The von Neumann Syndrome


Arthur Schopenhauer: "Approximately every 30 years, we declare the scientific, literary and artistic spirit of the age bankrupt. In time, the accumulation of errors collapses under the absurdity of its own weight."
RH: "Mesmerized by the Gordon Moore Curve, we in CS slowed down our learning curve. Finally, after 60 years, we are witnessing the spirit from the Mainframe Age collapsing under the von Neumann syndrome."

Abstract. Because of high energy consumption our computer-based infrastructure will become unaffordable - without reinventing the entire computing discipline, also due to cope with the manycore programming crisis. The paper highlights facts, trends, and a roadmap to by-pass this crisis and to reach new horizons.

The term von Neumann syndrome has been coined by C. V. "RAM" Ramamoorthy in reply to my talk at San Diego [1]. Most problems are caused by the Energy Wall, the Memory Wall, and the Education Wall. Still main focus of CS education, the von Neumann (vN) basic common model [2] lost its dominance decades ago [3], also having been criticized for overhead [5] [6]: its principles are fundamentally wrong, since data processing targets data streams - not instruction streams. In industry it has been replaced by a cooperation of vN CPU and non-vN accelerators (fig. 1). To-day, most MIPS equivalents are running on FPGAs [7]-[9] (Field-Programmable Gate Arrays [10]: fastest growing segment of the semiconductor market), where the microprocessor has become the tail wagging the dog and the basic accelerator model is data-stream-based - not instruction-stream-based. However, most published documentations of such symbiotic systems use a home-brewed confusing and/or strikingly selfish terminology and are (un)structured like Ratatouille, not at all straightening out an underlying twin paradigm common basic model.

von Neumann computing becoming unaffordable during the next decade!

The most disruptive revolution since the mainframe: it’s Reconfigurable Computing (RC) [11] [12] [14] [15] [16] [17], mostly FPGA-based. Its perversiveness is obvious. It comes with a second machine paradigm: the anti-machine, counterpart of vN [18] [19]. Meanwhile RC has become mainstream, not only in embedded systems. More than 170 international conference series deal with RC and its applications [20]. Not only in digital consumer electronics getting momentum from market convergence, RC is the key for future architectures: field-programmability is a must [21]. Since 2006, RC is also a hot spot in supercomputing [22] [23]: The personal supercomputer is near [24].

The 1st Reconfigurable Computing Paradox. From software to configware migrations, speedup factors by up to 4 orders of magnitude (e.g. 3000 in image processing [26] and 28514 in DES breaking [27] [28]) have been published (fig. 2), despite of bad FPGA technology parameters [12]. The effective integration density of a large FPGA is tremendously behind the Gordon Moore curve (fig. 3). Compared to speedup the discrepancy is up to 8 orders of magnitude. What id the explanation? (Instead of „simple FPGAs“ (fig. 3) some more recent projects from fig. 2 used platform FPGAs having a better integration density by including a domain-specific mix of hardwired module blocks embedded in FPGA fabrics.) Part of the explanation is the Gordon Moore gap. The Moore curve does not show the actual computational density (effective MIPS per area unit) of microprocessor chips which has drastically decreased with the sequence of generations [12] [13]. The discrepancy also indicates, that our common...
models and implementation principles are fundamentally wrong: the von Neumann syndrome. Following the vN-centric spirit from the mainframe age in using technology of the silicon age we are navigating with a completely wrong road map. We need to think out of the box. Reconfigurable Computing is leading us to new roads we need to escape from the vN paradigm trap, tunnel of horror. The M’soft [instruction-stream-based] programming model for upcoming manycore microprocessors has been predicted to be 10 years off" [25]. For an earlier solution we need a twin paradigm approach: instruction-stream-based solutions coordinated with data-stream-based antimachine concepts. Each core should have the option to run as a vN CPU, or, as an antimachine’s DPU (DataPath Unit: has no program counter), s. fig. 5.

We must reinvent computing [44].

ANTI MACHINE: Each core should have the option to run as a vN CPU, or, as an antimachine’s DPU (DataPath Unit: has no program counter), s. fig. 5.

Missing the point: because of tunnel view. For a hammer everything looks like a nail. For some mathematicians many phenomena look like an algebraic problem. Around 1980 it has been found out, that systolic arrays are beautiful schemes to solve complex algebraic problems [29] [30] [31], and, have come up with an excellent definition of the term data stream - nicely illustrated by a time/space schematics (fig. 6). Their several synthesis methods to derive the pipe network of a systolic architecture from a mathematical formula has been, of course, algebraic. Since this means linear projection yielding only uniform linear pipes, these synthesis methods have supported only applications with strictly regular data dependencies, which is a far-ranging limitation. It took almost 15 years, to overcome this limitation, which stems from the tunnel view of the algebraic perspective. In 1995 Rainer Kress replaced their algebraic synthesis methods by simulated annealing [32], which in fact, means a generalization of the systolic array. The resulting supersystolic array methodology supports all kinds of irregular pipe schemes, including (not only) spiral, zigzag and even much more wild schemes, even also with fork and join features. By the way, the original conference series on systolic arrays [33] [34] extended its scope [35] [36].

von Neumann syndrome: an important strategic issue at least at national level.

Datastream i.e. transport-triggered execution triggered by dataitems arriving at particular ports of the DPU at particular times.

Fig. 5. Duality of paradigms: von Neumann vs. antimachin.

Anti machine: delayed by paradigm trap. Who organizes the data streams to run systolic arrays? The reply of that systolic array scene around 1980 has been: „this is not our job. [This is the job of hardware people.]“ Since computing resources without a sequencer are not a computational „machine“ in the sense of a machine paradigm, those mathematicians have missed to invent the new machine paradigm of the antimachine [37] [38] [39] - the counterpart of the von Neumann machine. This „transdisciplinary“ break-through had been delayed by their reluctance due to their algebraic paradigm trap. Each individual data stream is generated by an auto-sequencing memory (ASM) block including a data counter. For the array example in fig. 6 we need 12 ASMs, 6 of them as data stream sources, and 6 of them as data stream sinks (fig. 7). So this machine has 12 data counters in total.

Compared to von Neumann the main differences of the antimachine are: (mostly multiple) data counters, co-located with memory - in contrast to vN’s single program counter, co-located with the single datapath unit (DPU). The anti machine does not have a CPU; it only has (mostly multiple) DPUs, i.e. without program counters. (Datastreams ≠ dataflow). Note: only use the term datastreams, but avoid to use the obsolete term dataflow! [40] The two machine paradigms are twins, because to express sequencing the same language primitives are used (fig. 8) [41]. Both paradigms have the same syntax rules. Their sequencers use the same circuitry. Their semantics is only slightly different. The only external asymmetry is the fact, that data stream loops can be internally parallel at this level, whereas instruction stream loops cannot. A von Neumann machine can have only a single DPU (inside the CPU), whereas an antimachine can have multiple DPUs.

The von Neumann tunnel of horror. The vN paradigm is preferred by rationally bounded humans for reasons of Denkoekonomie ([Ernst Mach] [42]). Scarce resources (intelligence) are substituted as soon as possible. vN’s beneficiaries Intel and Microsoft gain from the fact that the programmer does not need to think a lot about many difficult aspects of computing [43]. However, our von-Neumann-centric cyber-infrastructure is a tunnel of horror: astronomical code sizes cause a massive array of overhead phenomena, due to the von Neumann syndrome. 1000 processors running in parallel means that 1000 instruction streams with all their overhead phenomena yield a drastic programmer productivity decline. [44]: „In practice we are limited to a few instructions per clock cycle.“ Traditional software engineering problems are now topped by the manycore programming crisis. The human wave approach toward improving components not being the main system bottlenecks to come up with a variety of speculative and other indeterministic methodologies, recently topped by transactional memory efforts. A huge waste of researcher capacity to obtain mostly marginal results: the mountain screamed and bore a mouse. Also
multithreading is speculative and is not the silver bullet. Overhead phenomena also lead to microprocessor chips featuring a highly disappointing computational density [13]. This is only an incomplete list of indications to the von Neumann syndrome. The table in fig 10 lists some of the von Neumann overhead phenomena which can be avoided by software to configure migration, i.e. by RC.

**A new paradigm cannot be avoided.** Most researchers and implementers in HPC are reluctant to go for a paradigm extension. This is (not only) illustrated by following panelists’ statements from SC06: “It is feared that domain scientists will have to learn how to design hardware. Can we avoid the need for hardware design skills and understanding?” [45], “A leap too far for the existing HPC community” [45]. *Trying to avoid paradigm revisions leads to a completely wrong road map.* The following statement sounds somewhat better: “We need a bridge strategy by developing advanced tools for training the software community to think in fine grained parallelism and pipelining techniques.” [45]. Such a bridge strategy also makes sense, because of the manycore programming crisis running in parallel with the break-through of Reconfigurable Computing.

**Everything we know is wrong.** The vN syndrome reminds of a Freudian repression of material parallelism. RC is more equivalent to our natural unconscious intelligence. However, the revolution is painful, since consciously parallel thinking is hard, is rarely systematically trained, and is badly supported by established tools [43]. Since parallelism now becomes increasingly ubiquitous, HPC should be capable to exploit and extend mainstream programming languages, operating systems, development tools, libraries, and even applications written for smaller scale systems [44]. Also grid computing requests delivering its functionality through far simpler programming interfaces: “The Grid is sometimes its own worst enemy. Grid computing misses the point.” [46] [47]: “If we want to enable new science then we need to empower the user.” To make it much easier for developers to implement parallel software and systems, we urgently must reinvent not only computing but also the computing profession [44]. Everything we know is wrong [49]. This requires a paradigm revision for execution and programming models. [50].

**Twin paradigms approach is needed.** The von Neumann syndrome tells us, that instead of physical limits, fundamental misconceptions of algorithmic complexity theory limit the progress and will necessitate new breakthroughs. Not processing is costly, but moving and storing data and messages. We’ve to completely rethink basic assumptions behind computing. Moving data (at run time) in the vN domain turns for the antimachine domain into (at compile time) moving the locality of execution by pipe network synthesis (line 7 and 8 in fig. 8). Another example are reconfigurable address generators moving address computation overhead (lines 9 and 10 in fig. 10) from run time to compile time [51] [52] [53]. Supporting compilation techniques, also featuring automatic software / configware partitioning, have been demonstrated, accepting C language sources [54] [55] [56] [57], and mathematical formula input sources [58]. *Cores of manycore systems should be heterogeneous,* also including reconfigurable cores like FPGAs etc. Because we cannot afford to discard non-von-Neumann accelerators, we have to support a twin paradigm approach: a well and clearly organized dual model supporting both, von Neumann and the antimachine paradigm.

**The 2nd Reconfigurable Computing Paradox.** Compared to hardwired accelerators, FPGAs have bad technology features, such as slow clock frequency, and, higher energy and space requirements. Many publications report speedup factors obtained from software to configure migration (onto FPGAs [7] [12]) - up to a factor of 6000 (fig. 2). But only one of those publications reports slashing the electricity bill and space needed down to less

<table>
<thead>
<tr>
<th>#</th>
<th>feature</th>
<th>von Neumann machine</th>
<th>hardwired antimachine (e.g. [59])</th>
<th>reconfigurable antimachine</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>machine code schedules</td>
<td>instruction stream</td>
<td>data stream(s)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td># of programming sources</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>programming source 1*</td>
<td>software</td>
<td>none</td>
<td>configware</td>
</tr>
<tr>
<td>4</td>
<td>programming source 2**</td>
<td>flowware</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>[source 2] sequenced by</td>
<td>1 program counter</td>
<td>1 of more data counter(s)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>counter co-located with</td>
<td>DPU (data path), CPU</td>
<td>memory block(s), ASM</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>inter PU communication</td>
<td>via common memory</td>
<td>piped through</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>data meet PU (processing unit)</td>
<td>move data at run time</td>
<td>move execution locality at compile time</td>
<td></td>
</tr>
</tbody>
</table>

* To set up resources
** for scheduling

than 10 percent, for a speedup factor of only 17 [48]. This means a discrepancy factor of substantially more than 100 in terms of Watts per effective MIPS. The (semi-RC) GRAPE machine shows, that the electricity consumption per MIPS might go down to about one tenth of a percent [67] [68] [69]. Recently a power saving factor of 3439 for a 28514x speedup factor has been reported for DES breaking (fig. 9), also causing a massive impact on security overhead. Megabit keys needed soon would make by far unaffordable a von Neumann machine use for encryption.

The von Neumann syndrome: the Energy Wall. The electricity consumption of computers has been mainly ignored (fig. 11). But recently a discussion has been kicked off within the HPC scene [23]. The energy consumption of future supercomputers is heading for astronomic dimensions. The discussion now also includes server farms. Google’s annual electricity bill amounts to 50 million US$ - more than the value of its compute equipment. 25% of Amsterdam’s electricity consumption goes to server farms. New York city servers occupy a quarter square km of building floor area. A study predicts, that (from currently >20%) by the year 2020 the electricity needed soon would make by far unaffordable a von Neumann machine use for encryption.

The von Neumann syndrome: very high cost. From software to configware migration a hardware cost reduction to ~10% has been reported, also reducing the number of racks needed to ~10% [48]. Much more is expected for the future. A drastic reduction of floor area needed is a substantial cost factor, because of the building size needed. Also, if needed at all, the cost of air conditioning equipment and the electricity bill coming with it is massively reduced: another motivation for a paradigm revision.

Mapping from time to space: the Education Wall. In contrast to vN, the anti machine is data-stream-based: no instruction fetch at run time. Mapping an application from software to configware means mapping from time to space - a domain hated by people with a software-only background. A Japanese CTO when introduced [63] to rDPAs [64]: „But you cannot implement decisions!“ We immediately see the Education Wall in his brain. How to map a flowchart's decision box into the space domain? It turns into a demultiplexer controlled by the decision bit running on an extra wire instead of sitting in a CPU’s register: it’s branching in space. (The Register Transfer Module system (1972) sold by Digital Equipment [65], and a similar system (1967) from academia [66], are based on such a mapping of flowcharts.) See, how this important finding was commented in the 70ies by the HDL scene: „This is so simple. Why did it take 25 years to find out?“ This backlog was due to the tunnel view of the software-only mind set. Meanwhile time to space mapping is an old hat: we should stop ignoring it. This request is urgently addressed to our curriculum recommendation groups like the joint ACM/IEEE-CS task force, still hopelessly reluctant to discuss RC issues. Such a software-only mind set misses the IT job market. To create better visibility of these problems and to encourage activities to cope with the Education Wall the International Annual Conference Series on RC education has been founded. [70] [71].
Dynamically Reconfigurable FPGAs [72] are partially reconfigurable where parts of it can be running while other parts are being reconfigured. Dispatching, scheduling and swapping configure macros are the job of a configure OS (configure operating system [73]). „No instruction fetch at run time“ still holds, when the model used here follows a clean definition. This is a bit difficult to explain to beginners without some advanced FPGA background.

<table>
<thead>
<tr>
<th>Source</th>
<th>is compiled into:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software</td>
<td>an instruction schedule</td>
</tr>
<tr>
<td>Flowware</td>
<td>a data schedule</td>
</tr>
<tr>
<td>Configure</td>
<td>a pipe network by placement and routing</td>
</tr>
</tbody>
</table>

Fig. 14. Sources by compilation targets.

**Fully fledged paradigm shift not needed.** Classical CS knowledge is still important (also to run legacy software). We need only the additional adoption of a second paradigm, which is only partially different from the von Neumann mind set. The second paradigm, the antimachine, is a twin brother of the von Neumann paradigm. The control syntax is mainly the same (fig. 8). Only the semantics is different: controlling data streams instead of instruction streams. This helps us to find a good bridge strategy to go from von Neumann single paradigm to a twin paradigm methodology. Most ingredients of the antimachine methodology are rather old stuff - mostly ignored by the CS community. Most of the enabling technologies of RC and to cope with the von Neumann syndrome, have been published at least 20 years ago, like for instance, about loop transformations, routing algorithms, languages to express parallelism, compilation techniques, data streams, systolic and supersystolic arrays, software to hardware migration etc. The point of view may be slightly different to day.

**Understandable modelling scheme needed.** A global system view is required for grasping the principles and essential issues of the contemporary heterogeneous twin paradigm systems, not only in undergraduate education, we need an intuitive terminology and an understandable common modeling scheme. A style of schematics with a clear distinction between von Neumann subsystems and antimachine blocks helps a lot. We distinguish 3 different types of programming sources (fig. 14) [74]. Instead of only type of source (“software”) in von-Neumann-only systems (fig 15 b), two more kinds of sources are added by RC (fig. 15 c), which we should not call software: it’s configware to set up the structure of reconfigurable resources, and flowware for scheduling the data streams, according to configware compilation results. To create and understand such schemes we should clearly distinguish different types of programming sources: source type 1 (row 3 in fig 8.) to set up resources (not needed for hardened machines), and source type 2 (row 4 in fig 8) for scheduling (instruction streams by software for von Neumann machines, or, by flowware for data streams at antimachines). This twin paradigm approach means the interweavement of 2 cultures: a transdisciplinary approach (fig. 16), affiliating the instruction-stream-based mind set (computing in time: procedural semantics) with the data-stream-based mind set (computing in space: structural semantics). It is easy, since the syntax is mainly the same (fig. 8). Affiliating should not mean mixing. Although the semantics is different, it should avoid confusion by navigating with a clean coordinate system: this challenge to educators can definitely be mastered.

![Reconfigurable Computing: for mastering the education wall.](image)

**Platform FPGAs** include a domain-specific mix of hardwired module blocks like LUTs, multipliers or DSP cells, memory objects, ALUs, etc., specialized ALUs, vN processors, sometimes with customizable instruction set processors, and even analog components. Such a heterogeneous mix poses significant new challenges for programmers and for synthesis software. This makes quantification of the performance and capacity of modern FPGAs – and particularly comparison of various arrays – almost impossible.

![Reconfigurable Computing (reconfigurable antimachine):](image)

The von Neumann paradigm is tremendously overhead-based - in contrast to the AntiMachine.

Using rDPAs (coarse-grained reconfigurable arrays) is the best strategy to bridge the education wall - by raising the abstraction level.

![Configware](image)
strategy to bridge the software/configware chasm. Well-known and easily understandable loop transformations are smoothly mappable into pipe networks to be configured on rDPAs [54] [55]. Configware code size (much faster configuration time) and effective technology parameters of rDPAs (fig. 4) are much better than those of FPGAs (fig. 3) by about 4 orders of magnitude. Techniques for rDPA-based co-compilation and design space exploration, including automatic software/configware partitioning and interface generation, have been demonstrated [54] [64] [77].

rDPA coarse-grained arrays to avoid the memory wall. Fig. 16 illustrates the performance advantage of a coarse-grained rDPA array (row 2) over a manycore array of CPUs (row 1). Moving data between CPUs goes through common memory needing instructions slowed down by the memory wall for both: moving the data and to evoke executions on the CPUs (line 1). However, via compilation techniques for a coarse-grained reconfigurable array (placement and routing) the interconnect between rDPUs is configured to form a pipe network such, that data are directly pushed without needing common memory. This avoids data memory cycles. The execution within each rDPU is triggered by handshake, i.e. by the arrival of data piped through directly from another rDPU. This avoids instruction memory cycles.

rDPA coarse-grained arrays vs. platform FPGAs. rDPUs inside rDPAs are reconfigurable, whereas hardware blocks in platform FPGAs are not. Following a less understandable model, platform FPGAs are less suitable for a bridge strategy than rDPAs. In contrast to FPGAs, rDPAs like the XPP array from PACT [76] provide higher speedups and lower energy consumption - coming with a compilation environment including tools for automatic interfacing CPUs with rDPUs - closing a zipper (fig. 16).

Conclusions. RC is an essential qualification for the manycore future [43], HPC and supercomputing. The performance of von Neumann computing systems is dramatically behind expectations. Gordon Moore’s curve is far away from indicating computing performance. Von Neumann-based computing comes along with a tremendous array of instruction stream overhead phenomena, which are not coming with RC methodologies having no instruction fetch at run time. Von-Neumann-based programmer productivity progressively declines with an increasing number of processor cores involved. The equipment cost and energy consumption of von-Neumann-based computing are too high by more than an order of magnitude. Available RC methodology to cope with this syndrome is mainly ignored. An Exception are embedded systems. Curriculum recommendations miss this present and future job market, ignoring that by far most software is written for embedded systems and most MIPS run on FPGAs. Educational deficits hamper the development of better development tools for better acceptance. We urgently need (1) Reconfigurable Computing education for the entire CS / IT community, and (2) to update CS curricula by a twin paradigm zipper strategy throughout entire course programs. RC should be established as a vehicle to fascinate learning for the manycore future and to reverse the enrolment downturn.

The personal supercomputer is near.

RC is an essential qualification for the many-core future.

von Neumann Antimachine

instruction-stream-based

programming: time domain (software)

CPU cores

co-compilation [53] [57] [76], OS [73] et al., interfacing, etc.

rDPU cores (coarse-grained array)

& other accelerators

configware

programing: space domain (flowware)

data-stream-based

Antimachine (reconfigurable)

Fig. 16. Twin paradigm zipper strategy to jazz up CS education by Reconfigurable Computing.

Literature


