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 may 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. 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 In industry it has been replaced by a cooperation of vN CPU and non-vN accelerators (fig. 1).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] (Field-Programmable Gate Arrays [8]: the 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 confusing and/or selfish terminology and are structured like stirred up Spaghetti Bolognese, not at all straightening out the underlying twin paradigm common basic model.

Energy Wall, Memory Wall and Education Wall fuel the von Neumann syndrome.

The most disruptive revolution since the mainframe: it’s Reconfigurable Computing (RC) [9] [10] [12] [13] [14] [15], mostly FPGA-based. Its pervasiveness is obvious. It comes with a second machine paradigm: the anti-machine, counterpart of vN [16] [17]. Meanwhile RC has become mainstream, not only in embedded systems. More than 170 international conference series deal with RC and its applications [18]. Not only in digital consumer electronics getting momentum from market convergence, RC is the key for future architectures; field-programmability is a must [19]. Since 2006, RC is also a hot spot in supercomputing [20] [21]: The personal supercomputer is near [22].

The 1st Reconfigurable Computing Paradox. From software to configware migrations, speedup factors by more than 3 orders of magnitude (e.g. 3000 in image processing [24]) have been published (fig. 2), although FPGA technology parameters are very bad [10]. 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 explains such excellent results by such a bad technology? (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 [10] [11]. 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, from this tunnel of horror. The M’soft [instruction-stream-based] programming model for upcoming manycore microprocessors has been predicted to be 10 years off [23]. 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. The vN paradigm is preferred by rationally bounded humans for reasons of Denkoekonomie ([Ernst Mach] [38]). 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 different aspects of computing [39]. 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. [40]: “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

<table>
<thead>
<tr>
<th>type of PU (processing unit)</th>
<th>instruction sequencer included?</th>
<th>execution triggered by</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>yes</td>
<td>instruction fetch</td>
</tr>
<tr>
<td>DPU, or, rDPU</td>
<td>no</td>
<td>arrival of data*</td>
</tr>
</tbody>
</table>

*) transport-triggered

Figure 5. Duality of paradigms: von Neumann vs. antimachine. 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 [28], 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 [29] [30] extended its scope [31] [32].

Anti machine: delayed by paradigm trap. Who organizes the data streams to run systolic arrays? The reply of the mathematicians’ 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 [33] [34] [35] - 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! [36]) The two machine paradigms are twins, because to express sequencing the same language primitives are used (fig. 7) [37]. 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] [38]). 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 different aspects of computing [39]. 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. [40]: “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 [11]. This is only an incomplete list of indications to the von Neumann syndrome. The table in fig 9 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?“ [41]. „A leap too far for the existing HPC community“ [41]. 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.“ [41]. Such a bridge strategy also makes sense, because of the manycore programming crisis running in parallel with the break-through of RC.

Reconfigurable Computing means: no instruction fetch at run time.

Twin paradigms approach is needed. The von Neumann 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 [39]. Since parallelism now becomes increasingly ubiquitous, HPC should be able to exploit and extend mainstream programming languages, operating systems, development tools, libraries, and even applications intended for smaller scale systems [40]. 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.“ [42] [43]: “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 [40]. Everything we know is wrong [45]. This requires a paradigm revision for execution and programming models. [46].

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] [101]) - up to a factor of 6000 (fig. 2). But only one of those publications reports shaving 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. [55])</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>Hi-ware</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>(source 2) sequenced by:</td>
<td>1 program counter</td>
<td>1 or more data counter(s)</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>counter co-located with:</td>
<td>DPU (data path) „CPU“</td>
<td>memory block(s): „ASM“</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>inter PU communication</td>
<td>via common memory</td>
<td>piped through</td>
<td></td>
</tr>
<tr>
<td>7</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

Figure 8. von Neumann vs. Anti machine: Partitioning scheme.

than 10 percent, for a speedup factor of only 17 [44]. This means a discrepancy factor of substantially more than 100 in terms of Watts per effective MIPS. What about migrations with much higher speedups, e.g. 6000? The (semi-RC) GRAPE machine shows, that the electricity consumption per MIPS might go down to about one tenth of a percent [62] [63] [64].

The von Neumann syndrome: the Energy Wall. The electricity consumption of computers has been mainly ignored (fig. 10). But recently a discussion has been kicked off within the HPC scene [21]. 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-Dollars - more than the value of its computing equipment. And, about 25% of Amsterdam’s electricity consumption goes into server farms. Servers in New York city occupy a quarter square kilometer of building floor area. A study predicts, that - from currently more than 20% - by the year 2020 the electricity consumption of the entire cyber infrastructure in the US will amount to 35 - 50% of the US electricity production (fig. 10) [57]. The crude oil price development (fig. 11) and market predictions (fig. 12) lead to the question, whether computing will be affordable in the future. Conclusion: the von Neumann syndrome is a strategic issue, at least at national level.

<table>
<thead>
<tr>
<th>#</th>
<th>type of overhead</th>
<th>vN machine</th>
<th>hardwired</th>
<th>reconfigurable</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>state address computation over head run time</td>
<td>instruction stream</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>data address computation over head run time</td>
<td>instruction stream</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>inter PU communication over head run time</td>
<td>instruction stream</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>instruction fetch over head run time</td>
<td>instruction stream</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>data meet PU over head run time</td>
<td>instruction stream</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>synchronization over head run time</td>
<td>instruction stream</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>multiplexing [56] over head run time</td>
<td>instruction stream</td>
<td>reduced or none</td>
<td></td>
</tr>
</tbody>
</table>

Figure 9. Overhead avoided by antimachine w. distr. on-chip memory.

The Memory Wall. Dave Patterson’s law shows the memory chip bandwidth orders of magnitude behind microprocessor clock frequency: main reason of the tremendous instruction stream overhead. Because of usual vN-based software code size, it mostly cannot be stored at on-chip memory - in contrast to configure code tending to be massively more compact. Commercially available platform FPGAs can have up to more than 700 local on-chip memory blocks, so that data streams for many applications can be scheduled to/from fast local memory (fig. 8) [58]. This also helps to explain the first reconfigurable computing paradox.

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 also a reduction of hardware cost down to about 10% has been reported [44]. Also the number of racks needed has been reduced to roughly 10 percent. This yields a drastic reduction of floor area needed, which is also a substantial cost factor. 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.

The von Neumann syndrome: very high cost. From software to configure migration also a reduction of hardware cost down to about 10% has been reported [44]. Also the number of racks needed has been reduced to roughly 10 percent. This yields a drastic reduction of floor area needed, which is also a substantial cost factor. 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.

Undiscovered, the antimachine paradigm has been around for almost 3 decades: software used it indirectly via instruction streams.

Dynamically Reconfigurable FPGAs [67] 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 [68]). „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.

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. 7). 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 modelling 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. 13) [69]. Instead of only type of source („software“) in von-Neumann-only systems (fig 14 b), two more kinds of sources are added by RC (fig. 14 c), which we should not call software: it’s configure to set up the structure of reconfigurable resources, and flowware for scheduling the data streams, according to configure 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. 7). 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.

<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>

Figure 14. Nick Tredennick’s machine classification scheme.

a) early machines* (e.g. DDA):

\[
\begin{array}{ccc}
\text{no source code needed} & \text{resources fixed} & \text{algorithms fixed} \\
\end{array}
\]

b) von Neumann machine:

\[
\begin{array}{ccc}
\text{resources fixed} & \text{algorithms fixed} & \text{hardwired accelerators} \\
\end{array}
\]

c) Reconfigurable Computing (reconfigurable antimachine):

\[
\begin{array}{ccc}
\text{resources variable} & \text{algorithms variable} & \text{flowware source code} \\
\text{configware source code} & \text{scheduling the data streams} \\
\end{array}
\]

d) the hardwired antimachine*:

\[
\begin{array}{ccc}
\text{resources fixed} & \text{flowware source code} & \text{algorithms variable} \\
\text{scheduling the data streams} & \text{configware source code} \\
\end{array}
\]

Figure 14. Nick Tredennick’s machine classification scheme.

Coarse-grained Reconfigurable Computing: for mastering the education wall. The mental models of hardware and software engineers are unnecessarily set apart from each other [39]. But the market trend goes away from a bit-level FPGA hardware mind set, over to functional level with MAC, ALUs, DPUs and CPU-cores inside platform FPGAs [39] and rDPAs. A bridge strategy to cope with programmer’s reluctance is the use of coarse-grained reconfigurable data path arrays (rDPAs) [70] [59]. Their cores are rDPUs (reconfigurable data path units) not having a program counter (fig. 5). In contrast to using FPGAs the modeling with rDPUs reaches functional level, coming much closer to the software-based mind set. Much better than any kind of FPGAs, the rDPs (e.g. [71]) are the best educational strategy to bridge the software/configware.

We see a trend away from a bit-level FPGA hardware mind set, over to functional level with MAC, ALUs, DPUs, CPU-cores etc [39].

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

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.

The personal supercomputer is near.

Figure 15. Avoiding the memory wall by coarse-grained reconfigurable array instead of many core CPU array.

chasm. Well-known and easily understandable loop transformations are smoothly mappable into pipe networks to be configured on rDPAs [50] [51]. 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 [50] [59] [72].

**rDPA coarse-grained arrays for avoiding the memory wall.** Fig. 15 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 xFP array from PACT [71] 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 [39], 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. Exceptions are many areas in embedded systems. Curriculum recommendations fail to hit this present and future IT job market, missing to consider that 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 and training for the entire CS and 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 for fascinating learning for the manycore future [39] and to reverse the CS enrolment down trend. The potential performance gains and massive reductions of equipment cost and energy consumption by RC are by far too high to pass up such golden opportunities.

**Literature**
