A FRAMEWORK FOR MAPPING SCALABLE NETWORKED MULTIMEDIA APPLICATIONS ON RUN-TIME RECONFIGURABLE PLATFORMS

N. Pham Ngoc¹, G. Lafruit², J-Y. Mignolet², S. Vernalde², G. Deconinck* and R. Lauwereins²

¹Katholieke Universiteit Leuven-ESAT/ELECTA
Kasteelpark Arenberg 10
B-3001 Leuven-Heverlee, Belgium
{Nam.Phamngoc, geert.deconinck}@esat.kuleuven.ac.be

²IMEC-DESICS, Kapeldreef 75, B-3001 Leuven-Heverlee, Belgium
{lafuit, mignolet, vernalde, lauwerei}@imec.be

* Geert Deconinck is a postdoctoral fellow of the Fund for Scientific Research-Flanders-Belgium

ABSTRACT

Due to the heterogeneous nature of networks and end-systems in distributed multimedia systems, multimedia applications should ideally be designed to counteract fluctuations in network bandwidth and end-system processing capacities for providing end users a certain degree of Quality of Service (QoS). This requirement can be satisfied with scalable applications. In addition, with the current evolution in run-time reconfigurable computing, run-time reconfigurable multimedia platforms are becoming increasingly viable. In this paper, an end-to-end delivery chain framework for mapping scalable networked multimedia applications on reconfigurable platforms is presented. The framework is demonstrated by a case study of a 3D game running on a prototype run-time reconfigurable platform.

1. INTRODUCTION

In distributed multimedia systems, the processing power of end-systems may vary from a workstation, a portable computer, a set top box (STB), to even a PDA. Moreover, end-systems can be connected to the network at different transmission bitrates. Due to this heterogeneous nature of networks and end-systems, multimedia applications should ideally be designed to adapt to aforementioned variations, therefore providing end users a certain degree of Quality of Service (QoS). Fortunately, a number of scalable encoding/decoding schemes for multimedia contents have been developed recently, e.g. MPEG-2, MPEG-4 and JPEG-2000, which can deal with such requirements. In addition, with the current evolution in run-time reconfigurable computing techniques and field programmable device technologies, run-time reconfigurable multimedia platforms are becoming increasingly viable [1,2]. The reconfigurable aspect provides a much higher performance than software microprocessors, while maintaining a higher level of flexibility than custom-designed Application-Specific Integrated Circuits (ASICs) [3]. Consequently, it will become possible in the future to download applications that can run on the reconfigurable hardware as nowadays applications can be downloaded and run on a PC.

Consider, for example, an application scenario where a scalable MPEG-4 movie is downloaded from a movie server to an end-user system with a microprocessor and run-time reconfigurable hardware (e.g. field programmable gate arrays FPGA). The movie is encoded into three layers: a base layer which provides a basic quality of 15 fps and two enhancement layers which enhance the framerate to 20 fps and 25 fps, respectively. If the software MPEG-4 movie player can run at a maximum of 10 fps, while the user would prefer to watch the movie at 20 fps, it could be envisaged to additionally download a hardware bitstream configuring the FPGA into a 20 fps MPEG-4 movie player.

If some time later, the user prefers to play a 3D game for a while, he/she can download a 3D game engine from a game server to configure the FPGA into a 3D rendering device. Since the FPGA is not available any more for the MPEG-4 player, the remaining part of the movie is run in software and rendered at e.g. only 10 fps (by dropping some frames of the base layer), or in a smaller window at 20 fps. This penalty is not cumbersome, since the user puts the focus on the 3D game anyway.

It is obvious that the above scenario can readily be extended to situations where some tasks can be running in software, while others are running in hardware, imposing a tedious mapping process from applications to run-time reconfigurable platform. Our previous work [4] and that of others in the QoS community [5] only dealt with the problem of mapping scalable applications on microprocessor platforms, which are purely software. On the other hand, research on reconfigurable computing mainly concentrated on hardware/software (HW/SW) partitioning problem at the client side for non-scalable
applications [6,7]. In this paper a hybrid framework for mapping scalable applications on run-time reconfigurable platforms, with instruction-set processors (ISP) and a run-time reconfigurable hardware is presented. The framework involves both the service provider and end-user (or client) sides to take advantage of the end-to-end delivery chain characteristics.

The paper is organised as follows. Section 2 presents the application model for scalable multimedia applications. Section 3 describes our framework. The framework is demonstrated by a case study of a 3D game in section 4. Conclusions are drawn in section 5.

2. APPLICATION MODEL

Scalable applications are represented by a scalable directed acyclic task graph $G(V, E)$ as shown in Figure 1. The vertices of the graph are tasks, which can be implemented in hardware or software. The edges represent data dependencies between tasks. Tasks may be scalable to provide different quality levels at different processing requirements. The tasks in dotted lines are optional: they provide additional functionalities to improve the quality of the application and hence may be dropped when needed. This kind of scalability is called layered scalability, where the quality of the application is enhanced by adding one or more enhancement layers and their accompanying tasks. MPEG-2 and MPEG-4 are examples of layered scalable video codecs [8].

![Figure 1. Task graph of scalable applications](image)

For a different kind of scalability, called input-data driven scalability, the quality level of a task depends on the input data. The execution time and other resource requirements of the task are also a function of the input data. Usually, for this kind of scalability, the code sequence the task executes is fixed, but the amount of time spent in the code depends on the richness of the incoming data. For example, in 3D rendering, the rendering algorithm is fixed and the rendering time depends on the mesh resolution, corresponding to the quality of the 3D object [4].

Two types of tasks are considered: SW only and HW/SW tasks. SW only tasks are implemented only in SW, while HW/SW tasks are implemented in both HW and SW versions at the service provider side and one of these versions is selected at the client side.

Consider one application which has $L$ different quality levels $Q_1$-$Q_L$ with $N$ tasks $T_1$-$T_N$.

Each quality level $Q_i$ corresponds to a benefit value $B_i$ which represents the degree of user satisfaction when receiving this quality level.

Let us further define:

- $L^*N$ mapping matrix $\alpha$ that maps quality levels on tasks: $\alpha_{i,k}=1$ if task $T_i$ is needed in order to obtain quality level $Q_k$, $\alpha_{i,k}=0$ otherwise.
- $N^*N$ communication matrix $\chi$, $\chi_{i,j}=1$ if tasks $T_i$ and $T_j$ communicate with each other, $\chi_{i,j}=0$ otherwise.
- For each task $T_i$, a variable $\beta_i$ is 1 if $T_i$ is implemented in HW, $\beta_i=0$ if $T_i$ is implemented in SW.
- $t_{Ti}(Q_j, \beta_i)$ is the execution time of task $T_i$ at quality $Q_j$ with implementation $\beta_i$.
- $M_{Ti}(Q_j, \beta_i)$ is the memory needed by task $T_i$ at quality $Q_j$ with implementation $\beta_i$.
- $P_{Ti}(Q_j, \beta_i)$ is the power consumption of task $T_i$ at quality $Q_j$ with implementation $\beta_i$.
- $A_{Ti}(Q_j, \beta_i)$ is the hardware area needed by task $T_i$ at quality $Q_j$ with implementation $\beta_i$. Therefore $A_{Ti}(Q_j, 0)= 0$.
- $t_{km}(Q_j, \beta_k, \beta_m)$ is the communication time between task $k$ and $m$ at quality level $Q_j$ with implementations $\beta_k$ and $\beta_m$.

3. FRAMEWORK

The structure of the framework is given in Figure 2. Some components are part of the server; others belong to the client. These components are explained in more detail in the next subsections.

![Figure 2. Framework structure](image)

3.1. Components at the service provider

At the service provider side, the scalable HW/SW tasks (SHST) database and the scalable multimedia objects (SMO) database are managed by the database manager.

The scalable HW/SW tasks database stores SW and HW code of tasks in the task graph of applications (e.g. MPEG-4 decoder) provided by the service provider. A
design flow which allows the service provider to create SW and HW tasks from the application functional model was presented in [9]. These tasks are encoded in the form of software bytecode and hardware bytecode, which are target-platform independent. This property allows the service provider to provide services to a wide range of clients.

The scalable multimedia objects database contains bitstreams of actual multimedia objects and object description bitstreams. The object description bitstreams contain high level information (meta-data) about objects for estimating the resource requirements [4] wherever needed (in the network layer or the client side).

3.2. Components at the client side

The user interface handles user interaction and has information about the user profile. In the user profile the user can express his/her preferences about the resolution and framerate of the media objects, i.e. video, 3D, etc. The middleware layer consists of a QoS-based HW/SW partitioner (QHSP), a resource estimator (RE) and a HW/SW translator (HST). The middleware layer runs on top of an operating system, which provides services and a scheduling algorithm for HW/SW multitasking.

The QHSP decides which tasks should be put in SW, which tasks should be put in hardware and in both cases which quality level should be set, such that the user receives the highest possible quality level of the application given the constraints in resources and power consumption of his/her platform. In other words, QHSP finds the best combination of the parameters $Q_i$, $\alpha$ and $\beta$ described in section 2.

The RE is in charge of estimating the resource requirements for each task based on high level information from the application (e.g. framerate, resolution...) and specific information for the platform (which is typically obtained by profiling). These resource requirements are input for the QHSP to decide on the partitioning.

The HST translates the platform independent SW and HW code to platform specific code. The translated SW code runs on the CPU, while the translated HW code is used to configure the FPGA. The HST plays the role of the software and hardware virtual machine in [9].

4. EXPERIMENTAL RESULTS

4.1. System setup

The prototype run-time reconfigurable platform consists of a Compaq iPaq™ PDA, running RT-Linux on its Strong-Arm processor SA-1110 (206 MHz) and controlling a Xilinx Virtex™ XC2V6000 FPGA (Figure 3). The FPGA is the reconfigurable hardware, which is divided into logical tiles of coarse granularity with fine-grain reconfiguration. The platform is connected via a serial port to a PC, which plays the role of the service provider.

4.2. The 3D game

A simplified Quake-alike 3D shooting game has been developed (Figure 4). The 3D scene is composed of a number of walls. The aim of the game is to shoot the targets on the wall. We also added water-ringing effects on the (wet) ground of the 3D scene to increase realism (and processing requirements).

Figure 5 shows the task graph of the game. Tasks T1, T2, T3 do the transformation of the 3D scene from the object space to the world space, from the world space to the view space and from the view space to the screen space, respectively. The texture mapping task (TM) does the mapping of textures on the walls. The water processing task (WP) does repetitive 2D filtering operations to create the water-ringing effects. WP is an optional task. Finally, task T6 displays the 3D scene on the screen.

The window size of the game can be adjusted (scalable) from LARGE (320 * 240) to MEDIUM (210 * 160) or SMALL (160 * 120). Therefore, there are in total six different quality levels for the game, these are $Q_1$(LARGE, water), $Q_2$(LARGE, no water), $Q_3$(MEDIUM, water), $Q_4$(MEDIUM, no water), $Q_5$(SMALL, water), $Q_6$(SMALL, no water).
4.3. Implementation and simulation results

Tasks T1, T2, T3 and T6 are software only tasks and are implemented in C, while TM and WP are HW/SW tasks which are implemented in both C and VHDL. The VHDL code of the TM and WP tasks have been generated by OC-API-XL [10], a C/C++ system-level hardware design environment. The synthesis has been performed with Synplicity’s Synplify on the Virtex XC2V6000. The C codes and hardware bitstreams of all tasks are stored in the HW/SW tasks database in the PC. The 3D scene description and the texture images are stored in the objects database.

For the sake of simplicity in implementation, both software and hardware codes are specific for the prototype platform and therefore, a HW/SW translator is not implemented.

At the current state of our work, the resource estimator only implements functions to estimate the execution time of tasks in HW and SW. Other resources like communication, memory, and power consumption are not yet considered in the estimator.

Analysis of the HW and SW code reveals that the execution time of the TM and WP tasks in HW and SW can be estimated based on the window size (X*Y), as follows:

\[ t_{TM}^{HW} = (44 + (25*Y + 28)*X)*0.02 \mu s \]
\[ t_{TM}^{SW} = 1.95 * X*Y \mu s \]
\[ t_{WP}^{HW} = 0.125 * X*Y \mu s \]
\[ t_{WP}^{SW} = X*Y \mu s \]

In our implementation, all tasks of the 3D game are executed sequentially and thus the time to execute one frame is the sum of the execution times of all tasks. The total execution time for tasks T1, T2, T3 and T6 is approximately 10% of the frame time. Table 1 shows some possible mappings based on the model presented in section 2.

<table>
<thead>
<tr>
<th>Q</th>
<th>Mapping</th>
<th>A_TM</th>
<th>A_WP</th>
<th>fps</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>β_TM=0</td>
<td>4</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>β_TM=1</td>
<td>4</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>β_WP=0</td>
<td>9</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>β_WP=1</td>
<td>9</td>
<td>1</td>
<td>9</td>
</tr>
</tbody>
</table>

Since the number of possible mappings is quite small, an exhaustive search algorithm has been implemented in the QHSP to find the best possible mapping. A number of simulations have been done by varying the user requested framerate and the number of available configurable logic blocks (CLB) on the FPGA. For example, for a framerate of 15 fps and the number of available CLB equal to 1300, the game will be played with LARGE window, without the water effects, while the TM task is running in hardware. However, if the user also wants to have the water effects, the game is played with MEDIUM window, while the TM task is running in hardware and the PW task is running in software (see Table 1). It is clear that, the mapping solution strongly depends on the quality of service requirements of the user.

5. CONCLUSIONS

We have presented a model for scalable multimedia applications and a framework to map these applications on a run-time hybrid HW/SW reconfigurable platform. The experimental results on a 3D game case study have proven the soundness of the proposed framework. A more effective QoS-based HW/SW partitioning algorithm is subject to future work. Other types of resources, including memory, communication and power consumption, will also be considered in the near future.

REFERENCES


