A Tool for Generating Partition Schedules of Multiprocessor Systems

Hans-Joachim Goltz and Norbert Pieth

Fraunhofer FIRST, Berlin, Germany
{hans-joachim.goltz,nobert.pieth}@first.fraunhofer.de

Abstract. A deterministic cycle scheduling of partitions at the operating system level is supposed for a multiprocessor system. In this paper, we propose a tool for generating such schedules. We use constraint based programming and develop methods and concepts for a combined interactive and automatic partition scheduling system. This paper is also devoted to basic methods and techniques for modeling and solving this partition scheduling problem. Initial application of our partition scheduling tool has proved successful and demonstrated the suitability of the methods used.

1 Introduction

Particularly in safety-critical areas such as medical applications and the aerospace and automotive industries, the behavior of both simple and highly complex embedded systems must be exactly known. This is achieved by defining the workflow patterns of the individual subtasks, so-called scheduling. In many computer applications a dynamic scheduling of the processes is used. By contrast, systems operating in safety-critical areas execute defined and sequenced work steps, with execution being continuously repeated (see also [8], [9]). Often, the execution of a workflow pattern takes only a few seconds.

In this paper, we suppose a deterministic cycle scheduling of partitions at the operating system level and a given scheduling method among tasks within each partition. The tasks in one partition can only be executed during the fixed time slices allocated to this partition. When constructing such a scheduling of partitions, the execution sequence of the individual work packages is often defined manually. Here, developers soon encounter problems, given the very large number of possible variations and constraints that have to be taken into account. For instance, a specific sequence of work steps must be taken into consideration in the scheduling process. At the same time, a component such as a processor should, if possible, be able to execute a work step in one piece to avoid unnecessary switching overhead.

We developed a scheduling tool that generates the partitions schedules for such a multiprocessor system using constraint based programming methods. Here, all the constraints of these complex scheduling tasks are taken into account even before the actual systems control program - the scheduler - is configured.

The basic idea is to avoid conflicts and optimize schedules beforehand rather than troubleshooting after the event.

Our research is concerned with the development of methods, techniques and concepts for a combination of interactive and automatic scheduling. The automated solution search will be implemented in such a way that it normally allows a solution to be found in a relatively short time, if such a solution exists. The scheduling tool will be flexible enough to take into account special user requirements and to allow constraints to be modified easily, if no basic conceptual change in the problem specification is necessary. An essential component is the automatic heuristic solution search with an interactive user-intervention facility. The user will, however, only be able to alter a schedule such that no hard constraints are violated.

An exemplary application of a combined interactive and automatic partition scheduling system was developed. The first test phase has been successfully completed. The scheduling tool is designed to be user-friendly. Its graphical interface and combined automatic and interactive solution search component allow the quick generation of schedules, which can be individually tweaked within the given ranges.

2 Problem Description

In this section, we explain the used notions and describe the problem. It is assumed that a multiprocessor system and a set of applications are given. Each application consists of a set of tasks. Each application will be located in one partition on one processor. The dedicated processor may be statically configured or one part of the scheduling process.

A *time slice* is the non-preemptive (i.e. uninterrupted) usage of processor time, solely allocated for one partition. A partition consists of a set of time slices, which have to follow the given constraints and which in total will be repeated periodically. Note that we do not consider the operation or inherent schedule of the tasks within a partition. This is outside of the scope of this paper. A *period* is specified by the time distance between beginning of cycle N until

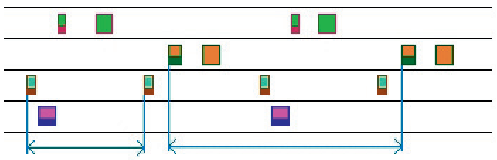

Fig. 1. periods of partitions

beginning of cycle N+1 of a partition allocation of the processor time. Since a period may have also other properties, we call this distance by *period length*, too. The *duration per period* is the duration (sum of time slices) the processor is allocated to the partition per period. A partition may get one or several time slice(s) per period. In Figure 1 the periods of two partitions are marked. One partition consists of one time slice per period while the other partition consists of two time slices per period.

Furthermore an overall time interval is given for which the partition scheduling will be generated. In the following this time interval is called *scheduling period* (also known as hyper period). The goal is the generation of a deterministic partition scheduling for this given scheduling period such that the given constraints described below are satisfied. This generated scheduling will be repeated continuously on the discrete processors.

For each partition there are constraints on the period and the duration per period. There may exist partitions which have to follow strong defined periods, so called *fixed periods*. Variations are not allowed in these cases. We distinguish between those and *flexible periods*, which allow certain variations. These flexible periods are very useful during solution search, in particular when the processor load reaches the limit of the period capacity.

A special focus must be held on the situation at the end of each period and the entry situation for the following period. All constraints must still be met in this intersection. For a coupled system some synchronization activities will take place and have to be considered. It should be aimed to allow larger variants in period and duration for these overall scheduling periods from one to its follower. These special aspects are not presented in depth in this paper.

If one application should be spread over different processors than in our system there should be used one copy of this application for each processor involved and there must be specifications of the relations of the processes.

Between two partitions on different processors there may be definitions of relations of various kinds. These relations always reference to the beginning of one period or the end of the last time slice belonging to that period. The pattern of these relations may be e.g.:

begin(...) $+X \leq$ begin(...)
end(...) $+X =$ begin(...)
begin(...) $+X \leq$ begin(...)
end(...) $>$ begin(...) $+X$
end(...) $+X <$ end(...)

A problem is specified by an amount of special definitions and constraints. The important components of a complete problem specification consist of:

1. a definition of the basic problem parameters:
   (a) the length of the scheduling period,
   (b) the basic time unit such that all time values are integers (for instance 1 *ms* or 1/10 *ms*),
   (c) the worst case waiting time at the end of a scheduling period (time, which may be necessary for the synchronization of loosely coupled processors);
2. a set of definitions for each processor:
   (a) the time for changing a partition on this processor,

   (b) the general time for writing data after the end of a time slice (communication activity);
3. a set of definitions and constraints for each partition (application):
   (a) the processor allocation or constraints for that (e.g.: not processor X; another processor than the processor dealing with partition Y)
   (b) the period length of the partition,
   (c) for the period length, the allowed difference is specified by its minimum and its maximum,
   (d) the duration of a period,
   (e) for the duration of a period, the allowed difference is specified by its minimum and its maximum,
   (f) the minimal CPU load (e.g. per mill) within the scheduling period,
   (g) the maximal number of time slices within a period,
   (h) the minimal duration of a time slice,
   (i) special constraints on the end of the scheduling period;
4. a set of definitions for the relations between the partitions; the relations can be of different kind and related to the begin and/or the periods of two partitions belonging to different processors.

3 Problem Modeling

The problem of generating partition scheduling for multiprocessor systems can be suitably modeled in terms of a set of constraints and a constraint based programming language can be used for solution search. Constraint Logic Programming with constraints over finite integer domains, CLP(FD), has been established as a practical tool for solving discrete combinatorial problems (e.g. [4], [10], [11]). The success of the search often depends directly on the chosen model and the modeling options on the chosen constraint solver. Global constraints use domain-specific knowledge and specialized consistency methods to obtain better propagations results. They can be applied to large problem instances and in general improve the efficiency for solving real-life problems. The global constraints built into the Constraint Logic Programming language CHIP are particularly useful for problem modeling. Examples of global constraints are `cumulative` and `diffn` (see e.g. [1], [2]). Note that global constraints which are similar to `diffn` exist also in other constraint based programming languages with other names.

The basic method for the problem representation by constraint programming is described in the following and refers to the problem specification given in Section 2. Note that the special constraints related to the end of the scheduling are not considered in this paper. Concerning the definition of the basic problem parameters it is supposed that a scheduling horizon (hyper period) is given and that all values are integers (see (1) of the specification). The given processors are numbered by natural numbers $1, 2, \ldots$. Let $A_1, \ldots, A_{n_A}$ be the given applications (we identify these also with the partitions they are belonging to). For each A_k we define a domain variable $proc(A_k)$ for the allocation of the processor. The domain of this variable is equal to the numbers of the allowed processors (see (3a) of the specification). If a processor allocation is given then $proc(A_k)$ is equal to the corresponding number of the processor.

A sequence of periods $p_1^k, p_2^k, \ldots, p_{m_k}^k$ is defined for each A_k . The length of the sequence depends on the scheduling period and the sum of the periods belonging to a partition. For each period p_i^k domain variables for the length of the period $l(p_i^k)$ and the duration of the period $d(p_i^k)$ are defined by the given values and the allowed differences (see (3b,c,d,e) of the specification). If differences of the period length and the duration per period are not allowed then $l(p_i^k)$ and $d(p_i^k)$ are integers.

Furthermore, a sequence of time slices $s_{i,1}^k, s_{i,2}^k, \ldots, s_{i,n_{k,i}}^k$ is defined for each period p_i^k . The length of such a sequence $n_{k,i}$ is given the maximal number of time within a period (see (3g) of the specification). For each time slice $s_{i,j}^k$ we define domain variables $start(s_{i,j}^k)$ for the starting time and $d(s_{i,j}^k)$ for duration of this slice. Firstly, the domain of $start(s_{i,j}^k)$ is given by the scheduling period (from 0 to Max, the scheduling period). Let min_s be the minimal duration of a time slice and max_p be the maximal duration of a period (see (3h,e) of the specification). The domain of $d(s_{i,j}^k)$ is defined by the union of $\{0\}$ and the interval $[min_s, max_p]$. A time slice $s_{i,j}^k$ is only relevant if $d(s_{i,j}^k)$ is different from 0 . Since one time slice of each period has to be relevant we can suppose that the first time slice of each period $d(s_{i,1}^k)$ is different from 0 . Furthermore, we can suppose that

$$start(s_{i,j}^k) \; + \; d(s_{i,j}^k) \; < \; start(s_{i,j+1}^k)$$

$$start(s_{i,1}^k) \; + \; d(p_i^k) \; \leq start(s_{i+1,1}^k)$$

for all the corresponding time slices and periods. Then the following equation is to be satisfied for the period length:

$$l(p_i^k) \; = \; start(s_{i+1,1}^k) \; - \; start(s_{i,1}^k)$$

The duration of a period $d(p_i^k)$ is equal to the sum of the durations of time slices within this period:

$$d(p_i^k) \; = \; d(s_{i,1}^k) \; + \; \ldots \; + \; d(s_{i,n_{k,i}}^k)$$

The minimal CPU load within the scheduling period (see (3f) of the specification) corresponds an integer $minD_k$. Then, for each partition A_k, this constraint can be modeled by the inequality

$$d(p_1^k) \; + \; \ldots \; + \; d(p_{m_k}^k) \; \geq \; minD_k$$

It is important to state by a constraint that all time slices must not overlap. Since the time for changing a partition has to be integrated into such a constraint we define the extended duration $d_1(s_{i,j}^k)$ of a relevant time slice by the sum of $d(s_{i,j}^k)$ and the time for changing a partition. If a processor allocation is not given and the time for changing is different on the processors then the symbolic `element`-constraint can used for computing this duration. In the case of a time slice with $d(s_{i,j}^k) \; = \; 0$, the extended duration $d_1(s_{i,j}^k)$ is also equal to 0 .

We consider a time slice as a "two-dimensional rectangle" with the dimensions *"time"* and *"processor"*. Such a rectangle can be represented by

$$[start(s^k_{i,j}), proc(A_k), d_1(s^k_{i,j}, 1]\,.$$

The use of the global `diffn`-constraint ensures that these rectangles must not overlap. For our problem we need only one `diffn`-constraint. Note that time slices with duration of 0 are not relevant for the `diffn`-constraint. If for each application the processor allocation is given and is fixed then a non-overlapping constraint can generated for each processor separately. In this case, the `diffn`-constraint with "one-dimensional rectangle" can be considered or the global constraint `cumulative`-constraint can be used with a resource limit of 1.

The relations between the periods of the partitions (see (4) of the specification) can be easily represented by arithmetic constraints (equalities, disequalities, inequalities). The time for writing data after the end of a time slice (see (2b) of the specification) has to be integrated into these constraints.

4 Solution Search

A solution of a constraint problem is an assignment of the domain variables to values of their domains such that all the constraints are satisfied. A constraint solver over finite domains is not complete because consistency is only proved locally. Thus, a search is generally necessary to find a solution. Often, this search is called "labeling". The basic idea behind this procedure is to select a variable from the set of problem variables considered, choose a value from the domain of this variable and then assign this value to the variable; if the constraint solver detects a contradiction backtracking is used to choose another value. This is repeat until values are assigned to all problem variables such that the constraints are satisfied or until it is proven that there is no solution. In our scheduling system, the domain-reducing strategy is also used for the search. This strategy is a generalization of the labeling method and was presented in [5]:

- The assignment of a value to the selected variable is replaced by a reduction of the domain of this variable.
- If backtracking occurs, the not yet considered part of the domain is taken as the new domain for a repeated application of this method.

Practical applications have shown that a reduced domain should be neither too small nor too large. A solution is narrowed down by this reduction procedure, but it does not normally generate a solution for the problem. Thus, after domain reduction, assignment of values to the variables must be performed, which may also include a search. The main part of the search, however, is carried out by the domain-reducing procedure. A conventional labeling algorithm can be used for the final value assignment. If a contradiction is detected during the final value assignment, the search backtracks into the reducing procedure.

Since a constraint solver is used for the partition scheduling tool the search space is reduced before the solution search begins. In each search step the search space is further reduced by the constraint solver. Nevertheless, in most cases, the search spaces of relevant problems are too large and it is not possible to use a complete solution search within an acceptable time amount. Therefore, the complete search spaces cannot be investigated and heuristics are needed for a successful solution search.

The search includes two kinds of nondeterminism: *selection of a domain variable* and *choice of a reduced domain* concerning the selected variable. If labeling is used, the reduced domain consists of a single value. The success of the domain-reducing strategy depends on the chosen heuristics for the order of variable selection and for the determination of the reduced domain.

Our experience has shown that in many cases either a solution can be found within only a few backtracking steps, or a large number of backtracking steps are needed. We therefore use the following basic search method: *the number of backtracking steps is restricted, and different heuristics are tried out*. This means that backtracking is carried out on different heuristics. With regard to the problems discussed in this paper, the user can choose between different methods for the solution search. In particular, the user can control the following parameters: the number of attempts with different heuristics, the number of permitted backtracking steps for one attempt, and the priorities of the partitions.

In the recent partition scheduling tool, a static ordering is used for the heuristic of variable selection. This ordering is defined by the priorities of the partitions and the following ordering of the relevant domain variables related to the selected partition A_k:

$$\begin{aligned}&proc(A_k),\ l(p_1^k),\ d(p_1^k),\\ &start(s_{1,1}^k),\ d(s_{1,1}^k),\ start(s_{1,2}^k),\ d(s_{1,2}^k),\ \ldots,\\ &l(p_2^k),\ d(p_2^k),\ start(s_{2,1}^k),\ d(s_{2,1}^k),\ \ldots\ .\end{aligned}$$

The domain-reducing strategy is used for the following domain variables: the length of a period $l(p_i^k)$, the duration of a period $d(p_i^k)$, and the starting time of a time slice $start(s_{i,j}^k)$. The used search strategy prefers the allocation of the domain maximum to the domain variable $d(s_{i,j}^k)$ (duration of a time slice). The goal of this strategy is to minimize the number of time slices of a period. Thus the switching overhead can be reduced.

The following properties should be taken into consideration for the determination of priorities of the partitions:

– the allowed difference of the period length (partitions with fixed periods should be scheduled firstly),
– the number of allowed time slices per period (if this number is equal to 1 then this partition should be scheduled earlier);
– the durations per period;
– the desired period length;
– the relations between partitions.

If a domain variable of period length is selected then the heuristics for the choice of a reduced domain or a value can be controlled by a parameter such that one of the following heuristic is used:

- the given value of the period length is preferred,
- the minimum of the domain value is preferred,
- the maximum of the domain value is preferred.

Furthermore, there is a parameter such that the choice of a reduced domain (or a value) for the starting time variable of a time slice can be controlled by the parameter values: minimum, maximum, middle. Additionally, there is a heuristic for minimizing the number of time slices per period.

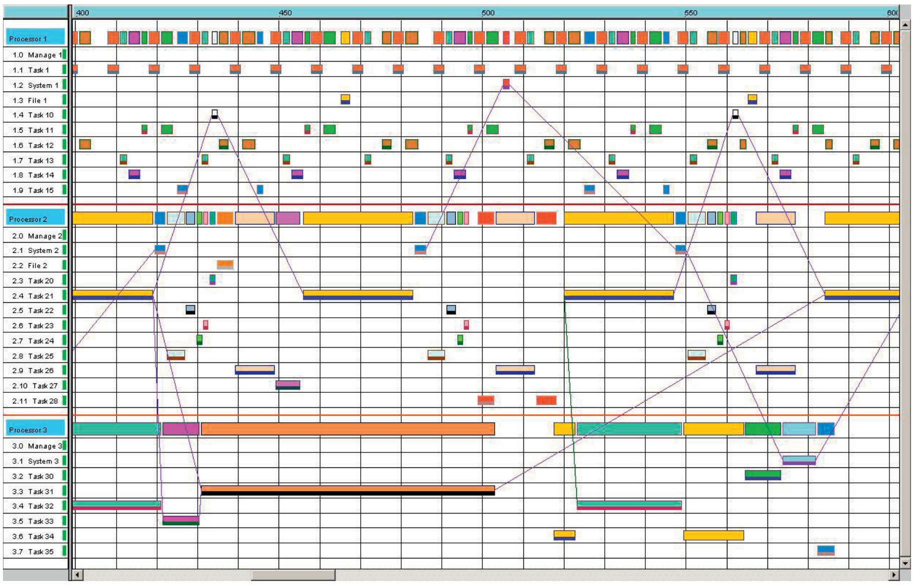

Fig. 2. Example of a partition scheduling, excerpt of approx. 200 ms

5 Graphical Interface

The scheduling tool is designed to be user-friendly by its graphical interface. The generated schedule can be graphically displayed in a clear form with a flexible layout and a user-defined level of detail. Figure 2 shows a part of an exemplary partition schedule for three processors. This partition schedule is generated by our tool and displayed by the graphical interface. For each application the partition schedule is represented in one row of this figure. Moreover, for each processor, all partitions of the processor are graphical represented in one row. The relations between two partitions are marked by lines. The following interactive-scheduling actions are possible with the help of the graphical interface:

- scheduling an individual partition,
- scheduling marked partitions automatically,
- removing marked partitions from the schedule,
- moving time slices of a partition within the schedule,
- scheduling the remaining partitions automatically.

These actions can be performed in any order or combination, and no automatic backtracking is caused by such user actions. The user can, however, only alter a schedule in such as way that no hard constraints are violated.

The user interface and the combined automatic and interactive solution search component allow the quick generation of schedules, which can be individually tweaked. For instance, different scenarios can be easily tried out and changes swiftly implemented, consistency with respect to the specifications being guaranteed at all times. The targeted development of variants enables the solution process to be made far more flexible and efficient, even when adopting an iterative approach and when the initial tolerance limits are exceeded. With the help of the graphical interface the user can interactively generate and evaluate different variants, depending on the optimization criterion. This enables the user to incorporate his expertise.

6 Implementation and Results

The Constraint Logic Programming language CHIP ([3]) was selected as the implementation language. The global constraints and the object oriented component built into CHIP are particularly useful for problem modeling.

For the representation of the problem, we used three phases: the definition of the problem, the internal relational representation, and the internal object oriented representation. For the first phase, the definition of the problem, we developed a declarative language for problem descriptions. All components of a partition scheduling problem can be easily defined using this declarative language. Thus, the graphical user interface is only needed to set the parameters. In the second phase, the problem definition is transformed into an internal relational representation. In the third phase, the internal object-oriented representation is generated from the internal relational representation. The definition data are converted into a structure that is suitable for finite integer domains. The object-oriented representation is used for the solution search and the graphical user interface.

Our scheduling tool can generate in less than a minute a consistent schedule for complex multiprocessor systems with many thousands of time slices for an arbitrary interval. Additionally, the generated schedule is guaranteed to be error-free and executable. Even extreme optimizations are properly manageable because it is possible to generate schedules that allow over 90 per cent CPU load. In addition, the results of the scheduling process can easily be converted into other formats, enabling them to be integrated into the overall system development process. It should be noted that the currently implemented version of our partition scheduling tool supposes that an allocation of the partitions to the processors is given.

7 Conclusions and Future Work

The initial application of our partition scheduling system has been proved successful and has demonstrated the suitability of the used methods. From this application, we were able to obtain useful information for our future work. Our future research on partition scheduling problems will include investigations of heuristics for variable and value selection and continued study of the influence of different modeling techniques on the solution search. Furthermore we will extend our implementation of a partition scheduling system such that scheduling process can also allocate partitions to processors. The development of special search methods is necessary for this goal. Moreover, a graphical interface for the problem specification will be implemented. The methods, techniques and concepts developed or under development will also be tested on other applications.

References

1. A. Aggoun and N. Beldiceanu,"Extending CHIP in order to solve complex scheduling and placement problems", *Math. Comput. Modelling*, 17(7):57–73, 1993.
2. E. Beldiceanu and E. Contejean, "Introducing global constraints in CHIP", *J. Mathematical and Computer Modelling*, 20(12):97–123, 1994.
3. M. Dincbas, P. van Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier, "The constraint logic programming language CHIP", in *Int. Conf. Fifth Generation Computer Systems (FGCS'88)*, pages 693–702, Tokyo, 1988.
4. M. Dincbas, H. Simonis, and P. van Hentenryck, "Solving large combinatorial problems in logic programming", *J. Logic Programming*, 8:75–93, 1990.
5. H.-J. Goltz, "Reducing domains for search in CLP(FD) and its application to job-shop scheduling", in U. Montanari and F. Rossi, editors, *Principles and Practice of Constraint Programming – CP'95*, volume 976 of *Lecture Notes in Computer Science*, pages 549–562, Springer-Verlag, 1995.
6. J. Jaffar and M. J. Maher, "Constraint logic programming: A survey", *J. Logic Programming*, 19/20:503–581, 1994.
7. K. Marriott and P. J. Stuckey, "Programming with Constraints: An Introduction", The MIT Press, Cambridge (MA), London, 1998.
8. Y. Lee, D. Kim, M. Younis, and J. Zhou, "Partition Scheduling in APEX Runtime Environment for Embedded Avionics Software", in *Proc. IEEE Real-Time Computing Systems and Applications*, pages 103109, Oct. 1998.
9. Y. Lee, D. Kim, M. Younis, and J. Zhou, "Scheduling Tool and Algorithm for Integrated Modular Avionics Systems", in *Proc. Digital Avonics Systems Conference (DASC)*, Oct. 2000.
10. P. J. Stuckey (editor), "Principles and Practice of Constraint Programming – CP 2008", volume 5202 of *Lecture Notes in Computer Science*, Springer-Verlag, 2008.
11. M. Wallace, "Practical Applications of Contraint Programming", *Constraints, An International Journal*, 1:139–168, 1996.