Chapter 18

The Paramountcy of Reconfigurable Computing

Reiner Hartenstein

Abstract. Computers are very important for all of us. But brute force disruptive architectural developments in industry and threatening unaffordable operation cost by excessive power consumption are a massive future survival problem for our existing cyber infrastructures, which we must not surrender. The progress of performance in high performance computing (HPC) has stalled because of the „programming wall“ caused by lacking scalability of parallelism. This chapter shows that Reconfigurable Computing is the silver bullet to obtain massively better energy efficiency as well as much better performance, also by the upcoming methodology of HPRC (high performance reconfigurable computing). We need a massive campaign for migration of software over to configware. Also because of the multicore parallelism dilemma, we anyway need to redefine programmer education. The impact is a fascinating challenge to reach new horizons of research in computer science. We need a new generation of talented innovative scientists and engineers to start the beginning second history of computing. This paper introduces a new world model.

18.1 Introduction

In Reconfigurable Computing, e. g. by FPGA (Table 15), practically everything can be implemented which is running on traditional computing platforms. For instance, recently the historical Cray 1 supercomputer has been reproduced cycle-accurate binary-compatible using a single Xilinx Spartan-3E 1600 development board running at 33 MHz (the original Cray ran at 80 MHz) ⁰. Reconfigurable Computing²³ is the paramount issue for continuing the progress of computing performance and for the survival of world-wide computing infrastructures. Section 19.2 of this chapter stresses, that all our computer-based infrastructures worldwide are extremely important⁵, also for avoiding a massive crisis of the global and local economy. Section 19.3.1 warns of the future unaffordability of the electricity consumption of the entirety of all computers worldwide, visible and embedded and, that low power circuit design⁷-¹⁰ and other traditional „green computing“¹¹-¹³, although important and welcome, are by far not sufficient to guarantee affordability and not at all to support further progress for future applications of high performance computing. Thousands of books have been published about world economy, energy, CO2, climate, survival on the globe, water, food, health, etc. Hundreds of them are about peak oil. I have listed just a few of them¹⁹-⁹⁷.

In contrast to the currently still dominant von Neumann (vN) machine, Reconfigurable Computing⁹⁸-¹⁰¹ (RC), the second RAM-based machine paradigm, introduced in section 19.4, offers a drastic reduction of the electric energy budget and speedup factors by up to several orders of magnitude - compared to using the von Neumann paradigm, now beginning to lose its dominance¹⁰²-¹⁰⁷. Sections 19.5.4 and 19.6.2 stress the urgency of moving RC from niche to mainstream and urges, that we need a worldwide mass movement of a larger format than that of the VLSI design revolution around 1980, where only an urgently needed designer
population has been missing\textsuperscript{104-117}. This time a properly qualified programmer population is missing. But we need to push the envelope into two different directions. The VLSI design revolution has been the most effective project in the modern history of computing. But this time we need even more. A dual rail effort (section 19.6) is needed for simultaneously developing the scene toward parallel programming for manycore architectures and to structural programming for reconfigurable computing (RC), as well as heterogeneous systems including the cooperation of both paradigms.

![Figure 1: Lufthansa Reservation anno 1960](image)

Currently the dominance of the basic computing paradigm is gradually wearing off (see section 19.4) with the growth of the area of Reconfigurable Computing (RC) applications - bringing profound changes to the practice of scientific computing, cyber physical systems (CPS) and ubiquitous embedded systems, as well as new promises of disruptive new horizons for affordable very high performance computing. Due to RC also the desk-top personal supercomputer is near\textsuperscript{116}. To obtain the payoff from RC we need a new understanding of computing and supercomputing, as well as of the use of accelerators (section 19.6.3). For bridging the translational gap, the software / configware chasm, we need to think outside the box\textsuperscript{5}.

### 18.2 Why Computers are Important

Computers are very important for all of us. By many millions of people around the world computers are used everywhere. Typical orders of magnitude in the computer application landscape are: hundreds of applications, consisting of ten-thousands of programs, with millions of lines of code, having been developed by expenditure of thousands of man-years investment volumes up to billions of dollars\textsuperscript{5}. We must maintain these important infrastructures. Wiki “Answers pages” nicely tell us, why computers running this legacy software are indispensable in the world\textsuperscript{118}. The Computer is an electronic device used in almost every field even where it is most unexpected\textsuperscript{120}. Here computer crashes or software crashes may cause widely spread disasters by domino effects\textsuperscript{91-97}. In his novel Hermann Maurer depicts the worldwide total chaos caused by a network crash in the year 2080 where millions of people die and the life of billions is threatened\textsuperscript{92-93}. A thrilling novel? Yes, but also a textbook about possible solutions.
BANKS use computers to keep record of all transactions and other calculations. It provides speed, convenience, security. Communication is another important aspect, very easy through internet and E-mail. Computer communicates by telephone lines and wireless. Through E-mail we can send messages to anybody in any part of the world in just a second while if we write letter then it will reach in some days. So the internet has made the earth a global village and above all saves time. This would not be possible without computers. Internet helps to find information on every topic. It’s the easiest and fastest way of research. Computer network makes the user capable of accessing remote programs and databases of same or different organizations. Without computers we also would not have any automated teller machines (ATM).

**Business.** Computers have now become an integral part of corporate life. They can do business transactions very easily and accurately and keep the record of all the profit and loss. Today computers can be found in every store, supermarkets, restaurants, offices etc. special software is used in these computers to calculate the huge bills within seconds. One can buy and sell things online, bills and taxes can be paid online and can also predict the future of business using artificial intelligence software. It also plays a very important role in the stock markets.

**Business Information Systems.** For the economy: business information systems are as essential as materials, energy and traffic. Without business information systems the economy would be extremely ineffective and inefficient. Business information systems are essential for globalization. Their significance for each enterprise: improving the productivity of the business processes (= rationalization), mastering complexity and volume, making information available fast and everywhere: for any operations, for decisions, as well as strategically for entrepreneurial planning on the creation of new business opportunities, i. e. by e-business. If automobile manufacturers would not have PPC systems (product planning & control system), cars could not be manufactured in desired wide variety. It would be like at the early times of Henry Ford, who said: cars can be delivered in any color, provided it is black.

![Figure 2: Some grand challenge examples for CPS](image)

- Blackout-free electricity generation and distribution,
- Extreme-yield agriculture,
- Safe and rapid evacuation in response to natural or man-made disasters,
- Perpetual life assistants for busy, senior/disabled people,
- Location-independent access to world-class medicine,
- Near-zero automotive traffic fatalities, minimal injuries, and significantly reduced traffic congestion and delays,
- Reduce testing and integration time and costs of complex CPS systems (e.g. avionics) by 1 to 2 orders of magnitude,
- Energy-aware buildings and cities,
- Physical critical infrastructure that calls for preventive maintenance,
- Self-correcting cyber-physical systems for “one-off” applications,
- Disaster Response: Large-Scale Emergency Evacuation,
- Assistive Devices.

**Biological And Medical Science.** Diagnostics of diseases and also treatments can be proposed with the help of computer. Many machines use computers which allows the doctor to view the different organs of our body like lungs, heart, kidneys etc. There is special software which helps the doctor during the surgery.

**Education.** Today the computer has become an important part of one's education because we are using computers in every field and without the knowledge of computer we cannot get a job and perform well in it. So computers can secure better jobs prospects. Knowledge about computer is must in this time.

**Media.** Almost every type of editing and audio-visual compositions can be made by using special software especially made for this purpose. Some software can even make three dimensional figures like used in the cartoon films. Special effects for action and science fiction movies are also created on computer.
Travel and Ticketing. Computers do all the work of plane and train reservation. It shows the data for vacant and reserved seats and also saves the record for reservation. Let us imagine, Lufthansa would handle reservations like in 1960 (Figure 1). To-day they could not handle their flight operations by this method.

Weather Predictions possible by experts using supercomputers: another important application.

Sports. It is also used for umpiring decisions. Many times the umpire has to go for the decision of a third umpire where the recording is seen again on computer and finally reaches to the accurate and fair decision. Simulation software allows the sportsman to practice and improve his skills.

Car Safety. Here the ultimate goal is a zero-fatality vehicle. Auto companies use computers for crash simulations to figure out how to build safer cars. GM, Ford, Honda, Mercedes Benz and other companies use this technology. For computer simulations, such as one vehicle crashing into another, carmakers have the supercomputing power in-house. It has been publicly demonstrated that even a complex simulation of a full crash test with 1 million elements can take just five minutes to render using a cluster of Intel Xeon 5500 processors. The latest HPC technology has enabled GM to move to an interactive design process for the entire vehicle, and run a simulation with up to four million elements. American Honda has more than 3000 processors devoted to crash analysis. Mercedes-Benz is now running approximately 5,000 crash simulations for every new vehicle design. The ultimate goal is a zero-fatality vehicle. More sophisticated technology should help make much safer cars a reality in the not too distant future.

Other HPC applications. HPC is pervasive enough so that it is used today not just by government and university researchers but to design products ranging from cars and planes to golf clubs, microwave ovens, animated films, potato chips, diapers and many other products.

Cyber Physical Systems (CPS) are computers or computer networks ready for real-time response, directly coupled to biological organisms or systems, to sensor networks, to organizations, or to technical networks and much more. Figure 2 lists some CPS application examples.

Daily Life. Computers are everywhere. Washing machines, microwave oven and many other products use software. Moreover we can store all the information about our important work, appointments schedules and list of contacts. Computers are playing a very important role in our lives. We cannot imagine the world
without computers. And this technology is advancing in both, industry and home. It is creating new mass markets by a variety of wireless smart portable devices. It is necessary for everyone to have some basic computing knowledge. Otherwise he cannot get a job as computers have invaded almost all the fields.

**Survival risk of mainframe software.** Tracing their roots back to IBM's System/360 from the 60ies mainframes became popular in banking, insurance, and other industries. Quite a number of companies still employ older mainframes with 3270 terminal emulator and Disk Operating System (DOS). Representing cutting-edge technology when the oldest Baby Boomers were still teenagers, mainframe survival is in danger because of recruiting problems.

**Risks of Domino Effects.** The computer-based worldwide interconnectedness of all areas of life is highly risky. For instance public power supply infrastructures are computer-controlled by intelligent networks, so that any local malfunction can trigger cross-border blackouts, causing widely spread breakdowns: employees do not reach their workplace, subcontracted supply does not reach the assembly line, perishable goods do not reach their destination and/or cannot be cooled, etc. Highly vulnerable are also all our communication infrastructures. For instance, blackouts have a lot of follow-up problems: phones and even cell phones do not work. Due to highly complex global interconnectedness a minor bug may cause by chains of reactions a huge disaster. The World Economic Forum (WEF) comes to the conclusion, that by a wide variety of reasons such risks have a very high probability (Figure 3). Our computer-controlled economic and technical infrastructures have reached such an enormous complexity that we hardly can estimate all possible domino effects.

### 18.2.1 Computing for Sustainable Environment

Computing for the Future of the Planet is more and more important for us because computing (computers, communications, applications) will make a major and crucial contribution to ensuring a sustainable future for society and the planet. The “Power Down for the Planet” challenge is a national competition to fight global warming by pledging to reduce the amount of energy used by computers on campus and elsewhere. Also here computers are important: computing for our sustainable environment is important for preserving our civilization, for avoiding its collapse.
The goal of Green Computing is simple: reduction in the use of harmful materials, maximize energy efficiency, and promote recyclability. Green Computing is the science of efficient and effective designing, manufacturing, using, disposing, and recycling of computers and computer related products (servers, networks, peripherals, etc.), also by creating technologies helping reduce harmful impact on the environment and to preserve natural resources. Wasting energy is costly and leading to a climatic change from burning coal and oil.

With the major goal to minimize the carbon footprint of computing, "Green IT" consists of 3 parts: 1) designing products that are less polluting, less energy-consuming and easier to recycle, 2) more efficient data centers, 3) innovative projects that will enable, via IT contributions, the building of a more sustainable world. Green IT supports smart measuring the energy consumption of housing, public buildings and other facilities, in order to be able to optimize the use of energy (smart meters). Green IT also supports education of the data centre operators on enhanced energy optimization (green data centers) - Use of telecommuting and teleconferencing in order to reduce travel requirements. - Installation of web sites offering better information on carpooling or public transport possibilities in order to reduce the traffic on our roads. Optimization of road traffic and transport logistics. Andy Hopper even sees four levels at which innovation-driven developments in computing being effective:

1) Simulation and modeling are important tools which will help predict global warming and its effects. Much more powerful computing systems are required to make the predictions better, more accurate, and relevant.

2) The amount of infrastructure making up the digital world is continuing to grow rapidly and starting to consume significant energy resources.

3) Computing will play a key part in optimizing use of resources in the physical world.

4) We are experiencing a shift to the digital world in our daily lives as witnessed by the wide scale adoption of the world wide web. Let me add a fifth level:

5) To help generate momentum and achieve these goals, it is important that a coordinated set of challenging international projects are investigated.

![Figure 5: PELAMIS wave power: electricity by the sea: Courtesy of Pelamis Wave Power.](image-url)

The World Economic Forum proposed to help existing institutions by IT networking to enable existing institutions to unleash public value, catalyzing initiatives and unleashing human capital in the world. Klaus Schwab: "Our existing global institutions require extensive rewiring to confront contemporary chal-
lenges in an effective, inclusive and sustainable way." Organizations like UN, GATT, G8 and G20 are becoming increasing inept at fixing what ails the world: goals of economic growth, climate protection, poverty eradication, conflict avoidance, human security and promotion of shared values. The topics are the three Rs: Rethink, Redesign and Rebuild[141].

The Forum's "Global Redesign Initiative" report notes[143] „how the digital world has brought about cross-border integration by new technologies enabling virtual interaction have created a world that is much more complex and more bottom-up than top-down." The world has become economically, politically and environmentally more interdependent - without a new set of international bureaucracies piled on the existing ones. It has been argued for a global system with graphic visualization tools to measure success, for a complete redesign the global legal system, for a global vaccine protocol, global intellectual property system, global risk management, etc. This means taking a Wikinomics approach -- embracing more agile structures enabled by global networks for new kinds of collaboration such, that do not need a new set of international bureaucracies piled on the existing ones.

![Figure 6: Beyond Peak Oil: declining future crude oil production][175,176]

Governments need to launch a new paradigm to involve the citizens of the world through mass collaboration by a new medium of communications including tools like digital brainstorms and town hall meetings: decision-making initiatives like citizen juries and deliberative polling; execution tools like policy wikis and social networks within government and evaluation programs. This initiative demonstrates how important to reinvent computing and the growth of IT and the internet is for broad engineering issues insuring sustainability issues of the world like smart energy production and distribution, intelligent water management, strengthening welfare, dealing with ageing and young population - mitigating risks[145].

18.3 Performance Progress stalled

Not only disruptive architectural developments in industry stall further progress of IT with respect to energy-inefficiency and performance improvements. Unaffordable operation cost by excessive power consumption are a massive future survival problem for existing cyber infrastructures, which we must not surrender. Because of the inevitable manycore architecture contemporary computer systems are in an all-dominant programmability crisis. The progress of performance is massively stalled because of this „programming wall“ caused by lacking scalability of parallelism and an ubiquitous programmer productivity gap[146,147,150]. Later in this chapter we show that reconfigurability is the silver bullet to obtain massively bet-
ter energy efficiency as well as much better performance by the upcoming heterogeneous methodology of HPRC (high performance reconfigurable computing). We also believe in the need for a massive campaign for migration of software over to configware. Also because of the multicore parallelism dilemma, we anyway need to reinvent programmer education. The impact is a fascinating challenge to reach new horizons of research in computer science. We need a new generation of talented innovative scientists and engineers to start the beginning second history of computing. This chapter discusses its new world model.

### 18.3.1 Unaffordable Energy Consumption of Computing

The future of our world-wide total computing ecosystem is facing a mind-blowing and growing electricity consumption, together with a trend toward growing cost and shrinking availability of energy sources. The electricity consumption by the internet alone causes more Greenhouse Gas emission than the worldwide air traffic. Will The Internet Break? Consumer broadband connections in North America, Mexico and Western Europe have reached 155 millions by the end of 2007 and are predicted to reach 228 millions 4 years after 2007. The network if the internet is being stressed more than ever with new technologies and larger e-mails, and the trend will accelerate. An explosion in services integrating video and software will intensify by increasing popularity of games, massive use of video on demand, high-definition video and pay-TV to the living room, as well as by newer services by mobile phone companies. and multiple connected PCs and devices using connections. The internet service providers need to be able to assess how much more bandwidth will be required and how much headroom they have.

It has been predicted that by the year 2030, if current trends continue, worldwide electricity consumption by ICT infrastructures will grow by a factor of 30, reaching much more than the current total electricity consumption of the entire world for everything, not just for computing. The trends are illustrated by an expanding wireless internet, and by a growing number of internet users. as well as with tendencies toward more video on demand, HDTV over the internet, shipping electronic books, efforts toward more cloud computing and many other services. Other estimations claim, that already now the greenhouse gas emission from power plants generating the electricity needed to run the internet is higher than that of the total world-wide air traffic. There are more predictions.

![Figure 7: The end of cheap oil.](image)
of 16 millions (an earlier estimation for one system is 120 MW\textsuperscript{158}). The electricity bill is a key issue not only for Google, Microsoft, Yahoo and Amazon with their huge data farms at Columbia River\textsuperscript{129} (Figure 4). That’s why Google recently submitted an application asking the Federal Energy Regulatory Commission for the authority to sell electricity\textsuperscript{167}, and has a patent for water-based data centers, using the ocean to provide cooling and power (using the motion of ocean surface waves to create electricity: Figure 5)\textsuperscript{169}. Already in the near future the electricity bill of most facilities will here be substantially higher than the value of their equipment\textsuperscript{170}. Already in 2005, Google’s electricity bill was with about 50 million US-Shigher than the value of its equipment. Meanwhile the cost of a data center is determined solely by the monthly power bill, not by the cost of hardware or maintenance\textsuperscript{171}. Google’s employee L. A. Barroso said\textsuperscript{172}: „The possibility of computer equipment power consumption spiraling out of control could have serious consequences for the overall affordability of computing.“

Rapidly growing energy prices are predicted since the oil production has reached its peak by about the year 2005\textsuperscript{174}. Already currently 80% of crude oil is coming from decline fields (Figure 6). However, the demand is growing because of developing standards of living in China, India, Brazil, Mexico and newly industrializing countries. The world Energy Council estimates, that the demand will double until the year 2050\textsuperscript{177}. We need at least “six more Saudi Arabias for the demand predicted already for 2030” (Fatih Birol, Chief Economist IEA\textsuperscript{179}). I believe, that these predictions do not yet consider the rapidly growing electricity consumption of computers. Maybe, we will need 10 more Saudi Arabias. About 50% of the shrinking oil reserves are under water\textsuperscript{180}. In consequence of the Gulf of Mexico oil spill not all deepwater explorations will be allowed, insurance rates rise, and the crude oil prices will go further up (Figure 7). Transitions from carbon fuels to renewables cannot completely fill the gap within at least two decades. This will cause a massive future survival problem for running our cyber infrastructures, which we must not surrender because this is an important global economy issue. Or, should we dig more coal\textsuperscript{181}? It makes sense, to measure computing performance not just by MIPS (million instructions per second), but by MIPS/watt or FLOPS/watt instead\textsuperscript{182}.

### 18.3.2 Crashing into the Programming Wall

For 40 years, semiconductor technology has followed Moore’s Law. Until about 2004 we obtained better performance by just waiting for the next generation microprocessor with higher clock speed. Because of this free ride on Gordon Moore’s law the improvement of software performance has been the successful job of hardware designers. This development ended, when the microprocessor industry changed strategy from a single CPU on a chip to manycore by increasing the number of on-chip processor cores instead of growing clock frequency. The “golden” CMOS era is gone\textsuperscript{147}. Technology scaling does not deliver anymore significant performance speedup and increasing power density poses severe limitations. High performance is no more the job of hardware designers. We hit the „programming wall“ since high performance now requires rare parallel programming skills\textsuperscript{147,183}. Great challenges for RC provide the answer\textsuperscript{147,150}.

This „programming wall“\textsuperscript{184} we know from supercomputing is not new. The dead supercomputer society list\textsuperscript{187} demonstrates, that almost all much earlier supercomputing projects and start-ups failed, since parallel programming has been required, which crashed into the parallel programming wall. This list is not even complete. More dead projects are listed elsewhere\textsuperscript{188}. Even to-day the vast majority of HPC or supercomputing applications was originally written for a single processor with direct access to main memory. But the first peta-scale supercomputers employ more than 100,000 processor cores each, and distributed memory. Three real-world applications have broken the petaflop barrier (10\textsuperscript{15} calculations/second) (Jaguar at ORNL)\textsuperscript{185}. A slightly larger number have surpassed 100 teraflops (100 x 10\textsuperscript{12} calculations/second), mostly on IBM and Cray\textsuperscript{185}. The scene hopes, that dozens of applications are inherently parallel enough to be laboriously decomposed, sliced and diced, for mapping onto such highly parallel computers. But a large number of applications is only modestly scalable. More than 50% of the codes do not scale beyond eight cores, only about 6% can exploit more than 128 PE, still a tiny fraction of 100,000 or more available cores\textsuperscript{185}.\textsuperscript{9}
A very important issue is saving energy\textsuperscript{16,189}. But multicore processors tend to have tuned-down speeds. Down from 3 to 4 GHz (single-core) each core meanwhile runs at about half that speed. Some HPC or supercomputing sites report that some of their applications were running more slowly on their newest HPC system\textsuperscript{185}. With almost 70% of this market x86-based intel or AMD processors are dominant (in 2009). But, multiplied collective peak performance comes without corresponding increases in NoC bandwidth, making it difficult to move data into and out of each core fast enough to keep the cores busy\textsuperscript{185,190}. We have to rethink not only Amdahl’s law. Adding accelerators via a slow PCI bus adds to the problem. Both hardware and software advances are urgently needed.

We see that massive hardware parallelism from skyrocketing core counts is racing ahead of programming paradigms and programmer productivity. This parallel performance “wall” will reshape the nature of HPC code design and system usage\textsuperscript{147}. The evolutionary path is not addressing the fundamental problems. A large number of HPC applications will need revolutionary changes to be fundamentally rethinked and rewritten within the next five to 10 years by serious algorithm development. We’ve seen examples of mathematical models and algorithms that broke when pushed beyond. There aren’t enough people with the right kind of brainpower\textsuperscript{14,15}. Universities should produce more.

Semiconductor technology has followed Moore’s Law throughout 4 decades. But continuing that pattern will require not only a breakthrough in energy-efficient design. With a very high probability we will be forced to seek an entirely new paradigm\textsuperscript{195}. This crisis and its key issues such as software scalability, memory, IO, and storage bandwidth, and system resiliency stems from the fact that processing power is outpacing the capabilities of all the surrounding technologies.

In parallel computing the realized real application performance is bad and getting worse\textsuperscript{196} Parallel computer programs are difficult to write: performance is affected by data dependencies, race conditions, synchronization, and parallel slowdown. The “parallel programming problem” has been addressed high performance computing for more than 25 years with disappointing results. The more formal way based on languages did not really help\textsuperscript{194}, neither more than hundreds of parallel programming languages (subset in Table 1), nor several hundred hardware description languages\textsuperscript{199-202} which are also capable to express parallelism (subset in Table 2). There are too many new prophets declared another new route out of the wilderness of software development and maintenance since at least the invention of COBOL (Common Business Oriented Language) in 1959. Only a much too small number of specialized developers are kind of half ways skilled to write parallel code. Given this shortage of parallel programmers, however, we need to accept that informal approaches are not working.
It has been proposed, that to succeed with parallel programming in the multi-core era, we must adopt a systematic approach obtained by insight into how programmers think. I do not agree. The thinking of programmers is far from being uniform. One expert predicts that shared memory computing will ultimately run out of steam. Another expert claims, that distributed memory is good because it is efficient, scalable and future-proof for increasingly distributed and non-uniform future hardware. A third one says that shared memory is good because it is compatible with existing sequential code and doesn't require re-training of developers. A forth one means that if you extrapolate current trends, a decade from now we'll be die-stacking cores and memory in NUMA (non-cache-coherent) architectures with thousands of cores. A fifth one says that it is obvious that stream processing applications (e.g. DSP and graphics) generally don't need globally shared memory. I believe, that we have to teach to programmers how to think. We have to teach them how to “think parallel”, how to find concurrent tasks in a program, how to synchronize for programming free from deadlock and race conditions, how to schedule tasks at the right granularity onto the processors of a parallel machine, and, how to solve the data locality problem for correctly associating data with tasks. Too many sequential-only textual languages are around (Table 3) not helpful to support locality awareness urgently needed to understand parallelism and its bottlenecks.

Thousands of programming languages existing (only a subset is listed in Table 3) demonstrate here the tradition that there is a very wide variety of opinions and proposals around, pointing toward too many different directions to go. (There are even esoteric languages. It is like seeking the needle in the haystack. It’s also part of the von Neumann syndrome. For instance (no direction at all): with multicore...)

### Table 1: Some Languages designed for Parallelism

<table>
<thead>
<tr>
<th>Language</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A+</td>
<td>Cthreads</td>
</tr>
<tr>
<td>ABCPL</td>
<td>CUMULVS</td>
</tr>
<tr>
<td>ACE</td>
<td>C++</td>
</tr>
<tr>
<td>ACT++</td>
<td>Cashmere</td>
</tr>
<tr>
<td>Active</td>
<td>D Agger</td>
</tr>
<tr>
<td>ActorSel</td>
<td>Chapel</td>
</tr>
<tr>
<td>Ada</td>
<td>DC++</td>
</tr>
<tr>
<td>ADDAP</td>
<td>Charm</td>
</tr>
<tr>
<td>Adl</td>
<td>Charm++</td>
</tr>
<tr>
<td>Adsmith</td>
<td>Chu</td>
</tr>
<tr>
<td>AFAPI</td>
<td>ChuckX</td>
</tr>
<tr>
<td>Aftrix</td>
<td>Cin</td>
</tr>
<tr>
<td>Alf</td>
<td>Cilk</td>
</tr>
<tr>
<td>Alice</td>
<td>Clojure</td>
</tr>
<tr>
<td>ALWAN</td>
<td>CM-Fortran</td>
</tr>
<tr>
<td>AM</td>
<td>Co-array</td>
</tr>
<tr>
<td>AMDC</td>
<td>Fortran</td>
</tr>
<tr>
<td>Amoeba</td>
<td>Code</td>
</tr>
<tr>
<td>Analystc</td>
<td>Concurrent</td>
</tr>
<tr>
<td>APL</td>
<td>Lua</td>
</tr>
<tr>
<td>AppLeS</td>
<td>Concurrent</td>
</tr>
<tr>
<td>ARIS</td>
<td>Pascal</td>
</tr>
<tr>
<td>Ateji</td>
<td>PX</td>
</tr>
<tr>
<td>Aurora</td>
<td>Cool</td>
</tr>
<tr>
<td>Automap</td>
<td>Corn</td>
</tr>
<tr>
<td>Axum</td>
<td>CORREL</td>
</tr>
<tr>
<td>Blaze</td>
<td>CparPar</td>
</tr>
<tr>
<td>BlockCrm</td>
<td>CPS</td>
</tr>
<tr>
<td>BSP</td>
<td>CRL</td>
</tr>
<tr>
<td>C4</td>
<td>CSP</td>
</tr>
</tbody>
</table>

...the list continues with many other languages...
systems becoming ubiquitous, there is some naive hope that “if you build it, they will come”\textsuperscript{207} (good ideas or skilled programmers). Parallel programming has developed along informal, empirical lines. For instance, Parallel Global Address Space (PGAS) programming languages have existed for over a decade and could be far more productive than the message passing interface (MPI) still dominating HPC programming today. But only a few HPC users are ready to learn a new language that would also require rewriting HPC applications that could contain tens or hundreds of thousands of lines of code. Are we as humans multicore-capable at all?\textsuperscript{208} There’s a school of thought telling us that humans are simply not built to comprehend multicore programming since our thought processes are inherently serial\textsuperscript{208}. But this is not true. We are able to succeed in goal-oriented activities w. r. t. maps, schematics, networks, graphs, all kinds games and sporting. Perhaps a whole new programming paradigm is required that uses symbols (for example), flowcharts or other schematics\textsuperscript{209}.

Table 2: Some Hardware Description Languages. \textsuperscript{199-202}

<table>
<thead>
<tr>
<th>Hardware Description Language</th>
<th>Model</th>
<th>Model</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABL</td>
<td>CoreFire</td>
<td>impulse</td>
<td>Ruby</td>
</tr>
<tr>
<td>ABE L</td>
<td>CoWareC</td>
<td>Impulse-C</td>
<td>SA-C</td>
</tr>
<tr>
<td>AccelChip</td>
<td>UPCL</td>
<td>JHDL</td>
<td>Scheme</td>
</tr>
<tr>
<td>AHDL</td>
<td>C2H</td>
<td>KARL</td>
<td>SML</td>
</tr>
<tr>
<td>AHP L</td>
<td>C2Verilog</td>
<td>Lava Haskell</td>
<td>SPARK</td>
</tr>
<tr>
<td>BDL</td>
<td>Dime-C</td>
<td>Lola</td>
<td>SpecC</td>
</tr>
<tr>
<td>Bluespec</td>
<td>DDS</td>
<td>MDL</td>
<td>Streams-C</td>
</tr>
<tr>
<td>BluespecLs</td>
<td>ELLA</td>
<td>Meta-HDL</td>
<td>SysGen</td>
</tr>
<tr>
<td>Brass</td>
<td>Erlang</td>
<td>Minnow-C</td>
<td>System-C</td>
</tr>
<tr>
<td>Brach C</td>
<td>F#</td>
<td>My-HDL</td>
<td>System-Studio</td>
</tr>
<tr>
<td>BDL</td>
<td>Forge</td>
<td>NAPA-C</td>
<td>System-TCS</td>
</tr>
<tr>
<td>CDL</td>
<td>HDGami</td>
<td>Ocapi</td>
<td>Transmogifier-C</td>
</tr>
<tr>
<td>CASH</td>
<td>HandelC</td>
<td>OpenVera</td>
<td>LystemVerilog</td>
</tr>
<tr>
<td>Catapult-C</td>
<td>HardwareC</td>
<td>PALASM</td>
<td>Tridget-Compiler</td>
</tr>
<tr>
<td>Carte-C</td>
<td>Hardware Join Java</td>
<td>PAL</td>
<td>Verilog</td>
</tr>
<tr>
<td>ChiIMPS</td>
<td>Hardw Verif Lan</td>
<td>Property Spec Lan</td>
<td>Verilog-AMS</td>
</tr>
<tr>
<td>Coecentic</td>
<td>Haskell</td>
<td>Ptolemy II</td>
<td>VHDL</td>
</tr>
<tr>
<td>Cones</td>
<td>HML</td>
<td>RC Toolbox</td>
<td>VHDL-AMS</td>
</tr>
<tr>
<td>Confluence</td>
<td>HVL</td>
<td>RHDL</td>
<td>Viva</td>
</tr>
<tr>
<td>ConvergentSC</td>
<td>Hydra</td>
<td>RTcode</td>
<td></td>
</tr>
</tbody>
</table>

What are the right models (or abstractions) to program for performance portability to all important parallel platforms? We should focus less on fashionable topics such as TM (Transactional memory), and multithreading which is considered harmful\textsuperscript{210} To be ready for discussions, also controversial discussions, we also face several issues, such as e. g. migrating code from a uniprocessor to SMP models (symmetric multiprocessors). We should be aware of several typical problems\textsuperscript{212}: why going to multi-core could make applications run slower, what are the sources of race conditions, what strategies to use for migrating uniprocessor code to a multi-core environment.

It’s a disaster response, that DARPA selected Sandia Labs to launch the Ubiquitous High Performance Computing (UHPC) program\textsuperscript{195} to design new supercomputer prototype (completed by 2018?) to overcome current limiting factors, such as power consumption and architectural and programming complexity, by developing for scalability entirely new computer architectures and programming models. The aim is to revolutionize the entire field of computing for fundamentally enabling a new model of computation by producing a highly dependable more energy-efficient computer that delivers 100 to 1,000 times more performance and is easier to program than current systems. There are more organizations, e. g. The HPC Advisory Council (HPCAC) creates local Centers of Excellence worldwide for education enhancing the HPC knowledge-base and exploring future solutions\textsuperscript{213}. 
We cannot afford to relinquish RC. It may provide a method to effectively circumvent the programming wall. We will also urgently need this technology to cope with threatening unaffordable operation cost by excessive power consumption of the entirety of all von Neumann computers world-wide. We need to migrate many application packages from software over to configware. A sufficiently large programmer population qualified for reconfigurable platforms is not existing.

Table 4: Computer System Model of the Mainframe Era.

This is a challenge to reinvent computing for providing the qualifications, needed not only for RC, but also to cope with the many-core programming crisis. Intel's cancellation of the Tejas and Jayhawk processors indicated in May 2004 the end of frequency scaling’s dominance to improve performance. „Multicore computers shift the burden of performance from hardware designers to software developers.”

requires the migration of many software packages from monoprocessors to manycore platforms.

A sufficiently large programmer population qualified for parallel programming is not existing. This causes a huge challenge to provide new educational approaches to create a dual-paradigm-savvy programmer population qualified for heterogeneous systems including both, parallel software and configware. To deal efficiently with FPGAs we also need robust and fast implementations of adequate configware compilers, e. g. automated by formal techniques based on rewriting\textsuperscript{225,226}. But new compilers alone are not sufficient. To programmers we also have to teach time to space mapping. Another source said: „Intrinsic dimensionality rooted scaling laws to favor reconfigurable spatial computing over temporal computing.“ [N. N.: Stone Ridge Technologies]. It is highly important, that the spatial character of RC introduces locality awareness in to the mind of students learning programming.

Time to space mapping dates back to the early 70ies and even the late 60ies, years before the first hardware description languages came up\textsuperscript{202,227}. „The decision box (in the flow chart) turns into a demultiplexer. This is so simple! Why did it take 30 years to find out?“\textsuperscript{229}. Due to notorious hardware / software gap CS education ignored for another 40 years, how simple this is. The impact is an encouraging challenge to reach new horizons of new computer science research. We need new generations of talented innovative scientists and engineers to start the second history of computing. But this is less difficult than it looks like at first glance. „The biggest payoff will come from Putting Old Ideas into Practice (POIP) and teaching people how to apply them properly.“\textsuperscript{230}.

18.4 The Tail is wagging the Dog (Accelerators)

Invented by Ted Hoff in 1969 the introduction of the first microprocessor has been a revolution. The EE types in digital design have been well qualified by using integrated circuits with gates and flip-flops inside to construct controllers. Just having been hired as employee no. 12 at intel Ted Hoff’s idea to replace the wide variety of circuits by a GP (general purpose) circuit using principles known from mainframes has not been welcome. This crew completely with EE background thought, this strange guy coming from computer science at Stanford is completely crazy. But Federico Faggin joining intel about a year later supported
Ted’s idea. In 1971 marketing the 4004 microprocessor has been started. But also to the customers of intel the idea of a programmable controller appeared to be strange. Replacing the soldering iron by a keyboard: isn’t this really crazy? To be able to sell this first microprocessor intel has been forced to give courses to about 100,000 people. But finally intel succeeded as we all know.

Table 5: The Post-Mainframe Machine.

<table>
<thead>
<tr>
<th>CPU / accelerator symbiosis</th>
<th>resources</th>
<th>sequencer</th>
</tr>
</thead>
<tbody>
<tr>
<td>property</td>
<td>programming source</td>
<td>property</td>
</tr>
<tr>
<td>1</td>
<td>ASICS hardwired accelerators</td>
<td>hardwired</td>
</tr>
<tr>
<td>2</td>
<td>instruction set processor</td>
<td>hardwired</td>
</tr>
</tbody>
</table>

Using machine principles from the 40ies and after a dozen technology generations the microprocessor meanwhile has become a Methusela, although some people still call it GP (general purpose: Table 4). However, meanwhile it cannot move forward without using crutches. Throughput requirements have grown faster than clock speed, so that the computer having this processor inside is even unable to drive its own display. This already happened in the 90ies or earlier. Meanwhile a growing variety of hardwired accelerators comes along with each PC, laptop, or other kinds computers, called ASICs (Application-specific Integrated Circuits). The tail is wagging the dog (Table 5). A variety of methods is available to provide massive speed-up: minimizing communication efforts, eliminating the need to store and communicate intermediate results, e. g. by merging many simple operations into mega functions, and many more techniques. Not only speed-ups are obtained by ASIC accelerators. Also massive energy saving can be the result. A more recent example is an ASIC designed for the modern HD H.264 video encoding standard. Here the 2.8 GHz Pentium 4 is 500x worse in energy\(^{231,232}\). Also a four processor Tensilica-based CMP is 500x worse in energy\(^{231,232}\).

18.4.1 Hardwired Accelerators

According to the state of the art in the 90ies an accelerator typically was a non-von-Neumann accelerator\(^ {233}\) (Table 5). But a good ROI (Table 15) with such ASICs used as accelerators was and is possible with a very high production volume. Along with the continuing progress by Gordon Moore’s law to shrinking feature sizes the general expenses like design cost, mask making and preparing a production charge, have been exploding. Meanwhile the cost of a fab line has exceeded several billion US dollars (Figure 8). Year by year the ratio between ASIC design starts and FPGA design starts went backwards\(^ {234}\) (Figure 9).

For this reason we now have to distinguish two kinds of such accelerators (line 1 and 3 in Table 6): ASICs made from hardwired logic or configured onto FPGAs (field-programmable gate arrays). These two kinds are distinguished by binding time of their functionality: (1) before fabrication for fixed logic or hardwired logic devices (HWD) vs. (2) after fabrication for (field-) programmable logic devices (PLD). The term „field-programmable“ indicates, that by reconfiguration the functionality can be changed also at the users site by receiving new configuration code: from some memory, or, even over the internet.

18.4.2 Programmable Accelerators

Meanwhile ASICs used as accelerators have massively lost market shares in favor of reconfigurable accelerators. Now FPGAs projects outnumber ASIC projects by >30 to 1 (Figure 9), or even by 50 to 1 due to another estimation\(^ {236}\). FPGAs\(^ {237}\) are structurally programmed from „configware“ sources, which is funda-
mentally different from the instruction-stream-based „software“ sources (Table 6). FPGAs come with a different operating system world organizing data streams and swapping configware for partially and dynamically reconfigurable platforms\(^{238,239}\). FPGAs are supposed to be in 70% of all Embedded systems\(^{228}\). Introduced in 1984, fastest-growing segment of the semiconductor industry for about two decades, now a 4 billion US-S world market (almost 30% increase from last year - Peter Clarke from iSuppli estimates 43%) FPGAs are a well proven technology rapidly heading for mainstream and also used in supercomputing by Cray, Silicon Graphics, and others\(^{98,241-243}\). Estimations of the number of designers actively involved in FPGA design range from about 100,000 to over 500,000 engineers worldwide.

The partition between hardware and software can be moved throughout the design cycle. Low-level hardware complexity is reduced or removed from design decisions. Many decisions can be removed from the early design process allowing a designer to focus on creating a product's unique functionality to serve a variety of marketing strategies\(^{247}\). Complex processor-based embedded systems can be created and changed easily. Here a key issue is the high-level product development approach implementing as much functionality as possible outside the hardware domain. Here software and configware can be easily updated during and after the design process\(^{247}\): a powerful flexible way to define product functionality, allowing a product's competitive IP and the crucial user experience to be defined almost entirely in the re-programmable domain\(^{249}\). Defining soft elements of the design can be updated at any time, even after the product has been deployed in the field. Systems on a Chip (SoC) can draw various benefits such as adaptability and efficient acceleration of compute-intensive tasks from the inclusion of dynamically reconfigurable platforms as a system component. Dynamic reconfiguration capabilities of current reconfigurable devices create an additional dimension in the temporal domain. During the design space exploration phase, overheads associated with reconfiguration and hardware/software interfacing need to be evaluated carefully in order to harvest the full potential of dynamic reconfiguration.

### Table 6: Contemporary (heterogeneous) Computer System Machine Model

<table>
<thead>
<tr>
<th>contemporary computer system machine model</th>
<th>resources</th>
<th>sequencer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 hardwired accelerators</td>
<td>wired (complex)</td>
<td>(hardwired)</td>
</tr>
<tr>
<td>2 instruction set processors</td>
<td>wired (ALU)</td>
<td>-</td>
</tr>
<tr>
<td>3 reconfigurable accelerators</td>
<td>programmable (complex)</td>
<td>Configware (configuration code)</td>
</tr>
<tr>
<td>4 flowware-programmable accelerators like BEE(^{218})</td>
<td>wired (complex)</td>
<td>-</td>
</tr>
</tbody>
</table>

Using schematic or flow chart graphical design interfaces, IP blocks, software routines and I/O systems can be quickly combined to explore and develop innovative product functionality without the need for low-level engineering\(^{247}\). The pool of design data, also library parts, holds a single model of each block that incorporates all its elements. This model and IP cores can be simply dropped into the design, using a high-level graphic-based capture system, regardless of their level of design abstraction. We can do much more than just programming the on-board FPGA\(^{247}\). By intelligent communication between hardware platform and the high-level design software, the system could directly interact with all parts of the development board. Peripherals can then be swapped on the fly, by automatically reconfiguring interface layers and configuration files. So the complete development system, including the physical hardware, acts as the one design environment. Hardware could conceivably become the final product in some circumstances.

The FPGA industry sprouted\(^{251}\) (Table 12) from programmable read-only memory (PROM, invented 1956),
field programmable read-only memory (FPROM), or one-time programmable non-volatile memory (OTP NVM), and programmable logic devices (PLDs), where the setting of each bit is locked by a fuse or antifuse. (For acronyms see Table 15.)

Table 15 Such PROMs are used to store its program patterns permanently, programmable after fabrication, in contrast to ROM. Programmable more often than just once\textsuperscript{251}, first floating-gate UV erasable PLD came 1971 from General Electric. In 1973 National Semiconductor and in 1975 Signetics had introduced the mask-programmable PLA, and so-called FPLAs came up in the early 80ies\textsuperscript{251}, also featuring very area-efficient layout similar as known from ePROM memory. Instead of just bits, coded forms of canonical Boolean expression patterns could be stored by PLAs or FPLAs.

<table>
<thead>
<tr>
<th>#</th>
<th>$g_0$</th>
<th>$g_01$</th>
<th>$g_{10}$</th>
<th>$g_{11}$</th>
<th>$f(A, B)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$A \text{ and } B$</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$B \text{ disables } A$</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>$A$</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$A \text{ disables } B$</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>$B$</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>$A \text{ xor } B$</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$A \text{ or } B$</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$\text{not}(A \text{ or } B)$</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$A \text{ and } B$</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$\text{no}(B)$</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>$B \text{ implies } A$</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$\text{not}(A)$</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>$A \text{ implies } B$</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>$\text{not}(A \text{ and } B)$</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Very high speed-up could be obtained by matching hundreds of Boolean expressions within a single clock cycle instead of computing them sequentially. Together with a reconfigurable address generator\textsuperscript{252} this brought a speed-up by up to 15,000\textsuperscript{254-257} for a grid-based design rule checker - already in the early 80ies. Via the multi project chip organization of the E.I.S. project such a FPLA (which was called DPLA) has been manufactured on a multi-project chip, with the capacity of 256 first FPGAs (field-programmable gate array) just appearing on the market (by Xilinx in 1984). In early to mid 80ies the multi university E.I.S. project has been the German contribution to the Mead-&-Conway VLSI design revolution\textsuperscript{104-115}.

Not only the particular acronyms FPLA and FPGA are confusing (Table 9, Figure 21). What is here the difference between „logic“ and „gate“? „Logic“ elements are very small with fixed interconnect: FPLAs
have very dense layout like memory. A „Gate“ (CLB) needs more space. But FPGAs are more flexible by
CLBs (configurable logic blocks) and routable wiring fabrics to interconnect CLBs (Figure 11). In contrast
to FPLAs, CLBs in FPGAs allow for instance to select one of 16 logic functions (Figure 10) from simple
LUTs (look-up tables, Figure 10).

Figure 12: Interconnect fabrics example of a routable GA; grey line:
every example of one routed wire connecting 2 CLBs with each other.

Beyond such fine grained reconfigurability the progress of Moore’s law leads to higher abstraction lev-
els with „coarse-grained reconfigurability“ featuring also CFBs (configurable function blocks), which may
be adders, multipliers and/or many other functions 256-261. The next step are mixed grained „platform
FPGAs“, which also include one or several microprocessors, like the PowerPC in earlier platform FPGAs
from Xilinx (also see section 19.5.4.3). However, FPGA technology is worse than microprocessors: slower
clock speed, and massive reconfigurability overhead. Orders of magnitude higher performance with a
worse technology: The Reconfigurable Computing Paradox (section 19.5.3). Software engineering’s plat-
forms are so massively inefficient, so that here Prof. C. V. Ramamoorthy from UC Berkeley has coined the
term „The von Neumann Syndrome“.

18.5 Reconfigurable Computing

Reconfigurable Computing Prospects on the Rise245. Meanwhile FPGAs are standard gear in high per-
formance streaming appliances like multimedia, medical imaging, routers, market data feeds, military sys-
tems, and others. The attractiveness of FPGAs is that they can be rapidly custom configured to run specific
application workloads efficiently. If a different workload needs to be run, the FPGA can be reconfigured
accordingly. Switching the configurations takes just milliseconds. Accelerating real-life applications using
FPGAs has really shown unprecedented levels of speed and savings in cost and energy for many applica-
tions264. Meanwhile, hardware acceleration in general has become the hot paradigm in computing also with
the top supercomputer systems in the world reaching the PetaFLOPS mark using hardware accelerators.
Many high-performance reconfigurable computing systems have been produced by top supercomputer
Again our common computer system model has changed\(^\text{100}\) (Table 6: rows 1-3): accelerators have become programmable\(^{267}\). In contrast to an instruction set processor (CPU) which is programmed only by software, a reconfigurable computing platform needs two program sources (Table 6: row 3): „configware“ and „flowware“, both not at all instruction-stream-based (configware: not procedural nor imperative). Flowware has been derived from the data stream defined for systolic arrays\(^{221,222}\) already in the late 70ies. The term „flowware“ makes sense to avoid confusion with a variety of data stream definitions. Flowware may also be used without configware for hardwired machines (Table 6: row 3), e. g. like BEE\(^{101}\). The term „Flowware“ avoids confusion with the term „dataflow“\(^{244}\).

![Figure 13: History of Reconfigurable Computing](image)

We now have to interface to each other two different programming paradigms: for programming the instruction set processors by traditional „software“, and for programming the accelerators by configware, still ignored by software engineering (SE). It is the dilemma of obsolete programmer qualifications. FPGAs require an unconventional programming model, to configure the chip’s logic elements and data paths before run time. Some tools at least halfways protect the developer from hardware design issues or even explicit parallel programming, making the sorting out of CPU and FPGA code mappings the responsibility of the compiler and runtime system. We should characterize this niche and the progress of High-Performance Reconfigurable Computing, as well as the associated challenges, and, we should characterize the systemic productivity problem\(^{146,242}\). We have to devise an orchestrated multilevel research agenda that is needed to move forward and identify the potential practical next steps for the community\(^{264}\).

A growing trend is the use of FPGAs in embedded systems: ERC (Embedded Reconfigurable Computing). FPGAs are supposed to be used in 70% of all Embedded systems\(^{240}\). Originally there has been a feeling that FPGAs are too slow, power-hungry and expensive for many embedded applications. This has changed. With low power and a wide range of small packages, particular FPGAs can be found in the latest handheld portable devices, including smartphones, eBooks, cameras, medical devices, industrial scanners, military radios, etc. Xilinx’s first attempt at this was an FPGA with a processor inside. This time around, it’s a processor with an FPGA grafted on. That’s not just semantic hair-splitting: it’s the big difference be-
tween these chips and the old ones. The new chips will boot up and run just like normal micro-processors, meaning there’s no FPGA configuration required at all\(^{200}\). FPGAs leap ahead. But embedded designers just don’t like FPGAs with CPUs inside\(^{205}\). FPGAs in this context have been very much seen as a hardware engineer’s domain, with the softies allowed in to play at some late stage\(^{206}\). Xilinx pre-announced a new family of devices „going beyond the FPGA“ which they called „EPP“.

This EPP („Extensible Programming Platform“) has a hardwired area with a top-end twin-core ARM Cortex-A9M processor unit and with a NEON multimedia processor, memory interfacing, peripherals, and a programmable fabric\(^{206}\). Instead of communicating across an FPGA, the two processors are connected by 2,500 wires, providing much capacity for an AMBA-AXI bus and other communications protocols. Xilinx was stressing that this approach recognizes the increasingly dominant role of software in systems and is pushing EPPs as a way to first define the system in software and then carry out software and hardware design in parallel.

![Figure 14: Better power efficiency by accelerators\(^{357}\).](image)

The totally changed concept of EPP makes these devices more like heterogeneous SoC, allowing to have significant benefits for high-performance applications like wireless communication, automotive driver assistance, intelligent video surveillance, etc. An EPP is a SOC with an embedded FPGA, boots like a processor and acts as a processor\(^{194}\). Instead of a classical FPGA an EPP is really a full fledged processor. EPPs make the processor the centre of the device with the programmable fabric as an extra. And this, argues Xilinx, now puts the software engineer first with the hardies following behind. In EPPs the FPGA logic and the CPU will be programmable separately. FPGA configuration will be handled by the processor(s) directly, not by a serial ROM. In other words, you have to tell the FPGA you want it configured. That’s very un-FPGA-like\(^{205}\). That’s EPP-like. The approach of using both a processor and programmable fabric allows design to start at high level and the system to be implemented as software\(^{206}\). EPPs are a result of the new research topic Network-on-Chip (NoC)\(^{332}\), which is a new paradigm for designing the on-chip hardware communication architecture based on a communication fabric, also including on-chip routers. NoC CAD tool flows also support mapping applications into NoC.

Apart from ERC (Embedded Reconfigurable Computing) we have another reconfigurable computing scene: HPRC (High Performance Reconfigurable Computing\(^{241,243}\)). This is the combination of supercomputing and the use of reconfigurable platforms. Well-known is the CHREC project\(^{272}\) in the US, heavily funded by the NSF. HPRC is a relatively new area, but has attracted a lot of interest in recent years, so much so that this entire new phrase has been coined to describe it\(^{273}\). HPRC uses FPGAs as accelerators for supercomputers\(^{272}\). HPC vendors like Cray, SILICON Graphics and others are already supplying machines with FPGAs ready-fitted, or have FPGAs in their product road maps. An example is the Cray XD1 incorpo-
rating Xilinx Virtex-II Pro FPGAs for application acceleration. What are the benefits of using FPGAs in HPC? Also here the first and most obvious answer is performance. HPC is renowned as that area of computing where current machine performance is never enough. This is scene leading us toward the dominance of heterogeneous systems. A problem yet to be solved here is programmer productivity. Programmers cannot work with hardware description languages like FPGA experts. We have to bridge this gap. It is an educational challenge, that programmers with this needed mix of skills are missing.

![Speed-up factors from software to configware migration.](image)

There is yet another embedded system scene, called Cyber Physical Systems (CPS), where real-time behavior is a key issue. The major obstacle to use multicores for real-time applications is that we may not predict and provide any guarantee on real-time properties of embedded software on such platforms. Also the way of handling the on-chip shared resources such as L2 cache may have a significant impact on the timing predictability. An interesting project proposes to use cache space isolation techniques to avoid cache contention for hard real-time tasks running on multicores with shared caches. Dynamic reconfiguration capabilities of current reconfigurable devices can create an additional dimension in the temporal domain. During the design space exploration phase, overheads associated with reconfiguration and hardware/software interfacing need to be evaluated carefully in order to harvest the full potential of dynamic reconfiguration.

But there is also an area of concern. Both FPGA giants (Xilinx and Altera) are hitting 28nm at end of 2010. With reduced feature size of integrated circuits transistors become less reliable. As feature size continues to shrink transistors become less reliable. Transistors will be defective at manufacture time, also by process variations, and more of them will degrade and fail over the expected lifetime of a chip. Also an increasing number of soft errors will occur. The failure rate is growing. Causing major degradation, such failures are based on the physical mechanisms like electromigration (EM), hot carrier degradation (HCD) and time dependent dielectric breakdown (TDDB). Usually manufacturers keep their failure rate statistics secret. However, with feature sizes of 20 nm and below the failure rate is a major problem (dark silicon).
so that fault tolerance methods should be applied\textsuperscript{278}. Such fault tolerance techniques can be implemented on FPGAs by re-routing methods\textsuperscript{280}. However, the fault detection required here is a non-trivial problem. The learning capabilities of artificial neural networks (ANN) would be a welcome capability to organize such fault detection methods. By the way, ANNs could be seen as a future innovation for reconfigurable computing by enabling self-healing reconfigurable platforms based on self-learning\textsuperscript{283-290}. But a very important component is still missing, requested by ANN pioneer Karl Steinbuch\textsuperscript{283-286} already in 1960, which is capable of non-destructively storing the results of ANN learning processes: the memristor\textsuperscript{292-292}.

Having a joint development agreement with Hewlett Packard, South Korea’s Hynix Semiconductor Inc. is going to develop new materials and process integration technology to implement the memristor technology in its research and development fab. Being analog components such Memristors are fundamentally distinguished from other non-volatile computer memory (NVRAM) based on future technologies like FeRAM, FRAM, MRAM, PRAM, and RRAM, which are digital. FRAM (ferro-electric RAM), MRAM (magneto-resistive RAM), PRAM (phase-change RAM) and ReRAM (resistive RAM) are other next-generation NVRAM technologies with low power consumption, because of their superior scaling characteristics and small cell size having the potential to replace flash memory, DRAM or even a hard drives. Maybe, even Memristors could also replace DRAMS, flash and hard disks and perhaps also CDs and DVDs. The small size of such memory elements would also be an important means to cope with the memory wall\textsuperscript{229} because the total size of multicore on-chip memory capacity could be dramatically increased.

Menta, founded 2007 in Montpellier, France, and focusing on domain-specific embedded FPGAs (eFPGAs), has announced to use MRAM instead of flash memory for configuration code to obtain architectural benefits for partial/dynamic reconfiguration as well as ease of fabrication with standard CMOS processes. Fujitsu just introduced an 8-bit microcontroller with embedded FRAM.

### 18.5.1 Speedup Factors by FPGAs

Energy efficiency of FPGAs is not new (fig. 13). 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. 14) by a number of papers\textsuperscript{293}. For example, most bioinformatics applications, image recognition (but not rendering), encryption / decryption, and FFT-based applications are ideally suited to FPGA silicon. More recently, for instance, a factor of 3000 has been obtained in 3-D image processing for computer tomography. Biology showed speed-up factors\textsuperscript{294} up to 8723 (Smith-Waterman pattern matching: Table 8)\textsuperscript{280-283}. Multimedia reports up to 6000 (real-time face detection). Cryptology reports for DES breaking a speed-up factor of 28514\textsuperscript{98} (Table 8). Some of these speed-up studies report energy saving factors, like 3439 for the DES breaker example\textsuperscript{98}. 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 factors reported by these studies tend to be roughly 10% of the speed-up factor: the golden bullet for saving energy.

Dozens of papers\textsuperscript{293} have been published on speed-ups obtained by migrating applications from software running on a CPU, over to configware for programming FPGAs\textsuperscript{282}. Fig. 14 shows a few speedup factors picked up from literature, reporting a factor of 7.6 in accelerating radioisity calculations\textsuperscript{288}, a factor of 10 for FFT (fast Fourier transform), a speedup factor of 35 in traffic simulations\textsuperscript{289}. A speedup by a factor of 304 is reported for a R/T spectrum analyzer\textsuperscript{300}. For digital signal processing and wireless communication, as well as image processing and multimedia, speed-ups by 2 to almost 4 orders of magnitude\textsuperscript{301,302}. In the DSP area for MAC operations a speedup factor of 100 has been reported compared to the fastest DSP on the market (2004)\textsuperscript{297}. Already in 1997 versus the fastest DSP a speedup between 7 and 46 has been obtained\textsuperscript{303}. In the multimedia area we find factors ranging from 60 to 90 in video rate stereo vision\textsuperscript{304} and
from 60 to 90 in real-time face detection\textsuperscript{305}, and of 457 for hyperspectral image compression\textsuperscript{306}. In communication technology we find a speedup by 750 for UAV radar electronics\textsuperscript{307}. For acceleration of H.264 video encoding a speed-up by a factor of „only“ 43.6 has been published\textsuperscript{309}. For cryptography speed-ups by 3 to >5 orders of magnitude have been obtained. For a commercially available Lanman/NTLM Key Recovery Server\textsuperscript{310} a speedup of 50 - 70 is reported. Another cryptology application reports a factor of 1305. More recently for DES braking a speed-up by x 28514 has been reported\textsuperscript{312} (Table 8). For Bioinformatics applications\textsuperscript{225,314} speed-ups have been obtained by 2 to 4 orders of magnitude. Compared to software implementations sensational speed-up factors have been reported for software to FPGA migrations. Speedups of up to 30 has been shown in protein identification\textsuperscript{316}, by 133\textsuperscript{317} and up to 500306 in genome analysis. The Smith-Waterman algorithm, used for protein and gene sequence alignment, is basically string-matching that requires a lot of computational power\textsuperscript{225}. Here another study demonstrates speedups of 100x using Xilinx Virtex-4 hardware matched against a 2.2 GHz Opteron\textsuperscript{320}. A speedup by 288 has been obtained with Smith-Waterman at the National Cancer Institute\textsuperscript{321}. More recently a speed-up higher by more than an order of magnitude has been obtained here\textsuperscript{322}. The CHREC project\textsuperscript{272} reports running Smith-Waterman on a Novo-G supercomputer, a cluster of 24 Linux servers, each housing four Altera Stratix-III E260 FPGAs. According to this CHREC study, a four-FPGA node ran 2,665 times faster than a single 2.4 GHz Opteron core\textsuperscript{323}.

Another Smith-Waterman DNA sequencing application that would take 2.5 years on one 2.2 GHz Opteron is reported to take only 6 weeks for 150 Opterons running in parallel. Using 150 FPGAs on NRL’s Cray XD1 (speedup by 43) is reported to further reduce this time to 24 hours, which means a total speedup of 7,350X over a single Opteron\textsuperscript{324}. These are just a few examples from a wide range of publications\textsuperscript{314-322} reporting substantial speedups by FPGAs. For the Smith-Waterman algorithm example the performance per Sand per Watt has been compared between FPGA, GPU (graphics processor), a cell processor, and a general purpose processor (GPP, see Table 7)\textsuperscript{394}.

Table 7: Normalized performance at Smith-Waterman\textsuperscript{186}. 
Software to configure migration and software to hardware migrations depend on the same principles, since both are time to structure mappings. The difference is binding time: before fabrication (hardware), or after fabrication time (configware). An example is the migration of the well-known $O(n^2)$ running time bubble sort algorithm fully based on memory-cycle-hungry CPU instruction streams, also for reading and storing the data. We map the inner loop into a bidirectional pipeline register array (Figure 15). But this solution comes with access conflicts which are removed by splitting the operation into 2 phases. However, this solution wastes resources, since at any time only 50% of the conditional swap units are busy. For optimization we change the algorithm into the shuffle sort algorithm having only half as many conditional swap units. For avoiding to break bubbling up we move the contents of the $k$ register pipeline alternatingly up and down, each time by a single step. That’s why we call it „shuffle sort“.

The algorithmic complexity turns from $O(n^2)$ into $O(n)$. In a similar manner, other well-known algorithmic methods can be transformed to explore parallelism and locality, like in dynamic programming as presented in \cite{225}. Needing no CPU instructions brings additional massive speed-up. Since software is usually stored outside CPU on-chip memory, the memory wall and overhead-phenomena typical to software cause performance by additional orders of magnitude worse than that of the migrated version.

### 18.5.3 The Reconfigurable Computing Paradox

Technologically FPGAs are much less efficient than microprocessors. The clock speed is substantially lower. The routable reconfigurable wiring fabrics causes a massive wiring area overhead. There is also massive other overhead: reconfigurability overhead, since of 200 transistors maybe about 5 or less that 1 of them (fig. 9 in \cite{333}) serve the application, whereas the other 195 are used for reconfigurability (Figure 11). Often there is also routing congestion, so that not all CLBs can be used, causing further degradation of efficiency. Why does software to configure migration yield such massive improvements in speed and power consumption, although FPGAs are a much worse technology? It’s the von Neumann paradigm’s fault.

Why does software to configure migration yield such massive improvements in speed and power consumption, although FPGAs are such a much worse technology? Measuring the Gap between FPGAs and ASICs yields 30-40x Area, 12-14x Power, 3-5x Speed. This means that for FPGAs the Area*Time*Power product is by about three orders of magnitude higher than for ASICs. FPGAs have an enormous wiring overhead, and massive reconfigurability overhead (from 100 transistors about 5 serve the application, whereas the other 95 provide the reconfigurability), and, have a much slower clock speed than a CPU. Routing congestion may even further degrade FPGA efficiency. Because of a rapid increase of the number of on-chip devices, currently billions of transistors, as well as the large number of metal layers, resources get “cheaper” and thus the area cost of reconfigurable hardware is not anymore a limiting factor. Due to power limitations with future technologies not all resources can be active at the same time. Such resources then can be used to offer reconfigurability and flexibility on a chip, also targeting fault-tolerance. The consequence is, that reconfigurable computing can fill, at least partially, the above gap.
Reconfigurable computing has the potential to completely fill the above gap. Why? It is a paradigm issue: instruction streams vs. datastreams. To answer, it is the von Neumann syndrome looks a bit unfair. It’s the typical environment which is so inefficient, that the much better processor technology is left behind the leading edge by orders of magnitude. It’s a software engineering issue, that multiple levels of overhead lead to code sizes which hit the memory wall. Nathan’s law says that software is a gas, which fills any available storage space (on-chip memory, extra semiconductor memory located outside the processor chip), as well as hard disks, and even the internet. Here the memory wall is a technology issue – not directly the paradigm’s fault.

Why are technologically much worse FPGAs by orders of magnitude more efficient than von-Neumann-based (vN) microprocessors. It’s a handicap of the von vN paradigm, that computing by instruction streams is highly memory-limited. It’s a handicap of the vN-type parallelism, that internode communication reduces computational efficiency. In RC the magnitude of parallelism overcomes the clock frequency limitations. RC massively accelerates tasks by datastreaming. RC minimizes memory size and memory bandwidth by data streams instead of instruction streams. Datastream computations across a long array (before storing results in memory!) can achieve by orders of magnitude improved use of memory. All this explains why RC performance and power consumption is by orders of magnitude more efficient than with vN.

Let us also look into history. Prototyped in 1884 the Hollerith tabulator was the first electrical computer ready for mass production. Punchcard-driven it has been datastream-driven reconfigurable computing. Since integrated circuits and transistors did not yet exist the LUT was configured manually by banana plug wiring. About 60 years later the von-Neumann-type ENIAC computer came up consuming the electrical power of 200 kiloWatts and requiring a hangar full of equipment (see pictures), whereas the Hollerith machine had just the size of about two refrigerators - note: 60 years earlier! Just for computing a few ballistic tables this gigantic difference of efficiency foreshadowed that von Neumann becoming fashionable turned out to become the most dramatic misinvestment of many hundreds of billions of dollars. This disruptive about-turn to von Neumann, the most disastrous decision in the history of computing, was the overtaking of all the problems of the von Neumann syndrome, such as the never-ending software crisis approaching its 50th anniversary. "In an environment which has represented the absence of the need to think as the highest virtue this is a decided disadvantage" [Daniel Slotnick, 1967].
Even to-day, based on modern microelectronics technology, already the principles of von Neumann hardware are massively inefficient. Just about 5% of the hardware, the ALU, is doing the processing, whereas the other 95% hardware are overhead (Figure 17)\textsuperscript{231}. But orders of magnitude more inefficiency is caused by multiple overhead phenomena in the software required to follow this machine paradigm. The flood created by much more than a thousand programming languages having been developed (a subset listed in Table 3: Some Programming Languages\textsuperscript{223}) seems to exhibit more a symptom of a lack of direction, than of cleverness. Another symptom of chaos is replacing languages of high abstraction level like Pascal by the assembler-like language C. The term „software crisis“ is almost 50 years old. This term has been coined by Prof. F. L. Bauer from TU Munich when, being its general chair, he opened the first NATO Software Engineering Conference 1968 in Garmisch, Germany. Meanwhile, literature, panels and keynote addresses investigating the variety of overhead phenomena is booming\textsuperscript{205-207,329}.

The vN paradigm was criticized also by celebrities\textsuperscript{337-340}. Peter Newman had for 15 years each month the critical „computers at risk“ back pages of Communications of the ACM\textsuperscript{341}. Nathan’s law (by Nathan Myhrvold, a former CTO of Microsoft) said that software is a gas, which fills any available storage space: on-chip memory, extra semiconductor memory located outside the processor chip, as well as hard disks. It even fills the internet. Nicklaus Wirth’s pre-manycore interpretation of Moore’s law is, that “software is slowing faster than hardware is accelerating“\textsuperscript{339}. Why, how often, and, to what extent software fails, is meanwhile its own subject era\textsuperscript{336-342}.
Why is von Neumann so inefficient? It’s not only the typical environment which is so inefficient, that the much better processor technology is left behind the leading edge by orders of magnitude. We can distinguish 2 different reasons: algorithmic complexity caused by the von Neumann paradigm, and, architectural issues. There is a number of attempts to explain at least particular symptoms of this syndrome (Figure 11). The most well known architectural problem is the memory wall (Figure 16). It’s also an architecture-related software engineering issue, that multiple levels of overhead lead to massive code sizes which hit the memory wall. The „memory wall“ means, that the time to access RAM outside the processor chip is currently slightly more than a factor of 1000 slower, than to on-chip memory. This difference is growing by 50% every year.

The memory wall is really not fully the paradigm’s fault. Smart cell phone architectures show embedded software approaches to cope with memory bandwidth problems. But instruction sequencing overhead is a consequence of the von Neumann paradigm. After a full migration to static reconfigurable computing an application uses zero instructions at run time and is run by data streams only. But by a migration also the amount of data streams may be minimized by changing the algorithm. Here an illustration example for reducing the algorithmic complexity is given by the migration of the well known O(n^2) complexity bubble sort algorithm away from von Neumann. The algorithmic complexity turns from O(n^2) into O(n). In a similar manner, other well-known algorithmic methods can be transformed to explore parallelism and locality, like in dynamic programming as presented in. The combination of these effects leads to massive speed-up and massive saving of energy (see section 19.5.4).

Why does the migration from instruction streams to data streams lead to such massive speed-up? How data are moved is also a key issue. CPUs usually move data between memory blocks and require instruction streams to carry it out (first line, Table 12: Twin paradigm fundamental terminology.). The movement of data is evoked by execution of instructions due to the von Neumann paradigm. Also the execution of operations inside a CPU requires reading and decoding of instructions. On a reconfigurable platform, however, which can be modeled as a pipe network, data are moved directly from DPU to DPU. This means, that operation execution inside a DPU (not having a program counter) is “transport-triggered” (second line, Table 12: Twin paradigm fundamental terminology.). It is triggered via handshake by the arrival of the data item, not needing an instruction to call it. Not looking at dynamically reconfigurable systems (only for advanced courses) we see, that reconfigurable fabrics don’t perform any instruction sequencing at run time.

Of course, the data entering or leaving such an array (Figure 8) have to be stored in memory. The data stream machine paradigm uses auto-sequencing Memory blocks (asM). Each asM has a reconfigurable address generator and data counter inside, so that no instruction streams are needed for address computation. All these data streams can be programmed via a data-imperative languages, being a kind of sisters of classical imperative programming languages.

18.5.4 Saving Energy by Reconfigurable Computing

Recently not only speed-up, but also energy saving factors have been reported, roughly one order of magnitude lower than the speed-up. Most recently has been reported for DES breaking (a crypto application): 28,500 (speed-up) vs. 3439 (saving energy) and for DNA sequencing 8723 (speed-up) vs. 779 (saving energy) etc. (Table 8). This paper also reports factors for saving equipment cost (up to x96) and equipment size (up to 1116, see Table 8). No hangar full of equipment is needed when FPGAs are used in Scientific Computing. The Pervasiveness of FPGAs is not limited to embedded systems, but is also spread over practically all areas of scientific computing, where high performance is required and access to a supercomputing center is not available or not affordable. The desk-top supercomputer is near.

This chapter introduces the highly promising and important future role of Reconfigurable Computing (RC) and emphasizes, that it is a critical survival issue for computing-supported infrastructures worldwide
and stresses the urgency of moving RC from niche to mainstream. It urges acceptance of the massive challenge of reinventing computing, away from its currently obsolete CPU-processor-centric Aristotelian CS world model, over to a twin-paradigm Copernican model supporting energy-efficient heterogeneous systems by including the massive use of RC. This chapter also gives a flavor of the fundamentals of RC and the massive impact on the efficiency of computing it promises. Furthermore the chapter outlines the educational barriers we have to surmount and the urgent need for major funding on a global scale to run a worldwide mass movement, of a dimension at least as far reaching as the Mead-&-Conway-style VLSI design revolution in the early 80ies\textsuperscript{104-115}. The scenarios are similar: around 1980 an urgently needed designer population has been missing. Now a properly qualified programmer population is not existing. This time the problem is more difficult, requiring a twin-paradigm approach for programming heterogeneous systems including both: many-core processors and reconfigurable accelerators.

The idea of saving energy by RC use is not new\textsuperscript{303,357}: the silver bullet to massively reduce the energy consumption of computing, by up to several orders of magnitude, RC is extremely important for the survival of the world economy. It has been reported more than a decade ago, that for a given feature size, microprocessors using traditional compilers have been up to 500 times more power hungry than a pure hardware mapping of an algorithm in silicon\textsuperscript{357} (Figure 14). Speedup factors up to more than 4 orders of magnitude were reported from software to FPGA migrations (Figure 14\textsuperscript{298-314}). The energy saving factor is roughly about 10\% of the speedup factor, i. e. still up to more than 3 orders of magnitude. Already a partial paradigm shift migrating only a part of the software into configware promises to save electricity by orders of magnitude.

18.5.4.1 Traditional Green Computing

Green Computing (compare section 19.2.1) uses conservative methods to save energy by more efficient modules, circuits, and components. For example LED flat panel displays need much less power than LCD-based plasma displays with 150 - 500 watts or more. Also much more power-efficient power supply modules are possible. The potential to save power is substantially less than an order of magnitude: maybe, a factor of about 3 or 4. A special scene within Green Computing is Low Power Circuit Design, now also called Low Power System on Chip Design (LPSoCD). Its most important conference series are about 30 years old: the PATMOS (oldest) and the ISLPED conference series\textsuperscript{358-128}, a brand new conference is the e-Energy\textsuperscript{164,359}.

Several aspects are known for LPSoCD, such as: leakage power, clock gating, Active Body Bias (ABB), Adaptive Voltage Scaling (AVS), Dynamic Voltage Scaling (DVS), Multiple Supply Voltages (MSV), Multi-Threshold CMOS (MTCMOS), Power Gating (PG), Power Gating with Retention (RPG), etc.\textsuperscript{171,360}. However, the order of magnitude of the benefit to be expected from this subarea LPSoCD is rather low. By MSV in using 3 Vdds the power reduction ratio at best is about 0.4\textsuperscript{171}. LPSoCD is a matter of ASIC design, e. g. of hardwired accelerator design. Only 3\% of all design starts are ASIC designs (Figure 9) with a trend leading further down. But in fact, low power design is also used for developing better power-efficient FPGAs - to the benefit of Reconfigurable Computing. But we need a much higher potential of saving energy because “Energy cost may overtake IT equipment cost in the near future”\textsuperscript{118}. “Green Computing has become an industry-wide issue: incremental improvements are on track”\textsuperscript{362}, “But „we may ultimately need revolutionary new solutions.”\textsuperscript{362} Let me correct this statement by „we will ultimately also need revolutionary solutions (like reconfigurable computing), since we need much higher efficiency”\textsuperscript{363}.

18.5.4.2 The Role of Graphics Processors
Accelerator use of General Purpose Graphical Processors (GPGPUs) is a big fashion, so that there seems to be a perception that there is a battle between FPGAs and GPGPUs for general-purpose HPC acceleration, w. r. to speed-up and power efficiency. Meanwhile the very busy hype on the accelerator use of GPGPU seems to be over-exaggerated. Depending on the class of algorithms speed-ups factors just between x1 and up to x3 are reported, compared to „normal“ x86-based manycore architecture use. Since a compute-capable discrete GPU can draw much more than 200 watts, other authors call this massive power draw a serious roadblock to the adoption, not only in embedded systems, but even for data centers.

FPGAs from a new Xilinx 28nm high-performance, low-power process, developed by Xilinx and TSMC-optimized for high performance & low power are much better off than GPUs. NVIDIA hardware has the advantage in of ECC memory support, local cache, asynchronous transfers, and a generally more sophisticated architecture geared for general purpose computing. But AMD's offerings have the advantage of better performance per watt, at least for the 150 watt FireStream 9350 product. A recent paper on GPU-accelerated software packet router says: “We believe that the increased power consumption is tolerable, considering the performance improvement from GPUs.” But Finally it turns out that The GPU remains a specialized processor, so that we still need the traditional CPU after all. Intel is coming up with both on the same microchip.

Nvidia needs Intel more than Intel needs the GPU designer. The truth is, that Nvidia's Tesla boards use Intel Xeon chips to demonstrate the performance gains of a CPU/GPU combination. So the question is: „why bother attacking the devil, if you have to dance with it?“: GPUs can be used for traditional graphics, advanced visualization, and floating point/vector processing. The rise of general-purpose GPU computing will inexorably push graphics-flavored logic onto the CPU die by the two big x86 chip vendors, AMD with their Fusion APU (Accelerated Processing Unit) processors, maybe for early 2011. CPUs and GPUs will share the same silicon real estate. When manufactured below 32nm the earliest CPU-GPU server chips may come 2012, or 2013. Maybe, GPGPUs as a kind of alternative manycore architectures are adding to the programming challenge. NVIDIA’s CUDA programming environment has become a premier software platform for GPGPU development., whereas AMD is sticking with the open standard OpenCL sometimes considered less capable and less mature. But Nvidia has the only conformant, publically available, production OpenCL GPU drivers." But in contrast to GPGPUs, socketed FPGAs can talk to a x86 CPU without host intervention, directly over the high performance native processor bus, like Intel's Front Side Bus (FSB) with PGA expansion modules by Companies like XtremeData, DRC Computer, and Nallatech and others and using compilers from tool makers like Mitronics, Celoxica, and Impulse Accelerated Technologies etc. compiling C (or C-like) code into an FPGA logic. At the Intel Developer Forum in September 2010 Xilinx showcased the Intel® QuickPath Interconnect (Intel® QPI) technology for enabling the integration of FPGAs in high performance computing applications.

18.5.4.3 Wintel versus ARM

Most of the referenced CPU to FPGA migration speed-up reporting papers (section 19.5.1) have compared FPGAs with earlier Wintel processors, however, also with older types of FPGAs, less power-efficient. Are those data still useful? Are ARM processors more power-efficient than x86 ? ARM processors are often powered by a small battery when being used extensively in consumer electronics, including PDAs, tablets, mobile phones, digital media and music players, hand-held game consoles, calculators and computer peripherals such as hard drives, printers, and routers. In 2007 about 98 percent of the mobile phones sold this year use at least one ARM processor. ARM licensed about 1.6 billion cores in 2005. In 2005, about 1 billion ARM cores went into mobile phones. Until January 2008 over 10 billion ARM cores have been built, and iSuppli predicts that 5 billion a year will ship in 2011. Cortex processors (ARMv7) now provide faster and more power-efficient options than all those previous generations. Cortex-A targets
applications processors, as needed by smartphones that previously used ARM9 or ARM11. Cortex-R targets real-time applications, and Cortex-M targets microcontrollers. "M" stands for an improved multiplier and a faster adder. In 2009, some manufacturers introduced netbooks based on ARM architecture CPUs, in direct competition with netbooks based on Intel Atom. The new ARM Cortex seems to be more energy-efficient than Intel Atom (but only in terms of GHz/watt), but is only a simple 32-bit reduced instruction set computer (RISC), whereas Intel has a powerful 64-bit CISC architecture. The relative simplicity of ARM processors made them suitable for low power applications. This has made them dominant in the mobile and embedded electronics market as relatively low cost and small microprocessors and microcontrollers. How much will this affect the CPU to FPGA speed-up and power-save comparison figures?

Current Intel x86 processors can deliver up to 3.6 GHz while consuming up to 130 watts, or, at the low end 1.8 GHz at 40 watts. The ARM line of chips has been reported to deliver 1 GHz at 700 milliwatts (down by x50 in terms of GHz/watt), and can reach up to 2 GHz while still consuming less than a watt (down by x75 in terms of GHz/watt). So the power savings seem to be substantial. But the situation is far from being that simple. Maybe, more reasonable conclusions can be obtained by benchmarking, after ending the controversial discussion on which benchmark to use. However, benchmarks are not real life. But meanwhile, we also have extremely power-efficient FPGAs (IGLOO, for instance) also useful for battery-powered handheld devices. You see, the evaluation of power saving figures from software to configure migration is not as simple as it has been.

Intel started challenging ARM with its Atom processor moving downmarket and towards smartphones, also by buying a unit of infineon. We see an emerging competition between ARM and x86 microprocessors. Led by the Intel Atom, x86 chips are quickly migrating downwards into embedded, low-power environments, while ARM CPUs are beginning to flood upwards into the more sophisticated and demanding market spaces currently dominated by x86 processors. Now Intel is working at an X86-based Ultra Mobile Personal Computer (UMPC) to "offering leading performance while reducing the footprint and power consumption" [Jon Jadersten], trying to invade ARM's traditional domain: low-power handhelds. Intel has produced three "platforms" called McCaslin, Menlow and Moorestown. Menlow consists of a Silverthorne 45-nanometer processor, a support chip called Poulsbo for controlling I/O and graphics, and a communications module that can be either Wi-Fi- or WiMax-capable. Moorestown combines the functionality of at least McCaslin and Menlow, reducing idle power consumption by an order of magnitude. Intel has also begun talking about Mobile Internet Devices (MIDs), a kind of more powerful iPhones. But this does not mean that FPGAs are not going to win. There are also indications that Apple plans to go FPGA. Is Intel planning to buy an FPGA vendor? If not, I would not understand.

Microsoft is largely a x86 vendor, which means that most of these ARM implementations don't run a version of Windows, but run a version of Linux instead. Intel spends a lot of R&D for x86 on process and architecture. One example is the cost of verification with making over 1,000 instructions work flawlessly, with predecoded logic, complex instruction caches, and many other techniques. It's versatility both in software and hardware made the x86 what it is. ARM vs. x86 is basically a RISC vs. CISC debate. ARM has significant limitations and not yet the potential to replace x86 on the market. Does ARM not even reach the performance of 14 years old pentium? What ARM brings to the table is very low power requirements for a given level of processor performance. ARM is fine when optimized for web browsing, writing, watching video if you have right video acceleration. For running something more it will be too slow. To accelerate non-standard codecs it does not have enough instructions (like extra multimedia ones) for playing such kind of video. Everything is getting compressed now for data saving. But ARM chips could hardly handle heavy compressed files like WinRAR or so. Even the video subsystem of ARM Cortex-A8 is limited. and memory is a slow 32-bit, DDR2-200MHz. It needs faster RAM and more RAM. Poor is also double-precision floating-point throughput of ARM Cortex-A8. Summary: FPGA superiority is not threatened.

IMHO with all the improvements ARM cores still have a weaker architecture than first Intel Core (Banias) and Athlon (not 64). Where the ARM platform historically falls short is in multitasking, soft-
ware breadth, and consistency between versions. Especially the last one, where x86 remains relatively constant so that a piece of software can generally run on x86 products over a decade old, each version of ARM generally requires a platform rewrite and this will make software offerings like application stores rather interesting to manage over the long term. Hardware virtualization, at least for the short term, will be difficult, because there isn’t a lot of extra performance overhead to run a virtual machine on ARM yet. The upcoming/next-generation Cortex-A class processor code-named "Eagle," for quad-core symmetric multiprocessing, intends to help redefine the smartphone landscape again. Will it be successful? So we do not yet need to declare our dramatic CPU to FPGA comparison figures (section 19.5.1) for being obsolete.

Why does energy efficiency matter to ROI (acronym: see Table 15) and the environment? Intel's x86 chips have gained dominance in data centers. But because of the need to add so many more servers to meet our rising demand for computing, power considerations begin to determine the calculation. Cooling makes up nearly half the capital expenditure and almost two-thirds of the operation expense. However, for an Atom to ARM conversion huge masses of software have to be rewritten which means to port programs like Windows. But before an end of x86 domination a transition to lower-power server chips would take many years, if happening at all. FPGAs are still very far from being obsolete.

Meanwhile much more power-efficient FPGAs are available. The exploding market for handheld smart devices creates pressure for low power. This has radically altered electronics design choices and decisions upstream. Expensive ASICs or custom ICs simply do not work in markets where cost is a factor, but the ability to hit tight market windows and adapt to changing technology standards is paramount. This paradigm shift requires FPGAs, which offer both low power capability and system design flexibility to meet time-to-market demands and changing user requirements and standards. For instance, GE Intelligent Platforms is developing a range of digital receiver, digital transceiver and FPGA processor products based on the Virtex-6 and 7 series FPGA families from Xilinx for applications such as software defined radio, signals intelligence, tactical communications and radar\textsuperscript{186}, requiring more raw processing performance, greater capacity, higher speed I/O and lower power consumption. An example is the power consumed by the FPGA of a converter design\textsuperscript{269} The comparison of Low Power FPGAs versus Low Power CPUs is still massively in favor of the FPGAs as reported in section 19.5.1. For instance, some authors reported a 103 mWatt FPGA\textsuperscript{269}. With 3 new product families (Virtex-7, Kintex-7, Artix-7) fabricated by TSMC’s 28 nm high-k metal gate (HKMG), high performance, low power (HPL) process technology, Xilinx has substantially improved power and performance, capacity, and price\textsuperscript{270}.

Table 8: Recent speed-up & power save data from software to configware migration\textsuperscript{98,311}.

<table>
<thead>
<tr>
<th>SGI Altix 4700 w. RC 100 RASC vs. Beowulf cluster</th>
<th>speed-up factor</th>
<th>save factor</th>
<th>power</th>
<th>cost</th>
<th>size</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNA &amp; Protein sequencing</td>
<td>8723</td>
<td>779</td>
<td>22</td>
<td>253</td>
<td></td>
</tr>
<tr>
<td>DES braking</td>
<td>28514</td>
<td>3439</td>
<td>96</td>
<td>1116</td>
<td></td>
</tr>
</tbody>
</table>

Being claimed to be the industry's lowest power and widest range of small packages, Actel’s flash-based IGLOO FPGAs can be found in the latest handheld portable devices\textsuperscript{365}. It has been designed for a wide array of handheld devices, including smartphones, eBooks, cameras, medical devices, industrial scanners, military radios, and more. Actel also offers a Low-power 15,000-gate IGLOO FPGA for 99 Cents with a power consumption of only 5µW, advertised as „more than 200x less static power than competitive FPGA offerings“\textsuperscript{372}. Also Xilinx came up with much more power-efficient FPGAs. Our observation is, that lower power consumption of current and future microprocessors stands against newer FPGAs also featuring much less power and much higher performance\textsuperscript{11,374}. The conclusion is, that the role of migrations from software to Reconfigurable Computing as the silver bullet to massively save energy is not really affected, even in case that this bullet would be slightly slower (what has not yet been proved).
18.5.5 Reconfigurable Computing is the Silver Bullet

Since not offering improvements by orders of magnitude, traditional green computing (section 19.5.4.1) is not threatening the paramount role of RC. The publication of speed-ups from software to configware migration started around 1995 (Figure 14) Many of these papers (section 19.5.1) have compared FPGAs with earlier Wintel processors, however, also mostly with old types of FPGAs being less power-efficient. Although to publish saving energy by RC (section 19.5.4) started a decade later we may have a second look. But meanwhile, we also have extremely power-efficient FPGAs (IGLOO, for instance), which are also used for battery-powered handheld devices. You see, the evaluation of power saving figures from software to configware migration is not as simple as it has been.

Also microprocessors are going low power (section 19.5.4.3). Intel has begun talking about Mobile Internet Devices (MIDs), a kind of more powerful iPhones. This does not mean that FPGAs are not going to win. There are indications that Apple plans to go FPGA. I would understand if intel would be planning to buy an FPGA vendor. The upcoming/next-generation Cortex-A class processor "Eagle," intends to redefine the smartphone landscape again. However, we do not yet need to declare our dramatic CPU to FPGA comparison figures (section 19.5.1) for being obsolete. Meanwhile also much more power-efficient FPGAs are available. This has radically altered electronics design choices and decisions upstream. Expensive ASICs or custom ICs simply do not work in handheld smart devices markets where cost is a factor. The conclusion is, that the role of migrations from software to Reconfigurable Computing as the silver bullet to massively save energy is not really affected, even in case that this bullet would be slightly slower (what has not yet been proved). Not only the very high energy consumption (section 19.3.1) urges us to revolutionize the fundamentals of programmer education. The conclusion is that also because of the programming wall (section 19.3.2) we cannot avoid the need to reinvent computing. Since a lot of software has to be rewritten anyway for manycore (section 19.6.2.1) a major migration campaign really makes sense (section 19.6.2).

18.5.4.3 A New World Model of Computing

Meanwhile ASICs used as accelerators have massively lost market shares in favor of reconfigurable accelerators. Now FPGAs projects outnumber ASIC projects by 30 to 1 (Figure 9 [Gartner, 2009]), or even by 50 to 1 due to another estimation. FPGA stands for „Field-Programmable Gate Arrays“. FPGAs are structurally programmed from „configware“ sources, which are fundamentally different from the instruction-stream-based „software“ sources (Table 6). FPGAs come with a different operating system world organizing data streams and swapping configware for partially and dynamically reconfigurable platform. Introduced in 1984 and now a 5 billion US-S-world market. Modern successors of FPGAs (Figure 12) reach the market like EPPs. Also used for HPRC in supercomputing by Cray and Silicon Graphics, well proven technology is rapidly heading for mainstream.

The traditional CPU-centric world model of the CS world is obsolete. It resembles the old Aristotelian geo-centric world model. Its instruction-stream-based software-only tunnel view perspective hides structural and data stream aspects - massively threatening the progression of system performance, where we have to confront a dramatic capability gap. We need a generalized view, comparable to the Copernican world model not being geo-centric. We need a model which also includes structures and data streams and supports time to space mapping, since scaling laws favor reconfigurable spatial computing over temporal computing. Exercising time to space mapping, also by programming data streams and by software to configware migration, provides important skills: e. g. locality awareness, understanding and designing efficient manycore architectures and their memory organization being essential to cope with bottlenecks caused by bandwidth problems.

This new direction has not yet drawn the attention of the curriculum planner within the embedded sys-
tems scene. For computer science this is the opportunity of the century, of decampment for heading toward new horizons, and, to preserve the affordability of its electricity consumption. This should be a wake-up call to CS curriculum development. Each of the many different application domains has only a limited view of computing and takes it more as a mere technique than as a science on its own. This fragmentation makes it very difficult to bridge the cultural and practical gaps, since there are so many different actors and departments involved. We need the new CS world model to avoid the capability gap caused by that fragmentation. Computer Science should take the full responsibility to merge Reconfigurable Computing into CS curricula for providing Reconfigurable Computing Education from its roots. CS has the right perspective for a trans disciplinary unification in dealing with problems, which are shared across many different application domains. This new direction would also be helpful to reverse the current down trend of CS enrolment.

Not only for the definition of the term “Reconfigurable Computing” (RC) it makes sense, to use a clear terminology – not only to improve education about how to reinvent computing. It is a sluttish use of terms if “soft” or “software” is used for everything, which is not hardware. The term “software” should be used only for instruction streams and their codes. However, we generalize the term “programming” (Figure 18) such, that procedural programming (in time domain) creates sequential code, like instruction streams (software), or data streams, which we call “flowware”, and, that ”structural programming” (programming in space) creates “structural code”, which we call “configware”, since it can be used for the configuration of FPGAs (Field-Programmable Gate Arrays) or other reconfigurable platforms. Summary: Table 12.

![Figure 19: new CS world model.](image)

This established terminology reveals (Table 9), that a software to configware migration means a paradigm shift, away from the traditional programmer’s CPU-centric world model of computing, resembling the geo-centric Aristotelian world model. To reinvent computing we need a multi paradigm hetero system world model of computing science (Figure 18), which models the co-existence of, and the communication between: (1.) the traditional imperative software programming language mind set with the CPUs running by software (instruction streams), (2.) the reconfigurable modules to be structurally programmed by configware, and (3.) an imperative datastream programming language mind set with\(^\text{344}\) data stream machines programmed by flowware for generating and accepting data streams (asM in Table 12 stands for “auto-sequencing Memory”, also containing the data counter inside a reconfigurable address generator). We obtain an almost fully symmetric methodology: the only asymmetry is intra-loop parallelism, possible for data streams, however not for instruction streams (Table 11). The semantic difference of these machine paradigms is the state register: the program counter (located with the ALU) for running the instruction streams in executing software, and the data counter(s) (located in memory block(s)\(^\text{265,327}\)) for running data streams.
streams in executing flowware.

Figure 18 illustrates this triple-paradigm “Copernican” world model replacing the von-Neumann-only-centric obsolete “Aristotelian” narrow tunnel view perspective of classical software engineering, which hides almost everything, which is not instruction-stream-based. (The term “supersystolic” in Figure 8 stands for the generalization214-220 of the systolic array221-222: non-linear and non-uniform pipes are allowed like spiral, zigzag and any other irregular shapes.) This generalized model will help us to come up with a new horizon of programmer education17,382 which masters overcoming the hardware / software chasm, having been a typical misconception of the ending first history of computing. The impact is a fascinating challenge to reach new horizons of research and development in computer science. We need a new generation of talented innovative scientists and engineers to start the beginning second history of computing, not only for the survival of our important computer-based cyber infrastructures, but also for developing and integrating exciting new innovative products for the transforming post PC era global information and communication markets344. Masses of highly qualified new kinds of jobs must be created to meet the fascinating challenges of reinventing computing sciences, following the wide horizon of the new world model.

18.6.1 The Twin Paradigm Approach to tear down the Wall

By going from hardwired accelerators to programmable (reconfigurable) accelerators the traditional hardware software chasm within CS education is turning into new horizons of configware / software interfacing. Mainstream academic software engineering education is crippling itself by ignoring, that we now live in a twin-programming-paradigm world. Mainstream education also ignores the many-core programming crisis395. Why does software engineering still ignore this highly potent silver bullet candidate? Why? NIH effect? Not Invented Here?

Our contemporary model (Table 6: rows 1-3) now includes 2 procedural programming 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.

Most primitives of a software language and of a flowware language are mainly the same (Table 11). Different is only the semantics: A software language deals with sequencing a program counter. A flowware language programs one or more data counters (generalized DMA = direct memory access): for sequencing data streams. This is the only asymmetry: just a single program counter (located in the CPU). But datastream machines may have several data counters running in parallel (located in asM data memory: auto-sequencing memory - generalized of DMA). Two exceptions make flowware languages are more simple than software: 1) no data manipulation, since being set up by reconfiguration via configware; 2) parallelism
inside loops: a datastream machine may have several data counters.

Since accelerators have become programmable, the traditional hardware/software chasm has become extremely intolerable. The supercomputing scene is on the way to learn, that via a von-Neumann strategy, the exascale computer will become unaffordable. We are forced to completely reinvent computing\textsuperscript{383}. We need a generalization of Software Engineering by Program Engineering covering both, time and space domains by including 3 paradigms: Software, Flowware\textsuperscript{389}, and Configware\textsuperscript{390} (Table 6, Figure 19).

We need to rearrange undergraduate courses\textsuperscript{391}, following the advice of David Parnas\textsuperscript{230}: „The biggest payoff will not come from new research but from putting old ideas into practice and teaching people how to apply them properly“\textsuperscript{.} Examples are two old simple rules of thumb: a) loops turn into pipelines [the year 1979 and later\textsuperscript{394}], b) decision box turns into demultiplexer\textsuperscript{396}. In the 70ies, when hardware description languages came up a celebrity said: „A decision box turns into a demultiplexer\textsuperscript{396}. This is so simple. Why did it take 30 years to find out?\textsuperscript{“}. It’s the tunnel view perspective not only of software engineering. Also the flowware paradigm is based on the data stream definition published in the late 70ies. We all need to extend our horizon to rediscover old stuff.

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

(1.) The Machine Paradigm Dichotomy (von Neumann vs. Datastream machine\textsuperscript{35,392,393}), and,

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

We are still affected by the „Software Crisis“ although this term has been coined already in 1968\textsuperscript{399}. Wikipedia says\textsuperscript{377}: „The software crisis was a term used in the early days of software engineering, before it was a well-established subject.“ I disagree! IMHO, SE is not yet a well-established subject! The software crisis has been and still is manifested by software difficult to maintain, very inefficient, of low quality, not meeting requirements, and by projects running over-budget and/or over-time, being unmanageable or canceled. Only about a decade ago Niklaus Wirth’s law says\textsuperscript{305}: „Software gets faster slower than hardware gets faster.\textsuperscript{“} (Apropos „slower faster“: shouldn’t there be sometimes even a comma between „slower“ and „faster“?) It is widely agreed that here no ”silver bullet“ has yet been found. The software crisis is still far from being tamed. Dijkstra explained its causes\textsuperscript{402} by the overall complexity made possible by growing processor performance, i. e. by Moore’s Law. More recently, Microsoft’s Nathan Myhrvold even argues that Nathan's Law says that Moore's Law by the demand software creates like a gas which fills its container. Let me even go further: This gas is not only filling growing memory microchips (Moore’s law), but also growing disc space by Kryder’s Law\textsuperscript{403}, leaving Moore’s law behind as a snail’s pace. This gas is also filling the internet with its communication bandwidth capacity for growing numbers of surfers and of e-mails growing in size, video on demand, internet radio and TV, voice over IP, etc., and growing numbers of smart mobile phones and Blackberries and what else is using it and its server farms\textsuperscript{129} and cloud computing space\textsuperscript{162}, (who’s law is this?).

Table 10: Similar scenario: SE revolution vs. Mead-\&-Conway VLSI design revolution\textsuperscript{104-115}

<table>
<thead>
<tr>
<th>#</th>
<th>scene</th>
<th>problem</th>
<th>interactions</th>
<th>claims</th>
<th>solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>M-&amp;-C revolution</td>
<td>VLSI designer population missing</td>
<td>(semiconductor) technology vs. computer science</td>
<td>technology claimed it is their job (but couldn’t follow Moore’s law)</td>
<td>separating: create VLSI design education scene</td>
</tr>
<tr>
<td>2</td>
<td>proposed SE revolution</td>
<td>innov. programmer population missing</td>
<td>Software Engineering (SE) vs. Reconfigurable Computing (RC)</td>
<td>SE claims, it is not their job</td>
<td>merging: include RC into SE education</td>
</tr>
</tbody>
</table>
The software crisis has become worse by the parallel programming wall now having become more dramatic by the manycore dilemma beginning around 2004. Now the programming model is different. Software reuse and task partitioning has become more difficult. We encounter multiple challenges, such as shared resource contentions, how to implement timing behavior and to control timing precision and safety requirements, as well as to achieve both simultaneously: low power and high performance. It is questionable, whether a larger number of slower processors is more efficient. We need to construct new timing models describing all possible timed traces of bus and memory requests. We have to cope with a wide classification variety of hardware architectures: multi-threading, homogeneous vs. heterogeneous, message passing, shared memory, UMA or NUMA, symmetric (SMP) vs. asymmetric (AMP), etc. 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. Future unaffordability of our total computer operating cost is looming. The further predominance of the von Neumann programming paradigm is intolerable. We urgently need to motivate opinion leaders in software engineering and curriculum recommendation task forces.

18.6.2 A Mass Movement needed as soon as possible

The scenario resembles the VLSI design revolution, the most effective project in the history of modern computer science (row 1 in Table 10). Originally the semiconductor technology experts claimed, that they master circuit design with the left hand. The Moore curve approaching 1000 transistors/chip turned this into a design crisis with a missing qualified designer population. VLSI design education has been founded as a separate discipline outside technology, supported by a new text book, especially written for people without a technology background. Within 3 years these courses have been introduced by more than a hundred universities world-wide.

We now have a similar scenario (row 2 in Table 10). Isn’t this a strong motivation for the software engineering scene? No, not yet. In contrast to the VLSI revolution, the SE community claims, that RC is not their job, and the solution is merging instead of separating (last 2 columns in Table 10). We again need such innovative education efforts: professors back to school! We need a world-wide mass movement qualifying most programmers for twin-paradigm programming, ready for a world-wide change-over of many applications, what will create a lot of jobs at least for a decade. But in contrast to the VLSI design revolution we have to merge two so far separate disciplines: software-based CPU programming methodology with flowware- and configware-based reconfigurable computing. New horizons in massively saving energy and very high performance computing will be opened up by this generalization of Software Engineering.

Until to-day masses of IBM-360 compatible software packages are still running. Since most of the programmers (the baby boom generation) having written that stuff are retired, a bunch of universities is starting courses on "mainframe programming" for their (hopefully) successors. We see the massive inertial effects of legacy software. What are the consequences for planning major software to configware migration campaigns to save energy by reducing the energy consumption of all our IT infrastructures? We will need much effort in selecting the candidates for migration.

18.6.2.1 Legacy Software from the Mainframe Age

This section discusses, why massive software to configware migration makes sense. What are the reasons, why a migration from microprocessors, over to FPGAs makes sense, although from a technological point of view FPGAs seem to be massively less effective. The explanation of this paradox is the unbelievable inefficiency of software based on the von Neumann paradigm.
Origins of legacy software may reach back by almost half a century, like that of mainframe software. IBM and CA Technologies (Islandia, N.Y., USA, maker of mainframe software) are hard-pressed to replace the aging corps of Baby Boomers who support their still-indispensable mainframe business\(^{125}\) (SW still compatible with IBM 360 mainframes). IBM commands 85 percent of the mainframe market with some 10,000 mainframes used by 4,000 to 5,000 customers around the globe\(^{125}\). Some companies still employ an older mainframe with a screen known as a 3270 terminal emulator, which evokes the decades-old Disk Operating System, or DOS, that predated Microsoft (MSFT) Windows\(^{122}\). Modern mainframes trace their roots to the introduction of IBM's System/360 in the mid-1960s, when the oldest Baby Boomers were still teenagers. With their ability to reliably process millions of instructions per second, mainframes became popular in banking, insurance, and other industries that required high-power computing. For that era's computer science students, the mainframe represented cutting-edge technology.

For IBM mainframes are a high-margin business, generating additional software and services revenues. Margins for mainframes are about 70 percent, compared with 46 percent for the company's margins as a whole\(^{125}\). The resulting worker shortage poses a threat to IBM. If unresolved, the lack of engineers adept at designing, programming, and repairing mainframes could curb demand for one of IBM's most profitable products by alternatives to including Hewlett-Packard (HPQ) or Dell (DELL) servers to run networks, Web operations, and a growing range of the computing tasks once entrusted to mainframes.

Teaching such mainframe skills is out of vogue at many universities. Many of the engineers capable of tinkering with the refrigerator-sized machines are nearing retirement. "This inescapable demographics will be trouble for the platform"\(^{125}\). So IBM has created a curriculum designed to encourage the teaching of mainframe skills and distributed it to institutions of higher learning in 61 countries, and, began distributing its System Z Academic Initiative to 24 colleges and universities in 2003. The number swelled to 700 this year and is expected to reach 1,000 institutions by the end of 2011\(^{125}\). Aside from training new mainframe workers, CA aims to keep existing mainframe experts in place longer. It offers flexible work schedules such as three-day work weeks. The company will even consider letting seasoned engineers take summers off. CA hires about 40 to 50 people a year and also encourages retiring workers to mentor younger ones.

18.6.3 How to Reinvent Computing

The key issue is the Computer Science education dilemma. To save massive amounts of energy we need a world-wide changeover of many applications from vN machines to FPGAs. This may take more than a decade and also will create a lot of jobs. However, FPGAs are still a niche technology, since we have yet to train and qualify a sufficiently large population of programmers for FPGA programming. Advanced training of such programmers needs support everywhere: at regional, national and at global level. This subsection emphasizes that RC is a critical survival issue for computing-supported infrastructures worldwide and stresses the urgency of moving RC from niche to mainstream. It urges acceptance of the massive challenge of reinventing computing, away from its currently obsolete CPU-processor-centric Aristotelian CS world model, over to a twin-paradigm Copernican model. A massive software to configware migration campaign is needed. First this requires clever planning to optimize the effort versus its expected results. Which software packets should be migrated first. All this requires massive R&D and education efforts taking many years. Lobbying for the massive funding should be started right now. We should address politicians at all levels: community level, state level, national level, and European Union level.

To explain all this to politicians is very difficult. Since politicians always watch the sentiment of their voter population, we efficiently have to teach the public, which is a challenge. Without a strong tailwind from the media a successful lobbying seems to be almost without success. All this has to be completed as soon as possible, as long as we can still afford such a campaign. To succeed with such a challenging educational campaign the foundation of a consortium is needed for running an at least Europe-wide project.

Before going to reinvent computing let us have a look at the current scenario. The rate of ASIC-to-
FPGA conversions continue escalating. But Methodology and tool flow questions impact the bottom line. Successful switching from ASICs to FPGAs depends very much on the tools, practices, and processes chosen for the FPGA development work. With high-quality tools and complete, well-integrated solutions of vendor-independent ESL, IP re-use, verification, synthesis, and PCB flow, designers need not learn a new tool set for every FPGA vendor’s products, and, his company does not lose their freedom of FPGA vendor choice. This frees to pragmatically select the device that best fits the project needs, without concern for prior tool usage. For PCB development it is necessary to carefully consider, whether the process is predictable, and well integrated, whether it is consistent from project to project, and, whether it offers the flexibility to move to another FPGA vendor.

The remarkable phenomenon of Electronics Intellectual Property (IP) providers exists not only for ASICs, but also for configware onto FPGAs. First IP providers have been founded in the 90ies and meanwhile some of them have been acquired by others: Altium, Ansoft, ANSYS, ARM, Artisan, Cadence, CEVA, Logic Vision, Magma, Mentor Graphics, MIPS, Monolithic, Mosys, Nassda, Simplicity, Synopsys, Parthus, Rambus, Verisity, VirageLogic, Total. MIPS has been founded in 1984, but turned to the IP business later on, since 1998 officially called „MIPS Technologies“. The IP eco-system is also for FPGAs still RTL-dominated because most FPGA vendors use RTL flow so porting the design to another FPGA is not extremely complicated like it is in microprocessors where legacy code plays a high role for market dominance. The inherently parallel RTL application is mapped automatically parallelized by CAD tools, however, not necessarily free of bugs like issues of nanosecond real-time response times or data throughput requirements. The required visibility is not provided by traditional debug/trace tools. So it is not always easier to port IPs both for ASICs and FPGAs as both use RTL (Register Transfer Languages). All this still is one of major difficulties also for manycore solutions. Merging both worlds is really a challenge.

Meanwhile FPGAs are also used everywhere for high performance in scientific computing, where this is really a new computing culture - not at all a variety of hardware design. Instead of H/S codesign we have here software / configware co-design (SC co-design), which is really a computing issue. This major new direction of developments in science will determine how academic computing will look in 2015 or even earlier. The instruction-stream-based mind set will loose its monopoly-like dominance and the CPU will quit its central role - to be more an auxiliary clerk, also for software compatibility issues.

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

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

The remarkable phenomenon of Electronics Intellectual Property (IP) providers exists not only for ASICs, but also for configware onto FPGAs. First IP providers have been founded in the 90ies and meanwhile some of them have been acquired by others: Altium, Ansoft, ANSYS, ARM, Artisan, Cadence, CEVA, Logic Vision, Magma, Mentor Graphics, MIPS, Monolithic, Mosys, Nassda, Simplicity, Synopsys, Parthus, Rambus, Verisity, VirageLogic, Total. MIPS has been founded in 1984, but turned to the IP business later on, since 1998 officially called „MIPS Technologies“. The IP eco-system is also for FPGAs still RTL-dominated because most FPGA vendors use RTL flow so porting the design to another FPGA is not extremely complicated like it is in microprocessors where legacy code plays a high role for market dominance. The inherently parallel RTL application is mapped automatically parallelized by CAD tools, however, not necessarily free of bugs like issues of nanosecond real-time response times or data throughput requirements. The required visibility is not provided by traditional debug/trace tools. So it is not always easier to port IPs both for ASICs and FPGAs as both use RTL (Register Transfer Languages). All this still is one of major difficulties also for manycore solutions. Merging both worlds is really a challenge.

Meanwhile FPGAs are also used everywhere for high performance in scientific computing, where this is really a new computing culture - not at all a variety of hardware design. Instead of H/S codesign we have here software / configware co-design (SC co-design), which is really a computing issue. This major new direction of developments in science will determine how academic computing will look in 2015 or even earlier. The instruction-stream-based mind set will loose its monopoly-like dominance and the CPU will quit its central role - to be more an auxiliary clerk, also for software compatibility issues.

Table 12: Twin paradigm fundamental terminology.
An introduction to Reconfigurable Computing (RC) should regard the background to be expected from the reader. This chapter of the book mainly addresses a bit IT-savvy people in the public and its mass media, as well as „software engineers“. Here an introduction is difficult, since in both communities people typically know nothing or almost nothing about RC. To move RC from its niche market into mainstream massive funding is needed for R&D and to reinvent programming education. To yield the attention of media and the politicians we need a highly effective campaign by mass media.

Up to 32 cores per chip have already been pre-announced. Already six, eight or more CPUs mostly talk to each other instead of getting work done. Transitioning from single core to multi-core can certainly be a difficult design challenge. What issues are encountered when moving from single core to multi-core? Prior to multicore, programmers had it easy. The same old software runs on the new chips much faster, just by a free ride on Moore’s curve. However, software designed for single core may run slower on a multicore machine. Computer games and some telecommunications and compression/decompression applications are relatively easy to parallelize. But not only complex applications that involve a great deal of data are not easy to parallelize. Often the algorithm has to be changed and the application has to be re-coded. Parallel applications require highly sophisticated debugging tools, also to cope with new kinds of bugs introduced by the parallelism. To take advantage of the additional cores most applications have to be rewritten, unfortunately also introducing new types of bugs. Often the algorithm has to be changed. Programmers are not trained to think toward parallel processing. For the required style of parallel programming a sufficiently large qualified programmer population is far from existing.

<table>
<thead>
<tr>
<th>#</th>
<th>source</th>
<th>controlled by</th>
<th>machine paradigm</th>
<th>state register</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>software</td>
<td>instruction streams</td>
<td>von Neumann</td>
<td>program counter</td>
</tr>
<tr>
<td>2</td>
<td>configware</td>
<td>(configuration memory)</td>
<td>(reconfigurable datapaths)</td>
<td>none (hidden)</td>
</tr>
<tr>
<td>3</td>
<td>flowware</td>
<td>reconfigurable address generator</td>
<td>data stream machine</td>
<td>data counter</td>
</tr>
</tbody>
</table>

An introduction to Reconfigurable Computing (RC) should regard the background to be expected from the reader. This chapter of the book mainly addresses a bit IT-savvy people in the public and its mass media, as well as „software engineers“. Here an introduction is difficult, since in both communities people typically know nothing or almost nothing about RC. To move RC from its niche market into mainstream massive funding is needed for R&D and to reinvent programming education. To yield the attention of media and the politicians we need a highly effective campaign by mass media.

Up to 32 cores per chip have already been pre-announced. Already six, eight or more CPUs mostly talk to each other instead of getting work done. Transitioning from single core to multi-core can certainly be a difficult design challenge. What issues are encountered when moving from single core to multi-core? Prior to multicore, programmers had it easy. The same old software runs on the new chips much faster, just by a free ride on Moore’s curve. However, software designed for single core may run slower on a multicore machine. Computer games and some telecommunications and compression/decompression applications are relatively easy to parallelize. But not only complex applications that involve a great deal of data are not easy to parallelize. Often the algorithm has to be changed and the application has to be re-coded. Parallel applications require highly sophisticated debugging tools, also to cope with new kinds of bugs introduced by the parallelism. To take advantage of the additional cores most applications have to be rewritten, unfortunately also introducing new types of bugs. Often the algorithm has to be changed. Programmers are not trained to think toward parallel processing. For the required style of parallel programming a sufficiently large qualified programmer population is far from existing.

Table 13: Some Functional Languages.

<table>
<thead>
<tr>
<th>Language</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Curi</td>
</tr>
<tr>
<td>APL</td>
<td>Curry</td>
</tr>
<tr>
<td>CAL</td>
<td>Dylan</td>
</tr>
<tr>
<td>Charity</td>
<td>Erlang</td>
</tr>
<tr>
<td>Clean</td>
<td>F#</td>
</tr>
<tr>
<td>Clojure</td>
<td>Haskell</td>
</tr>
<tr>
<td>Common Lisp Hop</td>
<td>Lush</td>
</tr>
<tr>
<td>Opal</td>
<td>Loco</td>
</tr>
<tr>
<td>Scala</td>
<td>Ocaml</td>
</tr>
<tr>
<td>Scheme</td>
<td>Poplog</td>
</tr>
<tr>
<td>Miranda</td>
<td>Standard ML</td>
</tr>
<tr>
<td>OPS5</td>
<td>Q</td>
</tr>
<tr>
<td>R</td>
<td>Tea</td>
</tr>
<tr>
<td>Refal</td>
<td>Little b</td>
</tr>
<tr>
<td>Mythril</td>
<td>ML</td>
</tr>
<tr>
<td>R</td>
<td>Spreadsheets</td>
</tr>
<tr>
<td>Teal</td>
<td></td>
</tr>
<tr>
<td>Scala</td>
<td>Pure</td>
</tr>
</tbody>
</table>

Following list outlines the educational barriers we have to surmount and the urgent need for major funding on a global scale to run a world-wide mass movement, of a dimension as far reaching as the Mead-\&-Conway-style microelectronics revolution in the early 80ies. Problems We Must Solve:

1.) A mass migration from software to configware for the benefit of massively saving energy, of much higher performance, and, of gaining high flexibility.

2.) From time to time only a smaller part of legacy software can be migrated. For optimization we need to develop a migration priority list to identify the most promising candidates.

3.) In the future we also will have to decide: should we also use neurocomputing. To overcome the limits deriving by the increasing complexity and the associated workload to maintain such complex infrastructure, one possibility is to adopt self-adaptive and autonomic computing systems. A self-adaptive and autonomic computing system is a system able to configure, heal, optimize and
protect itself without the need for human intervention.

Table 14: Some Dataflow Languages

<table>
<thead>
<tr>
<th>AviSynth</th>
<th>G</th>
<th>Max/Map</th>
<th>Prograph</th>
<th>SISAL</th>
<th>vvv</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMDFM</td>
<td>JMax</td>
<td>Microsoft</td>
<td>Pure Data</td>
<td>SPACE-VHDL</td>
<td></td>
</tr>
<tr>
<td>Closure</td>
<td>LabVIEW</td>
<td>Monk</td>
<td>Quartz</td>
<td>AREVA</td>
<td>Verilog</td>
</tr>
<tr>
<td>Dup</td>
<td>LAU</td>
<td>MoPL</td>
<td>Composer</td>
<td>Tersus</td>
<td>XEE</td>
</tr>
<tr>
<td>Fastflow</td>
<td>Lily</td>
<td>VPL</td>
<td>Show and Tell</td>
<td>VBUS</td>
<td>X10</td>
</tr>
<tr>
<td>Hartmann</td>
<td>Lucid</td>
<td>OpenWire</td>
<td>Simulink</td>
<td>VEE</td>
<td>VBUS</td>
</tr>
<tr>
<td>pipelines</td>
<td>Lustre</td>
<td>Oz</td>
<td>SAC</td>
<td>VisSim</td>
<td></td>
</tr>
</tbody>
</table>

(4.) Another obstacle is, that a qualified programmer population needed for such a mass movement campaign is missing, and, should be available at least throughout this beginning decade.

Programmers have to be prepared for the migration of their software. In any case the result will mostly be hetero systems, where programmers with an instruction-stream-based sequential-only mind set are not qualified. Educating – or re-educating – programmers is mandatory. Or even to keep one’s head above water. And, as quick as possible, we have to reinvent courses at academia and all other kinds of schools, and, have to upgrade our highly obsolete curriculum plans and recommendations. We have to take care of a massive programmer productivity decline for four reasons:

(a.) to cope with the many-core crisis where more parallel programming qualifications are a must,
(b.) to resolve the extreme shortage of programmers qualified for RC, and
(c.) hetero systems must be programmed (like modern FPGAs featuring all 3: reconfigurable fabrics, hardwired blocks, and CPUs) requiring twin paradigm programming skills.
(d.) could by the upcoming memristor technology the area of neurocomputing lead us in the future to hetero systems requiring triple paradigm skills?

In consequence we need innovative undergraduate programming courses which also teach a sense for locality. Such a sense of locality needed for classical parallel programming is already coming along in RC with time to space mapping required to structurally map an application to the datastream side of the twin paradigm approach. This means, that teaching the structural programming of RC also exercises the sense of locality needed for traditional parallel programming. The extension of the non-sequential part of education should be optimized not to scare away undergraduate students. Twin-paradigm lab courses should be MathWorks-supported model-based, mainly at the abstraction level of pipe networks.

Von Neumann scales up cost, performance, power, cooling and reliability concerns. Bill Dally summarizes: "performance = parallelism / efficiency = locality": high performance requires parallelism of operations given by RC platforms. Efficiency is possible by optimum locality of data and program code, since the movement of data and instruction streams is the reason of bottlenecks by resource contention and bandwidth limitations. These locality problems are the reason, that programming embedded systems and supercomputers has become a very complex and difficult area of research and development. In hetero computing systems Reconfigurable Computing is a very important accelerator of locality and parallelism.

That’s why RC should urgently become mainstream. Several reasons have prevented RC from truly becoming mainstream. The execution model is inherently different from the traditional sequential paradigm were we can reason about state transition sequences much better than in a hardware or a concurrent execution model. As a consequence, the development and validation of tools is substantially a traditional hardware mind set. For software developers, it is always this mythical chip that got added by the hardware designer onto the board. The problem we have to solve is, how to teach programming FPGAs to programmers? What new models? what languages? what tools?

The key issue is Programmer productivity. It is a handicap of vN-oriented textual programming languages, that layers of abstraction hide critical sources and limit efficient programming for parallel execution by lacking locality awareness. We need to reinvent the tool flow for programming both, manycore and RC. Better tools are urgently also needed since acceleration by RC also requires more programming effort
because an in-depth application study is required. Tools are still limited and above all fairly bridle. This means programmers must master the details of not only software development but also of hardware design. Such a set of skills is also not taught as part of major electrical engineering courses severely constraining the pool of engineering with the "right" mindset for programming RC to a selected few. Moreover the recent evolution of FPGAs and to some extent coarse-grain RC architecture make programmer and performance portability a little less difficult at best.

We need to model and to program real-time and embedded applications. What model and what language should we use for hetero systems development? The language inflation does not motivate to invent a new language (Table 1, Table 2, Table 3: Some Programming Languages).

Table 3). We should investigate primitives and models available in existing languages with programming of multicore and the efforts on the way to address them. Currently popular at the manycore side are also open solutions like OpenMP and several tools from Intel to help programmers exploit its multicore processors. Since also exploiting locality of reference the partitioned global address space (PGAS), a parallel programming model, seems to be attractive for solving twin paradigm problems. The PGAS model is the basis of Unified Parallel C, Co-array Fortran, Titanium, Fortress, Chapel and X10. A candidate could be the programming language X10 developed by IBM which is designed for parallel programming using the partitioned global address space (PGAS) model. X10, also exploiting locality of reference, uses the concept of parent and child relationships for activities and supports user-defined primitive struct types; globally distributed arrays, and structured and unstructured parallelism. We definitely should look at SPEAR.
tool set having been developed for hetero systems within the framework of the MORPHEUS project\textsuperscript{263} funded by the European Union. What about UML? We should investigate whether UML offers us interesting features\textsuperscript{212}. What can we learn from OpenCL and other languages\textsuperscript{215}? What about functional languages (Table 13) or dataflow languages (Table 14)?

![Complexity of the Value Chain](source: Gartner)

Lowering the barrier of access of RC to the average programmers is one of the objectives of the REFLECT project\textsuperscript{267} by retaining the "traditional" imperative programming mindset in a high-level language environment such as MATLAB\textsuperscript{418} and rely on the concepts of Aspects to provide a clean mechanism (at source code level) for the advanced user to provide key information for a compilation and synthesis tool to do a good job in mapping the computation to hardware. The approach should be by no means fully automatic\textsuperscript{381}. Instead, we have the programmer involved but controlling the high-level aspects of the mapping while the tools take care of the low-level, error-prone steps. We may learn a lot from all this.

There’s been no lack of effort in this area. But the silver bullet has not yet been completed. But we see it coming up from the highly promising new horizon: a model-based twin-paradigm methodology to master hetero of all 3: Single-core, Multicore, & Reconfigurable Computing. Let me propose the acronym RC2RC (Reconfigurable Computing to Reinvent Computing), or R2R. We need to Rewrite Textbooks for R2R (RT4R2R), and all our friends should become Reinvent Computing Evangelists (RCE) (or R2R Evangelists (R2RE)). A qualified programmer population for hetero systems is not existing. We should kick off a world-wide mass movement by learning from the Mead-\&-Conway VLSI design revolution, started around 1980, since a designer population did not exist\textsuperscript{104,115}. 

\[IP = \text{Intellectual property, IDM = integrated device manufacturer, EMS = electronics manufacturing service, OEM = original equipment manufacturer, VAR = value-added reseller}\]
Within an increasingly complex value chain (Figure 20) we are hit by the impact of vertical disintegration (Figure 21). Currently we see here at IP core level the challenging battle between FPGAs vs. MPSoC-like platforms which results in a battle between Software programming and RTL programming (register transfer level programming using hardware languages like VHDL or Verilog)\(^{147}\). We must extend the “traditional” imperative programming mindset (for software) by a twin-paradigm imperative mind also including an imperative datastream programming methodology (for „flowware“- for terminology see Table 11)\(^{344}\). We obtain an almost fully symmetric methodology: the only asymmetry is intra-loop parallelism, possible for data streams, however not for instruction streams (Table 11). The semantic difference of these machine paradigms is the state register: the program counter (located with the ALU) for running the instruction streams in executing software, and data counter(s) (located in memory block(s))\(^{327,255}\) for running data streams in executing flowware. Using schematic or flow chart graphical design interfaces, IP blocks, software routines and I/O systems can be quickly combined to explore and develop innovative product functionality without the need for low-level engineering\(^{247}\) (see section 19.4.2).

We can do more than just programming the on-board FPGA\(^{247}\). By intelligent communication between hardware platform and the high-level design software, the system could directly interact with all parts of the development board. Peripherals can then be swapped on the fly, by automatically reconfiguring interface layers and configuration files. So the complete development system, including the physical hardware, acts as the one design environment. Hardware could conceivably become the final product.

Recently the drag-and-drop LabVIEW graphical programming environment\(^{419,420}\) came up as an FPGA development environment\(^{421}\). Unlike textual hardware description languages, it is inherently parallel. LabVIEW 2010 has some interesting features for FPGA specialists including extended IP input, fast cycle-accurate simulation, new compilation options, and a new route to system integration\(^{421}\). This environment now provides much closer coupling with the Xilinx Core Generator tool for loading code into FPGAs mounted in a range of boards\(^{421}\). It is possible to return to the LabVIEW screen to create the system surrounding the FPGA, adding IO and other peripherals. The features of this product are good examples to provide ideas for what we need for parallel programming both, manycore and RC platforms and hetero systems of both. We need a well designed graphic user interface showing the architecture of the NoC (network on chip) and beyond, providing the visualization of locality to identify critical locations with possible bottlenecks by shared resource contention and bandwidth limitations.

The sequential-only-oriented Turing-only world has perished. We need to recognize, that we are now living in the world of hetero computing. Getting familiar with LabVIEW may give us some inspirations about the direction to go. Also the call for papers of the 4th Workshop on Programmability Issues for Heterogeneous Multicore (MULTIPROG-2011) is very helpful by listing, what we really need in this world to solve the programmability crisis\(^{429}\). For discovering and understanding parallelism, performance, bottlenecks, dependency problems and debugging we need understandability tools, discovery tools, locality
Usually it is very difficult to understand the code generated by compiler optimizations and other compiler techniques. We urgently need visualization of locality. Maybe, also other understandability tools could be helpful here. We urgently need a much better understanding of memory system architecture, Network on Chip (NoC) architecture, and, entire manycore SoC architectures and their routes of code and data. We need more knowledge about what architectural support do we need for compilers and programming models. Textual language extensions tend to be not very helpful, since being sequential-only-based, textual descriptions are typically very difficult for understanding parallelism. IMHO model-driven understandability tools based on real-time graphics should be the way to go. Inspirations from labVIEW and Simulink\textsuperscript{418} and their user interfaces may be useful.

18.7 Conclusions

This chapter of the book has emphasized that Reconfigurable Computing (RC) is a critical survival issue for computing-supported infrastructures worldwide and has stressed the urgency of moving RC from niche to mainstream. It is a critical issue for two reasons. One problem is the trouble with multicore: Chipmakers are designing microprocessors that most programmers can’t program\textsuperscript{191}. Reconfigurable computing is a key part of the solution to cope with the parallel programming wall.

The other problem is the dramatically high energy consumption by von Neumann computing, caused by the von Neumann Syndrome. A mass migration of applications from software over to configware for running RC platforms is needed. For both problems the qualified programmer population is not existing. Although providing some remarks about the multicore dilemma and proposals for its solution, this chapter mainly covered the side of Reconfigurable Computing. But we cannot completely separate these problem areas from each other, since the same programmer population has to be retrained to be qualified for programming hetero systems including both paradigms: parallel data streams coming with reconfigurable platforms, as well as parallel instruction streams.

FPGAs have a relatively low technology maturity and small user base compared to software. This will change. Large parts of FPGA solution development is spent on learning specific FPGA board APIs and debugging in hardware FPGAs. Designers’ Heel is in their long development time, since relatively low level HDLs like VHDL or Verilog are still dominant. Other directions of development try to work with C language sources, like Stone Ridge Technologies with its FPGA board and the development kit with the Impulse C tool set from Impulse accelerated technologies. A product review\textsuperscript{411} reports that software developers write HLL (high level language) algorithms that rapidly compile to optimized RTL run time language targeting Stone Ridge’s RDX-11 FPGA board and development kit. They report claims that, for designs with significant non-sequential logic, the speed improvements can be 10 – 100x. Also that, compared to hand coded RTL, the design entry takes only two thirds the time and iterations one eighth the time. We have to check this and need to reinvent major parts of this area.

We urgently need a world-wide mass movement of R&D and education to be more massively funded and supported than the Mead-&-Conway VLSI design revolution in the early 80ies, which so far has been the most effective project in the history of modern computing science\textsuperscript{104-109}. This chapter of this book urges to accept the massive challenge of reinventing computing, away from its currently obsolete CPU-processor-centric Aristotelian CS world model, to a twin-paradigm Copernican model. A massive software to configware migration campaign is needed. First this requires clever planning to optimize all its aspects.
We also need to develop plans deciding, which software packets need to be migrated, and, which of them should be migrated first. All this requires many years, probably a decade of massive R&D and education efforts. We cannot afford to hesitate. Lobbying for the massive funding should be started right now. We should address politicians at all levels: community level, state level, national level, and European Union level. To explain all this to politicians is very, very difficult. Since politicians always watch the sentiment of their voter population, we efficiently have to teach the public, which is a dramatic challenge. How do we effectively reach as many as possible people by the media? Scientific studies point toward embedding into amusing or entertaining stories, presentation inside docu soaps, soap operas, reality shows or infotainment formats. Without the support by a strong tailwind from the media a successful lobbying does not seem to have any chance. All this has to be completed as soon as possible, as soon as we can still afford such massive activities. To succeed with such a challenging educational campaign the foundation of a powerful consortium to be funded at all levels is needed for running an at least Europe-wide project.

<table>
<thead>
<tr>
<th>acronym</th>
<th>meaning</th>
<th>acronym</th>
<th>meaning</th>
<th>acronym</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>Arithmetic/Logic Unit</td>
<td>ePROM</td>
<td>e-programmable PROM</td>
<td>MRAM</td>
<td>Magnetoresistive RAM</td>
</tr>
<tr>
<td>AMP</td>
<td>Asymmetric MultiProcessor</td>
<td>ERC</td>
<td>Electrical Rules Checker</td>
<td>NoRMA</td>
<td>No-Remote-Memory-Access</td>
</tr>
<tr>
<td>ASIC</td>
<td>Application-specific</td>
<td>ESL</td>
<td>Electronic System-Level design</td>
<td>NUMA</td>
<td>Non-Uniform MA</td>
</tr>
<tr>
<td></td>
<td>Integrated 'Circuit'</td>
<td></td>
<td>European Union</td>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>ASIP</td>
<td>Application-specific</td>
<td>FDMP</td>
<td>function-distributed MP</td>
<td>PL</td>
<td>Programming Language</td>
</tr>
<tr>
<td></td>
<td>Instruction Set Processor</td>
<td>FIR</td>
<td>Finite Impulse Response</td>
<td>PLA</td>
<td>Programmable LA</td>
</tr>
<tr>
<td>ASMP</td>
<td>Asymmetric MultiProcessor</td>
<td>FME</td>
<td>Flash Media Encoder</td>
<td>PROM</td>
<td>Programmable ROM</td>
</tr>
<tr>
<td>CABAC</td>
<td>Context-Adaptive Binary</td>
<td>FW</td>
<td>Field-Programmable</td>
<td>PU</td>
<td>Processing Unit</td>
</tr>
<tr>
<td></td>
<td>Arithmetic Coding</td>
<td>FPGA</td>
<td>Field-programmable GA</td>
<td>r  reconfigurable or FP</td>
<td></td>
</tr>
<tr>
<td>cNUMA</td>
<td>Cache coherent NUMA</td>
<td>FPL</td>
<td>Field-programmable Logic</td>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>CE</td>
<td>Computer Engineering</td>
<td>FPU</td>
<td>Floating Point Unit</td>
<td>RC</td>
<td>Reconfigurable Computing</td>
</tr>
<tr>
<td>CS</td>
<td>Computer Science</td>
<td>FRAM</td>
<td>Ferroelectric RAM</td>
<td>rDPA</td>
<td>Reconfigurable DPA</td>
</tr>
<tr>
<td>CHREC</td>
<td>NSF Center for High-</td>
<td>GA</td>
<td>(routeable) Gate Array</td>
<td>rDPU</td>
<td>Reconfigurable DPU</td>
</tr>
<tr>
<td></td>
<td>Performance Reconfigurable</td>
<td>HPC</td>
<td>High Performance Computing</td>
<td>RE</td>
<td>reconfigurable Element</td>
</tr>
<tr>
<td>CLB</td>
<td>Configurable Logic Block</td>
<td>IPRC</td>
<td>High Performance RC</td>
<td>ROI</td>
<td>Return On Investment</td>
</tr>
<tr>
<td>CMP</td>
<td>Chip MultiProcessor</td>
<td>HWD</td>
<td>Hard-wired Device</td>
<td>ROM</td>
<td>Read-Only Memory</td>
</tr>
<tr>
<td>CoMA</td>
<td>Cache-only-Memory-Access</td>
<td>ICT</td>
<td>Information and Communication</td>
<td>RTL</td>
<td>Register Transfer Language</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
<td>IEA</td>
<td>Intern 1 Energy Agency</td>
<td>R&amp;D</td>
<td>Research &amp; Development</td>
</tr>
<tr>
<td>DEPS</td>
<td>dynamic energy/performance</td>
<td>IME</td>
<td>Integer Motion Estimation</td>
<td>SCTR</td>
<td>Stream Control Transmission</td>
</tr>
<tr>
<td></td>
<td>Scaling</td>
<td></td>
<td></td>
<td>SE</td>
<td>Software Engineering</td>
</tr>
<tr>
<td>DNA</td>
<td>DeoxyriboseNucleic Acid</td>
<td>IP</td>
<td>Intellectual Property</td>
<td>SMP</td>
<td>Symmetric MultiProcessor</td>
</tr>
<tr>
<td>DPA</td>
<td>DataPath Array</td>
<td>IT</td>
<td>Information Technology</td>
<td>STM</td>
<td>Software Transactional Memory</td>
</tr>
<tr>
<td>DPU</td>
<td>DataPath Unit</td>
<td>LA</td>
<td>(compact) Logic Array</td>
<td>TDP</td>
<td>Thermal Design Power</td>
</tr>
<tr>
<td>DSM</td>
<td>Distributed Shared Memory</td>
<td>LUT</td>
<td>Look-Up Table</td>
<td>TM</td>
<td>Transactional Memory</td>
</tr>
<tr>
<td>DSP</td>
<td>Digital Signal Processing</td>
<td>LCPM</td>
<td>Loosely coupled MP</td>
<td>UMA</td>
<td>Uniform MA</td>
</tr>
<tr>
<td>DVES</td>
<td>dynamic voltage / frequency</td>
<td>MA</td>
<td>Memory Architecture</td>
<td>VLSI</td>
<td>Very Large Scale Integrated</td>
</tr>
<tr>
<td></td>
<td>scaling</td>
<td>MAC</td>
<td>Multiply/Accumulate Unit</td>
<td>vN</td>
<td>von Neumann</td>
</tr>
<tr>
<td>EDA</td>
<td>Electronics Design Automation</td>
<td>MIPS</td>
<td>Mio Instructions P. Second</td>
<td>WCET</td>
<td>worst case execution time</td>
</tr>
<tr>
<td>eASIC</td>
<td>partly FPGAsyle ASIC</td>
<td>MoPL</td>
<td>Map-oriented PL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ePLA</td>
<td>e-Programmable LA</td>
<td>MP</td>
<td>Multi-Processor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EPP</td>
<td>Extensible Programmable</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Platform</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
References

1. Clive Maxfield: Reproducing a Cray 1 supercomputer in a single FPGA; Sep 3, 2010, EETimes
3. Joao Cardoso, Michael Huebner (Editors): Reconfigurable Computing; Springer Verlag 2010
5. R. Hartenstein: The Grand Challenge To Reinvent Computing; XXX Congress of the SBC, 20-23 July 2010, Belo Horizonte, MG, Brazil
7. Raj Jammy (keynote): Emerging Low Power Technologies: CMOS or Beyond CMOS? 2010 Int’l Symp. on Low Power Electronics and Design (ISLPED), Austin, TX, USA, Aug 18-20, 2010
9. Dimin Niu et al.: Low-power Dual-element Memristor-Based Memory Design; 2010 Int’l Symp. on Low Power Electronics and Design (ISLPED), Austin, TX, USA Aug 18-20, 2010
10. Ajith Amerasekera (keynote): Ultra Low Power Electronics in the Next Decade; 2010 Int’l Symp. on Low Power Electronics and Design (ISLPED), Austin, TX, USA Aug 18-20, 2010
11. Celeste LeCompte: Green Data Centers; March 11, 2010, GigaOM (BusinessWeek
13. Fei Li Yan Lin, Lei He, Jason Cong: Low-power FPGA using pre-defined dual-Vdd/dual-Vt fabrics; 2004 ACM/SIGDA 12th int’l symp. on FPGAs, 2004, Monterey, CA, USA
15. Mark Guzdial: How We Teach Introductory Computer Science is Wrong; http://cacm.acm.org/blogs/blogcacm/45725
17. Joao M. P. Cardoso: New Challenges in Computer Science Education; 10th ann. SIGCSE conf. on Innovation and technology in computer science education, June 27-29, 2005, Monte de Caparica, Portugal
20. Th. Hartmann, G. Kretzschmar: Unser ausgebrannter Planet: Von der Weisheit der Erde und der Torheit der Moderne; March 1, 2000, Riemann Verlag
22. Stanley Donaldson: Giant Earthquake will drain world’s oil supplies like a flushing toilet; Dec 24, 2002, Weekly World News
23. C. J. Campbell: The Coming Oil Crisis; April 1, 2004, Multi-Science Publishing Co. Ltd.
24. R. Heimbeg: Powerdown: Options and Actions for a Post-Carbon World; Sep 1, 2004, New Society
30. R. Heimbeg: The Party’s Over: Oil, War and the Fate of Industrial Societies; June 1, 2005, New Society
32. J. H. Kunstler: The Long Emergency: Surviving the End of Oil, Climate Change, and Other Converging Catastrophes of the Twenty-First Century; March 2, 2006, Grove Press
33. Wolfgang Gründinger: Die Energiefalle: Ein Rückblick auf das Erdölzeitalter; March 15, 2006, Beck
35. Sonia Shah: Crude: The Story of Oil; May 1, 2006, Seven Stories Press
36. Leonardo Maugeri: The Age of Oil: The Mythology, History, and Future of the World’s Most Controversial Re-
77. J. W. Rawles: How to Survive the End of the World as We Know It: Tactics, Techniques, and Technologies for Uncertain Times; September 30, 2009, Plume
79. Jörg Schindler, Martin Held, Gerd Würdemann: Postfossile Mobilität: Wegweiser für die Zeit nach dem Peak Oil; Nov 28, 2009, Vas-Verlag für Akademische Schriften
83. Nouriel Roubini, Stephen Mihm, Jürgen Neubauer, Petra Pyka: Das Ende der Weltwirtschaft und ihre Zukunft: Crisis Economics; May 11, 2010, Campus Verlag
84. Reiner Hartenstein: Configware für Supercomputing: Aufbruch zum Personal Supercomputer; Praxis der Informationsverarbeitung und Kommunikation (PIK), 2005
118. http://wiki.answers.com/Q/Why_are_computers_important_in_the_world
122. N. N.: HPC Technology Makes Car Safety Job 1; HPCwire, August 05, 2010
125. Rachael King: Big Tech Problem as Mainframes Outlast Workforce; August 2, 2010, Bloomberg Business Week
126. Rebecca Scudder: Definitions in Green Computing; Jan 29, 2010 BRIGHT HUB
128. Kevin Nowka (keynote): Technology variability and uncertainty implications for power efficient VLSI systems; 2010 Int'l Symp. Low Power Electronics & Design (ISLPED), Austin, TX, USA Aug 18-20, 2010
129. R. Katz: Tech Titans Building Boom - Google, Microsoft, and other Internet giants race to build the mega data centers that will power cloud computing; IEEE Spectrum, February 2009
131. Pelamis Wave Power Ltd., Edinburgh, Scotland
135. Rebecca Scudder: Top Rated Energy Efficient Computer Monitors; Mar 7, 2010, BRIGHT HUB
139. Andy Hopper: Computing for the Future of the Planet; 2008, Trans. R. Soc. A 366(1881)
141. Don Tapscott: The Dubai Summit On Redesigning Global Cooperation And Problem Solving; Nov 26, 2009, The Huffington Post
143. Giovanni De Micheli: Nano-systems; Devices, Circuits, Architectures and Applications; ARTIST Summer School Europe 2010, Autrans, France.
145. Syed Zahid Ahmed, Gilles Sassatelli, Lionel Torres, Laurent Rougé: Survey of new trends in Industry for Programmable hardware: FPGAs, MPPAs, MSoCs, Structured ASICs, eFPGAs and new wave of innovation in FPGAs; 20th International Conference on Field Programmable Logic and Applications (FPL 2010), Aug 31 - Sep 2, 2010, Milano, Italy
149. http://www.lirmm.fr/~ahmed/files/DATE09/DATE09_SZA_03.3_4_slides.pdf
151. J. Garlick, Will the Internet Break? ISP Economics Assessment to 2012; Oct 2008, Screen Digest
156. Wu-chun Feng: On the Second Coming of Green Destiny¶; International Conference on Energy-Aware HPC; Sep 16-17, 2010, Hamburg, Germany
159. Stephen S. Pawlowski: Exascale Science — The next frontier in High Performance Computing; 24th International Conference on Supercomputing, June 1-4, 2010, Tsukuba, Japan
167. R. Nuez: Google is Going into the Electricity Business; The Huffington Post, JUNE 6, 2010
168. http://www.huffingtonpost.com/ramon-nuez/google-is-going-into-the_b_417035.html
171. J. Rabaey: Low Power Design Essentials; Springer Verlag, 2009
172. L. A. Barroso: The Price of Performance; Oct 18, 2005, ACMqueue
174. W. Blendinger: Post Peak – Abstieg vom Peak Oil; ASPO annual conference, Berlin, 18 Mai 2010
175. K. Aleklett: Post Peak – The Future of the Oil-Production; ASPO annual conf., Berlin, 18 Mai 2010
176. J. S. Gabrielli de Azevedo: Petrobras e o Novo Marco Regulatório; São Paulo, December 1, 2009
177. Stefan Hajek: Dynamik unterschätzt; Wirtschaftswoche 34, August 8, 2010
179. http://www.youtube.com/watch?v=m377Is4tGF0
180. H. Ginsburg: Unterm Wasser liegt die Zukunft; Wirtschaftswoche 23, June 7, 2010
185. Steve Conway: Multicore Processing: Breaking through the Programming Wall - Big challenges lie on the software side in efficiently corralling hardware’s runaway parallelism; Aug 10, 2010, Scientific Computing179
189. Erik Saxe: Power-efficient Software; Feb 2010 Comm_ACM
190. Mary Jane Irwin: Shared Caches in Multicores: The Good, The Bad, and The Ugly”; 37th ISCA, June 19-23, 2010, Saint Malo, France
191. D. Patterson: The Trouble with Multicore; IEEE Spectrum, July 2010
194. Patrick Lysaght (keynote): The Programmable Logic Perspective; FPL-2010, Sep 2010, Milano, Italy
196. N. N. (Sandia National Labs): DARPA selects Sandia National Laboratories to design new supercomputer proto-
279. R. Hecht-Nielsen: Neurocomputing, Addison-Wesley, 1990;
280. C. A. Mead: Analog VLSI and Neural Systems, Addison-Wesley, 1989
283. R. Hartenstein: Steinbuch; Neue Deutsche Biographie (NDB) vol. 25, Duncker & Humblot, Berlin 2010
287. R. Rojas: Neural Networks; Springer Verlag, 1996
289. C. A. Mead: Analog VLSI and Neural Systems, Addison-Wesley, 1989
298. A. A. Gaffar and W. Luk: Accelerating Radiosity Calculations; FCCM 2002
301. E. Chitalwala: Starbridge Solutions to Supercomputing Problems; RSSI Reconf. Syst. Summer Inst., July 11-13, 2005, Urbana-Champaign, IL, USA
Computing: In’t Conf. on Energy-Aware HPC; Sep 16–17, 2010, Hamburg, Germany

350. Sangjin Han, Keon Jang, KyounghSo Park, Sue Moon: PacketShader: a GPU-Accelerated Software Router; SIGCOMM’10, Aug 30–Sep 3, 2010, New Delhi, India

351. Michael Feldman: A GPU on Every Chip; August 19, 2010, HPCwire


353. Lawrence Latif: AMD turns up the heat on Nvidia's GPGPUs; Aug 17, 2010, The Inquirer


360. C. Freeman: Power Reduction for FPGA using Multiple Vdd/Vhh.; Monday April 3, 2006


362. H. Simon (invited presentation): Leibniz-Rechenzentrum, TU Munich, 2009, Garching, Germany


364. Stacey Higginbotham: Watch Out Intel! Marvell to Make ARM-based Server Chips; May, 11, 2010, Gigamon (Business Week)


368. Van Smith: The Coming War: ARM versus x86; April 8, 1020, Van's Hardware Journ


370. Tom Krazit: iPhone 3.0 code hints at future Apple hardware; March 19, 2009, CNET News


375. R. Porter: Evolution on FPGAs for Feature Extraction; Ph.D. thesis; Queensland Un. of Technology Brisbane, Australia, 200136

376. http://www.pooka.lanl.gov/content/pooka/green/Publications_files/imageFPGA.pdf


381. personal communication from authors of chapters of the book


383. Burton Smith (keynote): Reinventing Computing; LACSI Symposium 2006, Santa Fe, NM, USA


387. http://hartenstein.de/innovation/

430. Gerd Michelsen, Jasmin Godemann (Herausgeber): Handbuch der Nachhaltigkeits-kommunikation: Grundlagen und Praxis; 2005, Oekom Verlag