System-Level Synthesis of Application Specific Systems using A* Search and Generalized Force-Directed Heuristics

Chunho Lee, Miodrag Potkonjak, and Wayne Wolf†
Computer Science Department, University of California, Los Angeles, CA
† Department of Electrical Engineering, Princeton University Princeton, NJ

Abstract

This paper presents a system-level approach to the synthesis of multi-task, hard real-time applications. The goal is to select a set of off-the-shelf processors with minimal cost while satisfying timing constraints. Our approach has three design phases: resource allocation, assignment, and scheduling. With the observation that the resource allocation is a search for a set of processors which requires the minimum cost, we adopted A search based technique. For assignment we use a variation of the force-directed technique. Final task scheduling is based on the Earliest Deadline First (EDF) algorithm. Experimental results show that this approach is highly effective on a variety of examples.*

1.0 Introduction

The proper functioning of a real-time systems on the time at which the results are produced as well as the logical correctness of the result [Liu73]. Classical examples include automobile and airplane monitoring systems. Modern real-time systems are often intrinsically multiple-task applications. For example, a video-server has to handle simultaneous requests from several users and it should be able to assemble and deliver both video and sound components in response to such requests.

Both behavioral and system synthesis have been focused on synthesis of single task applications [McF90]. Only a few research groups addressed synthesis of multi-threaded, real-time [Pra94, Yen95] and multi-task applications [Pot95].

Our goal is to develop a modular, flexible, and reusable synthesis tools for system level synthesis of multi-task hard real-time application specific systems. We are on the brink of renaissance in scheduling due to the recognition of important scheduling problems at the task level. Task-level scheduling will provide new avenues for high-impact research and industry-relevant tool development.

2.0 Problem Formulation and Complexity

We assume that all tasks are defined on semi-infinite streams of data. All tasks are periodic. For each task, three periodic timing constraints are imposed: the *period*, the *start time*, and the *finish time*. For each task execution time or upper bound on the execution time on each of the available processors is tabulated. Since the tasks are independent, there is no communication cost.

Two implementation constraints are imposed. The first constraint is that no preemption is allowed. Preemption often drastically simplifies many synthesis problems. However, very high context switching times for modern operating systems suggest that context switching can become prohibitively expensive. The second restriction is that all instances (iterations) of a periodic task should be executed on the same processor.

We can now formulate the optimization problems and establish their computational complexity. The targeted synthesis problems can be defined as follows:

Allocation: A set of k processors and a set of n independent periodic hard real-time tasks are given. Each processor has an associated cost. Select a multisubset of processors (subset where some processors can be include more than once) so that each task is assigned to exactly one processor and that the sum of costs of the selected processors is at most K.

Assignment (Partitioning): A set of k processors and a set of n periodic hard real-time tasks are given. Assign each task to one of the processors in such a way that all tasks can be scheduled within their timing constraints.

Scheduling: A set of n independent periodic hard real-time tasks is given. The goal is to generate schedule of the tasks so that all timing constraints are satisfied.

We can prove that allocation, assignment, and scheduling for system-levl synthesis of hard real-time systems are

Proceedings of the 9th International Symposium on System Synthesis (ISSS '96)
1080-1820/96 $10.00 © 1996 IEEE

NP-complete problems.

3.0 Synthesis Approach: Overview

The overall synthesis flow is as follows:

System-level synthesis of hard real-time systems
repeat
Allocation();
Assignment();
Scheduling();
until (a set of feasible schedules is generated);

The allocation subtask proposes a set of allocated processors to the assignment and scheduling procedures. Partitioning, in turn, assigns tasks to processors and passes the result over to the scheduling procedure. Finally, scheduling generates a feasible schedule for each allocated processor if there exists one. If there is no feasible schedule, the allocation procedure enters again at the point where it left before and the procedures are repeated until a set of feasible schedules is obtained.

The allocation subtask finds a set of resources by searching the solution space using the A* search strategy [Rus95]. The solution can be represented by a path in a solution tree. The root node of the solution tree represents the empty initial solution. At each step of the search, one out of k branches is chosen. The search follows the A* search strategy. The partitioning procedure assigns a task at a time to a processor. The assignment heuristic is based on the force-directed scheduling [Pau89] . Our scheduler is based on the EDF scheduling, as explained in Section 6.

4.0 Resource Allocation

The allocation algorithm adopts the A* search strategy [Rus95]. The heuristic function used in the search is based on a relaxed partitioning and scheduling.

The lower bound of the implementation cost of each task is the minimum among the products of the costs of processors and the corresponding run time of the task. For example, given a set of tasks and processors as in Table 1, the minimum implementation cost of each task is computed as $M[i] = min_j\{C[j]*E[i][j]/T[i]\}$ where $M[i]$ is the minimum implementation cost for a task i, $C[j]$ is the cost of a processor j, $E[i][j]$ is the computation time of a task i on a processor j, and $T[i]$ is the period of a task i. The result is given in the Table 1.

The sum of $M[i]$'s signifies the lower bound of the implementation cost. That is, we have to spend at least 54.9 to implement all the tasks. Those $M[i]$'s are goals which guide our search for the low cost implementation.

	P1	P2	P3	P4	T	C1	C2	C3	C4
t1	7	5	3	6	5	-	20	18	-
t2	8	5	4	9	6	-	16.7	20	-
t3	7	4	2	5	10	10.5	8	6	5
t4	5	7	3	8	10	7.5	14	9	8
t5	9	8	5	7	15	9	10.7	10	4.7
t6	10	9	4	9	30	5	6	4	3
c	15	20	30	10					

Table 1: Synthesis problem and implementation cost on each processor: Pi, tj, and Ci denote available processors, tasks, and implementation cost respectively. "-" in [i][j] indicates the task i cannot be implemented on the processor , c - cost.

In each allocation step, before a processor is chosen to be allocated, we check how well the set of allocated processors will be utilized and how many more processors should be added in the following allocation steps if the processor being examined is chosen. These estimates are obtained through the relaxed partitioning and scheduling.

Our original problem has a set of timing constraints, the atomic execution constraint, and the non-preemptive scheduling constraint. We relax the atomicity restrictions and perform partitioning. Each instance of a task is divided into several pieces based on the number of allocated processors and its execution time on each processor. For example, consider the problem given in the Table 1. When *P1* and *P2* are allocated, the probability

	P1	P2	T	S	D	A
t1	3	2	5	1	3	3
t2	2	3	5	1	3	3
t3	3	5	10	3	7	5
cost	16	10				

Table 2: An instance of synthesis problem: T- period, S start time, D- deadline, and A - available time.

table shown in the Table 2 is computed. The numbers in

	P1	P2	P1- estimated cost	P2- estimated cost
t1	2/5	3/5	(2/5)*3	(3/5)*2
t2	3/5	2/5	(3/5)*2	(2/5)*3
t3	5/8	3/8	(5/8)*3	(3/8)*5

Table 3: Probability and execution time table for relaxed assignment and scheduling

the table are estimated execution times of corresponding tasks on the combined superprocessor which combines all the computing capacities of the allocated processors.

With the observation that the utilization factor reveals an upper bound [Liu73] for preemptive schedulability and the

Proceedings of the 9th International Symposium on System Synthesis (ISSS '96)
1080-1820/96 $10.00 © 1996 IEEE

fact that the non-preemptive scheduling is more difficult (in both terms of checking the schedulability and having a feasible schedule) [Sta95], we can incorporate it in our heuristic function as the means to estimate if a set of allocated processors can be a feasible solution. For example, the utilization of the processor set of *P1* and *P2* given in the Table 5 is (2/5)*(3/5) + (3/5)*(2/5) + (5/8)*(3/10) = 0.67. The utilization provides a good measure as to whether a feasible schedule can be found or not.

When the utilization is too high, we need to know what portion of the task sets can be scheduled on the allocated processors to proceed with the search. To do that, we perform a relaxed scheduling on the partitioned task set using our force-directed EDF. In the process, if we encounter a task that cannot be scheduled, the task is thrown away and the scheduling continues until it finishes identifying a set of tasks that is scheduled. As described in Section 7, our scheduling is based on the EDF which offers the optimal length schedule if it finds a feasible schedule. With the set of tasks that is scheduled, the utilization is computed. By combining the utilization and the estimate of the future cost that is required for the tasks that were thrown away, we get an estimate of the overall implementation cost. The estimate is given by

$$\sum_i C[j] + \sum_i C[j](1-U) + \sum_i min_j\{C[j]*E[i][j]/T[i]\}$$

where *C[j]* refers to the cost of processor *j*, *U* utilization of allocated processors.

When a set of processors is allocated, the cost, the utilization, and the estimated future cost is checked. If the cost is greater than the current minimum and the relaxed partitioning and scheduling are completed successfully for all the given tasks, the actual partitioning of the task set onto the set of allocated processors is performed.

Allocation is summarized by the following pseudo-code:

```
Allocation ()
repeat
   perform the relaxed assignment and scheduling for all
   processors;
   collect a processor at a time that is most promising;
   if the relaxed assignment and scheduling are successful
      then check the utilization factor;
         if the utilization is too high then allocate more
            resources;
         else go to the actual partitioning;
      else continue;
until all the tasks can be scheduled using
relaxed_scheduling;
```

5.0 Partitioning (Assignment)

Our partitioning procedure is based on the observation that we have the best chance of finding a feasible schedule if we assign tasks onto the allocated processors in such a way that the distribution graphs on all the allocated processors are as even as possible. We modify the force-directed scheduling algorithm for behavioral synthesis [Pau89] to find a good partitioning of a given task set.

First we define probabilities of each task based on execution times of a task on each processor with which a task is tentatively assigned onto processors as follows:

$$P[i][j] = 1/(E[i][j]*E[i][j]*\sum(1/(E[i][j]*E[i][j])))$$

By doing so, we take into account the fact that it we prefer to assign a task to a processor which executes it more quickly. Next, we use a modified force-directed assignment procedure to balance the loads among the processors and to make distribution graphs on all the processors as even as possible. In addition to the probabilities, we compute the overall distribution of the tasks over the allocated processors by:

$$DG(task\ i,\ processor\ j) = P[i][j]*(E[i][j]/A[i])$$

We illustrate the procedure using the following example. The Table 4 shows characteristics of each task and each

	P1	P2	T	S	D	A
t1	3	2	5	1	3	3
t2	2	3	5	1	3	3
t3	3	5	10	3	7	5

Table 4: An illustrative example for partitioning

	P1	P2
t1	4/13	9/13
t2	9/13	4/13
t3	25/34	9/34

Table 5: Probability table for tentative assignment.

processor. In the Table 5, probabilities associated with each task on each processor are computed. Figures 1 and 2 show the respective probabilities of tasks multiplied by

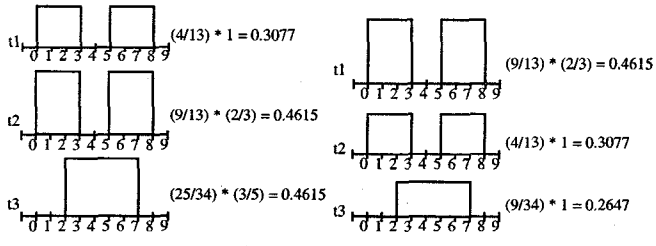

Figure 1: Distribution graph of each task on P1 (left)and P2 (right).

Proceedings of the 9th International Symposium on System Synthesis (ISSS '96)
1080-1820/96 $10.00 © 1996 IEEE

their respective distribution over the time frame during which they can be executed.

Clearly, it is not possible to assign all the tasks on a processor and to have a feasible schedule because there are time slots where the sum of distribution graphs is greater than 1. When we attempt to assign a task on a processor, the self-force is computed as follows:

$$self-force(task\ i,\ processor\ j) = \sum_i \sum_k DG[k,j]*X[i]$$

For example, self-force of task 1 on P2 is -0.7178. Of the values of the self-force, this one is the minimum. This can be interpreted as assigning the task 1 onto the processor 2 is the best choice in terms of maximizing schedulability.

The algorithm assigns tasks onto processors one at a time. When a task is identified to be assigned on a processor, the probability table is updated. As a result of assigning the task 1 on the processor 2 we get the probability table given in the Table 5. After updating the probability table, the values of the self force for the rest of the tasks with the new probability table are computed using the same procedure. In our example, the algorithm assigns the task 2 on the processor 1 and finally the task 3 on the processor 1.

Here is the complete assignment algorithm:

```
Assignment ()
repeat
   for all tasks t
      for all processors p
         compute self force(t,p);
      endfor;
   endfor;
   pick a combination of a task and a processor with least
   self force;
until all the tasks are assigned;
```

6.0 Task-Level Scheduling

The basis for our scheduling algorithm is EDF scheduling [Sta95]. We turn our attention to a heuristic procedure which transforms the given scheduling problem into a different form in such a way that the EDF can be applied to nearly optimally find a feasible schedule. By transforming the schedule to delay execution of one or more tasks, we can often find a feasible schedule even when pure EDF cannot. The scheduling algorithm is based on the following three easy-to-prove observations:

#1. Any sequence is optimal if all the tasks have the same start time and the same deadline.

#2. The EDF is optimal if all the tasks have the same deadline and different start times.

#3. The EDF is optimal if the deadlines are non-decreasing when the tasks are ordered in the non-decreasing order of the start times (i.e., tasks are released according to the order of their respective deadlines).

EDF gives the shortest schedule if it is possible for EDF to find a feasible schedule at all. If a task misses its deadline when EDF is used, then it would be beneficial to delay one or more tasks executed prior to the task missing its deadline. Therefore, the candidates that will be delayed should have deadlines that are later than that of our target task and start times earlier than those of the target task. The target task means the task that our algorithm will make meet its deadline. When a candidate is selected to be delayed, we want to make sure that the number of unused time slots and overlaps are minimized. If every deadline is met, we have a feasible schedule.

Fig. 3 depicts an instance of a scheduling problem. The rectangles refer to respective deadlines and start times of the tasks. The numbers inside the boxes indicate the execution times over available times. Pure EDF does not

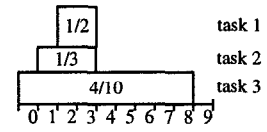

Figure 2: (a) Example #1: Distribution graphs of tasks to be scheduled

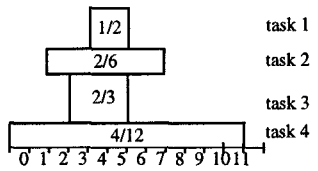

Figure 2: (b) Example #2: Distribution graphs of tasks to be scheduled

give a feasible schedule in this case. But if task 3 yields its execution turn in favor of meeting the deadline of the task 1 (or task 2), then there is a feasible schedule which can be found by applying EDF to the transformed system. Note that in order to have a feasible schedule we must not use the first time slot. The schedule, however, is optimal.

The example shown in Fig. 2b cannot be solved using EDF either. But if at most two tasks yield their execution turns, all tasks can meet their deadlines. When the standard EDF is used, the schedule would be task 4 →task 3→task 1 → .., which is not feasible. Note that there is only one task the start time of which can be delayed, namely, task 4. After changing the start time of task 4 to the start time of task 3 or task 1, the EDF generates the schedule task 2→task 3→task 1 → .. Again it is not feasible. By moving the start time of task 2 to the start time of task 3 or task 1, the EDF finds a feasible schedule.

Next consider the example shown in Fig. 3. The schedule by the EDF without transformations would be task 5→task 2→task 1 → .,,. There is no feasible schedule. By delaying the start time of task 5, the modified EDF finds a new

Proceedings of the 9th International Symposium on System Synthesis (ISSS '96)
1080-1820/96 $10.00 © 1996 IEEE

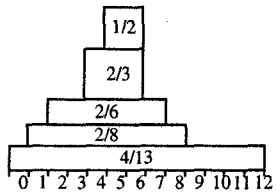

Figure 3: (a) An example of scheduling problem requiring the application of force-directed selection of a task to be delayed

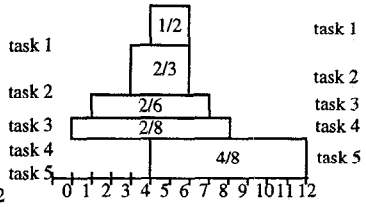

Figure 3: (b) The scheduling problem after one application of delaying a task.

sequence task 4 →task 3→task 2→task 1 → .. In this case there are more than one tasks that can be delayed (task 4 and 3). The choice for delaying its start time will impact the feasibility and effectiveness of the procedure. For example, if task 4 is chosen, the EDF cannot find a feasible schedule. On the other hand, if task 3 is chosen, the schedule length might be longer because the processor will be idle at time 1.

The first example shows that we can improve our chance of having a feasible schedule by checking to see if any task misses its deadline if a task which arrived prior to the task missing its deadline executes before the task. We do not check all the combinations of start time changes here. In the example, if task 3 executes according to its start time, then task 1 misses its deadline. So, it is our advantage to move the start time of task 3 to that of task 1 so that task 1 can go first. The second example, however, is more complex. No task misses its deadline if one of the prior task executes first. The third example is even more complex. To address it we propose force-directed delay-based EDF that is the described in the rest of this section.

The algorithm first tries EDF to find a feasible schedule. If it cannot, then it checks the given set of tasks to see if EDF is optimal using the criteria given in 1-3. If it is the case, there is no a feasible schedule. Otherwise, it selects a task at a time to be delayed using modified force-directed scheduling [Pau89].

The distribution is defined as the probability by which a task demands a particular time slot for its execution. By taking the summation of the probabilities of all tasks, we obtain distribution graphs. The resulting distribution graphs in our problem indicate the demand at a time slot for the resource requested by all the tasks. The DG at time 2 in Fig. 2a is 2/5 + 1/3 + 1/2 = 37/30, which means at least one task must be able to be scheduled not claiming the time slot in order for us to have a feasible schedule. Interestingly, if task 3 is scheduled solely based on the start time of it, it must use the time slot 2 and there is no feasible schedule for the task set. The distribution graphs are given by $\sum_i (E[i][j]/A[i])$

Each task has a self force associated with each time slot of its time frame which reflects the effect of an attempted delay of a start time of a task on the overall demand requested by all the tasks given by $force = DG[i] * x[i]$ where $DG[i]$ is the current distribution value and $x[i]$ is the change in the operation's probability. Our goal is that at each scheduling step every possible time slot at the origin side of time axis is used. Therefore, unlike the self force of force-directed scheduling [Pau89], our self force has a positive value when the DG is lower than 1.0 at the origin side of the time axis. We want to select a task to be delayed in such a way that it minimizes the chance of causing holes (unused time slots) and the possibility of overloading time slots, it is desirable to ensure distribution at each time slot to be as close as possible to 1.0.

Finally, the self-force associated with delaying the start time of a task is $\sum_i force(i)$ for all affected time slots i.

To illustrate the application of self force to choose a task to be delayed, we use example 3. We first obtain the distribution graphs in Fig. 3. We attempt to delay the start times of task 3 and 4, and calculate self-force(3) = 2.0208 and self-force(4) = 1.625.

From the two resulting self-force values, we see that changing the start time of task 4 has less overall adverse effect on the schedulability. In fact, changing the start time of task 3 will lead to no feasible schedule. The point that we have to have minimal amount of unclaimed holes at the origin side of the time axisis is valid in the sense that the schedulability might be hampered later on if we do not use all the possible time slots. In the example, EDF now finds a feasible schedule: task 3 -> task 2 -> task 1 -> task 4 -> task 5. Next scheduling will be continued at the time slot 14.

The following procedure checks to see if there is a feasible schedule for the given set of tasks.

1. Identify the set of released tasks. That is, look for the set of tasks that are arrived before the first deadline of a task in the set.
2. Try EDF for the task set. If there is no feasible schedule, compute the self-force for each task that is released prior to the deadline and have a deadline later than that of the target.
3. Select a task that will be scheduled after the target task based on the values of the self-force found in #2. Adjust the start time of the selected task to the start time of the target task.
4. Repeat steps #2-#3 until a feasible schedule is found or there is no more possible candidate that can be delayed.

Proceedings of the 9th International Symposium on System Synthesis (ISSS '96)
1080-1820/96 $10.00 © 1996 IEEE

5. When a feasible schedule is obtained, compute the schedule length and adjust start times of subsequent tasks. Repeat the procedure from this point of time on until done or found there is no feasible schedule.

7.0 Experimental Results

Table 6 shows improvement of new approach over one where the best random resource allocation is used for several sets of randmly generated tasks. The utilization of resources are high and indicate high quality results.

Number of Tasks	32	40	44	48
Best Random Solution	580	745	720	830
Optimized Solution	430	445	530	610
Resource Utilization	0.692	0.575	0.676	0.607

Table 6: Experimental Results

8.0 Related Research

The directly related research are ones conducted in behavioral and system synthesis, real-time scheduling, and search and heuristic optimization techniques.

A review of the early work on behavioral synthesis is given in numerous references [McF90]. System level synthesis is premier design and CAD research topics [Bar94, Gup93, Gaj96, Wol94]. The early work on scheduling of a set of periodic tasks with timing constraints on periodicity, start, and finish time of each task, resulted in a classic rate-monotonic scheduling [Liu73]. Consequently, numerous real-time scheduling algorithms has been proposed and analyzed [Sta95].

A* search and force directed heuristics are often used as optimization mechanisms for computationally intractable problems [Rus95]. Force-directed heuristics have been widely used. Paulin and Knight [Pau89] developed a force-directed approach for data-flow graph scheduling, which due to its clear intuitive foundations and strong performances have been used by many behavioral synthesis schedulers [McF90].

9.0 Conclusion

An approach for synthesis of application specific systems which implements a set of hard real-time tasks using general purpose processors is presented. The approach has three steps: A* search based resource allocation, force-directed assignment, and earliest-deadline first-based scheduling. The experimental results show the high effectiveness of the approach.

10.0 Acknowledgements

Wolf was supported by NSF grant MIP-9424410.

11.0 References

[Bar94] E. Barros, W. Rosenstiel, and X. Xiong, "A method for partitioning UNITY language in hardware and software," *EuroDAC '94*, pp. 220-225, 1994.

[Gaj96] D.D. Gajski, et al. "System design methodologies: aiming at the 100 h design cycle", *IEEE Transactions on VLSI Systems*, Vol.4, No.1, pp. 70-82, March 1996.

[Gar79] M. R. Garey, D. S. Johnson, "*Computers and Intractability: A Guide to the Theory of NP-Completeness*", W. H. Freeman New York, 1979.

[Gup93] R. K. Gupta and G. De Micheli, "Hardware-software cosynthesis for digital systems," *IEEE Design & Test of Computers*, Vol. 10, No. 3, pp. 29-41, 1993.

[Liu73] C.L. Liu, J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment", *Journal of ACM*, Vol. 20, No. 1, pp. 46-61, 1973.

[McF90] M.C. McFarland, A.C. Parker, R. Camposano: "The High-Level Synthesis of Digital Systems", *Proceedings of the IEEE*, Vol. 78, No. 2, pp. 301-317, February 1990.

[Pau89] P.G. Paulin, J.P. Knight, "Force-directed scheduling for the behavioral synthesis of ASICS", *IEEE Transactions on CAD*, Vol. 8, No. 6, pp. 661-679, June 1989.

[Pot95] M. Potkonjak, W.H. Wolf, "Cost Optimization in ASIC implementation of Periodic Hard-Real Time Systems using Behavioral Synthesis Techniques", *ICCAD95* pp. 446-451, 1995.

[Pra94] S. Prakash, A. C. Parker: "Synthesis of application-specific multiprocessor systems including memory components", *Journal of VLSI Signal Processing*, Vol.8, No.2, pp. 97-116, Oct. 1994

[Rus95] S. Russel, P. Norvig, "*Artificial Intelligence: A Modern Approach*", Prentice-Hall, Englewood Cliffs, NJ, 1995.

[Sta95] J.A. Stankovic, et al., "Implications of Classical Scheduling Results for Real-Time Systems", *IEEE Computer*, Vol. 28, No. 6, pp. 16-25, June 1995.

[Wol94] W.H. Wolf: "Hardware-Software Co-Design of Embedded Systems", *Proc. of IEEE*, Vol. 82, No. 7, pp. 967-989, 1994.

[Yen95] T.-Y. Yen, W. Wolf, "Communication Synthesis for Distributed Embedded Systems", *ICCAD95* , pp. 288-294, November 1995.

Proceedings of the 9th International Symposium on System Synthesis (ISSS '96)
1080-1820/96 $10.00 © 1996 IEEE