PREDICTIVE SCHEDULING OF JOB COMBINATIONS IN SDR SYSTEMS

David Guevorkian (Department of Signal Processing, Faculty of Computing and Electrical Engineering, Tampere University of Technology, Tampere, Finland; e-mail: david.guevorkian@tut.fi) and Jan Westmeijer (mimoOn GmbH, Duisburg, Germany; e-mail: jan.westmeijer@mimoon.de)

ABSTRACT

In SDR systems, different combinations of radios each consisting of a number of algorithms and having strict timing constraints must be implemented on top of a shared computational platform typically consisting of several processors and HW accelerators. To handle such complicated system, efficient scheduler design policies are needed. In this work, we propose a scheduling policy where during each established state where a fixed job combination (fixed set of active radios) is executed, new schedules are designed and stored for all possible states that may occur after the current state. Effectively, this means that static, highly optimized schedules are designed dynamically for the sequence of stochastically changing job combinations. This way, advantages of the static and dynamic scheduling policies are combined. The proposed method can be applied to scheduler design for any "piece-wise stationary" application where relatively small number of stationary jobs must be supported but the sequence of job combinations is unpredictable. Therefore, the system is stationary for a period of time when combination of jobs is fixed but is non-stationary during larger time periods when job combinations may stochastically change.

1. INTRODUCTION

Software Defined Radio (SDR) is in the focus of research community during last decade since it may provide new possibilities in Communication Technologies (see [1]-[5]). The computational platform of an SDR system typically involves several processors and HW accelerators that are shared between different radios [1] – [5]. Each radio, hereafter called job, consists of a number of algorithms, hereafter called tasks, and must be implemented under very hard real time constraints. To provide the required functionality where each task meets its deadline and to achieve efficient sharing of HW resources between the jobs, the system should use a sophisticated scheduler. Hence, creating efficient scheduling policies is an important and difficult stage in designing of SDR systems.

Scheduler creation involves processor assignment, decision on the order of task execution, and decision on the firing (execution) times of each task (see [6]). In conventional scheduler design strategies each of these steps may be implemented either statically (at compile time) or dynamically (at run time) depending on the features of the application [6], [7]. For static applications with fixed order of tasks having fixed execution times, all three steps may be implemented statically at compile time whereas for applications with unpredictable order of tasks and their execution times some or all these steps may only be implemented dynamically in run time.

Specific to SDR systems is that they involve relatively small number of jobs (three to ten different radios) and each job is more or less static with a nearly fixed order of algorithms having predictable worst-case execution times [1] - [5]. However, the overall system is far not static due to unpredictable (stochastic) sequence of switching radios On/Off and changing their modes. We call such applications with relatively long periods of static states but with unpredictable changes between these states "piece-wise static" applications. Another specific to SDR systems is that the scheduler for a set of active radios should be designed in a way such that the radios from this set that were active also at previous step keep running smoothly. Therefore, the scheduler should also take into account the history of arriving to the current set of active radios. Yet another specific feature of SDR systems is very short time periods where different radio algorithms should be completed. As discussed in the next section, these mentioned specifics limit the use of conventional schedulers for SDR systems.

In this work, we propose new scheduler design policy that combines advantages of static and dynamic scheduling policies by dynamically creating static schedules for each combination of active radios that may occur after the moment of scheduler creation. That is, at each static state in parallel to implementing active radios corresponding to that state, schedulers for all possible sets of active radios that may occur as the result of any possible change are created

Static assignment: compile time decision on processor assignment; run-time decision on task orderinfg and on firing instances.

Quasi-static: compile time decision on processor assignment and partially on task ordering and firing instances; partially run-time decision on task ordering and firing instances (for data dependent tasks)

Self-timed: compile time decision on processor allocation (but not on processor communication) and on task ordering; run-time decision on task firing instances.

Ordered transaction: compile time decision on processor assignment, and processor communication as well as on task ordering; run-time decision on task firing instances.

Fully static: compile time decision on processor assignment, task ordering and task firing times.

Figure 1. Classification of conventional schedulers according to decision time of three main scheduling steps: processor assignment, task ordering, and task firing.

and stored. Once a change happens, the corresponding scheduler is readily available to use. The proposed scheduler design approach can be extended to any piece-wise stationary application/system.

The rest of the paper is organized as follows. Section 2 presents a short background on schedulers from static versus dynamic point of view and with respect to suitability to SDR systems. In Section 3, formal description of the proposed method is presented. Section 4 is a discussion on advantages and drawbacks of the proposed approach. Finally, Section 5 is the conclusion.

2. BACKGROUND

In a real-time computing platform where several jobs, each consisting of several tasks and having own time constraints, in particular, in an SDR system, a scheduler is used to define the processing units where each task is to be implemented, as well as the timed sequence according to which these tasks are to be implemented. Efficiency and even applicability of different scheduling policies depend on the requirements of the set of jobs that are supposed to be implemented by the platform. Scheduler creation involves three main steps (see [6]):

1. processor assignment where a decision is made on allocating tasks to processors or hardware accelerators;
2. decision on the order of execution of tasks on each processor or hardware accelerator;
3. decision on the firing (or, equivalently, execution) times of each actor (task).

Conventionally, it is assumed that each of these three steps may be performed either at run-time (a dynamic strategy) or at compile time (static strategy) [6]. Thus scheduling policy implies not only the rule according to which tasks are assigned to processors, ordered or fired but also the relative time (either run-time or compile time) when these steps are performed. Classification of schedulers according to decision times of the above mentioned three scheduling steps is illustrated on Fig. 1. Applicability area of each type of schedulers is restricted to own set of applications. A conventional rule of thumb is that more dynamic schedulers are more general. That is, those schedulers where more of the above three steps are performed on run-time are having larger area of applicability (see Fig. 3).

For example, fully static schedulers may only be applied to jobs with fully stationary behavior or to fixed combinations of such jobs. Most general schedulers are fully dynamic ones.

Another conventional rule of thumb is that more dynamic is the scheduler less it is efficient because of two reasons. Firstly, in the dynamic scheduler the order of tasks or at least their firing times are unknown. Therefore, the

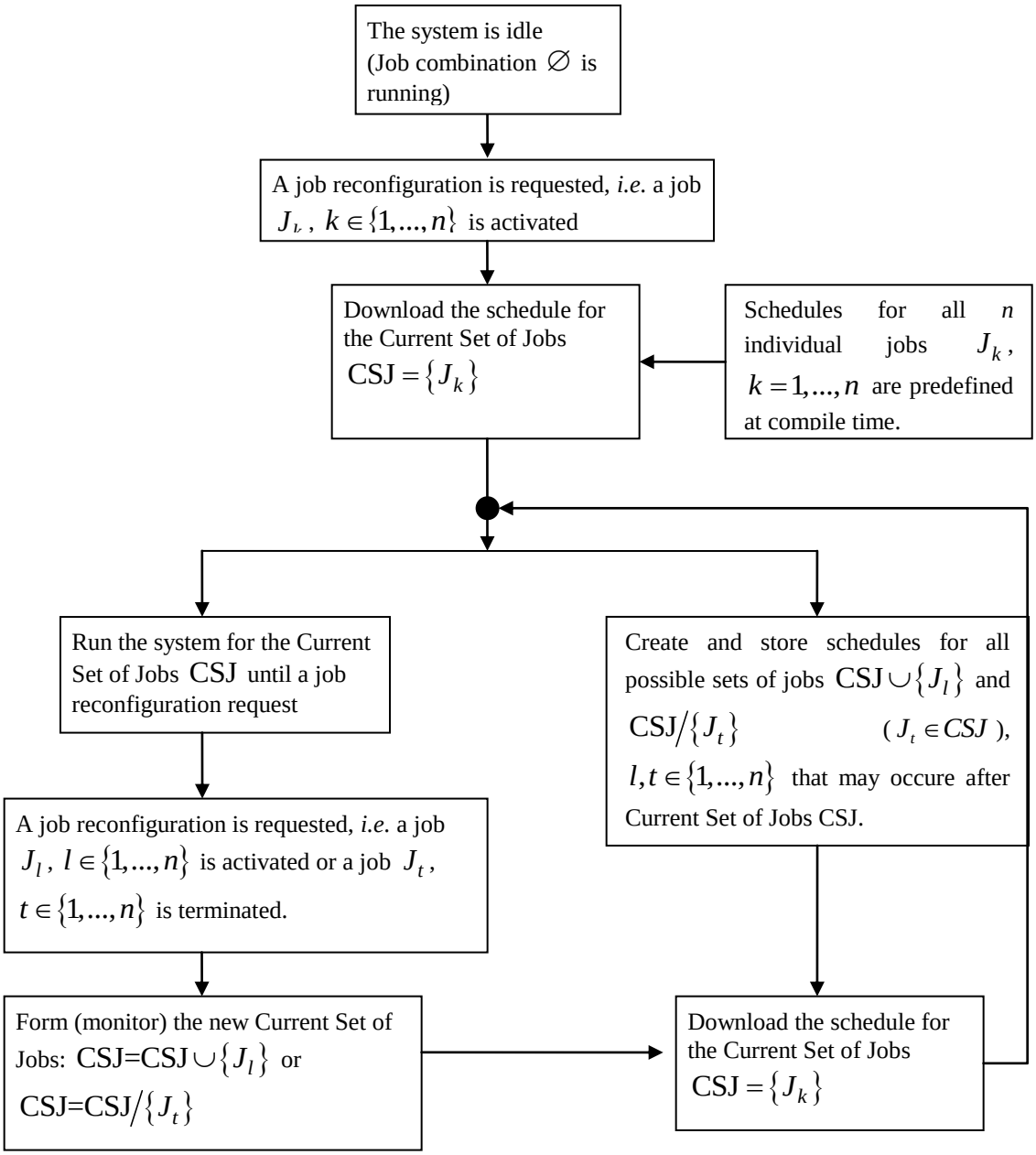

Figure 2. The block-diagram of the proposed predictive scheduling method for piece-wise stationary applications.

actual implementation of the task is delayed since the scheduler makes the decision only after the moment when the task needs be implemented. More importantly, some time is needed to call or download the programs executing the corresponding task. Secondly, as the decisions are made on run-time, real time constraints to the scheduling algorithm imply no or low optimization of the decisions. This is especially critical for SDR systems where execution times of individual tasks are very short and real-time constraints are very hard.

No a prior art scheduling policy is devoted to applications where each job itself is a stationary one but their combinations are dynamically changed in time. We shall call "piece-wise stationary" those applications where several stationary jobs may be initiated or terminated at arbitrary time instances.

Typical piece-wise stationary application is Software Defined Radio (SDR) systems where each radio standard is more or less a stationary job with a fixed sequence of tasks having predictable worst-case execution times but every radio may turn On/Off, or change its mode at an arbitrary time instance.

In the next section we propose new approach to scheduler design timing for piece-wise stationary systems. In the proposed approach during each established stationary state of the system schedules are created for all possible t which the system may arrive next.

3. PREDICTIVE SCHEDULING

The proposed predictive scheduling method for "piece-wise stationary systems" may be described using the block-diagram presented on Fig. 2.

Suppose the system is piece-wise stationary and is meant to support a set of jobs $J_1,...,J_n$. That is, each job J_k, $k=1,...,n$ is a stationary job which may externally be requested to start or to terminate at an arbitrary unpredictable time instance. At compile time only schedules for all individual jobs J_k, $k=1,...,n$ are created and are stored in a devoted memory space in the system. Any valid scheduling algorithm may be used to create the actual schedules. In an exemplary embodiment this may be the algorithm described in [7]. When the system is switch on due to a request to start a job (say, J_k), the corresponding stored schedule is used and the platform executes that job J_k. At this moment the Current Set of Jobs (CSJ) is $CSJ=\{J_k\}$. Later the CSJ is dynamically changed during operation of the system. Every time when CSJ is changed, new schedules for all possible sets of jobs that may occur after the CSJ are created. Creation of schedules may be considered as one of the jobs from the set of all supported jobs $J_1,...,J_n$. That is, schedules for $\mathrm{CSJ}\cup\{J_l\}$, and $\mathrm{CSJ}/\{J_t\}$ ($J_t\in\mathrm{CSJ}$), $l,t\in\{1,...,n\}$ that may occur after the CSJ are created (note that this way several instances of the same job are allowed, in the case that letting only one instance of each job (as in SDR) makes sense, we should restrict $J_l\notin\mathrm{CSJ}$). Again, any valid scheduling algorithm, for example the one described in [7], may be used to create the actual schedules. However, the schedules may then be optimized using one or another optimization technique. Created schedules are stored into the devoted memory space. Note that the maximum number of schedules that need be stored is $2n$, which is significantly far less than 2^{2^n}, the number of schedules that need be stored in the case where all possible job combination transactions are scheduled at compile time. When all the (optimized) schedules are created and stored the system only runs the CSJ. As soon as a request for changing the CSJ is received the corresponding schedule is downloaded from the devoted memory location and the system runs the new CSJ. In an unlikely case that the CSJ change request is received earlier than all the schedules are created the scheduler terminates creation of all possible schedules for $\mathrm{CSJ}\cup\{J_l\}$, and $\mathrm{CSJ}/\{J_t\}$ ($J_t\in\mathrm{CSJ}$), $l,t\in\{1,...,n\}$ but only creates or downloads the created schedule for the requested new CSJ.

In the proposed approach, the scheduling overhead is minimal because the schedules need not be created at run-time as well as because the real-time constraints on creating schedules is significantly relaxed and more sophisticated scheduling policies and optimization techniques may be utilized .

.

4. DISCUSSION

In Fig. 3 efficiencies of different scheduling policies classified according to relative time (either run-time or compile time) of performing the three main scheduling steps are summarized similarly as it was done in [6].

The efficiency in this figure is understood as the generality of the scheduling policy vs. the run-time overhead of that policy. It can be seen that the overhead is smaller for more static schedulers. This means that for a given application it is advantageous to use as more static scheduling policy as is possible for that application. Unfortunately, the applicability area (generality) of static schedulers is rather limited.

For example, an SDR system can only be scheduled using quasi-static or more dynamic scheduling approaches [7], which, however, require rather significant run-time overhead. In addition, only very simple scheduling algorithms may be used due to very tight real-time constraints on schedule creation.

In Fig. 3, also the proposed predictive scheduling method is positioned. Since the schedules for next sets of jobs are created in parallel with executing current set of jobs, the proposed method implies very low run-.time overhead, comparable with that of fully static scheduling policy. On the other hand, it has larger applicability area covering, for example the SDR application since the used schedules are dynamically changed as the system state is changed. The generality of the proposed method is at least comparable with that of quasi-static scheduling approach. In addition, since the real-time constraint in creating the schedules is significantly relaxed (compared to the case of quasi-static approach) rather sophisticated scheduling algorithms involving optimization techniques may be utilized.

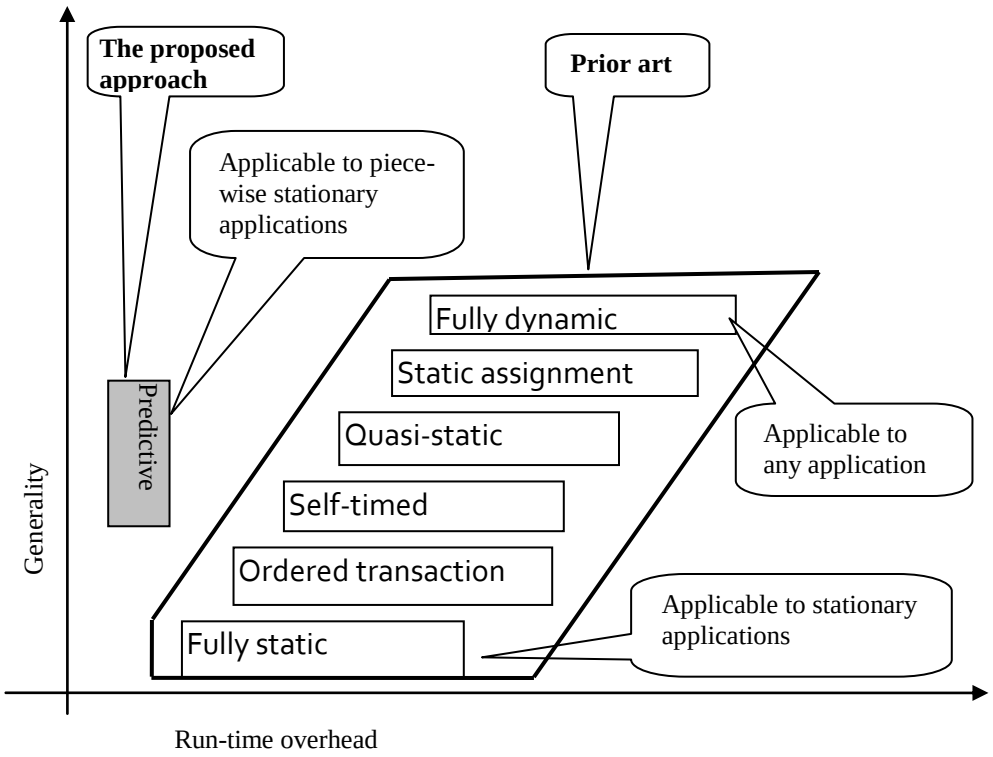

Figure 3. Efficiency (run-time overhead vs. generality) of scheduling methods depending on the time of making decisions on three main scheduling steps.

Therefore, the proposed predictive scheduling method provides high scheduling efficiency. The cost for achieving this efficiency is a devoted memory space where the schedules are to be stored and, optionally a dedicated small functional unit or a share of the system's CPU. However, these costs are rather small.

4. CONCLUSION

New predictive scheduling policy is proposed that dynamically creates static schedules every time when system's steady state is changed. The proposed method suits well to SDR systems since it requires minimal overhead andf allows dynamic creation of optimized schedulers with respect to the history of system evolution.

7. REFERENCES

[1] A. Ahtiainen, H. Berg, U. Lücking. A. Pärssinen. and J. Westmeijer, "Architecting Software Radio," *Proceeidngs of the SDR 07 Technical Conference and product Exposition*, 2007.

[2] A. Ahtiainen, K. van Berkel, D. van Kampen, O. Moreira, A. Piipponen, T. Zetterman, " Multi-radio Scheduling and Resource Sharing on a Software Defined Radio Computing Platform," *Proceeidngs of the SDR 08 Technical Conference and product Exposition*, 2008.

[3] L. Harju and J. Nurmi, "Hardware platform for software-defined WCDMA/OFDM baseband receiver implementation," *IET Comput. Digit. Tec.*, Vol.. No 5, pp. 640-652, 2007.

[4] F. Kasperski, O.Pierrelee, F. Dotto, M. Sarlotte, "High data rate fully flexible SDR modem advanced configurable architecture & development methodology," *Proceedings of Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE '09*, pp. 1040-1044, 2009.

[5] H. Berg, C. Brunelli and U. Lücking, "Analyzing models of computation for software defined radio applications," *Proceedings of International Symposium on System-on-Chip, 2008 (SOC-2008)*, pp. 1-4, 2008.

[6] S. Sriram, Sh. S. Bhattacharyya, *Embedded Multiprocessors. Scheduling and Synchronization.* Signal Processing and Communication Series, Marcel Dekker , Inc., 2000, 327.

[7] O. Moreira, F. Valente, M. Bekooij, "Scheduling multiple independent hard-real-time jobs on a heterogeneous multiprocessor," *Proceedings of the 7th ACM & IEEE international conference on Embedded software. 2007*, pp. 57-66. 2007.