Data-flow Timing Models of Dynamic Multimedia Applications for Multiprocessor Systems

Milan Pastrnak[a,b,1], Peter Poplavko[b], Peter N.H. de With[a,b], Dirk S. Farin[b]

[a] LogicaCMG Nederland B.V., Eindhoven, The Netherlands
[b] University of Technology Eindhoven, Eindhoven, The Netherlands
M.Pastrnak@tue.nl

Abstract

Efficient implementation of complex state-of-the-art multimedia applications on a system-on-chip requires a multiprocessor architecture. For this purpose, we have developed modeling techniques for application mapping on predictable multiprocessor systems. In the mapping experiments, it was found that the static nature of synchronous data-flow graphs are not suited for multimedia systems, which are increasingly characterized by dynamic workload. Therefore, we present an extension that also captures the dynamic properties of multimedia applications. We chose MPEG-4 shape-texture decoding, as a complex example for the modeling of an application with dynamic behavior. Experiments with a set of five test sequences show that our model has a high accuracy (3.4%). A key advantage of our modeling method is that it allows to analytically derive the performance characteristics.

1 Introduction

Real-time multimedia applications have become an integral part of embedded systems technology. In most cases, a single processor system cannot cope with the computational complexity of recently developed video-coding standards. For this reason, we assume a multiprocessor SoC (MP-SoC) as the target platform for executing such applications.

A commonly used model description for real-time applications is based on the Synchronous Data-Flow (SDF) graph. The SDF graph is well studied for mapping DSP applications to multiprocessor architectures [1]. However, the DSP applications have static characteristics, whereas general multimedia applications are dynamic. Therefore, in this work we extend the SDF formalism to expose their dynamism of multimedia systems behavior.

As a typical application, we explore the shape-texture decoding approved as MPEG-4 Part 2: Visual (ISO/IEC 14496-2). The arbitrarily-shaped video coding tools offer many extension techniques for a broad range of applications, such as videophones, surveillance systems, etc. In order to meet the real-time constraints at low costs, predictable system behavior of the application is an essential property.

This paper is organized as follows. Section 2 explains the SDF formalism. Section 3 provides a high-level overview of the MP-SoC. The application details are described in Section 4. Section 5 addresses the design flow. The next section presents the computation graph for arbitrarily-shaped video object decoding. Section 7 presents the obtained timing model and the last sections gives results and conclusions.

2 Synchronous Data-Flow Graph

The multiprocessor-level parallelism in our model is expressed by using (SDF) graphs, see e.g. [1]. More precisely, we use a restricted version of the SDF model, called *Homogeneous SDF* (HSDF).

The computations in an HSDF are represented by nodes of the HSDF graph, called *actors*. The *edges* of the graph represent dependencies between actors and carry tokens that are produced and consumed by the actors. Each edge points to the direction of its token flow and may contain a few initial tokens.

It is common to distinguish *data edges* and *sequence edges*. Passing of a token through a data edge represents the transfer of a block of data from one actor to another. On a sequence edge, the tokens represent events that do not carry data, e.g. the release of space in memory or the call of a subroutine by the processor.

[1]Supported by the European Union via the Marie Curie Fellowship program under the project number HPMI-CT-2001-00150.

Each actor waits until there is at least one token at each incoming edge. Then the actor performs computations on the contents of the first data token that is available at each data input. The computation takes a well-defined time interval, which only depends on the contents of the input data. We call this interval the *computation time* of the actor. When the computations have finished, one token is consumed from each incoming edge and one token is produced to each outgoing edge. HSDF has predictable timing behavior, which means that if the timing of the inputs is known, the timing of the outputs can be derived.

3 Multiprocessor System-on-chip

MP-SoC platforms represent an attractive architecture that can serve various future applications. Moreover, when combined with SDF, it intrinsically supports predictable timing.

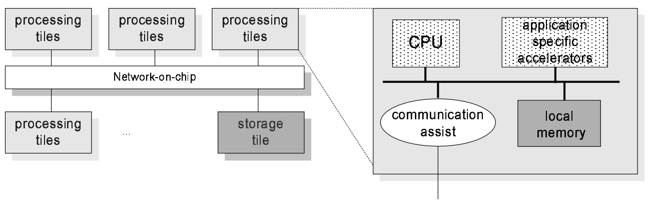


Figure 1. MP-NoC tile-based architecture.

The term “network” in Multiprocessor networks-on-chip (MP-NoCs) refers to the enclosed switch network that is used for global on-chip communication. In our case, we employ a *tile-based* architecture with distributed memory, as depicted in Figure 1. The MP-NoC platform contains processing tiles, storage tiles and the network-on-chip (or switch network). A processing tile represents a small embedded computer, consisting of one embedded CPU core (e.g. RISC), local memory and specific accelerators. The NoC transports data packets from one tile to another.

In general, NoC can be modeled using SDF graphs, provided that the following constraints on the architecture are satisfied. First, actors running in parallel on different processors use only the local memories of their processing tiles. Second, the memories are organized in single layers (no caching), or the caches are locked. This provides the predictability of the actor computation times.

The NoC should provide point-to-point connections with tightly bounded packet propagation delays. Similar to data edges in SDF graphs, the connections should be independent from each other and they should carry multiple tokens in FIFO order. Such connections can be implemented in NoCs at a reasonable cost [2].

4 Arbitrarily-Shaped Video Objects

The MPEG-4 Core Profile as well as the Advanced Coding Efficiency (ACE) Profile include arbitrarily-shaped object coding, which is the basis of our case study. Every video object (VO) is represented in several information layers, with the Video Object Plane (VOP) at the base layer. This video object plane is a rectangular frame composed from units called a macroblock (MB). An MB is the smallest data unit that we consider.

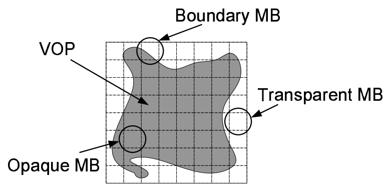


Figure 2. Macroblock classification within arbitrarily-shaped VOP.

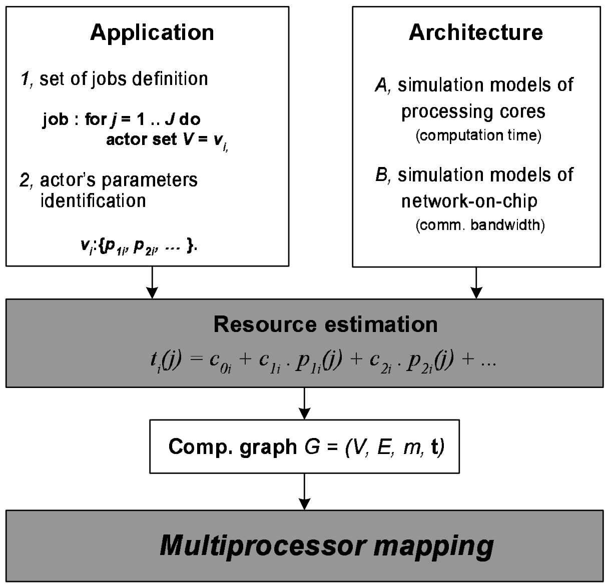
Figure 2 depicts the different MB types occurring during encoding of arbitrarily-shaped video objects. For intraframe-coded VOs, the standard distinguishes three types of macroblocks: boundary, opaque, and transparent. The boundary macroblock is defined as a macroblock that contains both transparent pixels and opaque pixels.

Transparent MBs are coded with a few bits only, since they contain neither shape nor texture information. Consequently, they require very little processing time. Opaque MBs require only the texture information to be decoded. Boundary MBs contain both texture and shape and require the most computations. For this reason, the required amount of computations can change drastically from one iteration to the other. Such a drastic change, has to be captured by our model.

5 Design Aspects

Within the context of the overall design methodology, we focus on on estimating the resource usage of actors (also called tasks). Resource estimation should precede the mapping process [3]. The main resources are the computing power, the required capacity of memories and the network communication bandwidth. In this paper, we investigate only the required computing power. Data storage exploration and bandwidth usage are currently studied as well.

Figure 3 outlines our view on the design flow. The resource estimation accepts inputs from the application and from the architecture. The application provides an executable specification of the application, where each job within the application is divided into actors. The hardware architecture is represented by accurate simulation models of the processing tiles, thereby allowing the measurement of the actor computation time.



Application
1, set of jobs definition
job : for j = 1 .. J do
actor set V = v_i
2, actor's parameters identification
v_i:{p_1i, p_2i, ... }.
Architecture
A, simulation models of processing cores
(computation time)
B, simulation models of network-on-chip
(comm. bandwidth)
Resource estimation
t_i(j) = c_0i + c_1i · p_1i(j) + c_2i · p_2i(j) + ...
Comp. graph G = (V, E, m, t)
Multiprocessor mapping

Figure 3. Design flow.

Every *application* activates dynamically a series of application *jobs*, e.g. a Video-Object (VO) decoding task activates a sequence of VOP-decoding jobs. Each job can use multiple processing tiles in parallel. We model a video-decoding job as an iterative "for" loop, taking J iterations to produce J video sample blocks (macroblocks).

The body of the "for" loop in Figure 3 is described by a set of actors $V = \{v_i\}$, representing the actors executed at each iteration. An actor can be described as a function in a programming language like C/C++, representing the atomic unit of computation that can be assigned to a processing tile. We assume that the computation time of actor v_i can be expressed as a linear function t_i. Input-data parameters $\{p_{1i}, p_{2i}, ...\}$, with fixed coefficients $\{c_{0i}, c_{1i}, c_{2i}, ...\}$ depend on the architecture of the tile to which the actor is assigned (see Figure 3). The individual timing functions t_i are combined in an overall timing-function vector $\boldsymbol{t}$ which represents the timing model adopted in our approach. Note that the values that both the individual timing functions and the parameters $p_i(j)$ may change in each j-th iteration in the "for" loop.

To exploit parallelism in multiprocessor mapping between actors in the loop body of the job, we use a computation graph which is an instance of a homogeneous SDF graph $G = (V, E, m, \boldsymbol{t})$ model of computation [1]. The parameter E is the set of edges that models the dependencies between the actors, and m (markings) is a function giving the number of initial tokens on every edge.

In full arbitrarily-shaped VO decoding for MPEG-4, the sequence of VOPs can activate jobs for VOP decoding. We only discuss jobs for the I- and P-VOP decoding and assume that actors process one MB per iteration. For the mapping, it is assumed that actors are executed on a RISC processor (ARM7) processor. The full-software implementation is a meaningful starting point for the design exploration [4], and it can lead to adopting hardware accelerators in a further design process.

6 Computation Graph for Shaped Video-Object Decoding

Figure 4 shows a computation graph for arbitrarily-shaped video object decoding. When comparing it with a fixed-shape decoding (e.g. MPEG-2), one can identify the extension for the shape-information decoding, followed by the texture data.

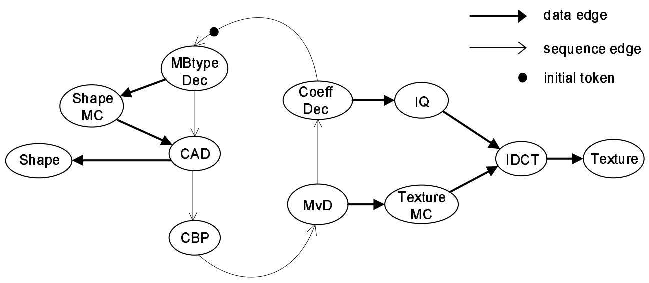


Figure 4. Computation graph of motion-compensated AS-VO decoder.

Every iteration of the job loop starts with the Macroblock type Decoding (MBtype Dec). The ShapeMC actor computes the motion compensation for the shape part and provides the referenced MB for the Context Arithmetic Decoding (CAD) actor. The CAD actor decodes an MPEG4-compliant shape representation of the macroblock. The shape for a macroblock is represented by a 16×16 Binary Alpha Block (BAB). As is depicted in Figure 4, the BAB should be sent to the output of the job (actor "Shape"). The Coded Block Pattern (CBP) extracts information about the parts of the texture that need to be updated.

The texture decoding comprises five steps: Motion vectors Decoding (MvD), IDCT Coefficients Decoding (Coeff Dec), Texture Motion Compensation (TextureMC), Inverse Quantization (IQ) and Inverse DCT (IDCT).

Actors MBtypeDec, CAD, CBP, MvD and CoeffDec have to be executed sequentially, because each actor depends on another actor to specify the next position in the input bitstream. For this reason, we have introduced a loop surrounding these actors, using so-called sequence edges that indicate the order between the actors. We observed possible parallelism with other actors, namely ShapeMC, TextureMC, IQ and IDCT. Actors “Shape” and “Texture” model the collection of the decoded video.

7 Timing model

For each actor of the computation graph, we identified a set of parameters (p_i in Figure 3), based on the study of the algorithm and experiments. We applied linear regression to derive the coefficients $c_{k,i}$. To measure the actor computation times $e_i(j)$, we executed the decoder software on an ARM instruction set simulator. The basic unit of time is the processor clock cycle. To provide an idea about the obtained linear model functions, we list three examples below.

$$
\begin{aligned}
&t_{MBtypeDec} = 913 + 1.34k \cdot \tau + 1.9k \cdot \tau' \\
&t_{CAD} = 14.13k + 135.12k \cdot \mu + 190 \cdot N_{bits1} \\
&\qquad 400 \cdot N_{bits2} + 390 \cdot N_{bits3} \\
&t_{CoeffDec} = 108 + 481 \cdot \varphi + 453 \cdot N_{VLD-Bytes} + 1.6k \cdot N_{DC} \\
&\qquad + 582 \cdot N_{DC+} + 1.78k \cdot N_{AC-NE} + 3.52k \cdot N_{AC-E1} \\
&\qquad + 3.55k \cdot N_{AC-E2} + 5.24k \cdot N_{AC-E3} + 303 \cdot \gamma
\end{aligned}
$$

For example, consider the CAD actor. It takes 14.13 kilocycles (indicated by a “*k*”) to initialize the actor. If the MB is a boundary block, we assign $\mu = 1$, otherwise we assign $\mu = 0$. For $\mu = 1$, 135.12 kcycles have to be spent on decoding the shape, plus 190-400 cycles for decoding of each bit contained in the arithmetic code.

It is important for our approach to evaluate the accuracy of the obtained timing model. To do this, we evaluate the statistical computation-time error. It is defined as a relative difference magnitude between the estimation given by the timing model and the measured value. For all actors we achieved a high accuracy with the average error below 5% for the test sequence. However, for reliable results, still more test cases have to be considered.

Figure 5 shows a typical example of the model behavior for e.g. the CBP actor. The plotted curves exhibit the required processing time for a sequence of 60 macroblocks. The gray curve shows the estimation of the model and the bold curve refers to the real computation. It can be seen that the model is quite accurate because both curves coincide mostly.

8 Conclusions

We have studied the estimation of the computing power resources for the mapping of the MPEG-4 arbitrarily-shaped video decoding onto a multiprocessor network-on-chip. For this purpose, we have developed a timing model

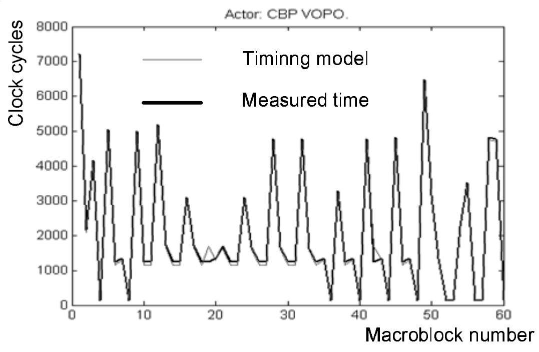


Figure 5. The timing model estimate and the measured computation time.

that estimates the computation times. The timing model is based on a set of linear equations, using parameters depending on the actual I-VOP and P-VOP MPEG-4 bitstream characteristics. The results show a high accuracy and the applicability of this approach for the estimation of processing-time usage in the architecture.

Future work will focus on modeling more complex applications like MPEG-4 encoding to prove the usability of our data-flow model, since we envision that our methodology can be reused for many other embedded system designs. Furthermore, we would like to apply our framework to heterogenous MP-NoCs using different cores and dedicated accelerators. We expect that different processing tiles will mostly influence the values of coefficients in the timing model, while maintaining the structure of the timing functions.

References

[1] N. Bambha, V. Kianzad, M. Kahndelia, and S. S. Bhattacharyyan, “Intermediate representations for design automation of multiprocessor dsp systems,” in *Design Automation for Embedded Systems*. 2002, vol. 7, pp. 307–323, Kluwer Academic Publishers.

[2] E. Rijpkema, K. G. W. Goosens, A. Radulescu, J. Dielissen, J. van Meerbergen, P. Wielage, and E. Waterlander, “Trade offs in the design of a router with both guaranteed and best effort services for networks on chip,” in *Proceedings of DATE’03*, 2003, pp. 350–355.

[3] R. Lauwereins, M. Engels, M. Ade, and J. A. Peperstraete, “Grape II: A system-level prototyping environment for DSP,” in *IEEE Transaction on Computer*, February 1995, vol. 28 of *2*, pp. 35–43.

[4] M. Wan, H. Zhang, V. George, M. Benes, A. Abnous, V. Prabhu, and J. Rabaey, “Design methodology of a low-energy reconfigurable single-chip DSP systems,” in *Journal of VLSI Signal Processing*, January 2001, vol. 28 of *1*.