# Architectures and Tools for Heterogeneous Reconfigurable Systems

Andreas Koch
Tech. Univ. Braunschweig, Dept. E.I.S., Mühlenpfordtstr. 23 D-38106 Braunschweig
koch@eis.cs.tu-bs.de

*We present a brief overview over multi-year research dealing with heterogeneous reconfigurable systems After discussing architecture issues, we show some initial results of work on a compiler automatically mapping ANSI C to a heterogeneous reconfigurable target system.*

## 1 Introduction

In the endless quest for increasing computing power, reconfigurable (sometimes called *adaptive*) architectures are a promising addition to conventional processors. Adaptive computers exploit the capability to mold part of their underlying hardware specifically to the needs of individual algorithms [1][2]. In contrast to ASICs, they retain the high degree of flexibility required by today's short time-to-market windows and fluid standards. Additionally, some experiments suggest that reconfigurable logic might be considerably more power-efficient than standard solutions [3].

Often, the configurable unit augments a standard processor, which still performs general system control tasks [4] [5] [6] [7] [8]. Only the compute-intensive parts of an application (called *kernels*) are implemented in configurable hardware. On a system-on-chip (SoC), a much higher degree of integration between these components can be achieved than with the currently prevailing board-level solutions. To simplify building such a device, IP blocks for reconfigurable arrays are already available [9] [10] [11]. Section 2 discusses some of the issues that need to be considered when designing a heterogeneous reconfigurable SoC.

One of the reasons that adaptive processors are still not in widespread use (despite their demonstrated performance potential) is the difficulty of programming them: In many cases, the design flows used require familiarity with hardware design techniques and architectures. Combined, these demands act as a severe obstacle, rendering the systems inaccessible to developers only having experience in programming conventional (pure software) computers.

Considerable effort has thus been expended to overcome these problems and raise the abstraction level of programming an adaptive processor closer to that of a conventional one. Examples include translating traditional languages such as C or even higher-level descriptions such as MATLAB into efficient hardware/software solutions. Our work aims at compiling standard dusty deck C to heterogeneous reconfigurable systems. Using such a tool set, only minimal effort is required for porting old code or developing new applications for a reconfigurable platform. A first prototype (Section 3) of such a flow has been implemented in cooperation with the Synopsys Advanced Technology Group, the BRASS project at the UC Berkeley, and Lockheed-Martin Advanced Technology Laboratories [12] [13].

## 2 Architecture Discussions

Given that we are considering heterogeneous systems consisting of one or more fixed and/or configurable parts, a broad spectrum of possible architectures and interfaces between components needs to be evaluated. For example, it is possible to integrate a reconfigurable function unit directly into the CPU. While this can be used to efficiently pack simple CPU instruction sequences (especially logical operations as occurring in crypto applications) into a single hardware function, the speed-up here is limited by the communications bandwidth (just the CPU register file) and the low degree of exploitable parallelism. Another approach attaches a reconfigurable co-processor (RC) to the processor bus and uses it to consume and produce data streams which are directed by the CPU or DMA engines. This architecture can efficiently exploit pipeline parallelism and is very well suited for applications such as filtering (audio/video) and sequence matching. However, when desiring to shift general-purpose algorithms to an RC, it becomes necessary to allow the unit full random access to the main memory, e.g., to perform independent table lookups and pointer operations. One of the more important choices to be made for this approach is whether and how to share caches between the CPU and and the RC.

The architecture of the RC itself also needs to be considered carefully. The size of the basic compute elements must be matched to the native size of the data to be processed and the processing style. E.g., for most compute-bound applications, the single-bit logic blocks of many FPGA architectures are rather inefficient. A better match for these algorithms might be achieved by composing 8b-wide operators into the 8b,16b,24b and 32b operand sizes commonly used. Alternatively, narrower blocks (such as 4b) might be used in a serial fashion.

The reconfiguration characteristics strongly influence the possible usage of the reconfigurable unit: Very slow rates suffice for "hardware" update/upgrade operations or global mode switches (e.g., the RC implements different communication protocols in different service areas). To better exploit the silicon area sacrificed to achieve reconfigurability, short configuration times are desirable. In some cases, this can be achieved by realizing a configuration cache that allows rapid switching between a small number of configurations. Note that continuous single-cycle configuration switching is unrealistic for large devices due to the amount of power consumed for each switch (and the corresponding cooling problem).

After introducing these basics, we can now characterize the target architecture for our design flow: We are aiming for an RC closely coupled to a conventional CPU. The RC has a dedicated memory interface offering access to main memory in both regular (streaming) and irregular (cached) access

patterns. This is required by our goal of compiling the full C language (including pointers etc) into software-hardware combinations. Following the size of C's most common data types, our logic block size is assumed to be 32b. Furthermore, we expect short configuration times to enable us to employ a specialized configuration for each kernel (generally a loop nest). The compiler prototype has two hardware targets: The GARP architecture [4] adds a configurable data path of 32b operators to a MIPS core. It comes very close to our wishlist, but is only available as a cycle-accurate simulator. For more precise measurements in the real world, we use the ACE-V [14] board-level platform that combines a microSPARC-II RISC processor with a Xilinx Virtex FPGA acting as RC. While the ACE-V also has all of the abilities required, the slow configuration speed and memory access limit the system to real-world tests instead of realizing actual speedups (as can be shown for GARP).

# 3 Tool Flows

The structure of the design flow of the prototype NIMBLE compiler is shown in Figure 1. Note that the compiler is mostly target-technology independent (the target is characterized by a configuration file). An input program is read by the front-end and analyzed to isolate the performance-critical and hardware-feasible parts of the applications (by various dynamic profiling mechanisms). Certain constructs cannot (e.g., printf) or can only be very inefficiently (e.g., floating-point operations) mapped to the RC. If they occur sufficiently rare (e.g., only for error handling), the kernel is partially executed on the RC: When the hardware-infeasible operation occurs, this exceptional case in handled by seamlessly switching back to software execution. In this manner, the program is partitioned between hardware and software components. Over a wide range of real-world applications analyzed in this manner, an average of 78% of the original software execution time can thus be handed over to the RC and potentially be accelerated. The software parts are then extended with automatically generated interfaces to the hardware components and processed by a normal software flow. The hardware parts are compiled into data flow graphs which are then mapped to the target device. A final link step merges both types of components into a single executable image, ready for execution either in the GARP simulator or the ACE-V adaptive computer. On a suitable architecture such as GARP, even the prototype compiler can accelerate a wavelet image compression program by a factor of 3 over an optimized MIPS software implementation [4].

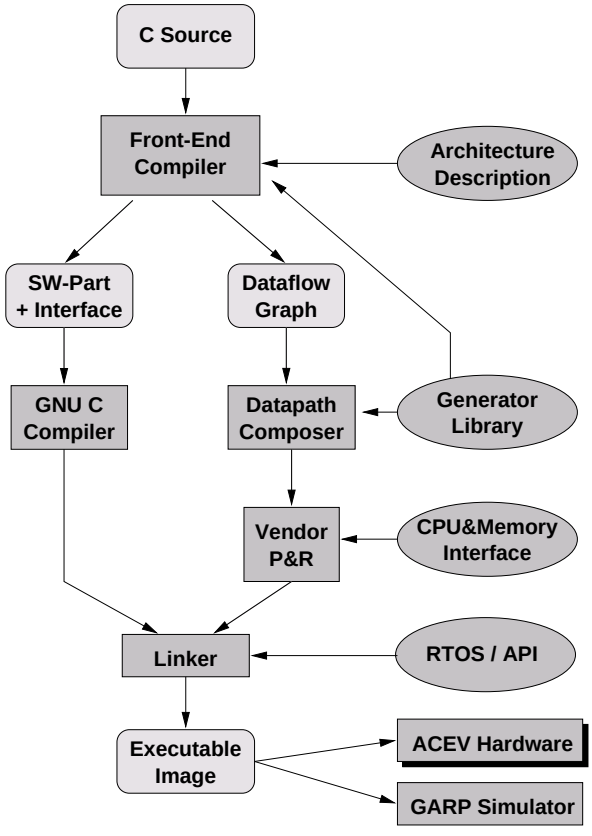


Figure 1: NIMBLE Compile Flow

We are now working on a second generation compiler system that builds on our experiences gained with NIMBLE, which served to validate our initial assumptions and acted as a proof-of-concept prototype. The new compiler is built from the beginning to support many powerful optimization (loop transforms, pipelining, speculative execution) that were not considered in the previous demonstrator. Subsystems such as the technology mapping, module generators and interface blocks have already been completed. Other research deals with automatically integrating other IP blocks with the automatically compiled data paths to reuse existing blocks or perform manual optimization for high-performance operation.

# References


[1] Tessier, R., Burleson, W., "Reconfigurable Computing for Digital Signal Processing: A Survey", *J. VLSI Signal Processing*, No. 28, pp. 7-27, Kluwer, 2001

[2] Hauck, S., "The Roles of FPGAs in Reprogrammable Systems", *Proc. IEEE*, pp. 615–638, 1998

[3] Zhang, H., Prabhu, V., George, V., Wan, M. et al, "A 1V Heterogeneous Reconfigurable Processor IC for Baseband Wireless Applications", *Proc. IEEE International Solid-State Circuits Conference*, 2000

[4] Callahan,T.,Hauser,R.,Wawrzynek,J., "The GARP Architecture and C Compiler",*IEEE Computer 33(4)*, pp. 62-69, 2000

[5] Triscend A7, www.triscend.com

[6] Chameleon CS2000, www.cmln.com

[7] Stollon, N., Sihlborn, B., "BAZIL: A Multi-Core Architecture for Flexible Broadband Processing", *Proc. Embedded Processor Conference*, 2001

[8] Xilinx VirtexIIPro, www.xilinx.com

[9] Adaptive Silicon MSA, www.adaptivesilicon.com

[10] Elixent RDA, www.elixent.com

[11] PACT XPP, www.pactcorp.com

[12] Li, Y.B., Harr, R., et al. "Hardware-Software Co-Design of Embedded Reconfigurable Architectures", *Proc. Design Automation Conference*, 2000

[13] Koch, A., "Adaptive Rechensysteme und ihre Entwurfswerkzeuge", *Proc. 10. E.I.S. Workshop, Dresden*, 2001

[14] Koch, A., "A Comprehensive Prototyping Platform for Hardware-Software Codesign", *Proc. IEEE Workshop on Rapid Systems Prototyping*, 2000