Extended Abstract: Estimation of Execution Times of On-chip Multiprocessor Stream-oriented Applications

P. Poplavko[1], T. Basten[1], M. Pastrnak[1,3], J. van Meerbergen[1,2], M. Bekooij[2], and P. de With[1,3]
[1]*Eindhoven University of Technology,* [2]*Philips Research,* [3]*LogicaCMG Nederland*

1. Introduction

Our work focuses on *stream-oriented* applications with real-time constraints for on-chip multiprocessors, e.g. video/audio coding. In such an application the same function, e.g. frame decoding, is executed over and over again. In general, the execution time is data-dependent. Thus, at run-time a situation may arise when the execution time exceeds the deadline specified in the real-time constraints and a preventive action should be taken. For example, a quality of service (QoS) manager can reduce the quality level. To be able to perform such preventive actions on time, many run-time resource managers predict the execution times in advance.

For stream-oriented applications pipelined execution of the function is very important for achieving the required throughput. An accurate execution time estimation method supporting pipelined execution on a multiprocessor architecture is proposed in this paper. It differs from existing estimation methods designed for run-time management, e.g. [2], which only assume sequential execution of the function rather than pipelined execution.

2. Problem and approach

We consider a generic loop of multiple tasks – we call it the *loop of interest* – that has to perform a specific number of iterations within a certain deadline. This set of iterations is called the loop execution. The total time to complete a loop execution is called the *loop execution time*. In video coding applications such a loop would decode one *video frame* per execution, and it would process one block of pixels per iteration.

The designer of an application provides the specification of the loop of interest in the form of a *process network* (e.g. [3]), consisting of processes, communicating through bounded FIFO channels. Each process executes a fixed subset of tasks in a local loop on a given processor. The problem is then to conservatively estimate the overall execution time of the loop of interest.

Similar to the work of Bavier et al [2] we require that each video frame header must contain a few *parameters*, characterizing the complexity of processing of the given frame a priori, e.g. giving the number of blocks of different types. Such parameters can be easily generated by the video encoder.

Our approach works as follows. First, we derive a timing model of the loop of interest [3]. Then we construct an algebraic expression, which, based on the timing model, yields the loop execution time. We report on a new timing analysis technique that significantly improves the applicability of our approach for data-dependent applications.

Our timing model is based on homogeneous synchronous dataflow graphs (HSDF). The tasks are modeled as dataflow *actors*. The restriction to HSDF is a limiting factor, because HSDF does not support conditional communication between actors. It is a subject of our future work to relax this restriction.

Timing analysis techniques have been proposed in the past employing similar timing models, e.g. for asynchronous circuits [1]. However, such techniques only work for *static* (worst-/best-case) *actor delays*, which may result in overly pessimistic estimations [3]. The analysis techniques working with probabilistic actor delays use assumptions that are not valid for typical stream-oriented applications [4, Chapter 7]. We propose alternative techniques that build upon known results for the case with static actor delays.

3. Static actor delays

In [3], we capture the timing behavior of process networks running on predictable on-chip multiprocessors using HSDF graphs. Figure 1 shows a simple HSDF graph modeling a loop of interest where tasks are placed in a pipeline. The nodes of the graph are actors. They may represent tasks or data transfers. The edges of the graph carry tokens from one actor to another. Each edge may contain a few initial tokens, available at the start of the loop of interest. An actor starts an execution immediately when it sees a token at each input. The duration of actor execution is given by the *actor delay*, d_k. In one execution, each actor consumes/produces one token at each input /output.

If actors have static execution delays, after a finite number of executions, all actors by themselves start to execute periodically with the same average period λ. Therefore, the execution time of a loop where each actor executes J times can be bounded from above by:

$$\hat{ET}_{static} = \lambda \cdot (J - 1) + \sigma \qquad (1)$$

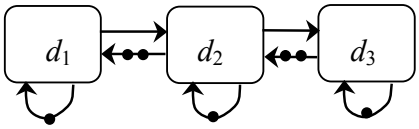

Figure 1 HSDF graph with delay annotations

λ and σ can be obtained from an analysis of the graph. We define σ such that $E\hat{T}_{static}$ bounds the loop execution time for any J, no matter whether the model enters the periodic regime within J iterations [3].

4. Dynamic actor delays

The static analysis – with e.g. worst-case delay values – cannot capture the dynamic properties of user applications. For example, for the decoding algorithm for the binary shape blocks of the MPEG-4 video standard we observed a broad distribution of the delay values (varying by almost a factor of seven).

To cope with this problem, we first model d_k as linear expressions on parameters ξ_i. For example, the time to decode the motion vector of a binary shape block – measured in the clock cycles of an ARM7 processor – can be tightly bounded from above by:

$$d_{\text{dec}} = 175 + 792\,\xi_{\delta} + 580\,\xi_{\delta d} + 464\,\xi_{\delta d+} + 128\cdot\xi_{\text{Nbyte}} \quad (2)$$

where e.g. ξ_{Nbyte} is the number of byte boundaries crossed when accessing the input bitstream. Note that, in effect, the coefficients of this formula provide us with an abstraction of the hardware architecture.

To reuse the results of the static analysis, we split the total loop execution into several intervals where the variation of actor delays is relatively small. For all parameters, we introduce a few quantization levels $\hat{\xi}_i(s)$, called *scenarios*. In every loop iteration, each parameter is rounded up to the closest scenario. At the interval boundaries, where parameters switch from one scenario to another, we identify a *scenario transition*. We apply Equality (1) to every interval and combine the results together, as follows:

$$E\hat{T}_{dyn} = \sum_{\text{scenario } s} \lambda_s \cdot J_s + (\sigma_s - \lambda_s)\cdot L_s - \sum_{s,t} \gamma_{s,t}\cdot K_{s,t} \quad (3)$$

where the first summation in the right part adds the contribution of all intervals of each scenario s; J_s is the total number of iterations and L_s is the total number of intervals of scenario s.

The second summation takes into account the timing overlap between the scenario intervals, which may result from the pipeline-like execution of the processes. $K_{s,t}$ is the number of transitions from scenario s to scenario t and $\gamma_{s,t}$ is the minimal timing overlap at such a transition. We can derive it by analyzing the various paths in the HSDF graph.

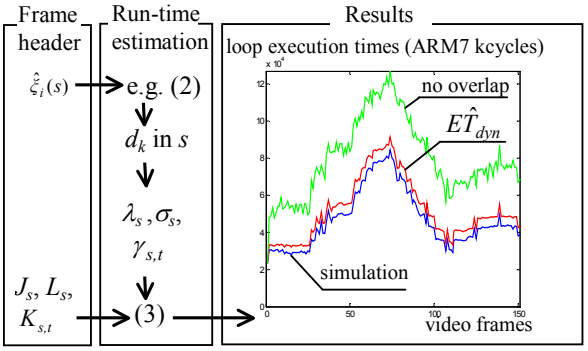

Figure 2: Shape decoding time estimation for video frames

The estimation and the required input data are summarized in Figure 2.

5. Results

We have both simulated and estimated the MPEG-4 shape decoding loop executed in the pipeline depicted in Figure 1, where d_1 represents the decoding of binary shape blocks, and d_2 and d_3 represent the communication and storage of data. For several MPEG-4 video sequences, we have observed that Equality (3) – with just three scenarios – yields a tight upper bound with an average error within 12%. Figure 2 shows the results for one of the sequences. To verify that the computation of minimum overlap is necessary, we also computed an estimate that ignores the second summation in Equality (3), which resulted in a huge overestimation, see Figure 2.

We conclude that we introduced essential ingredients for providing conservative run-time performance estimation for multiprocessors that works even for very dynamic applications, such as MPEG-4. We will investigate our approach for more benchmark applications.

References

[1] T. Amon, et al, "An Algorithm for Exact Bounds on the Time Separation of Events in Concurrent Systems", in: *Proc. ICCD-93*, pp. 166-173, IEEE, 1993.

[2] A.C. Bavier, A.B. Montz, and L.L. Peterson, "Predicting MPEG Execution Times", in: *Proc. Of the ACM SIGMETRICS'98*, pp. 131-140, ACM, 1998.

[3] P. Poplavko, et al, "Task-level Timing Models for Guaranteed Performance in Multiprocessor Networks-on-Chip", in: *Proc. CASES-03*, pp. 63-72, ACM, 2003.

[4] S. Sriram, and S.S. Bhattacharyya, *Embedded Multiprocessors: Scheduling and Synchronization*, Marcel Dekker, Inc., 2002.