENTAL RESULTS

Using a multiobjective search for hierarchical problems in combination with neighborhood estimation reduces the run-time and the cost of the final system implementation. We verified this by experiments based on given examples from the literature [11]. Fig. 3 gives an idea about the quality improvement achieved by sorted neighborhoods. The figure shows the two objectives, number of real-time violations and total system cost. The total system cost is the sum of binding and allocation cost. The figure shows the effect of sorted neighborhoods for the examples *random1* and *random2*. Tab. 1 gives a detailed overview, how the algorithm's run-time depends on the size of the neighborhood.

The table shows three experiments with a small neighborhood and a large neighborhood to find an optimal value for the length of a sorted neighborhood. The first number in the neighborhood size row gives the number of estimated moves and the second number gives the number of total evaluated (using a detailed real-time simulation) moves. This number is equivalent to a neighborhood size in standard tabu-search. The given run-time is the complete run-time for 10,000 iterations fixed given by starting the program. The number in the brackets gives the number of iterations after which the best solution was found. As can be seen, a neighborhood of 6000 yields better results than a small neighborhood of 600 moves. In detailed experiments, we found that a size of 4000 is a break-even in run-time and quality. A neighborhood with a size larger than 8000 again increases the run-time of the algorithm, without improving quality. However, a neighborhood size between 4000 and 8000 is a good value for optimization with sorted neighborhoods.

The experiments reveal that the result from tabu search literature, which states that the quality of optimization is independent from the neighborhood size, only holds for unsorted or randomly generated neighborhoods. Our newly found result allows the design of fast system synthesis algorithms based on tabu search. Note that the experiments in [1] show that the quality of tabu search is as good as the quality of genetic algorithms. Sorting the neighborhood gives the possibility to include embedded system designers knowledge to the heuristic search algorithm and to improve the results given in [1], [2], and [5].

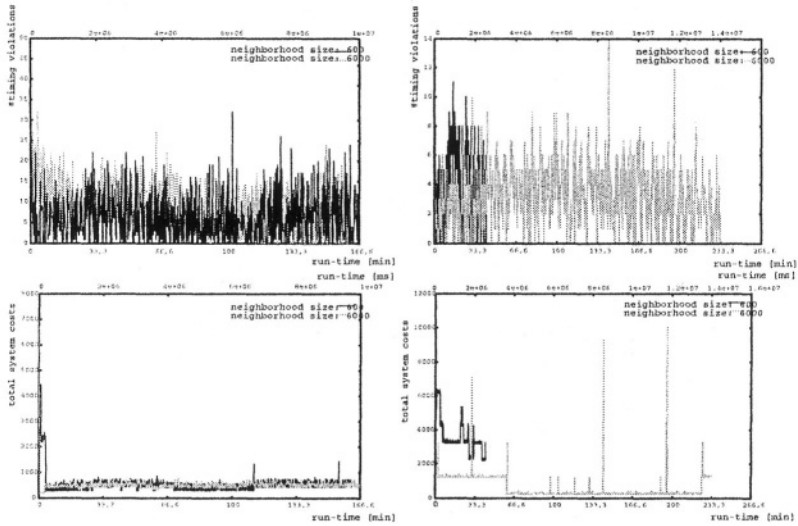

Figure 3. Results

Table 1. Results: Different Sizes of the Neighborhood

Experiment	*Small Neighborhood*			*Large Neighborhood*		
	Size	*Cost*	*run-time[ms]*	*Size*	*Cost*	*Run-Time [ms]*
Random-1.20	*600/20*	*310*	*424212 (4652)*	*6000/20*	*195*	*133910 (36)*
Random-2.20	*600/40*	*2192*	*2421560 (3397)*	*6000/40*	*222*	*4671360 (789)*
Random-3.100	*600760*	*500*	*6194830 (631)*	*8000/20*	*417*	*73440 (11)*

5. CONCLUSION

In this paper, a new multiobjective tabu search heuristic is presented: Including domain-specific heuristics into a general optimization algorithm improves the quality of the optimization results and reduces the algorithm's run time. For that reason, the heuristic was extended to work with both estimation and evaluation algorithms for the different objectives. This enables the use of large neighborhoods without loosing quality of the optimization results. It also combines for the first time tabu-search with Pareto-ranking.

REFERENCES

1. J. Axelsson. Analysis and Synthesis of Heterogeneous Real-Time Systems. Dissertation, Linköping Studies in Science and Technology, 502. 1997.

2. T. Blickle, J. Teich, L. Thiele. System-Level Synthesis Using Evolutionary Algorithms. Design Automation For Embedded Systems, Kluwer Academic Publisher, Boston, 3(1), 1998.
3. Dave, B.P. und Jha, N.K. COHRA. Hardware-Software Cosynthesis on Hierarchical Heterogeneous Distributed Embedded Systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 17(10). 1998.
4. B.P. Dave, G. Lakshminarayana, N.K. Jha. COSYN: Hardware-Software Co-Synthesis of Heterogeneous Distributed Embedded Systems. IEEE Transactions on Very Large Scale Integration (VLSI). 7(1), 1999.
5. R.P. Dick, N.K. Jha. MOGAC: A Multiobjective Genetic Algorithm for Hardware-Software Cosynthesis of Distributed Embedded Systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 17(10). 1998.
6. P. Eles, Z. Peng, K. Kuchcinski, A. Doboli. System-Level Hardware/Software Partitioning based on Simulated Annealing and Tabu Search. Design Automation For Embedded Systems, Kluwer Academic Publisher, Boston, 2(1), 1997.
7. C. Lee, ,M. Potkonjak, W. Wolf. Synthesis of Hard Real-Time Application Specific Systems. Design Automation for Embedded Systems. Kluwer Academic Publisher, Boston, 4(4), 1999.
8. C. Liu, J. Layland Scheduling Algorithms for Multiprogramming in Hard Real-Time Environments, Journal of the ACM, 20(1), 46-61, 1973
9. F. Slomka, J. Zant, L. Lambert. Schedulability Analysis of Heterogeneous Systems for Performance Message Sequence Chart. 6th International Workshop on Hardware/Software Codesign. IEEE Computer Society Press. Seattle, 1998.
10. F. Slomka, S. Kocher, A. Mitschele-Thiel. A Three-Level Heuristic for System Synthesis Based on Tabu Search. Technical Report IMMD7 3/99. University of Erlangen-Nuremberg.
11. T. Yen, W. Wolf. Hardware-Software Co-Synthesis of Distributed Embedded Systems. Kluwer Academic Publisher, Boston. 1996.
12. ITU-T. Z.100, Appendix I. ITU, SDL Methodology Guidelines. ITU, 1993