

# A Seamless Virtualization Approach for Transparent Dynamical Function Mapping targeting Heterogeneous and Reconfigurable Systems

Rainer Buchty, David Kramer, Fabian Nowak, Wolfgang Karl

Universität Karlsruhe (TH)  
Institut für Technische Informatik, Lehrstuhl für Rechnerarchitektur  
76128 Karlsruhe, Germany  
e-Mail: {buchty|kramer|nowak|karl}@ira.uka.de

**Abstract.** Future systems are not only heading towards increased parallelism, but also embrace heterogeneity and reconfigurability. We therefore present an approach targeting comfortable program development and execution, enabling full exploitation of the underlying hardware without burdening the application programmer with the details of the underlying hardware infrastructure. The approach employs lightweight resource virtualization by means of on-demand function resolution. By carefully extending the existing system infrastructure, the approach comes at virtually no cost and with highest compatibility to existing legacy code. The approach is suitable for a wide range of architectures from embedded systems to high-performance computing platforms.

## 1 Introduction and Motivation

Heterogeneity has become a key characteristic of current and future MPSoCs within converging commodity, embedded, and high-performance computing markets. This trend towards heterogeneous parallel systems is illustrated by current forecasts [4] as well as vendor road-maps [1,6]. When maximizing silicon use, an increased use of reconfigurability will be seen, enabling on-demand configuration of required application-specific hardware units.

Key problem of such architectures is fully exploiting these platforms' potential. As of now, application programmers typically have to concentrate on hardware-aware programming as opposed to the plain application itself, therefore spending most of the time for tailoring the implementation towards the architecture, producing a non-scalable, architecture-bound implementation. Naturally, this approach is not suitable for upcoming reconfigurable systems.

We therefore propose an approach that frees the programmer from the burden of hardware-aware programming. This approach dissects into two major building blocks, an augmented application description suitable for execution on heterogeneous, reconfigurable platforms, and an according runtime system performing transparent mapping of desired application subroutines or functions to individual implementations to be processed on hardware and software implementations. With respect to the increasing computational demands in embedded real-time devices, we put special focus on addressing the specific requirements of embedded systems concerning compatibility and efficiency.

By carefully using and extending the individual features of already existing system components, our approach was specifically designed towards low runtime overhead and high compatibility to existing legacy code and OS interfaces. It is therefore universally applicable to both high-performance computation platforms and embedded systems and offers significant benefits over existing approaches.

## 2 Related Work

One example of future heterogeneous systems is given by the *Cray XDI* [9] computing platform. It features AMD Opteron CPUs for general-purpose computation and dedicated, FPGA-based hardware accelerators as computation units or application-specific coprocessors. The system is interconnected using state-of-the-art communication buses.

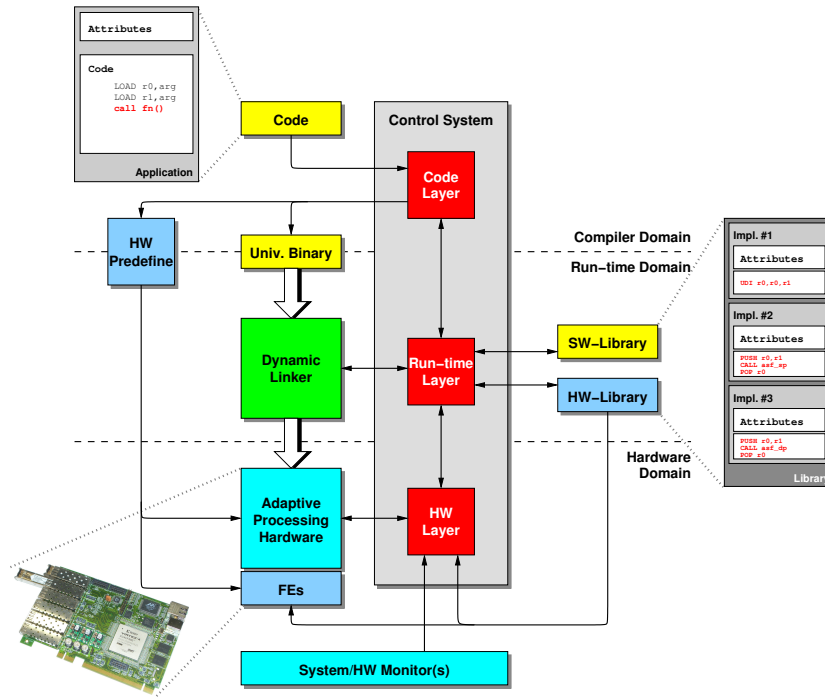
Programming of such heterogeneous multicore systems can be a tedious task; as of now, several approaches of different complexity exist. The *RapidMind* [11] is able to parallelize applications written in C/C++ across multiple cores and manages application execution, freeing developers from low-level optimizations for performance tuning. It does, however, not specifically address the use of reconfigurable systems.

Dealing with reconfigurability requires the introduction of certain abstraction, e.g. by introducing multiple stages of hardware service and software layers as it is the case in the *NoTA* architecture [16]. An abstraction on lowest hardware level is demonstrated by the *MOLEN* reconfigurable processor [12], one of several approaches of processors featuring a reconfigurable instruction set. Similar abstraction techniques are also employed within dedicated programming models for parallel systems. Examples are *EXOCHI* [15] and the derived *Merge* [10] framework that both target complete dynamization of program execution on heterogeneous systems using a library-based approach. IBM's recent *Liquid Metal* (Lime) program [13] targets the creation of a single unified programming language and environment that allows potentially all portions of a system to move fluidly between hardware and software, dynamically and adaptively. The concept bases on transparent and dynamic compilation and HW synthesis of Java programs to be co-executed on general-purpose processors and reconfigurable hardware.

The benefit of our approach compared to the listed concepts lies in its seamless expansion of existing system and development infrastructures, not requiring additional software layers, hence enabling highest level of compatibility, lowest overhead, and therefore easy integration into and interoperability with existing system architectures.

## 3 Platform Overview

Core of our approach depicted in Figure 1 is a feedback system formed by the processing hardware and its according runtime system. On this system, an application binary is executed on a reconfigurable processing hardware with according mappings to library-provided alternative implementations taking place. Hence, our approach divides into 3 distinct layers that are code generation, runtime system, and reconfigurable hardware. An application, written in conventional programming environments, is augmented in two ways: parts of the application may be marked as reconfigurable, meaning that affected code parts may be mapped at runtime to individual implementations



**Fig. 1.** Application Execution on Dynamically Reconfigurable Systems

with different properties, e.g. optimization for speed or for computational accuracy, and may require dedicated hardware to be executed on. Additional information is therefore provided defining the core requirements of affected code parts, such as hardware constraints, throughput, or computation accuracy. Using the possibilities of current binary file formats, this information is contained in each binary implementation, i.e. main program and linked libraries, and interpreted by the runtime system to guide mapping and configuration processes.

In order to adapt to the heterogeneous system in the most adequate way, an application will typically consist of a static control part and dedicated, accelerable computing functions to be mapped to the current hardware configuration at runtime. The hardware configuration might be changed according to the requirements of the application or a dedicated computing function. To guide runtime configuration and mapping, we propose an **augmented binary** where an application will be enriched by according attributes, defining specific requirements of this application. Likewise, attributes are provided for implementation alternatives of accelerable functions. Within the generated binary, these functions point to a function stub which gets resolved by the runtime system according to the provided guidance information. That way, a smooth integration and upgrade path into existing program development infrastructures is ensured. On compiler side, only the extraction of attributes needs to be performed by a pre-processor and an according declaration of a function being changeable at runtime must be applied.

This can be done by exploiting the features of the *Executable and Linkable Format* (ELF) [14]: a single ELF file dissects into several segments carrying individual sections, hence enabling monolithic, multi-architecture binaries, or including additional segments required by a specific runtime system. This way, a uniform, self-contained binary file or library can be provided using the underlying structures and ideas of the ELF specification. Both, binary file and library, can be used with either traditional runtime environments, disregarding the guidance information, or a dedicated virtualization and mapping environment targeting reconfigurable and heterogeneous systems.

To permit execution of a uniform application binary on an adaptive, heterogeneous platform, we use a lightweight **virtualization layer**, dealing with function resolution and according hardware configuration. This virtualization layer is realized as an extension of existing dynamic linking methods as outlined in [2]. We therefore kindly direct the interested reader to that publication.

As an appropriate **hardware platform** for our virtualization layer, we implemented an adaptive heterogeneous multicore system. This system comprises an AMD Opteron 870 dual-core processor and an FPGA-based accelerator board [7] featuring a Xilinx Virtex4-FX100 FPGA and on-board memory. The FPGA's resources are partitioned in a way that up to six application accelerators can be integrated into the FPGA logic. The accelerator board is connected to the Opteron processor using HyperTransport [8], therefore enabling easy and low-latency memory access to and from the accelerator board.

## 4 Evaluation

In [2] we showed that no measurable overhead exists for the basic virtualization, which during normal execution, i.e. if no switching occurs, behaves identical to the native operation mode of the OS's runtime system for single- and multi-threaded (OpenMP) execution.

Verifying our hardware architecture und demonstrating its general applicability, we chose en- and decryption of arbitrary data using Triple DES (3DES); we used the TripleDES Core provided by OpenCores [3] for hardware, and the TripleDES implementation of the standard OpenSSL library [5], available on most modern Linux systems, as the software implementation. In our experimental setup, both function implementations are encapsulated in C functions and registered using the provided API. These functions are then extended by code counting the number of needed cycles.

For evaluation, we used 6 data files of different size with a maximum chunk size of 1MB each as input data to the 3DES unit, i.e. larger files were split accordingly. To avoid hard drive latencies, the files were stored in main memory both at the beginning and after decryption. The Opteron processor operated at a clock frequency of 2GHz, the application accelerators at 100MHz with a derived HT Link frequency of 200MHz. Table 1 shows the obtained results for single-thread/accelerator execution. For small file sizes, the software implementation is faster, as no overhead for initialization of the data transfer to the hardware accelerator or to main memory occurs. For medium or large file sizes, the hardware implementation should be used, as the data transfer overhead becomes insignificant compared to computation time.

File Size	Hardware Implementation (1 Core)			Software Implementation (1 Thread)		
	Cycles	Time [ms]	Thr.put [MB/s]	Cycles	Time [ms]	Thr.put [MB/s]
1 KByte	864023	0.4	2.50	88425	0.08	12.50
10 KByte	1290012	0.64	15.63	1419408	0.70	14.28
100 KByte	4994667	2.49	40.16	14074467	7.02	14.24
1 MByte	49980110	24.93	41.08	144060568	71.86	14.25
10 MByte	499340518	249.0	41.12	1424615045	710.7	14.41
100 MByte	4993356084	2491	41.12	14210768748	7089	14.45

**Table 1.** Runtime depending on the data size

## 5 Conclusion and Outlook

Future systems do not only feature massive parallelism, but furthermore expose a high degree of heterogeneity and will employ reconfigurability as a means to maximize silicon use. In order to foster adoption of such systems and ease the migration of conventional approaches, we developed an infrastructure targeting the full exploitation of the underlying hardware of heterogeneous, reconfigurable parallel systems without burdening the programmer with details of the underlying hardware.

Core of this framework is a lightweight runtime system performing function resolution according to the current system configuration and adhering to application requirements. This process is guided by an augmented application description, enabling declaration of implementation alternatives of individual functions and providing according meta-data such as required or provided performance data. This augmentation takes place by exploiting specific features of the ELF specification, providing compatibility with conventional systems. Embracing and extending the ELF's so-called lazy-linking technique, functions are resolved on-demand to be adjusted to changed software requirements or hardware environment. The runtime system offers a dedicated control interface to monitor both application execution and hardware. Furthermore, an interface for controlling hardware reconfiguration exists.

The core components of our framework were thoroughly evaluated, proving that the virtualization layer does not contribute a significant amount to the overall execution time, but furthermore does tightly integrate into the existing runtime layer.

Based on industry-standard components, a reconfigurable accelerator hardware was built and an application example was presented, concentrating on the control aspects, i.e. whether and when to migrate from software-based implementation to a hardware-accelerated version. This is an initial step towards a fully automated control daemon employing runtime profiling and evaluation to guide the function switching process and, at a later stage, driving binary transformation.

Currently, the virtualization layer is refined to further enhance compatibility aspects. A manually driven version of the control daemon exists, which is developed into an automated version. In addition, work is conducted with respect to code generation and augmenting the ELF format, containing attributed application and library binaries.

Overall, the concept has proven to be versatile in use while maintaining maximum compatibility, providing a smooth upgrade path from conventional, static parallel systems to future dynamically reconfigurable heterogeneous multicore systems.

## References

1. Advanced Micro Devices. AMD Fusion Whitepaper. [http://www.amd.com/us/MarketingDownloads/AMD\\_fusion\\_Whitepaper.pdf](http://www.amd.com/us/MarketingDownloads/AMD_fusion_Whitepaper.pdf).
2. Rainer Buchty, David Kramer, Mario Kicherer, and Wolfgang Karl. A Light-weight Approach to Dynamical Runtime Linking Supporting Heterogenous, Parallel, and Reconfigurable Architectures. In *The 2009 International Conference on Architecture of Computing Systems (ARCS 2008)*, Delft, The Netherlands, March 2009. to appear.
3. Daniel Socek. 3DES/DES VHDL Core. 2006. [http://opencores.org/projects.cgi/web/3des\\_vhdl/overview](http://opencores.org/projects.cgi/web/3des_vhdl/overview).
4. Koen De Bosschere, Wayne Luk, Xavier Martorell, Nacho Navarro, Mike O'Boyle, Dionisios Pnevmatikatos, Alex Ramirez, Pascal Sainrat, André Sez nec, Per Stenström, and Olivier Temam. High-Performance Embedded Architecture and Compilation Roadmap. In P. Stenström and M. O'Boyle and F. Bodin and M. Cintra and S.A. McKee, editor, *Transactions on High-Performance Embedded Architectures and Compilers I (LNCS 4050)*, pages 5–29. Springer Berlin / Heidelberg, 2007. ISBN 978-3-540-71527-6.
5. Ralf S. Engelschall and The OpenSSL Project. OpenSSL: The Open Source toolkit for SSL/TLS. <http://www.openssl.org/>.
6. Jim Held, Jerry Bautista, and Sean Koehl. From a Few Cores to Many: A Tera-scale Computing Research Overview. *Research at Intel Whitepaper*, 2006. [http://download.intel.com/research/platform/terascale/terascale\\_overview\\_paper.pdf](http://download.intel.com/research/platform/terascale/terascale_overview_paper.pdf).
7. Holger Fröning, Mondrian Nüssle, David Slogsnat, Heiner Litz, Ulrich Brüning. HTX Board: A Rapid Prototyping Station. In *3rd annual FPGAworld Conference*, November 2006. <http://ra.ziti.uni-heidelberg.de/pages/papers/2006/1.pdf>.
8. HyperTransport Consortium. HyperTransport: Low latency Chip-to-Chip and beyond Interconnect. 2008. <http://www.hypertransport.org>.
9. Cray Inc. Cray XD1 Supercomputer, 2004. [http://www.cray.com/downloads/Cray\\_XD1\\_Datasheet.pdf](http://www.cray.com/downloads/Cray_XD1_Datasheet.pdf).
10. Michael D. Linderman, Jamison D. Collins, Hong Wang, and Teresa H. Meng. Merge: a programming model for heterogeneous multi-core systems. In *ASPLOS XIII: Proceedings of the 13th international conference on Architectural support for programming languages and operating systems*, pages 287–296, New York, NY, USA, 2008. ACM.
11. RapidMind, Inc. RapidMind Multi-Core Development Platform. 2008. <http://www.rapidmind.net/>.
12. S. Vassiliadis and S. Wong and S. D. Cotofana. The MOLEN  $\mu$ -coded Processor. In *11th International Conference on Field-Programmable Logic and Applications (FPL)*, Springer-Verlag Lecture Notes in Computer Science (LNCS) Vol. 2147, pages 275–285, August 2001.
13. Shan Huang and Amir Hormati and David Bacon and Rodric Rabbah. Liquid Metal: Object-Oriented Programming Across the Hardware/Software Boundary. In *Proceedings of the 2008 European Conference on Object-Oriented Programming (ECOOP)*, Jul 2008.
14. The Santa Cruz Operation, Inc. System V Application Binary Interface (Edition 4.1). 1997. <http://www.caldera.com/developers/devspecs/gabi41.pdf>.
15. Perry H. Wang, Jamison D. Collins, Gautham N. Chinya, Hong Jiang, Xinmin Tian, Milind Girkar, Nick Y. Yang, Guei-Yuan Lueh, and Hong Wang. EXOCHI: architecture and programming environment for a heterogeneous multi-core multithreaded system. *SIGPLAN Not.*, 42(6):156–166, 2007.
16. NoTA World. NoTA – Open Architecture Initiative. 2008. <http://www.notaworld.org/>.