Operating System Support for Interface Virtualisation of Reconfigurable Coprocessors

Miljan Vuletić, Ludovic Righetti, Laura Pozzi, and Paolo Ienne
Swiss Federal Institute of Technology Lausanne
Processor Architecture Laboratory
IN-F Ecublens, 1015 Lausanne, Switzerland
{Miljan.Vuletic, Ludovic.Righetti, Laura.Pozzi, Paolo.Ienne}@epfl.ch

Abstract

Reconfigurable Systems-on-Chip *(SoC) consist of large* Field-Programmable Gate-Arrays *(FPGAs) and standard processors. The reconfigurable logic can be used for application-specific coprocessors to speedup execution of applications. The widespread use is limited by the complexity of interfacing software applications with coprocessors. We present a virtualisation layer that lowers the interfacing complexity and improves the portability. The layer shifts the burden of moving data between processor and coprocessor from the programmer to the* Operating System *(OS). A reconfigurable SoC running Linux is used to prove the concept.*

1. Introduction and Goals

When interfacing application-specific coprocessors with the rest of the system, designers should respect the specific interface between the processor and the FPGA. Also, programmers must explicitly take into account availability and size of shared memories between processor and FPGA. Therefore, any new host platform requires redesigning software and hardware. Our contribution significantly reduces the complexity of the programming and design paradigms and improves the portability of codesigned applications.

The programmer of a traditional system equipped with an OS is abstracted from the characteristics of the memory system: he/she generates virtual memory addresses ignoring whether they physically exists. This illusion results in programming simplicity and code portability. The drawback is that the automatic allocation of pages by the OS is, in general, suboptimal. Figure 1 shows software (*Virtual Memory Manager*—VMM) and hardware (*Memory Management Unit*—MMU) components of a virtual memory system.

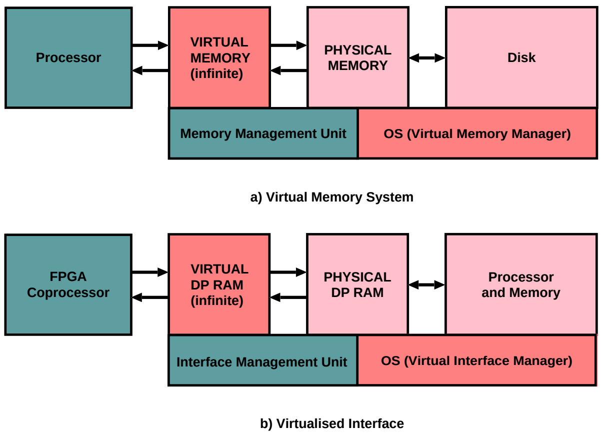

Figure 1. Virtualisation.

Our goal is to describe applications in high-level languages and corresponding coprocessors in hardware description languages independently of the target hardware. An augmented OS, a compiler, and a synthesiser must be sufficient to port the accelerated application across different systems.

Virtual Interface Management. The programmer of a reconfigurable computer should design data exchanges between the processor and the coprocessor independently of the physical system. The coprocessor designer should generate abstract addresses rather than physical ones. To support this abstraction, two components are added to the basic system: (1) A hardware support for the translation from abstract to real addresses (*Interface Management Unit*—IMU). (2) A software support in the OS to place dynamically data objects on the interface (*Virtual Interface Manager*—VIM). Figure 1 shows a practical instance of virtualised interface.

1530-1591/04 $20.00 (c) 2004 IEEE

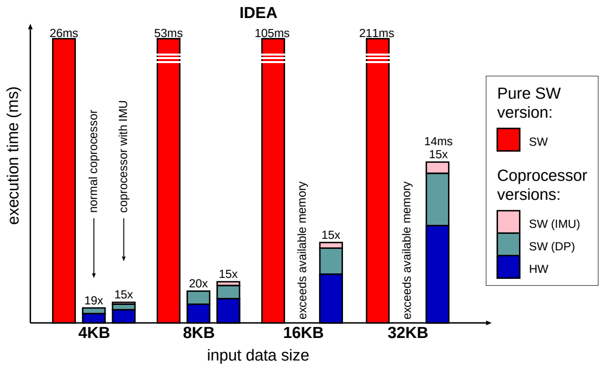

Figure 2. Measurements on IDEA kernel.

2. Interface Virtualisation Components

Three components implement the virtualisation:

(1) OS Services. Three system calls are provided to software designers. Firstly, `FPGA_LOAD` loads a coprocessor definition in the reconfigurable hardware. Secondly, `FPGA_MAP_OBJECT` identifies the data used by the coprocessor. Finally, `FPGA_EXECUTE` performs the data mapping, initialises the IMU, and launches the coprocessor.

(2) Hardware Interface. All coprocessor memory accesses go through the IMU, which translates them to real addresses. If no translation is possible, the OS is requested to handle the translation fault and dynamically place missing data to interfacing resources (e.g., shared or dual-ported memory) between processor and coprocessor.

(3) Interface Management. If interrupted by the IMU, the OS rearranges the current mapping to the on-chip memory—logically organised in pages—to resolve the page fault. Once the interrupt is resolved, the coprocessor exits from the stalled state and continues. During operation (if needed) and at task completion, the interface manager copies the produced data back to the user space.

3. Experimental Setup and Demonstration

A VIM system was implemented using a board based on the Altera Excalibur device (with an ARM processor and FPGA) and running the Linux OS. The IMU is designed in synthesizable VHDL and interfaces the coprocessor with a dual port memory. The VIM is realised as a kernel module. The IDEA cryptographic algorithm is implemented for demonstration. The software uses the VIM services to call the coprocessor interfaced with the IMU.

Figure 2 shows execution times for three versions of the application: pure software, a classic coprocessor, and a VIM-based coprocessor. For the VIM-based version, three components of the execution time are measured: (1) hardware time, (2) software time for the dual port memory management, and (3) software time for the IMU management. Even with virtualisation, coprocessors provide significant advantage. The overhead for the IMU management is acceptable (5–7% of the total execution time). There is a hardware execution overhead of up to 20% comparing to the typical coprocessor. This is due to the FPGA implementation of the IMU and it could be lowered. The experiments for the VIM-based version are performed by simply changing the input data size, *with no need to modify neither the application code, nor the coprocessor design, even when the data sets exceed the capacity of the available dual-port memory*.

4. Related Work

Standardised buses [3] and memory wrappers [2] make the details of the underlying memory interface transparent to the designer. Our idea is not in the standardisation of the interface details but in the dynamic allocation of the interfacing memory. Some researchers have considered managing reconfigurable lattice across different tasks [4] and reconfigurable hardware virtualisation [1] providing the illusion of infinite resources. The type of virtualisation we introduce is orthogonal and complementary to these.

5. Conclusions

In this work, we add a Virtual Interface Manager to a reconfigurable computing platform in order to achieve a straightforward programming paradigm, and ease the portability of applications. The approach is tested on a real system by running a complex cryptographic algorithm enhanced with an application-specific coprocessor. A significant speed-up is achieved while the virtualisation overhead is shown to be acceptable.

References

[1] M. Dales. Managing a reconfigurable processor in a general purpose workstation environment. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, Munich, Mar. 2003.

[2] F. Gharsalli, S. Meftali, F. Rousseau, and A. A. Jerraya. Automatic generation of embedded memory wrapper for multiprocessor SoC. In *Proceedings of the 39th Design Automation Conference*, New Orleans, La., June 2002.

[3] C. K. Lennard, P. Schaumont, G. De Jong, A. Haverinen, and P. Hardee. Standards for system-level design: Practical reality or solution in search of a question? In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, pages 576–583, Paris, Mar. 2000.

[4] H. Walder and M. Platzner. Online scheduling for block-partitioned reconfigurable devices. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, Munich, Mar. 2003.