The ReAl Computer Architecture
ReAl = Resource Algebra

Projektleiter
Prof. Dr. Wolfgang Matthes

Zeitraum
2009–2011

Kontakt
Prof. Dr. Wolfgang Matthes
Fachbereich Informations- und Elektrotechnik
Fachhochschule Dortmund
Sonnenstraße 96
44139 Dortmund
Tel.: 0231 9112-162
E-Mail: wolfgang.matthes@fh-dortmund.de

Abstract
ReAl computer architecture replaces the conventional processor core – essentially an autonomous state machine controlled by stored instructions – by ensembles of processing resources. The ReAl API allows for completely describing and exploiting the inherent parallelism of the application problems. To prove the principal feasibility, it has been shown that the basic operators together with some simple resources can emulate all the essential principles of the v.Neumann architecture. The approach has been vindicated further by investigating fundamental problems of efficiency. First experimental results have been obtained.

Computer Architecture as an Algebra of Resources
In a ReAl machine, the silicon real estate will be populated with a comparatively large number of control and operation units, designated as resources ([2]...[6]). The processor cores are decomposed into their functional units, which are put immediately under program control (Fig. 1). A resource represents an intermediate granularity between a fully-fledged processor and the logic

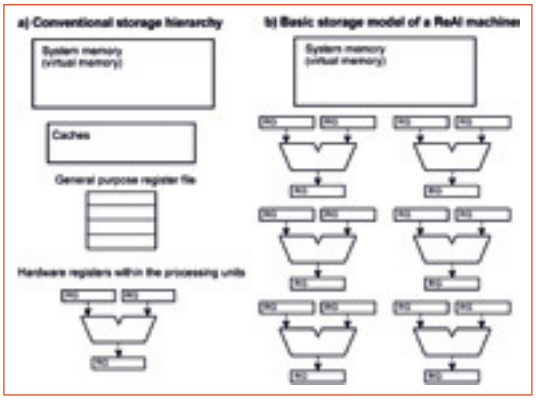

Fig. 1. Storage hierarchies compared. Because the intermediate variables reside within the processing resources, most of the transport operations are omitted.

cells of a FPGA. An arithmetic-logic unit (ALU) with some addressing, control, and storage means may serve as a typical example. Detailed investigations seem to confirm the advantages of a large number of comparatively simple resources instead of a smaller number of more complex ones. Resources are connected via bus systems or switched point-to-point interfaces (Fig. 1). Only some few topologies are of decisive importance ([1]), above all independent processing units (for exploiting true parallelism) and the inverted binary tree (for mapping nested expressions onto it). All other topologies could be emulated (virtualized). Therefore, cost could be expected to be kept reasonably low. At a first glance, block diagrams of ReAl machines resemble massively parallel or cellular systems. The peculiar feature is the application programming interface (API), which enables such configurations to execute conventional programs. To extract and describe the inherent parallelism in statu nascendi – in other words, immediately from the programmer's intentions –, the ReAl API assumes the pool of resources to be infinite.

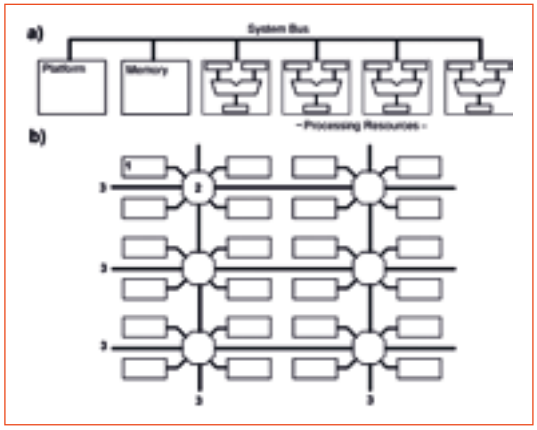

Fig. 2. Typical ReAl machines. 1 - resource cell; 2 - switching hub; 3 - point-to-point-interface.

In order to implement a certain programming intention, appropriate resources will be selected out of the resource pool. These resources will be fed with parameters. Then the processing operations will be initiated. Results will be stored in memory or written to I/O devices; intermediate results will be forwarded to other resources. Further steps of parameter passing, initiation and assignment will be executed until the processing task has been completed. Resources which are no longer needed will be returned to the resource pool. These processing steps are controlled by stored instructions. The instructions – operators in the ReAl terminology – describe only the basic processing steps, but not the operations to be performed (e.g., addition or multiplication). So-called platform resources are provided to fetch the instructions from memory. Additional operators establish concatenations between resources and to disconnect such concatenations. Once a concatenation has been set up, the steps of parameter passing, initiation of operations and assignment of results will be performed automatically.

Efficiency of Implementation
Each processor is basically a sequential state machine. It should do useful work. The task proper of a machine is not executing instructions but delivering output bit patterns according to input bit patterns. When an application problem is to be solved, intermediate variables, procedure calls and the like are essentially a waste of clock cycles or machine bandwidth. To quantitatively characterize architectures and machines, a performance metrics and a metrics of implementation efficiency have been introduced ([5], [7]).

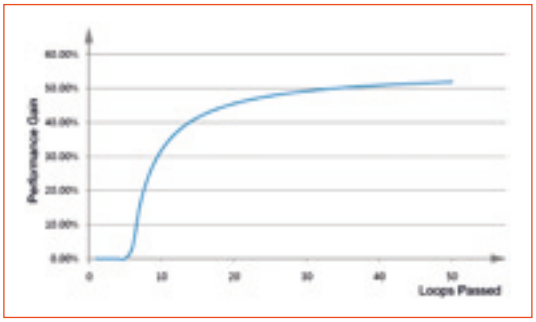

Fig. 3. A summary result of experimental investigations ([6]). After 50 loop cycles, the performance of the ReAl machine surpasses an x86 family processor by approximately 50%.

It has been found that this metrics can be used to evaluate efficiency problems of power consumption, too. Today, the primary design constraint is not transistor count, but power consumption. According to a strict power saving philosophy, the universal computer is to be considered only a makeshift solution. With respect to an application problem, the true optimum solution would be a dedicated machine whose cycles are spent exclusively to compute the desired final results. In such a machine, neither clock cycles and memory bandwidth nor power would be wasted for fetching instructions, loading and storing intermediate values, calling functions and the like. ReAl machines should be true universal machines whose characteristics come as close to this ideal as possible.

Experimental Results
It is difficult to evaluate new architectural proposals against existing processors, because that means to compare a fictitious machine to a fully-fledged high-performance processor. Thus, it is impossible simply to measure the execution times. Instead, the program execution is to be examined step by step.

An emulator program has been developed, which serves as a demonstration of feasibility as well as an evaluation tool ([6]). Thus, it was possible to compare the new architectural principles with conventional machines and programs and to obtain an approximate quantitative assessment of effectiveness. A ReAl machine is compared to a program written in C, translated by a state-of-the-art compiler and executed on the processor of a personal computer.

Initial investigations have been based on Bresenham's line drawing algorithm. Two programs were written: the one conventionally with the C language, the other with the ReAl API of the EmuRix emulator. The C program has been compiled using the Microsoft® Optimizing Compiler Version 16.0. The assembler code has been evaluated manually and compared to the statistics of the EmuRix code, generated by the emulator. The results are encouraging, showing an increase in performance between 20 and 50 % even for emulation on conventional machines (Fig. 3). It should be noted that emulation is not only a means for evaluation and comparison, but a viable technology for implementing the concept of bytecode, which can be executed everywhere (cf. the virtual machines JVM and Dalvik).

References:
[1] Matthes, W.: How many operation units are adequate? ACM SIGARCH Computer Architecture News, Vol. 19, Issue 4 (June 1991), pages 94-108.
[2] ReAl Design Documentation. Patent applications: DE 10 2005 021 749.4 and US 11/430,824. Internet: http://www.realcomputerarchitecture.de
[3] Matthes, W.: The ReAl Computer Architecture. Proceedings IDAACS 2007, pages 249-254.
[4] Matthes, W.: Ressourcen statt Prozessorkerne? NTZ 7/8 2009, pages 12 – 16.
[5] Matthes, W.: Resources instead of Cores? ACM Sigarch Computer Architecture News, Volume 38, Number 2, May 2010, pages 49 – 63.
[6] Kuczkowicz, L.: Verfahren zur Emulation von Hochleistungsrechnern. Bachelor Thesis, Fachhochschule Dortmund, 2011.
[7] Matthes, W.: Hardware und Software. Embedded Electronics, Band 3. Elektor, 2011.