FASTER: Facilitating Analysis and Synthesis Technologies for Effective Reconfiguration

D. Pnevmatikatos[1], T. Becker[5], A. Brokalakis[8], K. Bruneel[3], G. Gaydadjiev[4], W. Luk[5],
K. Papadimitriou[1], I. Papaefstathiou[8], O. Pell[6], C. Pilato[2], M. Robart[7], M. D. Santambrogio[2],
D. Sciuto[2], D. Stroobandt[3], and T. Todman[5]

[1]Foundation for Research and Technology - Hellas, Greece
[2]Politecnico di Milano, Italy
[3]Ghent University, Belgium
[4]Chalmers University of Technology, Sweden
[5]Imperial College London, United Kingdom
[6]Maxeler Technologies, United Kingdom
[7]ST Microelectronics, Italy
[8]Synelixis, Greece

Contact Email: pnevmati@ics.forth.gr

***Abstract*—The FASTER project aims to ease the definition, implementation and use of dynamically changing hardware systems. Our motivation stems from the promise reconfigurable systems hold for achieving better performance and extending product functionality and lifetime via the addition of new features that work at hardware speed. This is a clear advantage over the more straightforward software component adaptivity. However, designing a changing hardware system is both challenging and time consuming.**

The FASTER project will facilitate the use of reconfigurable technology by providing a complete methodology that enables designers to easily specify, analyse, implement and verify applications on platforms with general-purpose processors and acceleration modules implemented in the latest reconfigurable technology. To better adapt to different application requirements, the tool-chain will support both region-based and micro-reconfiguration and provide a flexible run-time system that will efficiently manage the reconfigurable resources. We will use applications from the embedded, high performance computing, and desktop domains to demonstrate the potential benefits of the FASTER tools on metrics such as performance, power consumption and total ownership cost.

I. Introduction

Extending product functionality and lifetime requires constant addition of new features to satisfy the growing customer needs and the evolving market and technology trends. Software component adaptivity is straightforward but in many cases it is not enough; recent products incorporate hardware accelerators to satisfy performance and energy requirements. These accelerators also need to adapt to the new requirements. Reconfigurable logic allows the definition of new functions to be implemented in dynamically instantiated hardware units, combining adaptivity with hardware speed and efficiency. However, designing a changing hardware system is both challenging and time consuming.

The FASTER project [1] aims to provide a complete methodology that will enable designers to easily implement and verify applications on platforms with one or more general-purpose processors and multiple acceleration modules implemented in the latest reconfigurable technology. Previous research and EU projects such as hArtes [2], Reflect [3], ACOTES [4], Andres [5], Morpheus and others focus on the necessary tool-chain and address similar issues as FASTER but focus more on system-level or architectural aspects of reconfiguration. Moreover, they do not explicitly emphasize on the design and runtime aspects of partial and dynamic reconfiguration, or on choosing the best reconfiguration grain-size.

This is exactly where FASTER intends to contribute. FASTER aims to introduce partial and dynamic reconfiguration from the initial design of the system all the way to its runtime use. The FASTER tool-chain (depicted at high-level in Figure 1) input will be based on high-level programming languages with an initial decomposition described using existing formalisms (such as OpenMP). This input will be transformed to the corresponding task graph, which in turn will be partitioned in space and time to identify candidates for reconfiguration. FASTER will support both region- and micro-reconfiguration (a technique that reconfigures very small parts of the device), an ability that opens up a new range of application opportunities for run-time reconfiguration.

FASTER will address the issue of verifying static and dynamic aspects of a reconfigurable design while minimizing

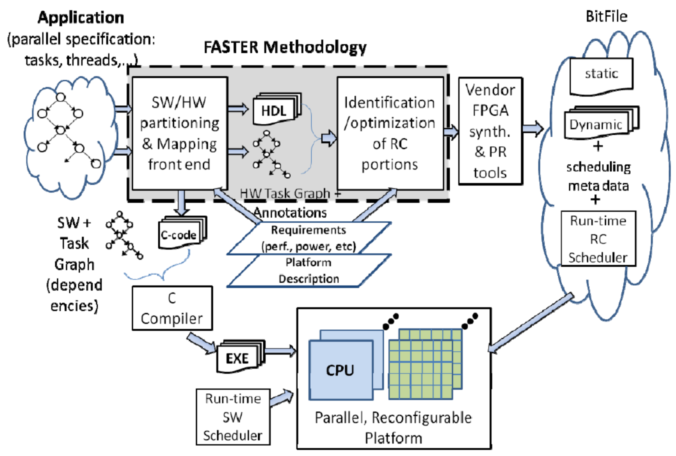
Application
(parallel specification: tasks, threads,...)
FASTER Methodology
SW/HW partitioning & Mapping front end
HDL
Identification /optimization of RC portions
Vendor FPGA synth. & PR tools
BitFile
static
Dynamic
+
scheduling meta data
+
Run-time RC Scheduler
HW Task Graph + Annotations
Requirements (perf., power, etc)
Platform Description
SW + Task Graph (depend encies)
C-code
C Compiler
EXE
Run-time SW Scheduler
CPU
Parallel, Reconfigurable Platform

Figure 1. Abstract View of FASTER Tool-Flow

run-time overheads on speed, area and power consumption, and provide a powerful and efficient run-time system for managing the various aspects of parallelism and adaptivity.

To demonstrate the potential benefits of the FASTER tool-chain we will use complex applications from the embedded, the HPC, and the desktop domains, and use metrics such as performance, power consumption, total ownership cost, verification effort, etc. We will also use prototype platforms to evaluate the speed, cost, and power consumption of the applications implemented within FASTER.

In the rest of this paper we describe the front-end analysis and the possible granularities of reconfiguration in Sections II and III. We discuss our approach to verification and the run-time support in Sections IV and V. We describe the platforms and applications we plan to use in Section VI.

II. The FASTER Front-End Analysis

A. *Context definition*

Different frameworks have been proposed to address the concurrent development of the architecture and the application. Ptolemy [6] is an environment for simulating and prototyping heterogeneous systems with mechanisms for modelling multiple abstraction levels. Daedalus [7] is a system-level design framework, composed of several tools that range from automatic parallelization for Kahn Process Networks to design space exploration of both the architectural and platform levels, and to the synthesis of the candidate platform architecture. The different components are interfaced through XML files. FASTER adopts a similar approach, where the different tools such as estimation tools and partitioning algorithms are interfaced with XML files, while the partitioned application is described with OpenMP. OpenMP is a standard format for describing parallel applications, successfully adopted also in the FP6 hArtes project [2]. Another framework for programming heterogeneous platforms is OpenCL, an open royalty-free standard for cross-platform, parallel programming of modern processors found in personal computers, servers and hand-held/embedded devices [8].

None of the existing approaches is able to automatically turn a large application into a complete design, due to the complexity of the problems and the size of the design space, especially when hardware accelerators are involved or reconfigurability is addressed. The goal of FASTER is to overcome these limitations by obtaining a general formulation, capable to deal with multiprocessor systems, supporting different hardware implementations for the same task (also exploiting micro-architectural optimizations) and proposing a design flow that efficiently partitions the application, while performing considerably more exploration on the possible solutions for the problem.

B. *The FASTER Front-End Flow*

The starting point of the FASTER Front-End can be a C application, whose initial decomposition could be described with any of the existing formalisms (e.g. OpenMP) potentially annotated with pragmas to specify additional information provided by the designer. The corresponding task graph is then derived, where every hardware task is to be treated as a region-reconfigurable module or a micro-reconfigurable module.

We consider both static and dynamic reconfiguration and assume devices are partially reconfigurable, either in a single-FPGA or a multi-FPGA system. We also perform hardware/software partitioning to determine which tasks can be executed in software instead. Finally, we will apply a methodology to schedule the resulting specification on the same reconfigurable architecture. With respect to this, one of our goals is to further reduce the dynamic hardware generation and reconfiguration overhead, especially by optimizing the applications and improving the locality of the parameter values that trigger a reconfiguration each time they change. In the past, several alternative configuration architectures were presented, such as multi-context, relocation/defragmentation and pipeline reconfigurable FPGAs. Configuration caching techniques try to keep configurations that will be needed in the future on chip. With respect to these techniques, our approach is compatible with these architectures, since we can bring different uses of a configuration closer in time with loop transformations.

To implmement the proposed front-end the work is divided in 4 tasks that can be potentially repeated in a loop to improve the partitioning of the application by means of the results obtained in the different phases. The methodology is outlined in Figure 2. In particular, the work is organized as follows:

- Task T2.1 aims to provide statistical prediction of metrics such as reconfiguration time, area, performance, energy efficiency and power consumption. It receives as input the source code (the initial one or the one

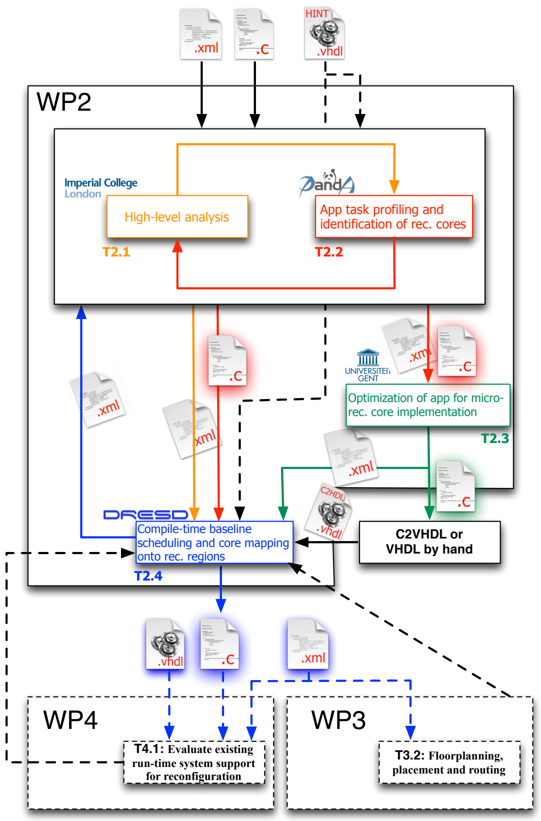
.xml
.c
HINT
.vhdl
WP2
Imperial College London
High-level analysis
T2.1
PandA
App task profiling and identification of rec. cores
T2.2
.c
.xml
.xml
UNIVERSITEIT GENT
.xml
.c
Optimization of app for micro-rec. core implementation
T2.3
.xml
.c
C2HDL
.vhdl
DRESD
Compile-time baseline scheduling and core mapping onto rec. regions
T2.4
C2VHDL or VHDL by hand
.vhdl
.c
.xml
WP4
T4.1: Evaluate existing run-time system support for reconfiguration
WP3
T3.2: Floorplanning, placement and routing

Figure 2. The overall presentation of the proposed flow.

after the decomposition) and the XML file containing information about the target architecture. Then, it regenerates the XML file containing the information about the characterization of the analyzed elements. In the same way, it removes obsolete implementations, updates the existing ones or introduces new ones.

- Task T2.2 takes care of decomposing the initial application into tasks and assigning them to the different components of the architecture. It receives the initial source code and the XML file containing the description of the target architecture. It can also receive information about the performance of the current tasks, if any, and feedbacks after the execution of the schedule (e.g. how the portioning is impacting the computed schedule). It decomposes the application or it applies further transformations in order to improve the current solution. Finally, it generates the different tasks in a C-based description and the representation of the corresponding task graph in the XML file (application part).
- Task T2.3 receives the descriptions of the different tasks (i.e., corresponding source code) and produces optimized versions for their implementation. Accordingly, it updates the XML file with the new implementations. These modifications will be done in terms of changes of parameters values using the micro-reconfiguration technique.
- Task T2.4 determines the schedule of the tasks, along with the mapping of hardware cores onto reconfigurable regions. It receives the XML file of the application and the architecture, focusing on the task assigned to the reconfigurable area. In particular, it determines the number and the characteristics (e.g., size) of the reconfigurable areas, the number and the size of each input/output point, and also takes into account the interconnection infrastructure of the system (e.g., bus size). It also schedules the resulting implementation and annotates the characterization part with such information to further refine the specification. Finally, it annotates the tasks with information about the feasibility of the implementation where the solution is specified.

Two different scenarios are shown in Figure 3. On one hand we envision a scenario where the framework proposed in FASTER has to take care of the HW/SW partitioning phase without having any constraints/hints provided by the application designer. This is depicted in Figure 3 as Scenario A. The task graph is provided as input with the information about the physical architecture that will be used to implement the final system. After the HW/SW partitioning phase (T2.2), the identified HW components are processed by T2.3 to investigate the opportunity of having an enhanced version of these elements. Finally, T2.4 will compute the baseline schedule and the floorplanning and placement constraints, via the definition of the UCF (User Constraint File). The baseline schedule will be used to map each HW element into an area defined in the UCF.

On the other hand, in Scenario B of Figure 3, we consider the case where the application designer provides some constraints/hints on the HW/SW partitioning of the desired application. A hint is a suggestion that can be considered or not by the HW/SW partitioning phase in T2.2, while a constraint is an information that cannot be ignored. Within this context, all the elements identified as HW components by the application designer will be treated in this way, without any further investigation of other possible SW implementation by the FASTER framework.

III. Region-based and Micro-reconfiguration Support

Both options will be supported with each one offering certain advantages in different cases.

A. *Region-based reconfiguration*

Partially reconfigurable regions are determined off-line. Their size can vary depending on the application and designer needs and they can be reconfigured while the rest of the chip is operational. We will use the vendor's tool flow to support the region-based reconfiguration. The research

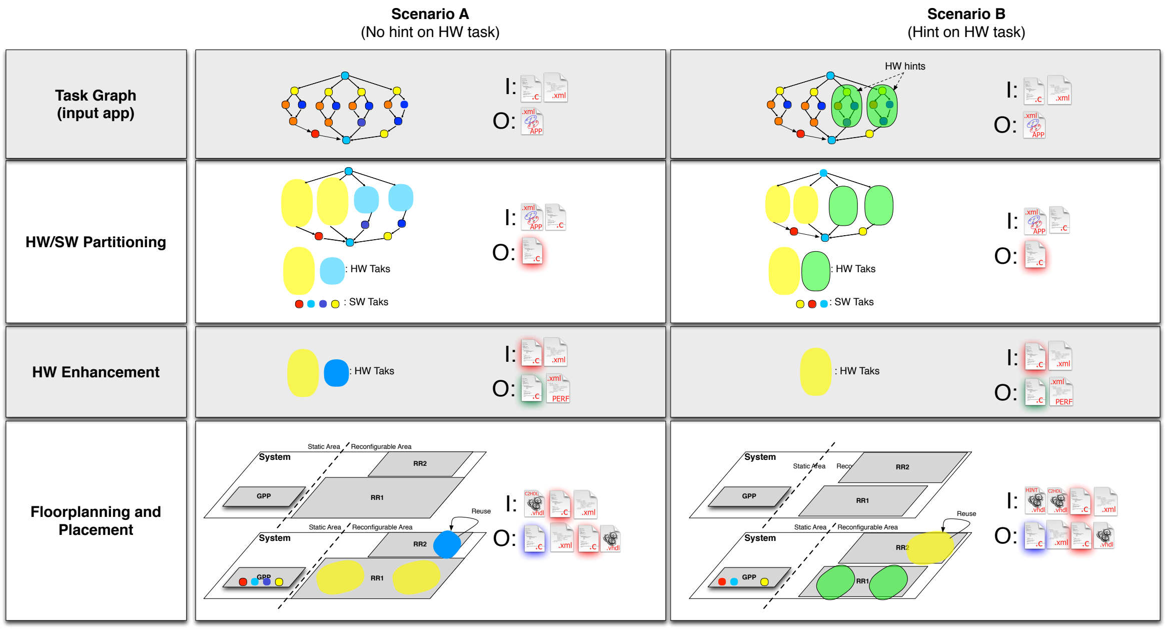

Figure 3. An overview of different scenarios for the proposed flow.

challenge is the proper identification of the regions and the support of relocatable modules at run-time, which could involve additional constraints for floorplanning and placement. The problem of floorplanning in the domain of partially reconfigurable FPGAs has steadily attracted the interest of the research community [9], [10]. Further, we intend to explore the capability of creating relocatable bitstreams that can be used in multiple locations of heterogeneous FPGA fabric. This will allow to reduce the size of configuration storage because only one bitstream version for each module has to be stored.

B. *Micro-reconfiguration*

One or more of the aforementioned regions can also be reconfigured on a much finer granularity (which we call micro-reconfiguration) to implement Runtime Circuit Specialization (RCS). RCS [11], [12] is a technique to specialize an FPGA configuration at runtime according to the values of a set of parameters. The general idea of RCS is that before a task is deployed on the FPGA a configuration that is specialized for the new parameter values is generated. Since specialized configurations are smaller and faster than their generic counterpart, the hope is that the system implementation will be more cost efficient when using RCS. Currently, the design tools of FPGA manufacturers only support region-based runtime reconfiguration (where a limited number of functionalities are timeshared on the same piece of FPGA area).

Since mapping a hardware specification to FPGA resources is an NP-complete problem [13] the specialization process will generate sub-optimal solutions. Therefore one can see that there will be a tradeoff between the resources spent on the specialization process and the quality of the specialized FPGA configuration. The more resources spent on generating the specialized functionality, the fewer resources needed to implement the specialized functionality. This means that there is a range of Pareto-optimal implementations for the specialization process. The optimal implementation depends on the design specification. One obvious choice would be to use the region-based reconfiguration tool flow. This approach works fine if the number of configurations is small. However, in RCS the number of parameter values grows exponentially with the number of bits needed to represent the parameter data. Hence, generating all configurations off-line and storing them in a database rapidly becomes infeasible for real-life applications. Therefore, the only option is to have the specialization process generate the partial configurations at run time, by using a (simplified) run-time tool flow, which is slow. In the FASTER project, we propose the use of parameterized configurations [14]. Using parameterized configurations, RCS implementations that have similar properties to handcrafted applications can be automatically built. The method builds on the observation that the specialization process actually implements a multi-valued Boolean function, which is called a Parameterized Configuration (PC). Indeed, both the input (a parameter value) and the output (a specialized configuration) of the specialization process are bit vectors.

The method make use of staged compilation. During the generic stage (Figure 4 (a)), a parameterized configuration is constructed and represented in a closed form starting from a parameterized HDL description. This stage is executed at

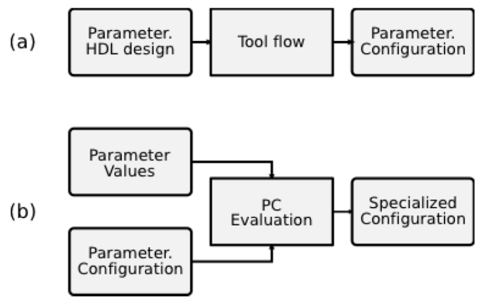

Figure 4. Staged compilation as used by RCS techniques that make use of parameterized configurations: (a) The ofline stage of the tool flow (b) The online stage

compile time, when the parameter values are not known. During the specialization stage (Figure 4 (b)), a specialized configuration is produced by evaluating the parameterized configuration given a parameter value. This stage is executed at run-time, after the parameter values have become available. After producing the specialized configuration, it is used to reconfigure the FPGA.

A parameterized HDL description is an HDL description in which distinction is made between regular input ports and parameter input ports. The parameter inputs will not be inputs of the final specialized configurations. Instead, they will be bound to a constant value during the specialization stage.

It is important to note here that while the problem that needs to be solved by the generic stage of our staged compilation tool flow is computationally hard, the problem that needs to be solved by the specialization stage, evaluating the parameterized configuration, is not. This drastically reduces the specialization time.

IV. Verification of Changing Systems

This section describes our approach to *verification*: ensuring that the optimized, reconfiguring design preserves the behaviour of the original one. In the overall FASTER tool flow, verification uses the mapped design as a reference to compare with designs using reconfiguration.

Traditional approaches simulate the reference and optimized designs with a set of test inputs, comparing the outputs. This approach works well, but the test inputs must cover all aspects of the design's behaviour. There is always a danger that the test inputs do not cover all the cases, or that the output is only coincidentally correct.

A. Our approach

Rather than relying on numerical or logical simulation, our approach combines *symbolic simulation* with *equivalence checking*. Symbolic simulation applies symbols rather than numbers or logic values to the design, and the outputs are functions of these symbolic inputs. For example, symbolically simulating an adder with inputs a and b might result in $a + b$. However, for larger designs it is harder to distinguish different but equivalent outputs ($b + a$ instead of $a + b$) from incorrect ones. The equivalence checker tests whether or not the outputs of transformed designs are equivalent to those of the reference design.

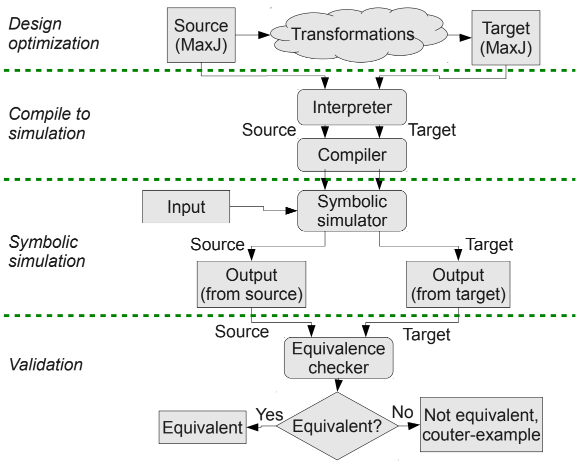

Figure 5. Verification design flow.

Figure 5 shows our approach, where we compare a reference design (the *source*) with a transformed design (the *target*). For the FASTER project, we compare designs implemented as Maxeler MaxJ kernels, though our approach could apply to other kinds of hardware design descriptions, or to software. The verification happens in four phases:

- *Design optimization*: the rest of the FASTER design flow transforms a source design to a target;
- *Compilation for simulation*: compile the MaxJ kernel into input for the symbolic simulator in two phases: (a) the program is interpreted to unroll any compile-time loops in the MaxJ design, and (b) the design is compiled to a symbolic simulation input using a syntax-directed compile scheme;
- *Symbolic simulation*: a symbolic simulator applies symbolic inputs to source and target designs;
- *Validation*: the Yices equivalence checker [15] compares the outputs of source and target designs, resulting in either success (source and target designs match), or failure, with a counter example showing why the designs are not equivalent.

B. Verifying dynamic aspects of the design

The FASTER tool flow generates designs using run-time reconfiguration, which is not supported by symbolic simulators or equivalence checkers. Rather than modify these tools, or try to switch between multiple configurations, we adapt the approach of Luk et. al. [16], modelling run-time reconfiguration as *virtual multiplexers*, which model switching between different configurations with the same configurations connected by a multiplexer-demultiplexer pair. We compile the run-time reconfigurable parts of designs to be enclosed by multiplexer-demultiplexer pairs. We modify the configuration controller to generate the control inputs to the multiplexers to choose the appropriate configuration. Our approach can apply equally to static, region-based reconfiguration, or micro-reconfiguration without modification.

V. Run-time System Support

The Run-Time System Manager (RTSM) is a software component supporting the execution of application workloads mainly in systems controlled by an Operating System (OS). Its scope is to undertake low-level operations so as to offload the programmer from manually handling fine grain operations such as scheduling, resource management, program optimization, memory savings and power consumption. RTSM can reside always in memory and is usually implemented as a standard library. It contains subroutines that realize functions by accessing the Operating System (system calls). Depending on the situation, part of the RTSM lies in the user space while another part can remain in kernel space.

In a partially reconfigurable FPGA-based system, in order to manage dynamically the HW tasks the RTSM needs to be extended with specific operations [17]. Figure 6 illustrates our envisioned system model which demonstrates the components participating in run-time system operation [18]. The FPGA is managed as a 2D area with regard to the HW task placement (a HW task corresponds to a HW module). Loading of tasks is controlled by a General Purpose Processor (GPP) while programming of FPGA configuration memory is done through the ICAP configuration port. All tasks can have both SW and HW versions available. Typically, HW tasks are predesigned (synthesized at compile time), and stored as partial bitstreams in a repository (omitted from Figure 6 for clarity), which accords with the restrictions of the Xilinx FPGA technology. Each task is characterized by three parameters: task area (width and height), reconfiguration time, and execution time. In Figure 6, four distinct components, most of them implemented outside the reconfigurable area participate in the control of tasks:

- Placer (P): responsible for finding the best location for the task in the FPGA
- Scheduler (S): finds the time slot in which a task will be loaded/starts execution
- Translator (T): resolves the task coordinates by transforming a technology independent representation of the available area into the low-level commands for the specific FPGA
- Loader (L): communicates directly with the configuration port

The system of Figure 6 is general enough to study similar systems in the sense that architectural changes will not affect

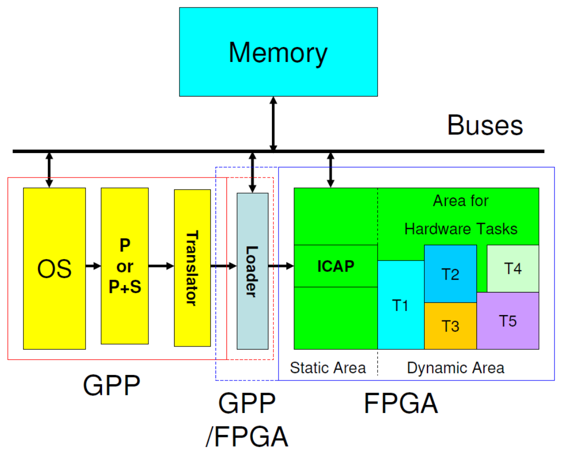

Figure 6. System model showing the components of the run-time system

its generality in terms of operation. Hence, instead of ICAP, external configuration ports can be employed such as the SelectMAP or JTAG. The GPP can be a powerful host processor (that implements Placer, Scheduler and Translator) communicating with the FPGA device through a PCI bus (e.g. desktop computing), or, it can be an embedded processor connected to the FPGA through a dedicated point-to-point link. In any case, communication latency and bandwidth should be evaluated and used effectively for optimal results.

The following operations will be supported by the RTSM to build a functional PR system. *Scheduling* will handle arriving tasks (or tasks that will arrive in the future) by placing them in proper time slots. *Area allocation* aims to define the regions that will accommodate the Partially Reconfigurable Modules (PRMs); currently in vendor's flow this is performed in design time, while dynamic area allocation is in its infancy. *Placement* should be efficiently supported within the boundaries of the partially reconfigurable regions, so as to find exact locations of the FPGA logic blocks under the constraint of minimizing the wire length required for routing. While during operation HW tasks are swapped in and out and thus the FPGA area is allocated and deallocated, it is likely that it will suffer from *fragmentation*. This can lead to a case where although the aggregate area will be sufficient to accommodate a task, at the same time it won't be able to accommodate a task that requires contiguous resources. The solution to this is offered through *relocation*, an operation that allows reorganizing the placement of the HW task. This should be performed in time periods in which the corresponding HW tasks are idle. A restriction applies with regard to the areas in which this is permitted; however there exist works that managed to relocate tasks amongst asymmetric areas and different resources [19]. Another operation that needs to be supported by the RTSM is *configuration caching*, which concerns placing the configuration data that will be needed in the future close to the configuration memory. Different configuration times have been observed depending on the type of memory used for caching and the configuration controller [20]. The RTSM will also support *configuration prefetching* which can alleviate the system from the reconfiguration overhead by configuring the logic ahead of time. Finally we consider managing power and thermal issues. We propose to do this dynamically, which can allow even distribution of power and thermal phenomena across the platform [21].

VI. Platforms and Applications

Regarding the applications that will benefit from the provided framework they cover the complete range from low-end embedded applications to high-end high performance computing (HPC) ones. In particular, in the HPC domain the efficiency of FASTER will be demonstrated with the implementation on the top of the provided framework of a Reverse Time Migration (RTM) scheme [22]. RTS is a computationally intensive geoscience algorithm that involves simulating wave propagation through the earth. The objective is typically to create an image of the subsurface from acoustic measurements performed at the surface. To create an image of the subsurface, a low frequency acoustic source on the surface is activated and the reflected sound waves are recorded by (typically) tens of thousands of receivers. We term this process a “shot”, and it is repeated many thousands of times while the source and/or receivers are moved to illuminate different areas of the subsurface. The resulting dataset is dozens or hundreds of terabytes in size. The problem of transforming this dataset into an image is computationally intensive and can be solved with a variety of techniques. RTM is a high end technique for generating images of the earth and is used in complex geologies to give detailed subsurface images.

Moving to the Desktop/Workstation domain the FASTER framework will also be utilized in the implementation of a CPU-intensive Global Illumination Scheme [23]. In particular, in future graphic applications (games, visualization, etc.), it will be important to achieve photorealistic rendering in a coherent manner, in order to greatly improve picture quality with an ever-increasing scene complexity, with support for real reflection, soft shadows, area light source, indirect illumination, etc. This is a computationally intensive problem, addressed by the increasing interest in real-time global illumination approaches. The system that will be produced by the FASTER framework should be flexible enough to accelerate different algorithms based on ray casting (ray tracing, path tracing, Monte Carlo ray tracing, etc.). In this scheme a 3D scene is described mathematically using simple primitives such as triangles, polygons, spheres, cylinders, etc. The properties of each primitive, e.g. position, orientation, scale and optical properties, are described by the scene. A virtual camera is placed into the scene, and an image is rendered accordingly in casting rays simulating the reverse path of ray of lights, from the origin of the camera through every pixel of its virtual focal plane. The color of a pixel is determined by the potential intersections of the primary ray casted through it, with the 3D scene. Photorealistic results, based on the global illumination scheme listed, can be achieved when sufficient rays are casted, simulating with fidelity the behavior of the light.

Moreover, the FASTER framework will be utilized in the implementation of a highly representative embedded application detecting possible intrusions in the network. Network Intrusion Detection Systems (NIDS)[24] are widely adopted as high-speed and always-on network access demand more sophisticated packet processing and increased network security. Instead of checking only the header of incoming packets (as for example firewalls typically do), NIDS also scan the payload to detect suspicious contents. The latter are described as signatures or patterns and intrusion signature databases are made available that include known attacks. These databases are regularly updated and an NIDS has to be able to provide a certain degree of flexibility so that it can incorporate the updated security information.

Concerning how these applications will exploit the novel features of the FASTER approach, RTM can use micro-reconfiguration to handle variations in domain size or in the number of timesteps for example. Partial region-based reconfiguration could be used to accommodate the change from imaging to propagation computation, while full reconfiguration may be necessary only when the physical model or other less frequent changes are necessary. Ray-tracing on the other hand may employ different reconfiguration methods to achieve the desire performances and flexibility. Different levels of reconfiguration can handle the various adaptable aspects of the application: from modification of the parameterization, such as secondary ray depth or pixel sampling, to more important reconfiguration of the type of supported geometric primitives, to even the global acceleration structure or the shading models used by the rendering process itself. It may even be beneficial to support reconfigurable accuracy (precision) of the intersection computations. Finally, the proposed NIDS system, may use micro-reconfiguration in order to accommodate frequent small changes to the detection rules (usually associated with IP addresses), while for larger changes to the ruleset (such as new regular expressions) region-based reconfiguration is an optimal solution. Full reconfiguration may be employed in cases where significant changes to the operation of the NIDS system are required, as for example the application of a new system policy.

Our applications target platforms from both the high performance and embedded domain. With regard to the former we will be using Maxeler Technologies platforms, which combine a pool of FPGAs for dataflow computing, conventional CPUs, networking and large storage means.

The standard compute element is the MaxNode compute node, which integrates 12 Intel Xeon CPUs and 4 dataflow compute engines. Each dataflow engine utilizes a large Xilinx Virtex-6 FPGA attached to up to 48GB of DDR3 DRAM. The FPGAs are connected to the CPUs via PCI Express. With respect to the embedded applications we will use Xilinx platforms carrying a single Virtex-5/-6 FPGA.

VII. Conclusions

Creating a changing hardware system is a challenging process, that requires additional effort in specification, implementation and verification, as well as increased support from the run-time system. We attempt to alleviate these overheads and streamline the design and implementation process providing a new tool-chain. Our contributions will span the analysis phase and the reconfigurable system definition, the support for multi-grain reconfiguration, the improved verification for the changing system, and the efficient run-time system to handle the run-time system reconfiguration requirements. Together, all these contributions will result in a design environment that will be friendly to reconfiguration and able to support multiple implementation platforms.

Acknowledgment

This work was supported by the European Commission in the context of FP7 FASTER project (#287804).

References

[1] http://www.fp7-faster.eu/.

[2] http://hartes.org/hArtes/.

[3] http://www.reflect-project.eu/.

[4] http://www.hitech-projects.com/euprojects/ACOTES/.

[5] http://andres.offis.de/.

[6] http://ptolemy.eecs.berkeley.edu/.

[7] http://daedalus.liacs.nl/.

[8] http://www.khronos.org/opencl/.

[9] A. Montone, M. D. Santambrogio, F. Redaelli, and D. Sciuto, "Floorplacement for Partial Reconfigurable FPGA-Based Systems," *International Journal of Reconfigurable Computing*, vol. 2011, no. 2, p. 12 pages, 2011.

[10] C. Bolchini, A. Miele, and C. Sandionigi, "Automated Resource-aware Floorplanning of Reconfigurable Areas in Partially-Reconfigurable FPGA Systems," in *Proceedings of the IEEE International Conference on Field Programmable Logic and Applications (FPL)*, September 2011, pp. 532–538.

[11] P. W. Foulk, "Data-folding in SRAM Configurable FPGAs," in *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines (FCCM)*, April 1993, pp. 163–171.

[12] M. J. Wirthlin and B. L. Hutchings, "Improving Functional Density Through Run-time Constant Propagation," in *Proceedings of the ACM International Symposium on Field-Programmable Gate Arrays (FPGA)*, 1997, pp. 86–92.

[13] K. Shahookar and P. Mazumder, "VLSI Cell Placement Techniques," *Computer Survey*, pp. 143–220, 1991.

[14] K. Bruneel, "Efficient Circuit Specialization for Dynamic Reconfiguration of FPGAs," PhD thesis, Ghent University, 2011.

[15] B. Dutertre and L. de Moura, "The YICES SMT Solver," Computer Science Laboratory, SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025 - USA, Tech. Rep., 2006.

[16] W. Luk, N. Shirazi, and P. Y. K. Cheung, "Modelling and Optimising Run-Time Reconfigurable Systems," in *Proceedings IEEE Symposium on FPGAs for Custom Computing Machines (FCCM)*. IEEE Computer Society Press, 1996, pp. 167–176.

[17] C. Steiger, H. Walder, and M. Platzner, "Operating Systems for Reconfigurable Embedded Platforms: Online Scheduling of Real-Time Tasks," *IEEE Transactions on Computers*, vol. 53, no. 11, pp. 1393–1407, November 2004.

[18] T. Marconi, "Efficient Runtime Management of Reconfigurable Hardware Resources," PhD thesis, TU Delft, 2011.

[19] T. Becker, W. Luk, and P. Y. Cheung, "Enhancing Relocatability of Partial Bitstreams for Run-Time Reconfiguration," in *Proceedings of the IEEE Symposium on Field Programmable Custom Computing Machines (FCCM)*, April 2007, pp. 35–44.

[20] K. Papadimitriou, A. Dollas, and S. Hauck, "Performance of Partial Reconfiguration in FPGA Systems: A Survey and a Cost Model," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 4, no. 4, pp. 36:1–36:24, December 2011.

[21] J. Donald and M. Martonosi, "Techniques for Multicore Thermal Management: Classification and New Exploration," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2006, pp. 78–88.

[22] E. Baysal, D. D. Kosloff, and J. W. C. Sherwood, "Reverse Time Migration," *SEG-Geophysics*, vol. 48, no. 11, pp. 1514–1524, 1983.

[23] K. Myszkowski, T. Tawara, H. Akamine, and H.-P. Seidel, "Perception-Guided Global Illumination Solution for Animation Rendering," in *Proceedings of the ACM Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 2001, pp. 221–330.

[24] I. Sourdis, D. Pnevmatikatos, and S. Vassiliadis, "Scalable Multi-Gigabit Pattern Matching for Packet Inspection," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 2, pp. 156–166, February 2008.