Constant-bandwith supply for priority processing

Constant-Bandwidth Supply for Priority Processing

Martijn M.H.P. van den Heuvel, Mike Holenderski, Reinder J. Bril and Johan J. Lukkien
Eindhoven University of Technology
Eindhoven, The Netherlands

***Abstract*—Today's consumer electronic devices feature multiple applications which have to share scarcely available resources. We consider a priority-processing-based video application, which comprises multiple scalable video algorithms (SVAs) that are executed on a shared, virtual platform. This application is given a guaranteed processor share by means of a constant-bandwidth server (CBS), which in addition efficiently reclaims all spare processor time. A decision scheduler distributes the assigned processor share among the SVAs, with the aim to maximize their overall output quality. To correctly distribute this processor share we introduce the concept of a virtual timer. This timer only advances when its associated virtual platform is executing.**

I. Introduction

The principle of priority processing provides optimal real-time performance for scalable video algorithms (SVAs) on programmable platforms with limited system resources [1]. According to this principle, SVAs provide their output strictly periodically and processing of images follows a priority order. After creation of an initial output by a basic function, processing can be terminated at an arbitrary moment in time, yielding the best output for given resources.

To distribute the available resources, i.e. processor time, among competing, independent priority-processing algorithms, a decision scheduler (DS) has been developed. The DS aims at maximizing the total progress of the algorithms on a frame basis. It therefore divides the available resources within a period into fixed-sized quanta, termed *time slots*, and dynamically allocates these time slots to the algorithms. Strategies and mechanisms for dynamic resource allocation have been addressed in [2]. In this paper we map a priority-processing application on an embedded, reservation-based platform [3].

A reservation-based approach makes it possible to concurrently schedule multiple applications with different timing constraints, and guarantee their resources. We inherently have a three-level scheduled system, i.e. a global scheduler to assign a reservation to the processor, a fixed-priority scheduler for tasks, and a DS for the SVAs. We extended a commercial real-time operating system, μC/OS-II, with constant-bandwidth servers (CBS) [4] to guarantee a minimum processor share.

II. Related Work

Virtualization techniques in which a guest operating system is hosted by a hypervisor or micro-kernel have become widely adopted in embedded systems to decouple a system in composable components. However, virtualization has shown to give considerable overheads [5], i.e. increased latencies with an order of magnitude. We therefore extended μC/OS-II with light-weight virtualization mechanisms based on the CBS [4]. The CBS guarantees a fraction of the processor time to applications whose computation time cannot be easily bounded. It is scheduled based on its deadline and automatically reclaims any spare time, i.e. a CBS immediately replenishes its budget upon exhaustion and postpones its deadline.

III. Mapping an Application on a Virtual Platform

We attach a CBS to a priority-processing application. This application consists of a DS and at least two independent SVAs. The DS is mapped on a task and is assigned the highest priority, so that upon activation it can immediately preempt the SVAs. These SVAs are each mapped on a strictly periodic task. All tasks comprising the priority-processing application are assigned to the same server and consume processor time *relative to the server's budget*. The consumed time is accounted to (and subtracted from) that budget. All SVAs are synchronous with the same period, P_f, i.e. each period the SVAs start with a new video frame and at the end of a period the processing is terminated. The SVAs are not blocked by their input and output and share no resources except the processor.

A server has a replenishment period, P_b, and a budget, Q_b. Activation of the DS, i.e. a virtually timed event, is triggered after consumption of a time slot, Δt_s, relative to the budget Q_b. For example see Figure 1 where the video-frame period $P_f = 20ms$, and the application is provided with a budget $Q_b = 5.5ms$ every period $P_b = 10ms$, and $\Delta t_s = 1ms$.

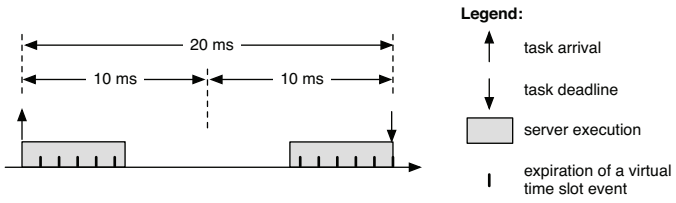

Fig. 1. Example of budget replenishments and virtual events, with $P_f = 20ms$, $P_b = 10ms$, $Q_b = 5.5ms$, and $\Delta t_s = 1ms$.

We need *virtual timers* to trigger timed events *relative to the consumed budget* to activate the DS. We therefore extended the work in [6] to implement such support.

IV. Virtual Platform Implementation

Intrinsic to our virtual platform are relative timed-event queues (RELTEQs) [6]. A *system queue* tracks all server events. Each server has its own *local queue* to track its tasks' events, e.g. task arrivals. When a server is suspended its local

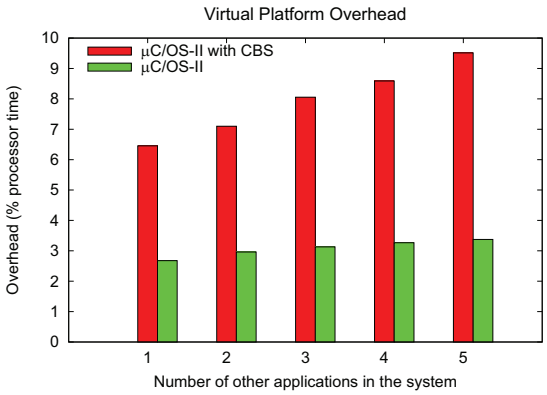

Fig. 2. Operating system overheads for timer-interrupt handling and scheduling of a defacto μC/OS-II versus a hierarchically scheduled system. The timer interrupt fires at 1 kHz on a 100 MHz OpenRISC architecture.

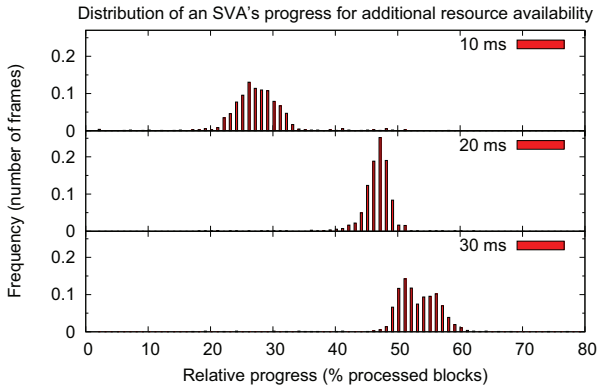

Fig. 3. Distribution of an SVA's additional progress for 10, 20 or 30 ms more processing time per frame. Measurement data is obtained from an application comprising two independent SVAs, each deinterlacing a video stream.

queues are deactivated for efficiency reasons, and when it is resumed its local queues are synchronized with global time. Since an inactive server does not consume any of its budget, a dedicated server queue managing virtually timed events is not synchronized when a server is resumed. Given this timed-event support, we built a three-level hierarchical scheduling framework within μC/OS-II:

1) Earliest deadline first (EDF): At the system level a RELTEQ is introduced to keep track of server deadlines. When the scheduler is called, it traverses this queue and activates the earliest *ready* server, i.e. according to the EDF policy.

2) Fixed-priority scheduling: After allocating a server to the processor, μC/OS-II's fixed-priority task scheduler determines the highest priority ready task within the server.

3) Decision scheduler (DS): The DS selects an SVA to execute for the duration of a given time slot. It accordingly manipulates task priorities [2].

The CBS builds on top of the priority-based EDF scheduler. A CBS has no periodic replenishment, but instead immediately replenishes its budget upon exhaustion and postpones its deadline by its period, P_b. A task within the CBS remains eligible for execution when the server's deadline is postponed albeit at a lower priority. Postponing a server's deadline is implemented by updating RELTEQ's deadline events. Budget replenishment is implemented using a virtual-timed event. When the virtual event expires the server's budget is depleted.

V. Evaluation

We created a port for μC/OS-II to the OpenRISC platform. First, we show the overhead imposed by our virtual-platform support, compared to a defacto fixed-priority μC/OS-II system. Secondly, we show the effect of the CBS' processor reclaiming mechanism on the priority-processing application.

1) Virtual platform overhead: We introduce an interfering application which is provided a hard reservation [3], i.e. it periodically consumes a fixed fraction of the processor resources. A single CBS is given a guaranteed share of 25% of the processor. We now increase the number of interfering applications, while keeping the system fully loaded, i.e. the sum of all evenly distributed reservations utilizes the entire processor. Figure 2 shows the overheads for our extended μC/OS-II compared to a fixed-priority task-scheduled system.

2) Reclaiming unused processor reservations: Since priority processing has a greedy nature, it exploits all unused processor time that is reclaimed by the CBS. A guaranteed budget of 10 ms every 40 ms is sufficient to complete an SVA's mandatory processing part. Figure 3 shows that an SVA reaches a higher progress value when given more processing time per frame, i.e. the processor time is not fully reserved.

VI. Conclusion

Virtual platforms become indispensable to guarantee secure and predictable behaviour in today's heavily loaded consumer platforms. We presented light-weight architectural support to (*i*) temporally isolate priority-processing applications by means of a CBS and (*ii*) efficiently distribute the available (spare) processor resources by means of a virtually timed DS. Our design is implemented in μC/OS-II, and evaluated on the OpenRISC platform. We showed that its overhead are affordable compared to hypervisor-based virtualization techniques [5]. These properties make virtualization a promising solution for future consumer electronics.

References

[1] C. Hentschel and S. Schiemenz, "Priority-processing for optimized real-time performance with limited processing resources," in *Int. Conf. on Consumer Electronics*, Jan. 2008.

[2] M. M. H. P. van den Heuvel, R. J. Bril, S. Schiemenz, and C. Hentschel, "Dynamic resource allocation for real-time priority processing applications," *Trans. on Consumer Electronics*, vol. 56, no. 2, May 2010.

[3] R. Rajkumar, K. Juvva, A. Molano, and S. Oikawa, "Resource kernels: A resource-centric approach to real-time and multimedia systems," in *Conf. on Multimedia Computing and Networking*, Jan. 1998, pp. 150–164.

[4] L. Abeni and G. Buttazzo, "Integrating multimedia applications in hard real-time systems," in *Real-Time Systems Symp.*, Dec. 1998, pp. 4–13.

[5] F. Armand and M. Gien, "A practical look at micro-kernels and virtual machine monitors," in *Consumer Communications and Networking Conf.*, Jan. 2009, pp. 1 –7.

[6] M. Holenderski, W. Cools, R. J. Bril, and J. J. Lukkien, "Multiplexing real-time timed events," in *Conf. on Emerging Technologies and Factory Automation*, July 2009.