Reiner Hartenstein

Generalization of Software Engineering via Reconfigurable Computing — Strong Motivation: toward New Horizons by Updating Programmer Qualifications

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 the absurdity of the von Neumann Syndrome."

The future of our computing ecosystem is facing a mind-blowing and growing electricity consumption, and a trend toward growing cost and shrinking availability of energy sources. And the strategy change from higher clock speed to manycore[1] is the biggest challenge our computing industry faces today. Six, eight or more CPUs[2] mostly talk to each other instead of getting work done. To take advantage of the additional cores mainstream applications have to be rewritten, unfortunately introducing new types of bugs. Here a sufficiently large qualified programmer population is far from existing.

Still a problem is the predominance of a CPU-centric monoprocessing mind set (table 1). At a US state governor summit meeting Bill Gates said, that the US educators teach computing like for the mainframe era, and, that he cannot hire such people. Mainstream academic software engineering education is crippling itself by ignoring, that we now live in a twin-paradigm world, since the traditional hardware / software chasm has turned into configware / software interfacing. Mainstream education ignores the many-core programming crisis[1].

computer machine model of the mainframe era	resources		sequencer		
	property	programming source	property	programming source	state register
„CPU“ instruction set processor	hardwired	-	**programmable**	**Software** (instruction stream)	**program counter**

Table 1: Computer Model of the Mainframe Era.

Accelerators are programmable

The „CPU“ (central processing unit) became less central, needing accelerator support (ASICs) like to run its own display. The CPU-centric old model became obsolete (table 1): the tail is wagging the dog. Rows 1-2 in table 2 show the de facto model. But later ASICs massively lost market shares in favor of reconfigurable accelerators like FPGAs.

On „FPGA“ Google finds 10 million hits. Why does software engineering still ignore this highly potent silver bullet candidate? Introduced in 1984 and now a 5 billion US-$ world market, FPGAs are a well proven technology rapidly heading for mainstream and also used in supercomputing by Cray and Silicon Graphics. Again our common model has changed[3] (table 2: rows 1-3): accelerators have become programmable[3], still ignored by software engineering. Why? NIH effect? Not Invented Here?

In contrast to a CPU programmed only by ***software***, a reconfigurable computing platform needs two program sources (table 2: row 3): ***„configware“*** and ***„flowware“***, both not at all instruction-stream-based (configware: not procedural nor imperative).

Now accelerators are programmable — but Software Engineering ignores: the hardware/ software chasm turning into configware/ software interfacing

Flowware, derived from the ***data stream*** defined for systolic arrays, may be used also without configware for hardwired machines (table 2: row 3), e. g. like BEE[4]. ***„Flowware“*** avoids confusion with the term „dataflow“ [5].

It's a Twin Paradigm World

Our model (table 2: rows 1-3) includes 2 procedural program paradigms: ***software*** to schedule instruction streams, and ***flowware*** to schedule data streams. This twin paradigm model is a dichotomy and supports interlacing both machine paradigms. The ISP (CPU) model is the ***von Neumann machine paradigm*** for sequencing by program counter. But flowware is based on sequencing by data counters. This counterpart and twin brother of the von Neumann paradigm is the ***data-stream machine paradigm***.

contemporary computer system machine model		resources		sequencer		
		property	programming source	property	program source	state register
1	hardwired accelerators	hardwired	-	hardwired	-	
2	instruction set processors	hardwired	-	**programmable**	**Software** (instruction stream)	**program counter**
3	reconfigurable accelerators	**programmable**	**Configware** (configuration code)	**programmable**	**Flowware** (data streams)	**data counter(s)**
4	flowware-programmable accelerators	hardwired	-	**programmable**	**Flowware** (data streams)	**data counter(s)**

Table 2: Contemporary Computer System Machine Model

The Dichotomy of Languages

The primitives of a software language and of a flowware language are mainly the same (table 3). Different is only the semantics: A software language deals with sequencing a program counter. A flowware language programs one or more data counters for implementing data streams. There is only one asymmetry: only one program counter (located in the CPU). But datastream machines may have several data counters running in parallel (located in asM data memory).

Language primitives to manipulate data counters are mainly the same as known for sequencing by program counter (table 3). There are 2 exceptions, so that flowware languages are more simple: 1) no data manipulation, since being set up by reconfiguration; 2) parallelism inside loops, since a datastream machine may have several data counters.

Software Engineering Education

Since accelerators have become programmable, the traditional hardware/software chasm has become extremely intolerable. We need a generalization of Software Engineering into Program Engineering covering both, time and space domains by including 3 paradigms: Software, Flowware, and Configware (table 2). We need to rearrange undergraduate courses, following the advice of David Parnas: „The biggest payoff will not come from new research but from putting old ideas into practice and teaching people how to apply them properly“. In the 70ies, when hardware description languages came up somebody said: „A decision box turns into a demultiplexer[6]. This is so simple. Why did it take 30 years to find out?“ It's the tunnel view perspective not only of software engineering. We need to extend our horizon.

#	language features	Software Languages	Flowware Languages
1	sequencing managed by	read next instruction, goto (instruction address), jump (to instruction address), instruction loop, and nesting, escapes instruction stream branching	read next data item, goto (data address), jump (to data address), data loop, and nesting, escapes data stream branching
2	parallelism	***no*** *parallel loops*	***yes,*** *parallel loops* (by multiple data counters)
3	data manipulation	yes	***no*** (hardwired or synth. from configware language)
4	state register	single **program counter** (located in CPU)	1 or more **data counter(s)** (located in ***asM*** memory)
5	instruction fetch	memory cycle overhead	***no*** memory cycle overhead
6	address computation	massive memory cycle overhead (depending on application)	reconfigurable address generator(s) in asM: ***no*** memory cycle overhead

Table 3. Software Languages versus Flowware Languages.

Programming education requires an interlacing twin paradigm approach. Two dichotomies alleviate dual-rail teaching:

1) The Machine Paradigm Dichotomy (von Neumann vs. Datastream machine)

2) The Relativity Dichotomy: time domain vs. space domain: helps parallelization:

Two old simple rules of thumb: a) loop turns into pipeline [1979], b) decision box turns into demultiplexer [1967][6].

In contrast to instruction set processors, where instruction stream scheduling is a task of the operating system, reconfigurable computing comes with a different OS world organizing data streams instead, and swapping configware for partially and dynamically reconfigurable platforms[7,8].

Speed-up by FPGA accelerators

From CPU software to FPGA configware migrations for a variety of application areas speedup factors from almost 10 up to more than 3 orders of magnitude have been published (fig. 1) by a number of papers I collected[9]. A factor of 3000 has been obtained in 3-D image processing for computer tomography. Biology showed speed-ups up to 8723 (Smith-Waterman pattern matching). Multimedia reports up to 6000 (real-time face detection). Cryptology reports for DES breaking a speed-up factor of 28,514[10].

Program Engineering — the Generalization of Software Engineering including 3 paradigms: Configware, and the twin paradigm dichotomy of Software and Flowware

FPGAs for Saving Energy

Some of these speed-up studies report energy saving factors, like 3439 for the DES breaker[10]. The same performance requires drastically less equipment. For instance only one rack or half a rack and no air conditioning, instead of a hangar full of racks. The energy saving factor tends to be roughly 10% of the speed-up factor

The von Neumann Syndrome

FPGA technology is worse than that of microprocessors: slower clock speed, and massive reconfigurability overhead. Orders of magnitude higher performance with a worse technology? We call this the ***Reconfigurable Computing paradox***. Software

By orders of magnitude more performance with worse technology: The Reconfigurable Computing Paradox — caused by the von Neumann syndrome

engineering's platforms are so inefficient, that they can be beaten easily with a worse technology. Prof. Ramamoorthy from UC Berkeley calls this scenario ***the von Neumann Syndrome***.

New Software Engineering: why?

The impact of very high energy consumption by all computers world-wide, offers a higher success potential than most other energy and climate policy issues. Google causes 2% of the world's electricity consumption. The internet alone causes more Greenhouse Gas emission than the world-wide air traffic. If current trends continue, a recent study sees an increase by a factor of 30 within 22 years[11]. Future unaffordability of our total computer operating cost is looming. We urgently need to motivate the opinion leaders in software engineering.

Conclusions

The scenario resembles the VLSI design revolution, the most effective project in the history of modern computer science. Isn't this a strong motivation for the software engineering scene? We again need such innovative education efforts: professors back to school! We need a world-wide mass movement qualifying most programmers for a world-wide change-over of many applications, what will create jobs for a decade. New Horizons in HPC and highly effective energy policy will be opened up by generalized Software Engineering.

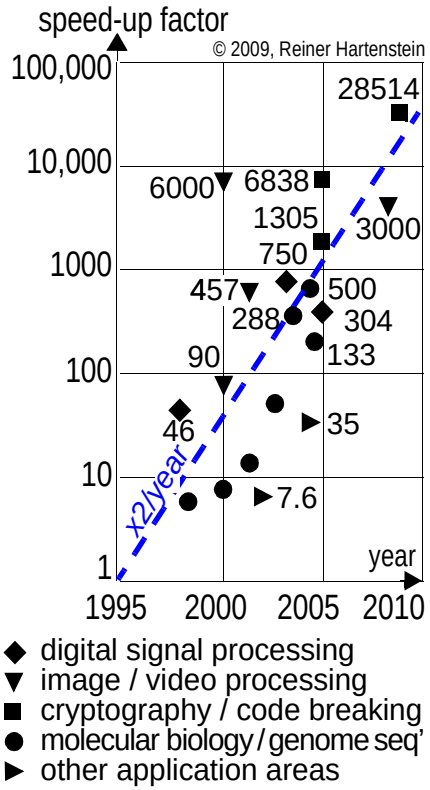

Fig. 1: Some speed-up factors reported from software to configware migration (mainly CPU to FPGA migration)[9].

A scenario like the VLSI design revolution: the most effective move in the history of modern computer science —

Isn't it a strong motivation toward new horizons of software engineering?

Literature

1. S. Bhattacharyya et al.: OpenDF, A Dataflow Toolset for Reconfigurable Hardware and Multicore Systems; Swedish Workshop on Multicore Computing (MCC-08), Ronneby, Sweden, 27-28 Nov, 2008
2. S. K. Moore: Multicore Is Bad News For Supercomputers; IEEE Spectrum, Nov. 2008
3. Ch. Bobda: Introduction to Reconfigurable Computing: Architectures, Algorithms and Applications; Springer, 2007
4. C. Chang et al: The Biggascale Emulation Engine (Bee); summer retreat 2001, UC Berkeley
5. D. Gajski et al.: A second opinion on data-flow machines and languages; IEEE Computer, Feb. 1982.
6. C. G. Bell et al: The Description and Use of Register-Transfer Modules (RTM's); IEEE Trans-C21/5, May 1972
7. K. Paulsson, M. Huebner, J. Becker: Exploitation of dynamic and partial hardware reconfiguration for on-line power/performance optimization; FPL 2008
8. J. Becker, M. Huebner (invited talk): Run-time Reconfigurabilility and other Future Trends; SBCCI 2006, Sept 2006, Ouro Preto, Brazil
9. Reiner Hartenstein: Why we need Reconfigurable Computing Education; Int'l Workshop on Reconfigurable Computing Education, March 1, 2006, Karlsruhe, Germany
10. T. El-Ghazawi et al.: The promise of high-performance reconfigurable computing; IEEE Computer, vol.41, no.2, pp.69-76, Feb. 2008
11. G. Fettweis, E. Zimmermann: ICT Energy Consumption – Trends and Challenges; Proceedings 11th Int'l Symp. on Wireless Personal Multimedia Comm. (WPMC'08), Lapland, Finland, 8-11 Sep 2008

Reiner Hartenstein, IEEE fellow, FPL fellow, professor at TU Kaiserslautern. For CV see http://hartenstein.de/Hartenstein-bio.pdf