DATA-STREAM-BASED COMPUTING: MODELS AND ARCHITECTURAL RESOURCES

Reiner Hartenstein,
Kaiserslautern University of Technology, Germany
http://hartenstein.de

ABSTRACT. *The paper addresses a broad readership in information technology, computer science and related areas, introducing reconfigurable computing, and its impact on classical computer science. It points out trends driven by the mind set of data-stream-based computing.*

1. INTRODUCTION

platform category	source "running" on platform	machine paradigm
hardware	(hardwired)	
morphware	configware	
ISP*	software	von Neumann
AM*	flowware	anti machine
rAM*	flowware & configware	

*) acronyms see fig. 4, terminology: fig. 8 and 9.

Fig. 1: Platform categories

An alternative general purpose platform. The dominance of the instruction-stream-based procedural mind set in computer science stems from the general purpose properties of the ubiquitous von Neumann (vN) microprocessor. Because of its RAM-based flexibility no costly application-specific silicon is needed. Throughput is the only limitation by its sequential nature of operation (von Neumann bottleneck). Now a second RAM-based computing paradigm is heading for mainstream: *morphware*, electrically reprogrammable by reconfiguration of its structure [1]. This is a challenge to CS curricula innovators, also an occasion to reconsider criticism of the von Neumann culture [2] [3] [4] [5].

CS to explore new horizons. From this starting point Computing Sciences (CS) are slowly taking off to explore new horizons: a dichotomy of two basic computing paradigms, removing the blinders from the still dominant von-Neumann-only mind set, which is still ignoring the impact of Reconfigurable Computing (RC). It has been predicted, that by the year 2010 more than 90% of all programmers will implement applications for embedded systems, where a procedural / structural double approach is a pre-requisite. Currently programmers do not yet have the background required for this new labor market. This challenge can be met only by the dichotomy of machine paradigms within CS.

language category	vN language (like e. g. C)	anti machine language
state register	program counter	data counter(s)
sequencing operation examples	read next instruction, goto (instruction address), jump (to instruction address), instruction loop, loop nesting instruction stream branching, escapes, ***no parallel loops***,	read next data item, goto (data address), jump (to data address), data loop, loop nesting, data stream branching, escapes, ***parallel loops***,
sequencing primitives	control flow	data stream management
other primitives	data manipulation	***none*** ←
address computation	memory cycle overhead	overhead avoidable
instruction fetch	memory cycle overhead	no fetch at run time

Fig. 2: Traditional Software languages versus Flowware languages.

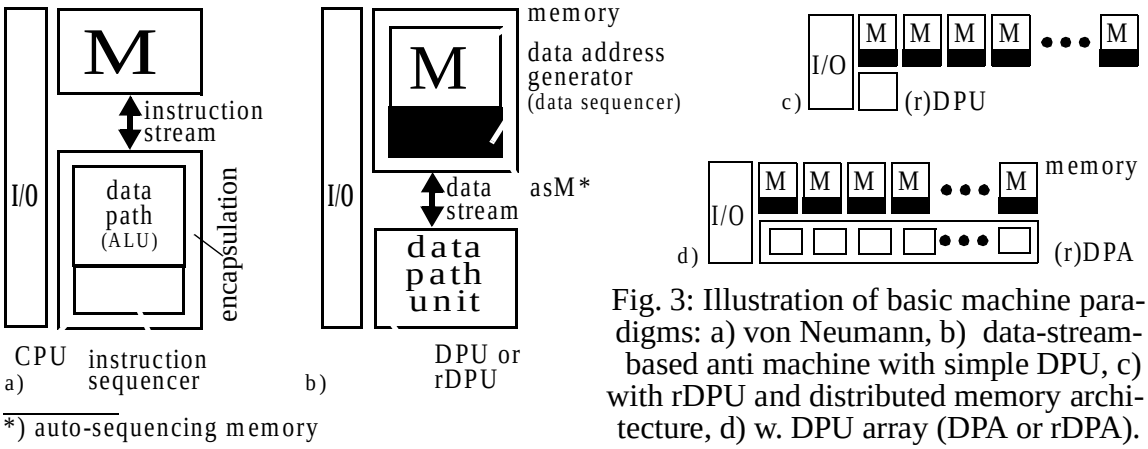


Fig. 3: Illustration of basic machine paradigms: a) von Neumann, b) data-stream-based anti machine with simple DPU, c) with rDPU and distributed memory architecture, d) w. DPU array (DPA or rDPA).

The education gap an be bridged. A rich supply of tools and research results is available to adapt fundamental courses, lab courses and exercises [6]. There are a lot of similarities between both branches, like between matter and anti matter. But also some challenges are waiting. Our basic curricula do not teach, that hardware and software are alternatives, and, how hardware / software partitioning is carried out. E. g. some urgently needed new directions of algorithmic cleverness are not yet taught. For instance, how to implement a high performance application for low power dissipation on 100 processors running at 200 MHz, rather than on one processor running at 20 GHz. A curricular revision is overdue [7].

2. RECONFIGURABLE COMPUTING

In morphware application the lack of algorithmic cleverness is an urgent educational problem.

Advancing maturity is indicated by a growing consensus on terminology (fig. 1). Occupied by other areas. the term "dataflow machine" [8] and the acronym DSP should not be used. So this paper uses the term *anti machine*.

The dichotomy of fundamental models. More important is the terminology from a global point of view (figure 1 a). Whereas classical CS deals with *software* (SW) running on *hardware (HW)*, the new branch deals with *flowware (FW)* [9] running on *HW*, or, *configware (CW)* [10] and *FW* "running" on *morphware (MW)* [11],. This paper gives introductions for a broad readership mainly with a CS background..

This paper does not deal with fine grain morphware (FPGAs, using single bit wide CLBs) already being mainstream. *Reconfigurable Computing* (RC) uses coarse grain morphware platforms: rDPUs (reconfigurable data path units), which, similar to ALUs, have major path widths, like 32 bits, for instance - or even rDPAs (rDPU arrays). Important applications are derived from the decay of "general purpose" vN computer architecture [2] [3] [4] and its performance limits [5], creating a demand for accelerators. For very high throughput requirements RC is the drastically more powerful and more area-efficient and energy-efficient programmable alternative [5] [12] to FPGAs (fig. 6), also providing a massive reduction of configuration memory and time needed for configuration [13].

AM	anti machine (DS machine)	DS	data stream
asM	autosequencing Memory	DSM	data stream processing machine
rAM	reconfigurable AM	EE	Electrical Engineering
CPU	"central" processing unit: DPU **and** instruction sequencer (vN)	ESW	embedded SW
		FW	flowware
CS	Computing Sciences, Computer Science	HW	hardware
CW	configware	ISP	instruction stream processor
DPU	data path unit **without** sequencer	MW	morphware
rDPU	reconfigurable DPU	RC	reconfigurable computing
DPA	data path array (DPU array)	SW	software
rDPA	reconfigurable DPA	vN	von Neumann (machine paradigm)

Fig. 4: Some acronyms.

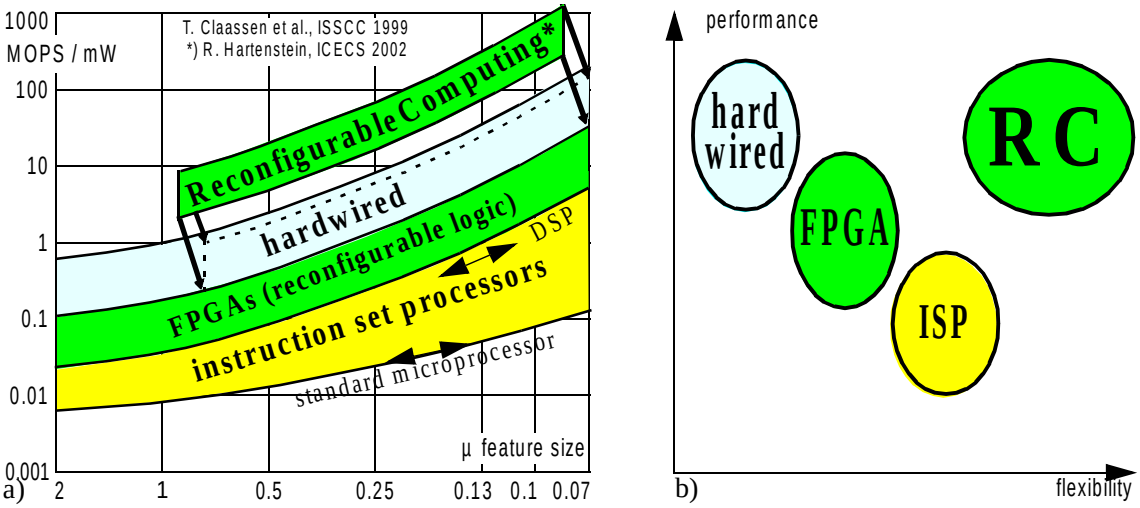


Fig. 6: Energy efficiency and performance vs. flexibility incl. Reconfigurable Computing.

Commercial architectures. In application areas like multimedia, wireless telecommunication, data communication and many others, the throughput requirements are growing faster than Moore's law, along with growing flexibility requirements due to unstable standards and multi-standard operation [14]. Currently the requirements can be met from commercial sources only by rDPAs from a provider like PACT [15] [16] [17] [18] [19] (fig. 11).

Domain-specific approach. A a currently viable solution appears the domain-specific approach [13], where a design space explorer may help to derive within a short time an optimum (r)DPU and (r)DPA architecture from a benchmark or domain-typical set of applications [20] [21].

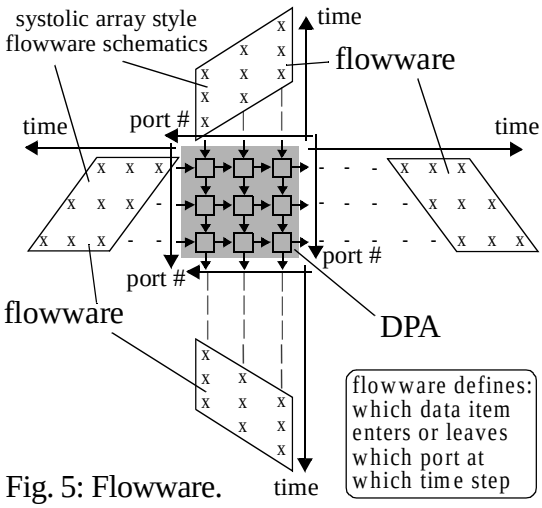


Fig. 5: Flowware.

3. DATA-STREAM-BASED COMPUTING

Traditional instruction-stream-based informatics is based on computing in the time domain, where a program deserves scheduling the instructions for execution (fig. 9). Classical basic structures and principles in computing are von-Neumann-centric, which are instruction-stream-based, where instruction sequencer and datapath are in the same CPU (fig. 3 a). Due to reconfigurable a second basic model has emerged, so that we now have a dichotomy of models: instruction-stream-based computing vs. data-stream-based computing. There is a lot of similarities, so that each of the 2 models is a kind of mirror image of the other model - like with matter and antimatter.

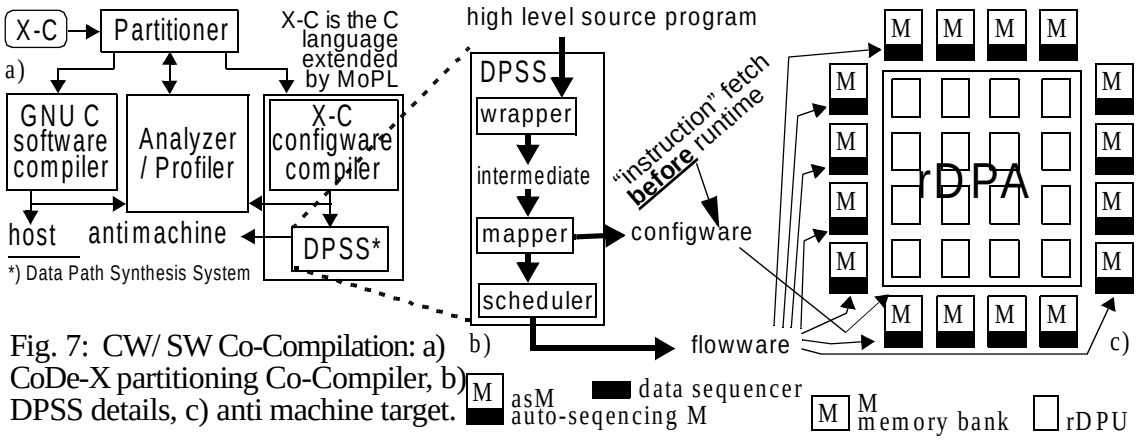


Fig. 7: CW/ SW Co-Compilation: a) CoDe-X partitioning Co-Compiler, b) DPSS details, c) anti machine target.

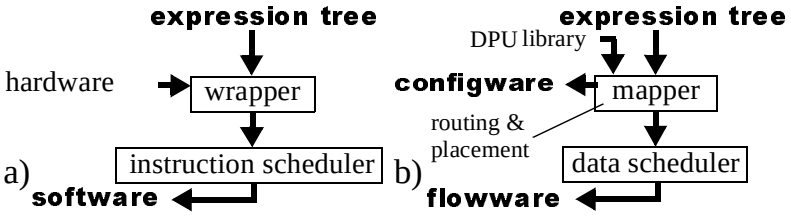


Fig. 8: Compilation: a) von-Neumann-based, b) for anti machines

Data counters replace the program counter. Data-stream-based computing, the counterpart of instruction-stream-based von Neumann computing (fig. 9), however, uses one or more data counters instead of a single program counter (example in fig. 3 b). However, there are some asymmetries, like predicted by Paul Dirac for antimatter. Figure 7 b shows the block diagram of data-stream machine with 16 autosequencing memory banks. The basic model allows the machine to have 16 data counters, where as a von Neumann machine cannot have more that one program counter. The partitioning scheme of the data-stream machine model assigns a sequencer (address generator) always to a memory bank, never to a DPU. This modelling scheme goes fully conform with the area of embedded distributed memory design and management (see section on Embedded Memory).

The vN microprocessor is indispensable. But because of its monopoly our CS graduates are no more professionals.

Flowware. Data streams have been popularized by systolic arrays [22] [23] [24] (fig. 5), the super systolic array [25], and more recently by projects like SCCC [26], SCORE [27] [28], ASPRC [29], BEE [30] [31] [32], the KressArray Xplorer [20] [21] and many other projects. In a similar way like instruction streams can be programmed from *SW sources*, also data streams can be programmed, but from *FW sources*. High level programming languages for flowware [33] and for software join the same language principles and have a lot in common, no matter, wether finally the program counter or a data counter is manipulated. Figure 8 illustrates the basic semantic principles of flowware by 12 data streams associated with the 12 ports of a DPA. The data schedule generated from a flowware source determines, which data object has to enter or leave which DPA port (or DPU port) at which time. This way flowware can be used to program the 12 autosequencing memory banks (asM) of the embedded distributed memory to generate the expected data streams.

machine category	(a) instruction set processor	(b,c) data stream processor	
		(b) hardwired	(c) morphware
machine paradigm	von Neumann (vN)	anti machine	
reconfigurability support	no	no	yes
programming	instruction-procedural	no	structural (super "instruction" fetch)
		data scheduling	
program source	software	flowware	flowware & configware
"instruction" fetch	at run time	at fabrication time	before run time
execution at run time	instruction schedule	data schedule	
operation spin	instruction flow	data stream(s)	
operation resources	CPU	DPU, or, DPA	rDPU, or, rDPA
	hardwired	hardwired	reconfigurable
parallelism	only by multiple machines	by single machine or multiple machines	
state register	single program counter	one or more data counter(s)	
state register located	within CPU	outside DPU or DPA:	outside rDPU or rDPA:
		within asM (autosequencing memory banks)	

Fig. 9: Asymmetry between machine and anti machine paradigms.

Two programming sources. Figure 7 a, Figure 8 a and Figure 10 d illustrate, why a von Neumann machine needs just software as the only programming source, since the resource part being hardwired is not programmable. Figure 7 b, Figure 8 b and Figure 10 e show, why a reconfigurable data-stream-based machine needs two programming sources: configware to program (to reconfigure) the operational resources, and, flowware to schedule the data streams. Figure 10 f shows why hardwired anti machines need only a single program source: flowware only. Figure 7 c illustrates the structure of the compiler (DPSS [25]) generating the code of both sources from a high level programming language source (here a C subset [25]): phase 1 performs routing and placement to configure the rDPA, and phase 2 generates the flowware code to program the autosequencing distributed memory, so that the data streams fit to the routing and placement result from phase 1.

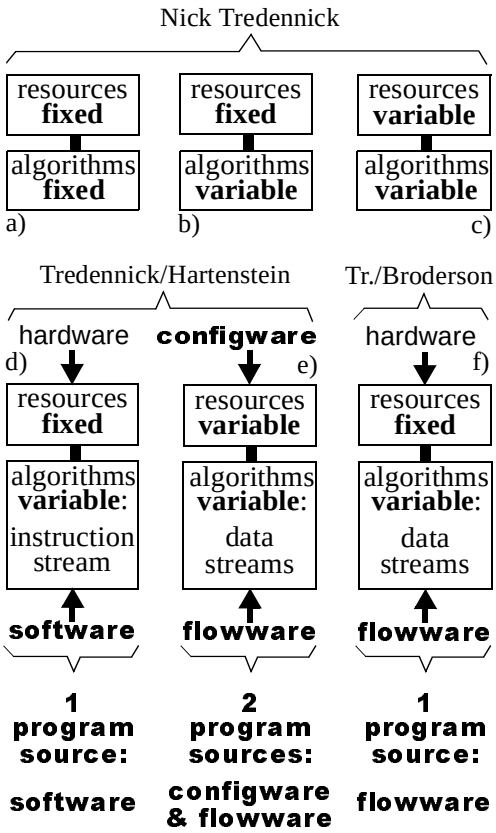


Fig. 10: Nick Tredennick's digital system classification scheme: a) hardwired, b) programmable in time, c) reconfigurable d) von-Neumann-like machine paradigm e) reconfigurable anti machine paradigm f) Broderson's hardwired anti machine. terminology also from: [5].

The same model for hardware and morphware. There is in principle no difference, whether a data-stream-based DPAs is hardwired or reconfigurable. The only important difference is binding time of placement and routing: before fabrication, or, after fabrication (compare fig. 9 b).

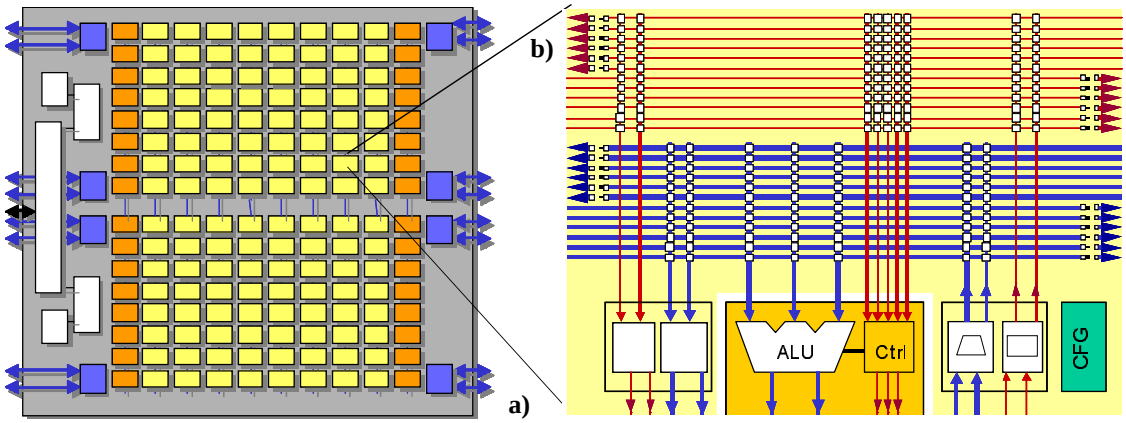
Embedded Distributed Memory. Together with application-specific embedded memory architecture synthesis also flowware implementation (for memory management strategies) is subject of performance and power optimization [34], also by loop transformations [35]. Good flowware may be also obtained after optimized mapping an application onto rDPA [20], where both, data sequencers and the application can be mapped (physically, not conceptually) onto the same rDPA [13].

Memory bandwidth. To solve the memory communication bandwidth problem the anti machine paradigm (datastream-based computing) is much more efficient than "von Neumann". There are alternative embedded memory implementation methodologies available [34] [36] [37] [38], either specialized memory architecture using synthesized address generators (e. g. APT by IMEC [34]), or, flexible memory architectures using programmable general purpose address generators [39] [40]. Performance and power efficiency are supported especially by sequencers, which do not need memory cycles even for complex address computations [34], having been used also for a smart memory interface of an early anti machine architecture [41] [42].

Data-Stream-based vs. concurrent Computing. Classical parallelism by concurrent computing has a number of disadvantages over the parallelism by anti machines having no von Neumann bottleneck, what is discussed elsewhere [32] [42]. Amdahls law explains just one of several reasons of inefficient resource utilization. vN-type processor chips are almost all memory, because the architecture is wrong. Here the metric for what is a good solution has been wrong all the time.

4. CONFIGWARE COMPILERS

Co-Compilation. Using coarse grain morphware (rDPAs) as accelerators changes the scenario: implementations onto both, host and accelerator(s) are RAM-based, which allows turn-around times of minutes for the entire system, instead of months for hardwired accelerators, and, supporting a migration of accelerator implementation from IC vendor to customer, who usually does not have hardware experts. This creates [43] a demand for



c) platform	application example	speed-up factor	method
PACT Xtreme 4-by-4 array [2003]	16 tap FIR filter	x16 MOPS / mW	straight forward
MoM anti machine with DPLA* [1983]	grid-based DRC** 1-metal 1-poly nMOS 256 reference patterns	x2000 (computation time)	multiple aspects

*) Kaiserslautern e-programmable PLA, manufactured by E.I.S. project MPC organization
**) Design Rule Check based on 4-by-4 pixel reference patterns

Fig. 11: Configurable System-on-Chip with XPU (xtreme processing unit) fromPACT AG): a) XPU array structure, b) the structure of a rDPU, c) speed-up factors (PACT & MoM).

compilers accepting high level programming language (HLL) sources. Partly dating back to the 70ies and 80ies know-how is available from the classical parallelizing compiler scene, like software pipelining [43], and, loop transformations [44] [45] [46] [47] (survey in [48]).

Mapping applications onto rDPAs. Classical systolic arrays could be used only for applications with regular data dependencies, because at that time linear projections or algebraic methods had been used for mapping, which yield only uniform arrays with strictly linear pipes. However, to-day for DPA synthesis or mapping applications onto rDPAs simulated annealing is used instead, to avoid the limitation to regular data dependencies [5] [25]. This *("super systolic array")* generalization of the systolic array by Kress [49] also supports inhomogenous irregular arrays, supporting also any wild shapes of pipes within rDPA pipe networks [20] [21].

Automatic partitioning. Until recently, not only for hardware / software co-design, but also for software / configware design, the compiler is a more or less isolated tool used for the host only. But accelerators are still implemented by CAD. *Software / configware partitioning is still done manually* [27] [50], requiring massive hardware expertise, particularly when hardware description language (HDL) and similar sources are used. Compilation from HLL sources [25] [26] [43] [51] still stem from academic efforts, as well as the first automatic *co-compilation* from HLL sources including automatic software/configware partitioning [52] (fig. 7 a) by identifying parallelizable loops [5] [35], having been implemented for the data-stream-based MoM (Map-oriented Machine) [21] [39] [42].

4.1 MACHINE PARADIGMS AND OTHER GENERAL MODELS

Simplicity of the machine paradigm. Machine paradigms are important models to alleviate CS education and for understanding implementation flows or design flows. The simplicity of the von Neumann paradigm helped a lot to educate zillions of programmers. Figure 3 a shows the simplicity of the block diagram, which has exactly one CPU and exactly one RAM module (memory M). The instruction sequencer and the DPU (datapath unit) are merged to be encapsulated within the CPU (central processing unit), whereas the RAM (memory M) does not include any sequencing mechanism. Other important attributes are the RNI mode (read next instruction) and a branching mechanism for sequential operation

(computing in the time domain.) Figure 9 compares both machine paradigms. Since compilers based on the "von Neumann" machine paradigm do not support morphware we need the data-stream-based anti machine paradigm (sometimes called Xputer paradigm[52]) for the rDPA side, (based on *data sequencer* [53]).

The anti machine has no von Neumann bottleneck.

The *Anti Machine* Paradigm for morphware [42] [55] and even for hardwired anti machines the data-stream-based anti machine paradigm is the better counterpart (fig. 3 b) of the von Neumann paradigm (fig. 3 a). Instead of a CPU the anti machine has only a DPU (datapath unit) without any sequencer, or a rDPU (reconfigurable DPU) without a sequencer. The anti machine model locates data sequencers on the memory side (fig. 3 b). Anti machines do not have an instruction sequencer. Unlike "von Neumann" the anti machine has no von Neumann bottleneck by allowing multiple data counters (fig. 3 c) to support multiple data streams from/to multiple auto-sequencing memory banks (fig. 3 c) allowing multi-port operational resources much more powerful than ALU or simple DPU: majorDPAs orrDPAs (fig. 3 d).

General purpose anti machine. The anti machine is as universal as the von Neumann machine. The anti programming language is as powerful as von-Neumann-based languages. But instead of a "control flow" sublanguage a "data stream" sublanguage like *MoPL* [33] recursively defines *data goto, data jumps, data loops, nested data loops,* and *parallel data loops.* For the anti machine paradigm all execution mechanisms are available to run such an anti language. Its address generator methodology includes a variety of escape mechanisms needed to interrupt data streams by decision data or tagged control words inserted in the data streams [55]. Figure 9 compares both paradigms.

Architectural ressources, conform with the discipline of embedded distributed memory. The anti machine model, where the DPUs are transport-triggered by arriving data, goes conform with the new and rapidly expanding R&D area of embedded distributed memories [34] [37] [37], including the architectural ressources, like application-specific or programmable data sequencers (see [40] [53] [54]).

5. TURNING PC INTO PS (PERSONAL SUPERCOMPUTER)

Many application areas. There is a number of HPC application areas, where the desired performance is hard to reach by "traditional" high performance computing. For instance, the gravitating n-body-problem is one of the grand challenges of theoretical physics and astrophysics [56]. Also hydrodynamic problems fall in the same category, where often numerical modeling can be used only on the fastest available specialized hardware. Analytical solutions exist only for a limited number of highly simplified cases. For interpretation of dense centers of galactic nuclei observed with the Hubble Space Telescope to unite the hydrodynamic and the gravitational approach within one numerical scheme. Until recently this limited the maximum particle number to about a 10^5 even on largest supercomputers available. The situation improved by the GRAPE special purpose computer [57]. To improve the flexibility a hybrid solution has been introduced with AHA-GRAPE, which includes auxiliary morphware (FPGA-based processors) [58]. Another morphware usage example is cellular wireless communication, where the performance requirements grow faster than Moore's law [59] [60].

6. CONCLUSIONS

The paper has given an introductory survey on reconfigurable logic and reconfigurable computing, and its impact on classical computer science. It also has pointed out future trends driven by technology progress and innovations in EDA. It has tried to highlight, that deep submicron allows SoC implementation, and the silicon IP business reduces entry barriers for newcomers and turns infrastructures of existing players into liability.

The paper tried to illustrate, why many system-level integrated future products without reconfigurability will not be competitive. Instead of technology progress better architectures by reconfigurable platform usage will be the key to keep up the current innovation speed beyond the limits of silicon. The paper advocates that it is time to revisit past results from morphware-related R&D to derive promising commercial solutions, and, that curricular updates in basic CS education are urgently needed. The exponentially increasing of CMOS mask costs demands urgently adaptive and re-usable silicon area, which can be efficiently

realized by integrating (dynamically) reconfigurable hardware parts on different granularities into sSoCs with great potential for short time-to-market (-> risk minimization), multi-purpose/-standard features incl. comfortable application updates within product life cycles (-> volume increase: cost decrease). This results in the fact that several major industry players are currently integrating reconfigurable cores/datapaths into their processor architectures and system-on-chip solutions.

7. LITERATURE

[1] J. Becker, R. Hartenstein (invited paper): Configware and Morphware going Mainstream; Journal on System Architecture (JSA), 2003

[2] Arvind et al.: A critique of Multiprocessing the von Neumann Style; Proc. ISCA 1983

[3] G. Bell (keynote): All the Chips Outside: The Architecture Challenge; Proc. ISCA 2000

[4] J. Hennessy: ISCA25: Looking Backward, Looking Forward; Proc. ISCA 1999

[5] R. Hartenstein (invited paper): The Microprocessor is no more General Purpose, Proc. ISIS'97

[6] R. Hartenstein (opening keynote): Are we really ready for the Break-through?; Proc. Reconfigurable Architectures Workshop (RAW 2003), Nice, France, April, 2003

[7] R. Hartenstein (keynote): A Mead-&-Conway-like Breakthrough is overdue; Dagstuhl, July 2003

[8] D. Gajski et al.: A second opinion on dataflow machines; Computer, Feb 1982

[9] : http://xputers.informatik.uni-kl.de/staff/hartenstein/lot/ICECS2002Hartenstein.ppt

[10] J. Becker et al.: Parallelization in Co-Compilation for Configurable Accelerators; Proc. ASP-DAC´98

[11] coined within the Adaptive Computing Programme, funded by DARPA

[12] A. DeHon: The Density Advantage of Reconfigurable Computing; Computer, April 2000

[13] R. Hartenstein (embedded tutorial): A Decade of Research on Reconfigurable Architectures - a Visionary Retrospective; DATE 2001, Munich, March 2001

[14] J. Becker, T. Pionteck, M. Glesner: An Application-tailored Dynamically Reconfigurable Hardware Architecture for Digital Baseband Processing; SBCCI 2000

[15] http://pactcorp.com

[16] V. Baumgarten, et al.: PACT XPP - A Self-Reconfigurable Data Processing Architecture; ERSA 2001

[17] J. Becker, A. Thomas, M. Vorbach, G. Ehlers: Dynamically Reconfigurable Systems-on-Chip: A Core-based Industrial/Academic SoC Synthesis Project; IEEE Workshop Heterogeneous Reconfigurable SoC; April 2002, Hamburg, Germany

[18] J. Cardoso, M. Weinhardt: From C Programs to the Configure-Execute Model; DATE 2003

[19] M. Vorbach, J. Becker: Reconfigurable Processor Architectures for Mobile Phones; Reconfigurable Architectures Workshop (RAW 2003), Nice, France, April, 2003

[20] U. Nageldinger et al.: KressArray Xplorer: A New CAD Environment to Optimize Reconfigurable Datapath Array Architectures; Proc. ASP-DAC 2000.

[21] U. Nageldinger et al.: Generation of Design Suggestions for Coarse-Grain Reconfigurable Architectures; Proc. FPL 2000

[22] J. McCanny et al. (Editors): Systolic Array Processors; Prentice Hall; 1989

[23] M. Foster, H. Kung: Design of Special-Purpose VLSI Chips: Example and Opinions. ISCA 1980

[24] H. T. Kung: Why Systolic Architectures? IEEE Computer 15(1): 37-46 (1982)

[25] R. Kress et al.: A Datapath Synthesis System for the Reconfigurable Datapath Architecture; ASP-DAC'95

[26] J. Frigo, et al.: Evaluation of the streams-C C-to-FPGA compiler: an applications perspective; FPGA 2001

[27] T. J. Callahan: Instruction-Level Parallelism for Reconfigurable Computing; FPL'98

[28] E. Caspi, et al.: Extended version of: Stream Computations Organized for Reconfigurable Execution (SCORE): Proc. FPL '2000

[29] T. Callahan: Adapting Software Pipelining for Reconfigurable Computing; CASES 2000

[30] C. Chang, K. Kuusilinna, R. Broderson, J. Rabaey: The Biggascale Emulation Engine; summer retreat 2001, UC Berkeley

[31] H. Kwok-Hay So, BEE: A Reconfigurable Emulation Engine for Digital Signal Processing Hardware; M.S. thesis, UC Berkeley, 2000

[32] C. Chang, K. Kuusilinna, R. Broderson: The Biggascale Emulation Engine; FPGA 2002

[33] A. Ast, et al.: Data-procedural Languages for FPL-based Machines; Proc. FPL'94

[34] M. Herz et al.: (invited paper): Memory Organization for Data-Stream-based Reconfigurable Computing; Proc. ICECS 2002,

[35] J. Becker: A Partitioning Compiler for Computers with Xputer-based Accelerators; Ph. D. dissertation, Kaiserslautern University, 1997

[36] F. Catthoor et al.: Data Access and Storage Management for Embedded Programmable Processors; Kluwer, 2002

[37] F. Catthoor et al.: Custom Memory Management Methodology Exploration of Memory Organization for Embedded Multimedia Systems Design; Kluwer, 1998

[38] P. Kjeldsberg, F. Catthoor, E. Aas: Data Dependency Size Estimation for use in Memory Organization; IEEE Trans. on CAD, 22/5 (July 2003)

[39] M. Weber et al.: MOM - Map Oriented Machine; in: E. Chiricozzi, A. D'Amico: Parallel Processing and Applications, North-Holland, 1988

[40] H. Reinig et al.: Novel Sequencer Hardware for High-Speed Signal Processing; Proc. Design Methodologies for Microelectronics, Smolenice, Slovakia, Sept.1995

[41] A. Hirschbiel et al.: A Flexible Architecture for Image Processing; Microprocessing and Microprogramming, vol 21, pp 65-72, 1987

[42] M. Weber et al.: MoM - a partly custom-designed architecture compared to standard hardware; IEEE CompEuro 1989

[43] M. S. Lam: Software Pipelining: an effective scheduling technique for VLIW machines; ACM SIGPLAN Conf. PLDI, 1988

[44] L. Lamport: The Parallel Execution of Do-Loops; CACM 17,2, Feb. 1974

[45] D. Loveman: Program Improvement by Source-to-Source Transformation; J.ACM Jan 1977

[46] W. Abu-Sufah et al.: On the Performance Enhancement of Paging Systems Through Program Analysis and Transformations; IEEE-Trans. C-30(5), (May 1981)

[47] J. Allen, K. Kennedy: Automatic Loop Interchange; Proc. ACM SIGPLAN'84 Symp. on Compiler Construction, June 1984

[48] K. Schmidt et al.: Automatic Parallelism Exploitation for FPL-based Accelerators; HICSS'98

[49] N. Petkov: Systolic Parallel Processing; North-Holland; 1992

[50] M. Budiu and S. Goldstein: Fast Compilation for Pipelined Reconfigurable Fabrics; FPGA'99

[51] I. Page, W. Luk: Compiling occam into FPGAs; Proc. FPL 1991

[52] J. Becker et al.: A General Approach in System Design Integrating Reconfigurable Accelerators; Proc. IEEE ISIS'96; Austin, TX, Oct. 9-11, 1996

[53] M. Herz, et al.: A Novel Sequencer Hardware for Application Specific Computing; . ASAP'97

[54] M. Herz: High Performance Memory Communication Architectures for Coarse-grained Reconfigurable Computing Systems; Dissertation 2001, Univ. Kaiserslautern

[55] R. Hartenstein et. al. (invited reprint): A Novel ASIC Design Approach Based on a New Machine Paradigm; IEEE J.SSC, Volume 26, No. 7, July 1991

[56] R. Hartenstein (keynote address): Data-Stream-based Computing and Morphware; Joint 33rd Speedup and 19th PARS workshop; Basel, Switzerland, March 2003

[57] N. Ebisuzaki et al.; 1997 Astrophysical Journal, 480, pp. 432,

[58] R. Männer, R. Spurzem et al.: AHA-GRAPE: Adaptive Hydrodynamic Architecture - GRAvity PipE; Proc. FPL 1999

[59] J.Becker (invited paper): Configurable Systems on Chip; Proc. ICECS 2002

[60] J. Rabaey (keynote): Silicon platforms for the next generation wireless systems; FPL 2000