Software Implementation vs. Hardware Implementation: The Avionic Test System Case-Study

George Afonso, Rabie Ben Atitallah, Jean-Luc Dekeyser

Software Implementation vs. Hardware Implementation: The Avionic Test System Case-Study

George AFONSO
EADS-INRIA Lille Nord Europe
george.afonso@inria.fr

Rabie Ben Atitallah
LAMIH, University of Valenciennes
rabie.benatitallah@univ-valenciennes.fr

Jean-Luc Dekeyser
LIFL, USTL, INRIA Lille-Nord Europe
jean-luc.dekeyser@lifl.fr

ABSTRACT

This paper presents a development methodology that helps designers to map efficiently applications onto a heterogeneous CPU/FPGA system. An industrial case study is presented and aims at meeting performance and real-time requirements with the help of our architecture capabilities and avionic model parallelization. Different avionic model implementations will be presented in order to explain how to find the best trade-off between performance and design-time.

1. INTRODUCTION

In order to meet real-time requirements, power, and flexibility goals, combination of general CPU and reconfigurable fabrics like Field-Programmable Gate Aarrays (FPGAs) is a promising solution leading to heterogeneous computing. In such systems, multi-core CPU provides high computation rates while the reconfigurable logic offers high performance per watt and adaptability to the application constraints. Due to the high parallelism rate of the application, FPGA technology could offer better performances comparing to CPUs or GPUs up to 10x [1] at lower frequencies. Designers could exploit the existing partitioning of the architecture which leads to several possible implementations with different performances. The main focus of this paper is the task mapping taking into account the different constraints of our application

2. DESIGN METHODOLOGY FOR CPU/FPGA ARCHITECTURE

In order to perform a complete Test and Simulation session, we need to simulate each embedded part of the helicopter (i.e. automatic pilot system, navigation, etc.), the environmental parameters (weather conditions, geographical factors, etc.), and the behaviour of the aircraft. This simulation models will be implemented in a software and/or hardware fashion in order to satisfy the timing constraints. In order to get the best performance, sequential model profiling must be performed in order to exploit the parallelism level inherent to our functions. Thus, we will determine the best trade-off between parallel software, hardware or software/hardware execution.

.

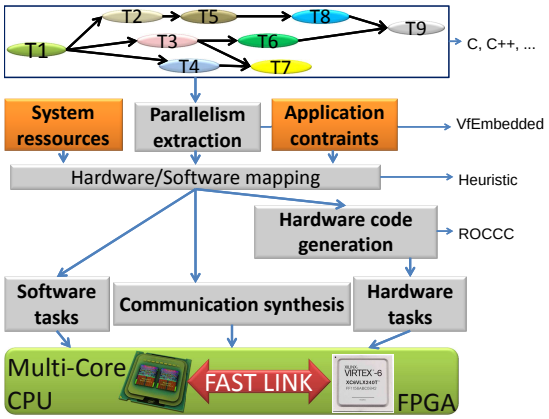

Figure 1: Design methodology for efficient test mapping

To overcome this challenge, we present a design methodology that covers the different development steps from software specification to the system implementation as shown in Fig. 1. First, we are considering a software application presented as a task graph containing different communicating functions (T0, T1, etc.). All applications are not adequate to be implemented onto heterogeneous CPU/FPGA architectures; a complete analysis of the source code is needed to verify what implementation could bring better performances. In order to leverage the parallelism of multi-core CPU/FPGA architecture, tools such as Vector Fabrics VfEmbedded [2] can find all data dependencies by analysing the C or C++ source code and extract the parallelism intrinsic in the application. vfEmbedded analyses partitions and maps applications on specific platforms from single processors to heterogeneous ones. It can also estimate the performance of the parallelized software before implementing it. Moreover, it can trim any overhead in your hardware to reduce cost and ensure that all critical behaviours in your program are exercised. The mapping step requires a heuristic method that takes as inputs system resources, application constraints, and the results of the analysis step. It will find the best configurations which satisfy timing requirements and resources utilization. After the mapping step, we need to develop some user hardware applications from the existing functions. To make this step more efficient, tools such as Riverside Optimizing Compiler for Configurable Computing (ROCCC) [3] can focus on FPGA-based code acceleration from a subset of the C language. ROCCC does not focus on the generation of arbitrary hardware circuits. Its objectives are to maximize parallelism within the constraints of the tar-

get device, optimize clock cycle time by efficient pipelining, and minimize the used area. Finally, a compilation step using GCC and ISE from Xilinx, can be easily performed in order to map functions respectively onto the CPU and the FPGA parts.

3. EXPERIMENTAL RESULTS

In this section, we will analyse different avionic models in order to obtain different possible implementations required to tune the design according to the needed performance and the real-time constraints. These implementations are also useful to switch between software and hardware configurations or vice-versa.

First, as our objective is to analyse our software models in order to get better performance in our heterogeneous multi-core CPU/FPGA architecture, we will profile six different software models. To do so, we used VfEmbedded [2] tool presented previously which offers a "parallelize" function analyses all loops and parallelizes them, correctly handling the dependencies.

Table 1: Avionic models analysis

Results	Speed-up	Max. of useful threads	Synchronization overhead
Model A	1.2	2	1%
Model B	0	1	0%
Model C	2.4	3	0%
Model D	3.5	6	39%
Model E	3	4	29%
Model F	486.7	10000	95%

Table I summarizes the experimental results obtained by analysing software avionic models on VfEmbedded with x86 Intel i5 processor execution support. First, we measure the speed-up obtained after optimization. Secondly, threads must be created for parallel implementation strategy. This might be implemented through the use of POSIX calls creating the threads. The maximum useful number of threads is directly linked to the parallelism degree of the application. But threads means synchronisation and more frequent synchronization of small amounts of data means less delay while waiting for data and therefore less latency. Synchronization requires overhead and this could require lot of time and make parallelization step fail. VfEmbedded shows a 1.2 speed-up for the model A with low synchronization overhead with only two threads. Model B cannot make profit from a parallelization strategy. These two first models are more suitable to be implemented on a single core architecture. Model C, D and E offer higher parallelism degree with low synchronisation overhead for model C. These models are suitable for multi-core architecture implementation or hybrid multi-core CPU/FPGA implementation by splitting the models in different functions and implement them in different calculation nodes. This, is also possible because of the low synchronization overhead. Finally, Model F shows a high parallelism degree with high synchronisation overhead, this model is very adapted to a hardware implementation, in the second part of our results we will generate VHDL code using ROCCC compilation framework.

Secondly, using our previous results, we decide to implement model F in a hardware fashion using a C to VHDL translator. The model F main loop fits perfectly to such transformation. With the help of the ROCCC tool, a compilation step from C to VHDL is performed. Furthermore, we make profit from the loop unrolling feature provided by ROCCC in order to obtain varied implementations for the model F main loop. We synthesised 17 samples with varying the loop unrolling value. Implementation results are reported in the Fig. 2. We get an execution time varying from 19 to 14 μs respectively with an area utilization varying from 5,800 to 14,300 slices. Indeed, more we parallelize the model F main loop, the occupied hardware increases and the execution time is reduced. The software implementation on the host offers an execution time equals to 18 μs. The selection of the best implementation will depend on the global system constraints, the data mapping, and the available resources. ROCCC shows some limitations on more complex models, in this case manual coding becomes necessary. Using our previous results, we decide to implement model A in order to observe the behaviour of such model in a pure VHDL hardware implementation. The pure software version, we obtained a 2 μs execution time with a Quad-core processor (4x2.5 GHz). Despite VfEmbedded does not show a large parallelism degree for this model, most of the mathematical operations do not have data dependency between them. A pure VHDL implementation offers a 2 μs execution time with 8% space occupation (Xilinx ML605 Board) which is related to the pure software execution time. This result offers the opportunity to move model A from a processor to the FPGA in order to bring performance and respect the needed real-time constraints at the price of the hardware design time.

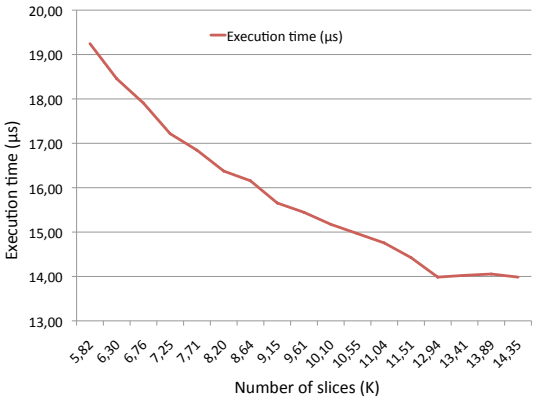

Figure 2: Different implementations for model F

4. CONCLUSION

In this paper, we have highlighted the benefits of some profiling tools making Test & Simulation model optimisation easier on the CPU/FPGA system. In future works, our investigation will concern a run-time task mapping for dynamically reconfigurable system which will map optimally highly communicating hardware and software tasks and deals efficiently with the violation of timing constraints.

5. REFERENCES

[1] S. Asano, T. Maruyama, and Y. Yamaguchi, "Performance Comparison of FPGA, GPU AND CPU in Image Processing," in *19th IEEE International Conference on Field Programmable Logic and Applications, FPL*, Prague, Czech Republic, Aug. 2009.

[2] Minjang Kim and Hyesoon Kim and Chi-Keung Luk, "A scalable approach to dynamic data-depedence profiling," in *43rd Symposium on Microarchitecture (MICRO)*, December 2010.

[3] N. W. Villarreal Jason, Park Adrian and H. Robert, "Designing Modular Hardware Accelerators in C with ROCCC 2.0," in *18th Symposium on Field-Programmable Custom Computing Machines (FCCM'10)*, Washington, USA, May 2010.