A PLATFORM FOR ALGORITHM / ARCHITECTURE DESIGN EXPLORATION

Christophe Lucarz, Marco Mattavelli
Ecole Polytechnique Fédérale de Lausanne
Research Group in Multimedia Architecture (LSM Lab)
{christophe.lucarz, marco.mattavelli}@epfl.ch

Julien Dubois
Université de Bourgogne
LE2I Lab
jdubois@u-bourgogne.fr

ABSTRACT
The efficient implementation of multimedia algorithms, for the ever increasing complexity of the specifications and the emergence of the new generation of processing platforms characterized by multicore and multicomponent parallel architectures, requires appropriate design space exploration procedures as preliminary step for any implementation. This paper describes a new platform aiming at supporting the algorithm and architecture co-exploration starting by a pure software specification that is gradually transformed into a into a possibly mixed SW and HW implementation. The process is based on profiling capabilities supported by the new platform specifically conceived to study and optimize data flows and data transfers between SW and HW modules. The process is based on profiling capabilities supported by the new platform specifically conceived to study and optimize data flows and data transfers between SW and HW modules. Different explicit or implicit (i.e. virtual memory extensions) data transfer modes can be profiled in the co-exploration process, by using minimal SW reconfiguration, thus minimizing any SW/HW re-writing effort in the co-exploration stage. Such optimization capabilities can be used to achieve different optimization objectives such as optimization of memory architectures or low power designs by appropriate minimization of data transfers. Experimental results and an example of the usage of the platform are provided for the design case of a motion estimation module for video encoding.

Index Terms— SW/HW co-design, design space exploration, virtual socket platform

1. INTRODUCTION
The increasing complexity of signal processing systems in all field, but particularly for video and multimedia processing has already lead to the need of developing the algorithms/standards specifications using software implementations that are the starting point of the implementation process. Thus, they are the input of any algorithm/architecture co-exploration step. In the case of MPEG such specifications were and are in the form of monolithic C/C++ reference SW model for all recent existing standards (AVC and SVC) or will be in the form of a higher level data flow model for past and future standards as is under development in the new MPEG Reconfigurable Video Coding Standard [1]. In both cases of monolithic sequential models or data flow models written in CAL [2] the analysis of the execution time of a function (for the C/C++ case) or the execution of the actions of a CAL actor (for the data flow case) implemented in HW and the analysis timing of the data or tokens exchanges with the other SW or HW component are essential information for an appropriate design space exploration stage aiming at optimizing concurrency and parallelism that will be massively available in the next generation processing platform. This paper, presents a design space exploration methodology and an associated tool supporting the processes of designing embedded systems starting from the C/C++ or CAL specification algorithm.

This paper is structured as follows: section 2 presents the need of design exploration for sequential and data flow based specifications, section 3 presents the virtual socket platform including the supported profiling features, section 4 provides an example of profile result, section 5 briefly presents some related works and finally section 6 concludes the paper.

2. DESIGN SPACE EXPLORATION OF COMPLEX MULTIMEDIA SYSTEMS
Initially, multimedia systems are specified by sequential C/C++ programs called reference models. However, generic sequential languages such as C/C++ are not good implementation independent descriptions of algorithms particularly for parallel and concurrent platforms. Moreover, when optimized they usually remain strongly related to specific processors and DSPs for which optimization is targeted. A better way to specify and describe a multimedia algorithm better exposing the intrinsic concurrency and parallelism as well as the data exchanges is, for instance, to use a high level dataflow language. CAL [2] is a good formalism to describe in a compact form multimedia algorithms [1] that has been recently adopted by ISO/IEC MPEG committee for unifying and specifying video coding technology in a modular reconfigurable form. However, in both cases, of specifications expressed as sequential models or CAL data flow models, the measure of the execution time and the measure of the data exchanges for functions or actor’s actions execution time for
each token consumption, is a very important information for exploring the design space of multicomponent heterogeneous platforms. In fact, if all actors of a CAL model are independent concurrent entities, the sequence of actions executed by a CAL is considered atomic within the data flow model simulations and requires the evaluation of the execution time and the timing of all data exchanges as they occur to correctly profile the system. For sequential models the need of accurate evaluations of the speed-up achievable and of the data bandwidth necessary by HW co-processing units is obvious. Such evaluations, when available in the early design exploration stage, can then be used with two different purposes for the two forms of specifications. In the case of a sequential model it is straightforward to understand how to use the timing information since it provides a direct measure of the achievable speed-up of one or more HW modules. In the case of a CAL model if execution time and timing of data transfers are available, it is possible to feedback such information to the CAL model execution scheduling thus enabling the actual profiling of a CAL model. The interest of such approach is that CAL models are intended to be the input of SW and HW synthesis tools. Therefore, any tool that evaluate execution time of HDL synthesized CAL actors and associated data transfers constitute a powerful design exploration tool because HDL descriptions of CAL actors can be directly synthesized by the CAL specification itself. Since it is very difficult to forecast how a synthesized (complex) module behaves in terms of execution time and data access, a platform that can profile such behavior abstracting from what is "outside" the actor itself constitute a very interesting design exploration tool.

3. THE VIRTUAL SOCKET PLATFORM WITH VIRTUAL MEMORY EXTENSION

The Virtual Socket concept implemented in a support platform has been presented in detail in [3, 4, 5, 6] and has been developed to support the mixed specification of MPEG-4 Part2 and Part 10 (AVC/H.264) specifications in terms of reference SW including the plug-in of HDL modules. The platform is constituted by a standard PC where the SW is executed and by a PCMCIA card that contains a FPGA and a local memory. Specifying explicitly the data transfers would not constitute a serious burden when dealing with simple deterministic algorithms for which the data required by the HDL module are known exactly. Unfortunately for complex design cases, when data transfers cannot be explicitly specified in advance by the designer or because the module has been synthesized by a HDL synthesis tool a system that can abstract from explicit data transfer can be extremely useful for design exploration purposes because it avoids some of the resource and time consuming design operations. The Virtual Socket platform supporting virtual memory capability allow automatic data transfers from the host, running the SW part, to the local HW memory. The goal of such platform implementation is to provide a "direct map" of any SW portion to a corresponding HDL specification without the need of specifying any data transfer explicitly. In other words, to extend the concept of Virtual Socket for plugging HDL modules to SW partition with the concept of virtual memory. HDL modules and software algorithm share a unified virtual memory space. Having a shared memory - enforced by a cache-coherence protocol - between the CPU running the SW sections and the platform supporting HW avoids the need of specifying explicitly all the data transfers. The clear advantage of such solution is that the data transfer needed to feed the HDL module can be directly profiled so as to explore different memory architecture solutions. Another advantage of such direct map is that conformance with the original SW specification is guaranteed at any stage and the generation of test vectors is naturally provided by the way the HDL module is plugged to the SW section.

The Virtual Socket platform is composed of a PC and a PCMCIA card that includes a FPGA and a local memory. The Virtual Socket handles the communications between the host (the PC environment) and the HDL modules (in the FPGA inside the PCMCIA).

Given that the HDL modules are implemented on the FPGA, they have only a physical access to the local memory (see figure 1). This was the case of the first implementation of the Virtual Socket platform, with the consequence that all the data transfers from the host to the local memory had to be specifically specified in advance by the designer himself. Such operation beside being error prone or be implemented transferring more data than necessary it is not straightforward and may become difficult to be handled when the volume of data is comparable with the size of the (small) local memory. Therefore, an extension has been conceived and implemented so as to handle these data transfers automatically. The Virtual Memory Extension (VME) is implemented by two components: the hardware extension to the Virtual Socket platform (Window Manager Unit) and a Virtual Manager Window (VMW) library on the host PC. The cache-coherence pro-

![Fig. 1. The Virtual Socket Platform](image)
The designer must configure the platform by using the Virtual Socket platform (see figure 1). This function drives the Virtual Socket platform and the VME to activate the HDL module. The designer must set a given number of parameters needed for the configuration of the HDL module. This can be done thanks to the data structure Module_Param. Sixteen parameters are available for each HDL module.

3. The HDL call function is started. This function writes the parameters in the register memory of the Virtual Socket platform (see figure 1). Start_module() drives the Virtual Socket platform and the VME to activate the HDL module. This function is from the VMW API.

4. When the entire job is finished, the platform is closed. The VMW library manages all the data transfers between the main memory (unified virtual memory) and the local memory of the platform because as the HDL module is in an FPGA, it has access only this local memory. Thanks to the VME, the HDL module has access to the host memory without intervention of the designer. Data are sent to the HDL module and results are updated in the main memory automatically thanks to the software library support. When the HDL module finishes its work, the hardware call function is terminated.

Fig. 2. Relations between the unified virtual memory, the reference software algorithm, and the HDL module.

Given a reference software so as to test each HDL modules separately, the designer needs to execute some parts of the reference algorithm using the host processor and to test and profile the HW modules on the Virtual Socket platform that implements the support for the hardware. So as to easily handle such mixed HW/SW specifications, it is very convenient that the HDL and C/C++ functions have access to the same user memory space. This memory is part of the host hardware and contains the data to be processed. Such host memory space is trivially available by the processor which executes the reference software, but it is much less evident for the Virtual Socket platform which is on the FPGA and is the support for the HDL modules. For instance, when the designer wants to run a piece of the code using the reference software and the other functions using one HDL module, the section of the reference code the designer intends to execute in hardware is replaced by the following piece of code which is called the HDL module calling procedure:
Table 1. Profiling results for the motion estimation module with a search window of 56x40.

<table>
<thead>
<tr>
<th>Access</th>
<th>Variable</th>
<th>Count</th>
<th>Timing</th>
</tr>
</thead>
<tbody>
<tr>
<td>read</td>
<td>pattern</td>
<td>64</td>
<td>0 ns</td>
</tr>
<tr>
<td>read</td>
<td>search_window</td>
<td>560</td>
<td>1.280 us</td>
</tr>
<tr>
<td>write</td>
<td>results</td>
<td>1</td>
<td>381.2 us</td>
</tr>
<tr>
<td>finish</td>
<td></td>
<td></td>
<td>381.300 us</td>
</tr>
</tbody>
</table>

Table 2. Profiling results for the motion estimation module with a search window of 80x40.

<table>
<thead>
<tr>
<th>Access</th>
<th>Variable</th>
<th>Count</th>
<th>Timing</th>
</tr>
</thead>
<tbody>
<tr>
<td>read</td>
<td>pattern</td>
<td>64</td>
<td>0 ns</td>
</tr>
<tr>
<td>read</td>
<td>search_window</td>
<td>800</td>
<td>1.280 us</td>
</tr>
<tr>
<td>write</td>
<td>results</td>
<td>1</td>
<td>603.000 us</td>
</tr>
<tr>
<td>finish</td>
<td></td>
<td></td>
<td>603.100 us</td>
</tr>
</tbody>
</table>

Table 3. Profiling results for the motion estimation module with a search window of 128x40.

<table>
<thead>
<tr>
<th>Access</th>
<th>Variable</th>
<th>Count</th>
<th>Timing</th>
</tr>
</thead>
<tbody>
<tr>
<td>read</td>
<td>pattern</td>
<td>64</td>
<td>0 ns</td>
</tr>
<tr>
<td>read</td>
<td>search_window</td>
<td>1280</td>
<td>1.280 us</td>
</tr>
<tr>
<td>write</td>
<td>results</td>
<td>1</td>
<td>1044.000 us</td>
</tr>
<tr>
<td>finish</td>
<td></td>
<td></td>
<td>1044.100 us</td>
</tr>
</tbody>
</table>

The Virtual Socket Platform can support the designer in the design exploration phase by providing, with reduced design efforts valuable information on the hardware performance and data exchanges of parts of algorithms specified as sequential or CAL models. In the case of CAL models the HDL code for the actor under profile can be generated automatically by a CAL2VHDL tool. The hardware block is then executed on the Virtual Socket Platform obtaining results on data transfers (how much, which one, when) and the execution time of the hardware module.

Tables 1,2 and 3 report the profiling results for a motion estimation module working at 50 Mhz for different configurations of the size of the search window. The motion estimation algorithm chosen for the example is a full search. It is not described in this paper for brevity, more information is available in [7]. The platform provides the type of access to the memory (Access), the names of the variables (relative to the SW algorithm) requested by the module during execution (Variable), the time at which each request is done (Timing), the amount of data transferred (Count) and the execution time.

The timing results reported in the tables correspond to the execution of the motion estimation algorithm on an unique macroblock: it computes only one motion vector (a pair of x and y coordinates) for the given macroblock. The time taken under account for the results are focused only on the HDL module. The time taken by the Virtual Socket Platform to feed the HDL module with data coming from the main memory (on the host) are not taken into consideration. The profiling apply only to the execution of the HDL module on the platform.

5. RELATED WORK

Based the same original platform, the work of Yifeng Qiu & al. [8] presents an integrated hardware accelerated co-design platform with a simplified system architecture for MPEG-4/H.264 video codec implementation. Generic HDL modules are used for data communications, signal exchange and video IP integration based on the concept of the virtual socket. Their work focuses on the implementation of one given standard by integrating generic HDL modules in the platform. Our work aims at profiling hardware characteristics of one or more HDL modules for design exploration purposes targeting heterogeneous platforms.

6. CONCLUSION

MARCO: TO BE DONE

7. REFERENCES


