The ReAl Computer Architecture

ReAl = Resource Algebra

Prof. Dr. Wolfgang Matthes, Fachbereich Informations- und Elektrotechnik

Abstract
The research effords are directed towards a hardware-software interface which can put into effect a practically unlimited number of processing resources and which allows for completely describing and exploiting the inherent parallelism of the application problems. The proposed architectural principles may lead to:

- Instruction set architectures which can cope with a transfinite number of hardware resources.
- Processing circuits containing resources of intermediate granularity and appropriately optimized interconnects.
- Machines which do not waste clock cycles – and hence energy – for purposes which are not related immediately to the computation of the desired final results (like fetching instructions or pushing register contents and intermediate variables around).

Computer Architecture as an Algebra of Resources
The proposed architecture is based on a set or pool of resources which can execute certain operations with data of certain types. This constitutes basically an algebraic structure. Hence the name ReAl = *Re*source-*Al*gebra. Similar paradigms have been used occasionally over more than one decade for performance analysis and abstract modeling of architectural principles ([1]). Here the basic idea will be applied to instruction set design.

The principal hypothesis: There will always be enough:

- Hardware does not matter.
- Memory Capacity does not matter.
- Hardware requirements for machine program generation do not matter.

The basic paradigm: When we want to do something, we will fetch an appropriate piece of hardware out of a magazine (like a hammer to drive in a nail or a wrench to fasten a nut) and use it to perform the information processing task to be executed. When we want to add two numbers together, we take an adder, when we want to compare two values, we take a comparator and so on. A piece of hardware which has done its duty will be returned to the magazine. We will take as many tools as we need, e.g., 50 hammers if 50 nails are to be driven in, or 50 adders if 50 pairs of numbers are to be added together.

An advanced architecture should permit to make use of the inherent parallelism. The ReaL approach is based on the idea to assume an infinite pool of resources. Of cause, each pool of resources is limited in size. Hence the programs are to be adapted to the limits of a given pool of resources. This can be done during compiling time or during run time.

A complementing idea is the efficiency of implementation. 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 (and, in consequence, energy). Obviously, it would be better to have the job being done (for example, the rendering of a graphics presentation) than to push the EDX register onto the stack in order to get it free for multiplication and later to fetch it back again (this is an example of an instruction sequence which contributes nothing to solve the application problem, but is to be executed to compensate for architectural quirks).

The architecture is based on the following principles:

- There will be always enough resources. Above all this is a theoretical assumption (hypothesis of a transfinite resource pool). Based on this assumption it is possible to request an arbitrary number of resources in order to exploit the inherent parallelism up to the utmost level. In practice however, each pool of resources is limited in size. Hence the programs are to be adapted to the limits of a given pool of resources.
- With respect to an application problem, the universal computer is considered to be only a makeshift solution. The true optimum solution would be a dedicated hardware whose machine 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, function calls and the like. ReAl machines should be true universal machines whose characteristics come as close to this ideal as possible.
- The basic paradigm of a resource is a piece of hardware with input registers, combinational circuitry and output registers.
- The instructions describe only the basic processing steps, but not the concrete operations to be performed (e.g., addition or multiplication).

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. So-called platform resources are provided to fetch the instructions from memory. Additional instructions 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.

A processing resource in a ReAl machine is a functional unit – more than a logic block and less than a complete processor. An arithmetic-logic unit (ALU) with some addressing, control, and storage means may serve as a typical example. Compared to the operation units in contemporary high-performance processors, the resources in ReAl machines are less complex.

Resources should be able to work autonomously. The principal goal is to avoid information transport and operation cycles which are not necessary. Therefore, instruction fetch cycles as well as load and store cycles have to be avoided whenever possible. In an ideal machine, only data related to the solution of the application problem would be fetched and stored. In a ReAl machine, instructions will set up concatenations of resources which correspond to parts of the dataflow graph of the application problem and leave the execution of operations and the transport of intermediate data to the processing resources.

The first more detailed investigations have been aimed at demonstrating the feasibility of the real approach by showing that the basic operators together with some simple resources can emulate all the essential principles of the v.Neumann architecture.

Next steps to be taken:

- ReAl development will be a time-consuming process.
- The first implementations will be emulators.
- Toy-like implementations are worthless.
- The first experimental ReAl architecture implementation should be a reasonable compiler target.
- Hence the complexity will be comparable to the architectures of contemporary high-performance processors (for example, Intel IA-32 and IA-64).
- Numerous details have to be taken into consideration. (Contemporary processors have 200 instructions and more.)
- A set of basic resources has to be defined in detail.
- The theory of operation has to be worked out and described in detail.
- All the primary evaluation work has to be done manually, as a number of iteration cycles cannot be afforded (design a architecture – write a compiler– evaluate the work based on real-world applications – improve the architecture and so on).
- The set of reference manuals will comprise more than 1000 pages.
- Algorithms for converting conventional programs into ReAl operator sequences have to be developed.

References:

[1] Matthes, W.: Hardware Resources: a generalizing view on computer architectures. ACM SIGARCH Computer Architecture News, Vol. 18 , Issue 2 (June 1990), pages 7-14.

[2] Matthes, W.: How many operation units are adequate? ACM SIGARCH Computer Architecture News, Vol. 19, Issue 4 (June 1991), pages 94-108.

[3] ReAl Design Documentation. Pending patent applications: DE 10 2005 021 749.4 and US 11/430,824. Internet: http://www.realcomputerarchitecture.de

[4] Matthes, W.: The ReAl Computer Architecture. Proceedings IDAACS 2007, pages 249-254.

The ReAl Computer Architecture

ReAl = Resource Algebra

Prof. Dr. Wolfgang Matthes, Fachbereich Informations- und Elektrotechnik

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 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.

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]).

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).

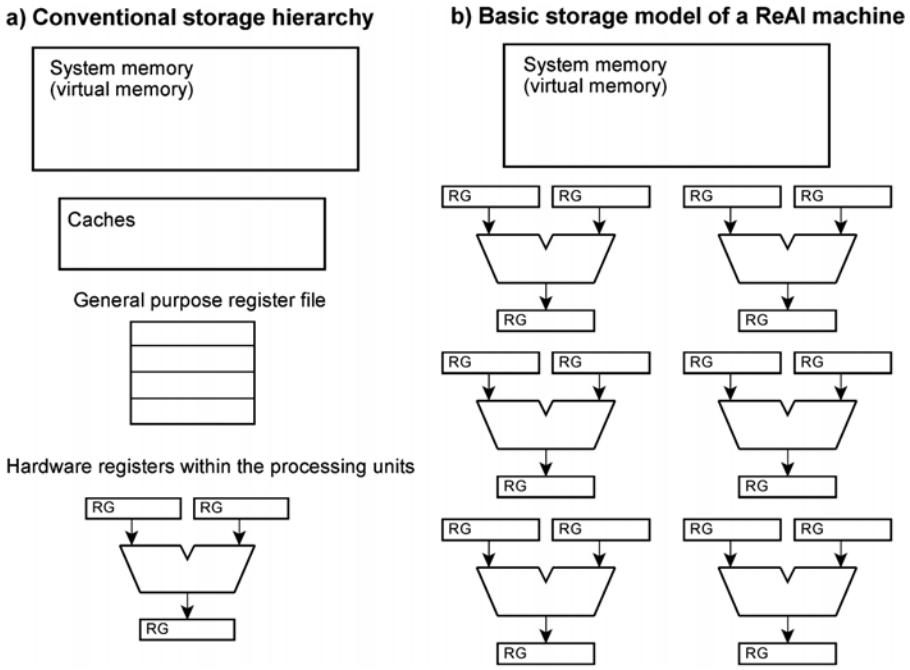

Fig. 1. *Storage hierarchies compared. Because the intermediate variables reside within the processing resources, most of the transport operations are omitted.*

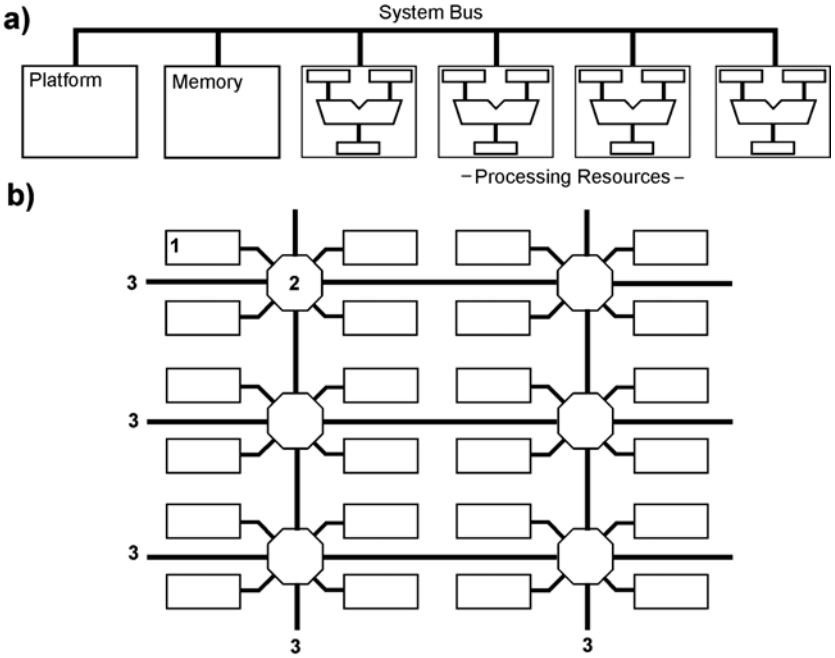

Fig. 2. *Typical ReAl machines. 1 - resource cell; 2 - switching hub; 3 - point-to-point-interface.*

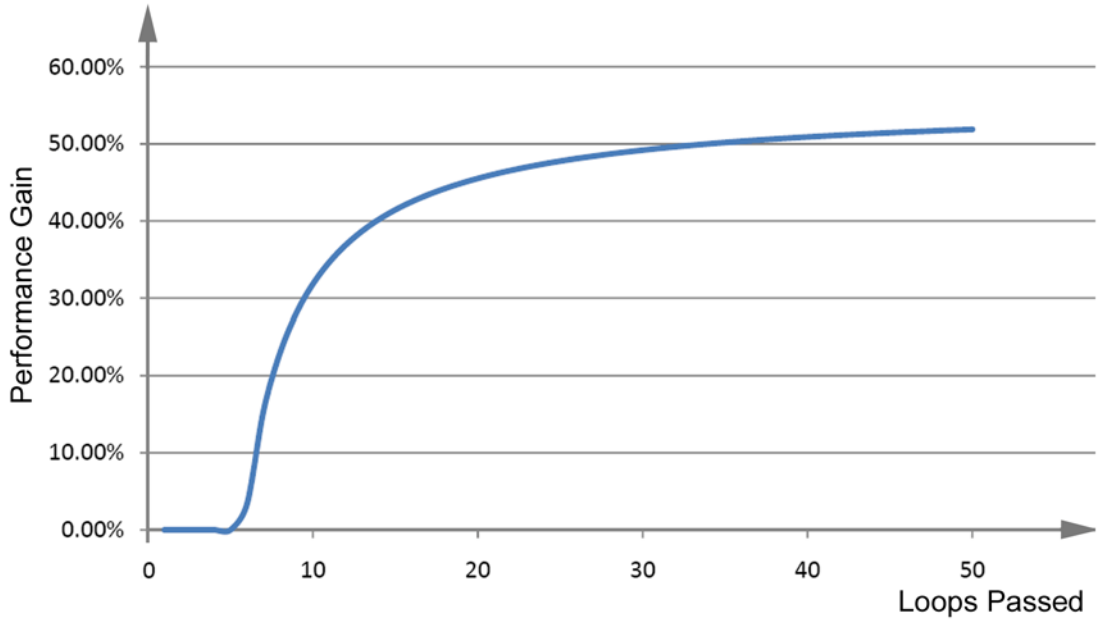

Fig. 3. *A summary result of experimental investigations ([6]). After 50 loop cycles, the performance of the ReAl machine surpasses an x86 family processor by approximately 50%.*

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.