Reducing Memory Requirements in a Multimedia Streaming Application

Mike Holenderski, Reinder J. Bril, Johan J. Lukkien
Eindhoven University of Technology,
Den Dolech 2, 5600 AZ Eindhoven, The Netherlands

***Abstract*—This paper investigates memory management for real-time multimedia applications running on a resource-constrained platform. It is shown how a shared memory pool can reduce the total memory requirements of an application comprised of a data-driven chain of tasks with a time-driven head and tail and a bounded end-to-end latency. The general technique targeted at memory-constrained streaming systems is demonstrated with a video encoding example, showing memory savings of about 19%.**

I. Introduction

Multimedia applications are known to be data intensive. Many of these applications are implemented on resource-constrained embedded systems where the memory space is scarce [1].

We consider multimedia streaming applications which are implemented as a chain of data-driven tasks, with a time-driven (i.e. periodic) head and tail task. One such application is a video encoder with a time-driven video digitizer and renderer at the head and tail (see Section IV).

An application consists of tasks which communicate via bounded buffers. Task execution is determined by priority, data availability, buffer sizes and time triggering at the boundaries of the system, however, we assume that the end-to-end latency of the complete chain is bounded. Task execution times may vary and depend on the data they process. Figure 1 shows an example of such a system.

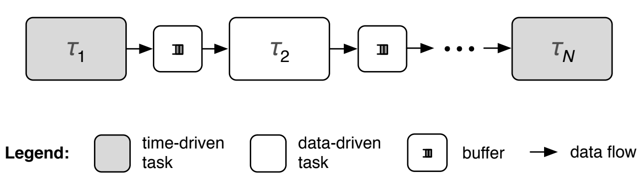

Fig. 1: A linear chain of media processing tasks communicating via shared buffers.

Weffers-Albu [2] explore how the assignment of task priorities and buffer capacities impact the behavior of multimedia applications composed of a linear chain of tasks. Let the first and last task in the chain be periodic with period T, with all other tasks being data-driven. The execution time of one iteration through the chain (i.e. for processing a single frame) may vary, assuming that processing a window of M consecutive frames is bounded by $M * T$. It can be shown that meeting the real-time constraints of the last task in the chain requires the first and last buffers in the chain to have capacity for $M + 1$ and M frames, respectively, with all other buffers having capacity for 1 frame.

This work has been supported in part by Information Technology for European Advancement (ITEA2), via the Content Aware Networked Systems Towards Advanced and Tailored Assistance (CANTATA) project.

Contributions

In this paper we introduce the concept of a *shared memory pool*, which encapsulates the memory shared between buffers in an application. We exploit the fact that in the above scenario the total number of frames in transit never exceeds $M + 1$, and propose to share a memory pool with capacity for $M + 1$ frames between all the buffers. As a result, in an application consisting of a chain of N tasks, we can save memory for storing $M + N - 3$ frames. We evaluate the memory savings in a real application by means of a H.264 video encoder.

II. System model

Below we describe our application and platform models.

A. Application model

An application consists of a chain of N tasks communicating via $N - 1$ shared buffers. The first and the last tasks in the chain are time driven. In this paper we do not consider variations in the number of packets produced or consumed by a component, and therefore assume the head and tail tasks share the same period T. All other tasks in the chain are data driven. Tasks use components to do their work. A component encapsulates a data structure with accompanying interface methods for modifying it and communicating with the system and other components. It can be regarded as a logical resource shared between different tasks. Buffer components are responsible for the majority of memory requirements of an application.

An application expresses its real-time requirements in terms of a minimum and maximum bound on the interarrival time between consecutive outputs generated by the tail task in the chain.

B. Platform model

We assume that memory is managed in terms of fixed-sized blocks. A component expresses its memory requirements in terms of *memory reservations* (or *memory budgets*) [3], where each reservation guarantees access to the requested number of blocks. Memory reservations are granted to components only if there is enough space in the system-wide memory pool, and memory allocations are granted only if there is enough space within the corresponding reservation.

Several components may request memory from the same reservation, giving rise to a *shared memory pool*. The memory pool guarantees that the *cumulative* requirement of all components using it does not exceed its capacity.

III. Reducing memory requirements

The total capacity of all buffers in an application consisting of a chain of N tasks, as shown in Figure 1, is equal to $M + (N - 3) + (M + 1)$ frames, where M represents the first buffer, $(N - 3)$ represents the buffers with capacity 1 between the first and last buffer, and $(M + 1)$ represents the last buffer. However, it can be shown that the total number of frames in transit never exceeds $M + 1$ frames.

Rather than allocating each buffer its required capacity, we can have them share a common memory pool, since all buffers together will never require more memory than for storing $M + 1$ frames. In this way can save the memory for storing $M + N - 3$ frames, for $3 \leq N$.

At different stages of the task chain frames may have different sizes (e.g. raw video frames are likely to be larger than the encoded frames). Let s_i be the frame requirement for a single frame at stage i, i.e the size of the *largest* frame ever stored in the i^{th} buffer in the chain. A chain of N tasks defines a collection of frame requirements:

$$\{\underbrace{s_1, \ldots, s_1}_{M}, \underbrace{s_2, s_3, \ldots, s_{n-2}}_{N-3}, \underbrace{s_{n-1}, \ldots, s_{n-1}}_{M+1}\}$$

If we order the frame requirements in the application in ascending order of s_i, then using a shared memory pool can save the memory space required by the $M + N - 3$ smallest frame requirements.

The memory reservations based on fixed-sized blocks simplify the reallocation of memory between components, allowing for an efficient implementation of a shared memory pool.

IV. Results

Figure 2 shows an application example of a H.264 video encoder, which is commonly used in the consumer electronics domain. We used it to evaluate the memory savings in a real application.

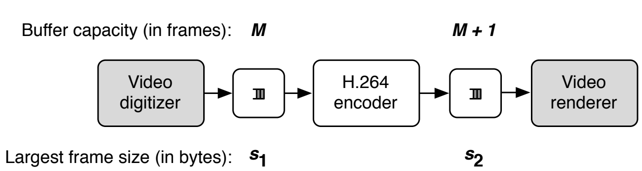

Fig. 2: A video encoding application.

The application consists of three components: the video digitizer provides raw frames in CIF format (with resolution 352x288 pixels) for the H.264 video encoder, which produces a 300kbs video stream with the same resolution for the video renderer. The s_1 parameter is equal to the size of a raw input frame, i.e $s_1 = 352 * 288 = 101376$ bytes. We have measured the largest frame ever produced by the H.264 encoder for a series of standard video sequences[1] to be $s_2 = 26002$ bytes. The relative memory savings are therefore given by $\frac{M * s_2}{M * s_1 + (M+1) s_2}$. Figure 3 shows the memory savings of our approach as a function of M.

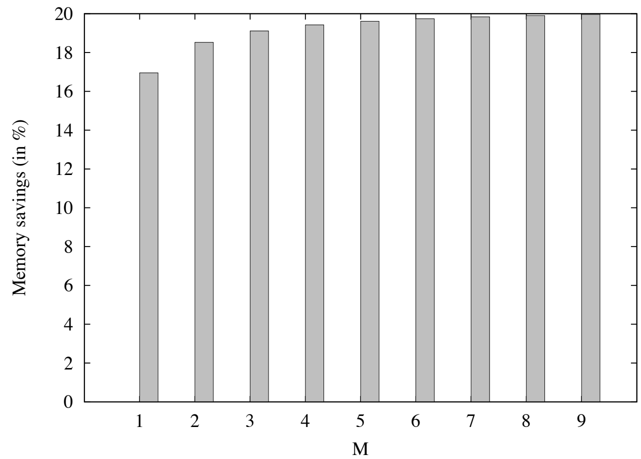

Fig. 3: Memory savings in our example application as a function of M.

Note that video scaling algorithms [4, 3] can be applied to guarantee that the M parameter holds, i.e. that the processing of any sequence of M frames does not exceed $M * T$.

In our video encoder application the raw frames were 4 times larger than the encoded frames. In general, the smaller the difference between the frame requirements at different stages, the larger the memory savings.

V. Conclusions

We have shown a general mechanism for reducing memory requirements in a streaming application comprised of a chain of tasks with periodic head and tail tasks communicating via shared buffers. The proposed method is based on having the buffers share a common memory pool. The results for an H.264 encoder show memory savings of around 19%. The approach is targeted at resource-constrained systems, such as those found in consumer electronics.

References

[1] F. Menichelli and M. Olivieri, “Static minimization of total energy consumption in memory subsystem for scratchpad-based systems-on-chips,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 2, pp. 161 –171, 2009.

[2] M. Weffers-Albu, “Behavioral analysis of real-time systems with interdependent tasks,” Ph.D. dissertation, Technische Universiteit Eindhoven, April 2008.

[3] M. Holenderski, C. G. Okwudire, R. J. Bril, and J. J. Lukkien, “Memory management for multimedia quality of service in resource constrained embedded systems,” in *International Conference on Emerging Techonologies and Factory Automation (ETFA)*, 2010.

[4] C. C. Wüst, L. Steffens, W. F. Verhaegh, R. J. Bril, and C. Hentschel, “Qos control strategies for high-quality video processing,” *Real-Time Systems*, vol. 30, no. 1-2, pp. 7–29, 2005.

[1] Available at http://media.xiph.org/video/derf/