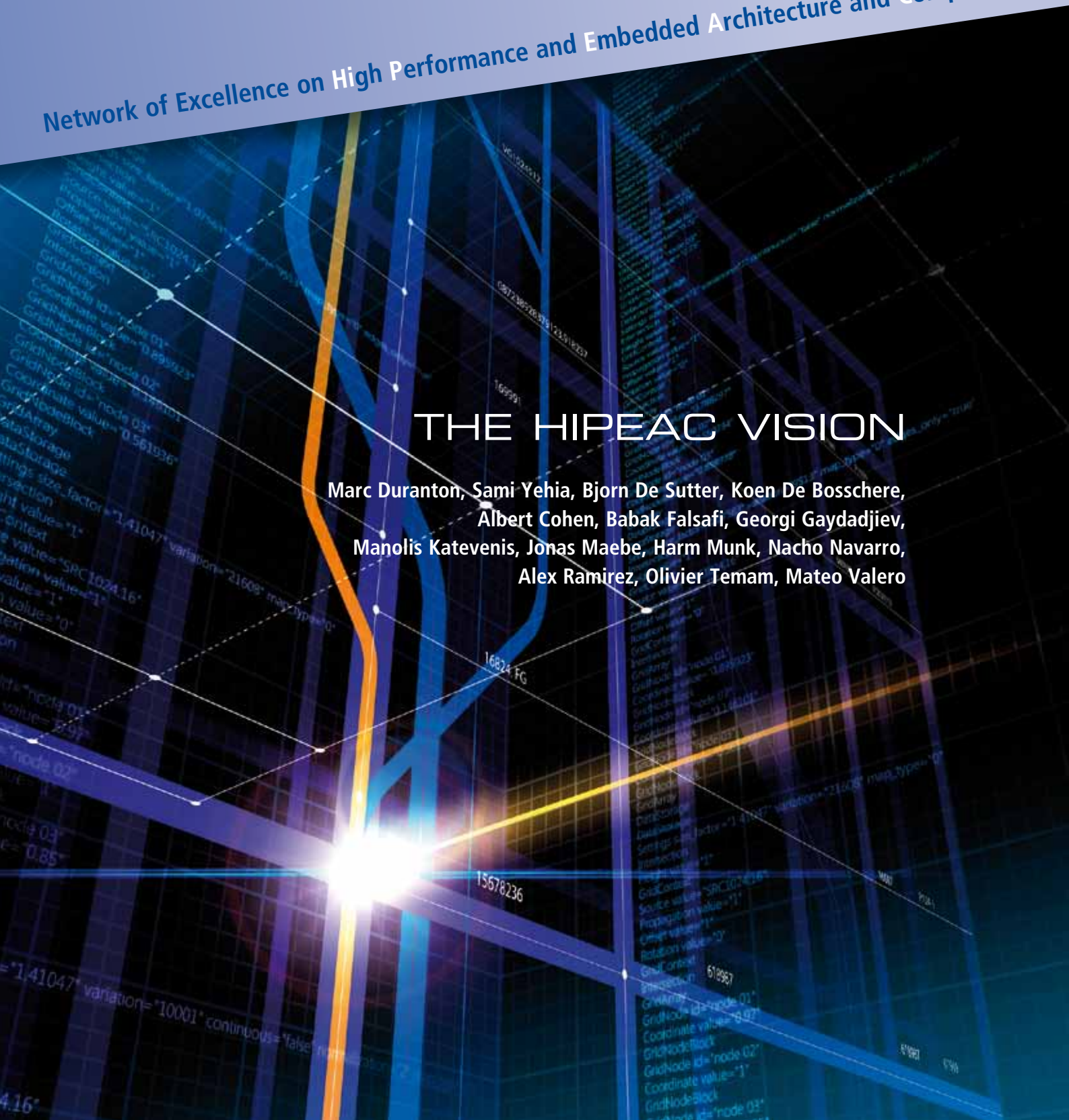
SEVENTH FRAMEWORK PROGRAMME

HiPEAC COMPILATION ARCHITECTURE

Network of Excellence on High Performance and Embedded Architecture and Compilation



THE HIPEAC VISION

Marc Duranton, Sami Yehia, Bjorn De Sutter, Koen De Bosschere,
Albert Cohen, Babak Falsafi, Georgi Gaydadjiev,
Manolis Katevenis, Jonas Maebe, Harm Munk, Nacho Navarro,
Alex Ramirez, Olivier Temam, Mateo Valero

Contents

Project Acronym: HiPEAC
Project full title: High Performance and Embedded Architecture and Compilation
Grant agreement no: ICT- 217068

DELIVERABLE 3.5

The Authors

Marc Duranton, NXP, The Netherlands
Sami Yehia, THALES Research & Technology, France
Bjorn De Sutter, Ghent University, Belgium
Koen De Bosschere, Ghent University, Belgium
Albert Cohen, INRIA Saclay, France
Babak Falsafi, EFPL, Switzerland
Georgi Gaydadjiev, TU Delft, The Netherlands
Manolis Katevenis, Forth, Greece
Jonas Maebe, Ghent University, Belgium
Harm Munk, NXP, The Netherlands
Nacho Navarro, UPC & BCS, Spain
Alex Ramirez, UPC & BCS, Spain,
Olivier Temam, INRIA Saclay, France
Mateo Valero, UPC & BCS, Spain

Executive Summary

Information & Communication Technology had a tremendous impact on everyday life over the past decades. In the future it will undoubtedly remain one of the major technologies for taking on societal challenges shaping Europe, its values, and its global competitiveness. The aim of the HiPEAC vision is to establish a bridge between these societal challenges and major paradigm shifts accompanied by technical challenges that the computing industry needs to tackle.

The HiPEAC vision is based on seven grand challenges facing our society in decades to come, as put forward by the European Commission: energy, transport and mobility, health, aging population, environment, productivity, and safety. In order to address these challenges, several technologies and applications will have to be pushed beyond their existing state-of-the-art, or even be reinvented completely.

Information Technology application trends and innovative applications evolve in parallel with societal challenges. The trends include the seemingly unstoppable demand for ubiquitous access, personalized services, and high-quality virtual reality. At the same time, we observe the decoupling of computing and storage together with an exponential growth of massive data processing centers. In terms of applications domestic robots, autonomous transportation vehicles, computational science, aerospace and avionics, smart camera networks, realistic games, telepresence systems, and the Human++ are all examples of solutions that aim to address future societal challenges.

The development of these applications is influenced by business trends such as cost pressure, restructuring of the industry, service-oriented business models and offloading the customer's hardware via "cloud computing". Other important aspects are the converging of functionality on devices of various sizes and shapes, and collaborative "free" development.

However, several technological obstacles block the path the computing industry has to take in order for these applications to become drivers of the 21st century. The following statements summarize major obstacles our industry needs to overcome:

1. Hardware has become more flexible than software;
2. Power defines performance;
3. Communication defines performance;
4. Application-specific integrated circuits (ASIC) are becoming unaffordable;
5. Worst-case design for ASICs leads to bankruptcy;
6. Systems will have to rely on unreliable components;
7. Time is relevant;
8. Computing systems are continuously under attack;
9. Parallelism seems to be too complex for humans;
10. One day, Moore's law will end.

These technological roadblocks or constraints lead to technical challenges that can be summarized as improvements in seven key areas: performance, performance/€ and performance/Watt/€, power and energy, managing system complexity, security, reliability, and timing predictability.

The HiPEAC vision explains how the HiPEAC community can work on these challenges.

The central creed of the HiPEAC vision is: keep it simple for humans, and let the computer do the hard work. This leads to a world in which end users do not have to worry about platform technicalities, where 90% of the programmers are only concerned with programming productivity and can use the most appropriate domain-specific languages for application development, and where only 10% of the trained computer scientists have to worry about efficiency and performance.

Similarly, a majority of hardware developers will use a component-based hardware design approach by composing functional blocks with standardized interfaces, some of them possibly automatically generated. Such blocks include various processor and memory organizations, domain-specific accelerators and flexible low-cost interconnects. Analogous to the software community, a small group of architects will design and optimize these basic components. Systems built from these components will be heterogeneous for performance and power efficiency reasons.

Finally, system engineers will be able to depend on a virtualization layer between software and physical hardware, helping them to transparently combine legacy software with heterogeneous and quickly changing hardware.

In tandem with these human efforts, computers will do the hard work of (i) exploring the design space in search of an appropriate system architecture; of (ii) generating that system architecture automatically with electronic design automation tools; of (iii) automatically parallelizing the applications written in domain-specific languages; and of (iv) dynamically adapting the hardware and software to varying environmental conditions such as temperature, varying workloads, and dynamic faults. Systems will monitor their operation at run time in order to repair and heal themselves where possible.

The HiPEAC vision also reminds us of the fact that one day the past and current technology scaling trends will come to an end, and when that day arrives we should be ready to continue advancing the computing systems domain in other ways. Therefore our vision suggests the exploration of emerging alternatives to traditional CMOS technology and novel system architectures based on them.

Finally this document presents a Strengths, Weaknesses, Opportunities, and Threats (SWOT) analysis of computing systems in Europe, and makes six recommendations for research objectives that will help to bring to fruition the HiPEAC vision. These are:

1. Design of optimized components;
2. Electronic Design Automation (EDA);
3. Design Space Exploration (DSE);
4. Concurrent programming models and auto-parallelization;
5. Self-adaptive systems;
6. Virtualization.

This vision document has been created by and for the HiPEAC community. Furthermore it is based on traditional European strengths in embedded systems. It offers a number of directions in which European computing systems research can generate impact on the computing systems industry in Europe.

Target audience of this document

This HiPEAC vision is intended for all stakeholders in the computing industry, the European Commission, public authorities and all research actors in academia and industry in the fields of embedded systems, computer architecture and compilers.

The executive summary of this document targets decision makers and summarizes the major factors and trends that shape evolutions in the HiPEAC areas. It describes the societal and economic challenges ahead that affect or can be affected by the computing industry. It is essential for all decision makers to understand the implications of the different paradigm shifts in the field, including multi-core processors, parallelism, increasing complexity, and mobile convergence, and how they relate to the upcoming challenges and future application constraints and requirements.

The more detailed trends and vision sections of this document target all industrials, academics and political actors, and in general all readers interested in the subject. The goal of these sections is to detail the challenges facing society and this particular sector of industry, and to map these challenges to solutions in terms of emerging key developments.

The last part of this document consists of recommendations for realizing the objectives of the vision, both for the HiPEAC community and for Europe. It therefore focuses on the gaps between the current developments and the directions proposed by the vision section. This part is mainly targeted at policy makers and the whole HiPEAC community.

Introduction

European Information & Communication Technology (ICT) research and development helped to solve many societal challenges by providing ever more computing power together with new applications that exploited these increasing processing capabilities. Numerous examples of the profound impact the computing industry had can be seen in medical imaging, chemical modeling for the development of new drugs, the Internet, business process automation, mobile communication, computer-aided design, computer-aided manufacturing, climate simulation and weather prediction, automotive safety, and many more.

Advances in these areas were only possible because of the exponential growth in computing performance and power efficiency over the last decades. By comparison, if the aviation industry had made the same progress between 1982 and 2008, we would now fly from Brussels to New York in less than a second. Unfortunately, several evolutions are now threatening to bring an end to the exponential growth path of the computer industry.

Until the early 90s, the computer industry's progress was mainly driven by a steadily improving process technology. It enabled significant speed as well as area improvements and on-die transistor budget growth at manageable power and power density costs. As a result, easily programmable uniprocessor architectures and the associated sequential execution model utilized by applications dominated the vast majority of the semiconductor industry.

One notable exception was the embedded systems domain, where the combination of multiple computing engines in consumer electronic devices was already common practice. Another exception was the high-performance computing domain, where large scale parallel processing made use of dedicated and costly supercomputer centers. Of course, both of these domains also enjoyed the advantages offered by an improving process technology.

From the late 90s on, however, two significant evolutions led to a major paradigm shift in the computing industry. In the first place the relative improvements resulting from shrinking process technology became gradually smaller, and fundamental laws of physics applicable to process technology started to constrain the frequency increases and indicate that any future increase in frequency or transistor density will necessarily result in prohibitive power consumption and power density.

Secondly, consumer electronic markets, and therefore industries, started to converge. Digital watches and pagers evolved into powerful personal digital assistants (PDA) and smartphones, and desktop and laptop computers were recently reduced to netbooks. The resulting devices demand ever more computational capabilities at decreasing power budgets and within stricter thermal constraints. In pursuit of the continued exponential performance increase that the markets expect, these conflicting trends led all major processor designers to embrace the traditionally embedded paradigm of multi-core devices and special-purpose computational engines for general-purpose computing platforms.

In the past decade, industrial developments were driven by mobile applications such as cell phones and by connectivity to the Internet. These were the applications that appealed the most to the general public and fueled the growth of the ICT industry. In the future, however, we expect to see less and less of such "killer applications": ICT will become as common in everyday life as, e.g., electrical energy and kitchen appliances. Today, most people already spend a lot of time with their PDAs, MP3-players and smartphones. This will intensify competition on a global scale and will drive a trend towards specialization. Even though globalization is supposed to break down borders, we expect to see clear demographic divisions, each with its own area of expertise. Europe has to capitalize on its own strengths in this global economy.

Today, Europe is facing many new challenges with respect to energy consumption, mobility, health, aging population, environment, productivity, safety, and, more recently, the worldwide economic crisis. The role of the ICT industry in addressing these challenges is as crucial as it was ever before. The aforementioned trends have, however, made this role much more challenging to fulfill. The two major trends, multi-core parallelism and mobile convergence, have pushed the semiconductor industry to revise several previously established research areas and priorities.

In particular, parallelism and power dissipation have to become first class citizens in the design flow and design tools, from the application level down to the hardware. This in turn requires that we completely rethink current design tools and methods, especially in the light of the ever-increasing complexity of devices. Additionally, these concerns now both span the entire computing spectrum, from the mobile segment up to the data centers.

The challenges arising from this paradigm shift, along with others such as reliability and the design space explosion, are exacerbated by the increasing industrial and application requirements. We nevertheless see them as opportunities for the European industry, especially given our historical leadership in the domains of embedded systems and low power electronics. However, to take advantage of these opportunities in the decade ahead, we require a vision to drive actions.

The HiPEAC Network of Excellence groups the leading European industrial enterprises and academic institutions in the domain of high-performance and embedded architectures and compilers. The network has 348 members affiliated to 74 leading European universities and 37 multinational and European companies. This group of experts is therefore ideally positioned to identify the challenges and to mobilize the efforts required to tackle them.

The goal of this document is to discern the major societal challenges together with technical constraints as well as application and business trends, in order to relate them to technical challenges in computing systems. The vision then explains how to tackle the technical challenges in a global framework. This framework then leads to concrete recommendations on research areas where more effort is required.

Approach taken

The HiPEAC community produced a first technical roadmap document in 2007. The current document complements it by a more global integrated vision, taking into account societal challenges, business trends, application trends and technological constraints. This activity was kicked off during the HiPEAC 2008 conference in January 2008.

It was followed by a survey that was sent to all HiPEAC clusters and task forces. The clusters discussed the survey at their spring cluster meeting, and produced their report by the end of June 2008.

The 13 HiPEAC clusters and task forces are:

- Multi-core architecture;
- Programming models and operating systems;
- Adaptive compilation;
- Interconnects;
- Reconfigurable computing;
- Design methodology and tools;
- Binary translation and virtualization;
- Simulation platform;
- Compilation platform;
- Task force low power;
- Task force applications;
- Task force reliability and availability;
- Task force education and training.

During the ACACES 2008 summer school, the industrial participants and the teachers of the school held a brainstorming session based on this report. This material was further supplemented by the personal vision of a number of HiPEAC members. This resulted in about 100 pages of raw material.

This material was analyzed, restructured, complemented and shaped during several workshops and teleconferences, and through numerous email exchanges and updates of the document by members of the HiPEAC community, under the supervision of an editorial board.

The ACACES Summer School 2009 gave the opportunity to the industrial participants and the teachers to brainstorm about the Strengths, Weaknesses, Opportunities and Threats (SWOT) that Europe is facing in the domain of Information & Communication Technology. The results were analyzed, complemented and included in the recommendations.

1. Trends & Challenges

The HiPEAC vision builds on several foundations in the form of challenges, trends, and constraints. The first foundation are the European grand societal challenges.

Secondly, we look into application trends and some future applications that can help in meeting these societal challenges.

Both of these foundations are situated outside the core competences of the HiPEAC community, but they help in illustrating the larger context in which HiPEAC operates.

The third foundation are general business trends in the computing systems industry and their consequences.

Finally, we consider technological evolutions and constraints that pose challenges and limitations with which our community has to deal, leading to a list of core technical challenges.

Societal Challenges for ICT

The main purpose of Information and Communication Technologies (ICT) is to make the world a better place to live in for everyone. For decades to come, we consider the following seven essential societal grand challenges [ISTAG], which have deep implications for ICT.

Energy

Our society is using more energy than ever before, with the majority of our current energy sources being non-renewable. Moreover, their use has a significant and detrimental impact on the environment. Solving the energy challenge depends on a two-pronged approach. On the one hand, we need research into safe, sustainable alternatives to our current energy sources. On the other hand, we also have to significantly reduce our overall energy consumption.

Currently computing is estimated to consume the same amount of energy as civil aviation, which is about 2% of the global energy consumption. This energy consumption corresponds to a production of, for example, 60g CO2 per hour a desktop computer is turned on. Along similar lines, a single Google query is said to produce 7g of CO2. Making computing itself more energy-efficient will therefore already contribute to the energy challenge.

Even though computers consume a lot of power, in particular in the data centers, some reports [Wehner2008] state that they globally contribute to energy saving (up to 4x their CO2 emission) due to on-line media, e-commerce, video conferencing and teleworking. Teleworking reduces physical transport, and therefore energy. Similarly, videoconferencing reduces business trips. E-commerce also has a significant impact. Electronic forms and administrative documents reduce the volume of postal mail.

An even greater indirect impact can be expected from energy optimizations in other aspects of life and economy, by introducing electronic alternatives for other energy-consuming physical activities, and by enabling the optimization of energy-hungry processes of all sorts.

Transport and Mobility

Modern society critically depends on inexpensive, safe and fast modes of transportation. In many industrialized areas of the world mobility is a real nightmare: it is an environmental hazard, the average speed is very low, and it kills thousands of people every year.

ICT can help with solving the mobility challenge by optimizing and controlling traffic flows, by making them safer through more active safety features, or by avoiding them altogether, e.g., through the creation of virtual meeting places.

Health

The use of advanced technologies is essential to further improve health care. There is a great need for devices that monitor the health and assist healing processes, for equipment to effectively identify diseases in an early stage, and for advanced research into new cures and improving existing treatments.

ICT is indispensable in this process, e.g., by speeding up the design of new drugs such as personalized drugs, by enabling personal genome mapping, by controlling global pandemics and by enabling economically viable health monitoring.

Aging population

Thanks to advances in health care, life expectancy has increased considerably over the last century, and continues to do so even today. As a result, the need for health care and independent living support, such as household robots and advanced home automation, is growing significantly. ICT is at the heart of progress in these areas.

Environment

The modern way of living, combined with the size of the world population, creates an ecological footprint that is larger than what the Earth can sustain. Since it is unlikely that the first world population will want to give up their living standard or that the world's population will soon shrink spontaneously, we have to find ways to reduce the ecological footprint of humans.

ICT can assist in protecting the environment by controlling and optimizing our impact, for example by using camera networks to monitor crops and to apply pesticides only on those specific plants that need them, by continuously monitoring environmental parameters, by optimizing the efficiency of engines, by reducing or optimizing the traffic, by enabling faster research into more environment-friendly plastics, and in numerous other ways.

Productivity

In order to produce more goods at a lower price or in order to produce them more quickly, economies have to continuously improve the productivity of their industrial and non-industrial processes. In doing so, they can also remain at the forefront of global competition. ICT enables productivity enhancements in all sectors of the economy and will continue to do so in the foreseeable future.

Safety

Many safety-critical systems are or will be controlled by information systems. Creating such systems requires effective dealing with failing components, with timing constraints and with the correctness of functional specifications at design time.

Advancements in ICT also enable society at large to protect itself in an ever more connected world, by empowering individuals to better protect their privacy and personal life from incursions, and by providing law enforcement with sophisticated analysis and forensic means. The same applies to national defense.

Application trends

The continued high-speed evolution of ICT enables new applications and helps creating new business opportunities. One of the key aspects of these future applications, from a user perspective, is the way in which the user interacts with computing systems. Essentially, the interfaces with the computers become richer and much more implicit, in the sense that the user is often not aware of the fact that he is interacting with a computer. This is known as "the disappearing computer" [Streit2005].

This second part of our vision lists a number of application trends that we envision for the next decade. This list is by no means exhaustive. Its main purpose is to establish a list of technical application requirements for future applications. We start with an outline of potential future ICT trends continued with a list of innovative future applications.

Future ICT trends

We envision at least the following major trends in the use of ICT during the following decade.

Ubiquitous access

Users want to have ubiquitous access to all of their data, both personal and professional. For example, music, video, blogs, documents, and messages must follow the users in their home from room to room and on the move in the car, at work, or when visiting friends. The way and user interface through which this data is accessed may however differ depending on the situation, and so may the devices used. These include, but are not limited to, desktop computers, laptops, netbooks, PDAs, cell phones, smart picture frames, Internet radios, and connected TV sets. Since these different platforms may be built using completely dissimilar technologies, such as different processors, operating systems, or applications, it is important to agree on high quality standards that will allow for information interchange and synchronization between all these devices.

Personalized services

We expect services to become more and more personalized, both in private and professional life. Our preferences will be taken into account when accessing remote web-based services. Other examples are personalized traffic advice, search engines that take our preferences and geographical location into account, music and video sources presenting media fitting our personal taste and in the format that best suits our mobile video device, and usability adaptations for disabled people.

Personalized video content distribution is another case of ever increasing importance. Video streams can be adapted to the viewer's point of view, to his or her personal taste, to a custom angle in case of a multi-camera recording, to the viewer's location, to the image quality of the display, or to his or her consumer profile with respect to the advertisements shown around a sports field.

Delocalized computing and storage

As explained in the previous sections, users want to access those personalized services everywhere and through a large diversity of hardware clients. Users thus request services that require access to both private and public data, but they are not interested to know from where the data is fetched and where the computations are performed. Quality of experience is the only criterion that counts. YouTube, Google GMail, Flickr and Second Life are good examples of this evolution. The user does not know the physical location of the data and computations anymore, which may be data centers, within access networks, client devices or still other locations.

Massive data processing systems

We envision that three important types of data processing systems will coexist:

- Centralized cloud computing is a natural evolution of current data centers and supercomputers. The computing and storage resources belong to companies that sell these services, or trade them for information, including private information such as a profile for advertisements. However, mounting energy-related concerns require investigating the use of "greener data centers". One promising approach, in which Europe can lead, is using large numbers of efficient embedded cores, as these may provide better performance/watt/ than traditional microprocessors [Asanovic2006, Katz2009].
- Peer-to-Peer (P2P) computing is a more distributed form of cloud computing, where most of the computing elements and storage belongs to individuals as opposed to large companies. Resources are located throughout a large network so as to distribute the load as evenly as possible. This model is very well suited to optimally exploit network bandwidth, and can also be used for harvesting unused computation cycles and storage space. It continues the decentralization trends initiated by the transition from centralized telephone switches to the Internet, but at a logical rather than at a physical level. Some companies already use this technique for TV distribution, in order to avoid overloading single servers and network connections.
- Personal computing follows from ICT trends that provide end users with increasingly more storage capacity, network bandwidth, and computation power in their personal devices and at home. These come in the form of large, networked hard drives, fiber-to-the-home, and massively parallel graphical processing units (GPUs). Hence many people may simply use their "personal supercomputers", accessible from anywhere, rather than some form of cloud computing. We might even envision a future where people convert their excess photovoltaic or other power into computing cycles instead of selling it to the power grid, and then sell these cycles as computation resources, while using the dissipated power to heat their houses.

High-quality virtual reality

In the near future, graphic processors will be able to render photorealistic views, even of people, in real time [CEATEC2008]. The latest generations of GPUs can already render virtual actors with almost photorealistic quality in real time, tracking the movements as captured by a webcam. These avatars, together with virtual actors, will enable new high-quality virtual reality (HQVR) applications, new ways to create content, and new forms of expression. This trend has already started in Japan, for example in the form of software that enables the creation of music videos with a virtual singer [Vocaloid].

It is obvious that these techniques will also allow new ways of communication, for example by reducing the need to travel for physical meetings.

Intelligent sensing

Many unmanned systems, security systems, robots, and monitoring devices are limited by their ability to sense, model or analyze their surrounding environment. Adding more intelligence to sensors and allowing embedded systems to autonomously analyze and react to surrounding events in real time, will enable building more services, comfort and secure systems and will minimize human risks in situations requiring dangerous manipulations in hard-to-access or hostile environments. As a result, we will see the emergence of "smart" cities, buildings, and homes. In the future we also envision advanced sensor networks or so-called "smart dusts", where clouds of tiny sensors will simply be dropped in locations of interest to perform a variety of monitoring and sensing applications.

Less automated, but at least equally important, are tele-manipulators or robots that enable remote manual tasks. Combined with high-quality haptic feedback, it opens the path to, e.g., telesurgery.

High-performance real-time embedded computing

Embedded computing has long ago outgrown simple microcontrollers and dedicated systems. Many embedded systems already employ high-performance multi-core systems, mostly in the consumer electronics domain (e.g. signal processing, multimedia).

Future control applications will continue this trend not just for typical consumer functionality, but also for safety and security applications. They will do so, for example, by performing complex analyses on data gathered with intelligent sensors, and by initiating appropriate responses to dangerous phenomena. Application domains for such systems are the automotive domain, as well as the aerospace and avionics domains. Future avionic systems will be equipped with sophisticated on-board radar systems, collision-detection, more intelligent navigation and mission control systems, and intelligent communication to better assist the pilots in difficult flight situations, and thus to increase safety. Manufacturing technology will also increasingly need high-end vision analysis and high-speed robot control.

In all cases, high performance and real time requirements are combined with requirements to low power, low temperature, high dependability, and low cost.

Innovative example applications

The above trends manifest themselves in a number of concrete applications that clearly contribute to the societal challenges.

Domestic robot

An obvious application of the domestic robot would be taking care of routine housekeeping tasks. In case of elderly or disabled people, the domestic robot could even enable them to live independently, thereby increasing the availability of assisted living. A humanoid form seems to be the most appropriate for smooth integration into current houses without drastic changes in their structure or organization. This poses major challenges for sensors, processing and interfacing. It also requires the robots to run several radically different types of demanding computations, such as artificial intelligence and video image processing, many of which need to be performed in real time to guarantee safe operation.

Furthermore, the robots will have to continuously adapt to changes in their operating environment and the tasks at hand. For example, depending on the time of day and the room in which they operate, the lighting will be different, as will the tasks they have to carry out and potentially even the users they have to assist. Furthermore, the reliability and autonomy of the robots needs to be guaranteed, for example when for some reason the power socket cannot be reached or when there is a power outage. In that case, non-essential tasks such as house cleaning can be disabled to save energy for life-saving tasks that must remain available, such as administering drugs or food, and calling for aid.

As such, domestic robots can clearly play an important role in dealing with the aging population. The domestic robot is currently a priority for the Japanese government [Bekey2008] and we expect that a strong social demand for domestic robots will be a solid driver for computing systems research and business in the future.

The car of the future

Cars can be equipped with autopilots. In order to drive safely and quickly to their destination, cars can stay in touch with a central traffic control system that provides personalized traffic information for each car, such that, e.g., not all cars going from A to B will take the same route in case of congestion. Cars can also contact neighboring cars to negotiate local traffic decisions like who yields at a crossing. Autonomous vehicles can also be used by children, disabled people, the elderly or people that are otherwise not allowed to drive a car, or that are not willing to drive themselves because, e.g., they want to work/relax while traveling. Furthermore, autonomous vehicles can be used unmanned to transport goods.

Advanced Driver Assistance Systems (ADAS) that combine high-end sensors enable a new generation of active safety systems that can dramatically improve the safety of pedestrians. ADAS systems require extreme computation performance at low power and, at the same time, must adhere to high safety standards. Stereovision, sensor fusion, reliable object recognition and motion detection in complex scenes are just a few of the most demanding applications that can help to reduce the number of accidents. Similar requirements are found in aerospace safety systems.

Clearly the automation and optimization of traffic on our roads can help in saving energy, reducing air pollution, increasing productivity, and improving safety.

Telepresence

A killer application for HQVR could be realistic telepresence, creating the impression of being physically present in another place. This could be achieved with high-resolution displays, possibly in 3D, with multi-view camera systems, and with low-latency connections. For example, at each participating site of a video-conference, a circle of participants around a meeting table can consist of some real participants and of a set of displays that show the remote participants from the point of view of the in situ participants. This way, participant A would see two participants B & C that participate from two different physical locations but are seated adjacent to each other in the virtual meeting as if they were facing each other when they have a conversation. At the same time, participants B & C will effectively face each other on their respective displays.

Such an application requires 3D modeling of all in situ participants, 3D rendering of all remote participants at all sites, and a communication and management infrastructure that manages the virtual world: who is sitting where, what background images are transmitted, the amount of detail to be transmitted, etc.

Typical applications of such systems are virtual meetings, advanced interactive simulators, virtual family gatherings, virtual travel, gaming, telesurgery, etc. In the future, these applications might be combined with, e.g., automated translation between different languages spoken during a telepresence session.

While relatively simple instances of such systems are currently designed and researched, many possible features and implementation options remain to be explored. For example, where will most of the processing take place? In centralized servers feeding images to thin set-top boxes? Or will fat set-top boxes at each participating site perform this task? What will the related business model of such systems look like? Are the participants displayed in a virtual environment or in a realistic environment? What happens if a participant stands up and walks out? Will he or she disappear in between two displays of the virtual meeting? How will the systems handle multiple

participants at the same physical site? With multiple multi-view cameras? With multiple display circles?
Telepresence applications clearly contribute to overcome the challenges of mobility, aging population, and productivity. By saving on physical transportation of the participants, telepresence can also reduce energy consumption [Cisco].

Aerospace and avionics
Aerospace and avionics systems will undergo a continued evolution towards tighter integration of electronics to increase safety and comfort. Future systems, both in the air and on the ground, will be equipped with sophisticated on-board radar systems, collision-detection, more intelligent navigation and mission control systems, and intelligent communication to better assist pilots in difficult flight situations in order to increase safety. Highly parallel on-board real-time computer systems will enable new classes of flight control systems that further increase safety in critical situations.

While on the one hand this is a continuation of ongoing automation in the aerospace and avionics industry, on the other hand it ushers in a new era in which many more decisions will be taken while airborne instead of before takeoff. This will lead to less strict a priori constraints, which will in turn lead to more efficient routes and procedures. As such, these new applications will help with the challenges of safety, the environment, and mobility.

Future space missions will be equipped with ever more complex on-board experiments and high-precision measurement equipment. Satellite-based systems will be getting more sophisticated sensors and communications systems, enabling new application domains, such as better surveillance and mobile terrestrial broadband communications.

To make this evolution economically viable, all devices that are launched should behave very reliably over a long period of time and should be light to limit launching costs. Achieving both goals will require new experimentation and application devices to include more reliability-enhancing features. By implementing those features in computing electronics themselves by means of adaptability and redundancy instead of using mechanical shields, we can save weight and thereby reduce launch costs. Furthermore, to increase the lifetime of devices and to optimize their use during their lifetime, their processing capabilities will become more flexible, enabling the uploading of new or updated applications.

Human++
A fascinating example of advanced intelligent sensing could be the augmented human, or the Human++. More and more, implants and body extensions will overcome limitations of the human body. For example, complex micro-electronic implants will restore senses for disabled people, as in case of cochlear implants or bionic eyes. Other implants will control internal body functions, for example by releasing hormones such as insulin precisely when they are needed, or by detecting epileptic seizures and releasing medicine in time to avoid the most dangerous stages of a seizure.
Exoskeletons will enable people to work more productively, for example by offering them finer gesture control. In order to steer the actuators in such exoskeletons, electronics will be connected to the human brain and nervous systems through interfaces that require no conscious interaction by the user [Velliste2008]. Augmented reality devices such as glasses and hearing aids, or recording and analyzing devices [GC3], can also help healthy people in their daily life.
Human++ can clearly help in meeting the challenges relating to health and the aging population. It can also help to improve productivity.

Computational science
Computational science is also called the third mode of science (in silico) [GC3]. It creates detailed mathematical models that simulate physical phenomena such as chemical reactions, seismic waves, nuclear reactions, and the behavior of biological systems, people and even financial markets. A common characteristic of all these applications is that the precision is mostly limited by the available computing power. More computing power allows using more detailed models leading to more precise results. E.g. in global climate modeling, results are more precise if not only the atmosphere and the oceans, but also the rainforests, deserts and cities are modeled. Computing all these coupled models, however, requires an insatiable amount of floating-point computing power.

Today's supercomputers offer petaflop-scale sustained performance but this is not yet sufficient to run the most advanced models in different disciplines, nor does it allow us to run the algorithms at the desired granularity. The next challenge is to develop exascale computing with exaflop-scale performance. Exascale computing differs from the cloud in the sense that exascale computing typically involves very large parallel applications, whereas the cloud typically refers to running many (often smaller) applications in parallel. Both types of computing will have to be supported by appropriate software and hardware, although large fractions of that software and hardware should be common.

The impact of computational science is huge. It enables the development of personalized drugs, limits the number of experiments on animals, allows for accurate long term weather predictions, helps us to better understand climate change, and it might pave the way to anticipate health care based on detailed DNA screening. Computers for computational science have always been at the forefront of computing in the sense that most high-performance techniques were first developed for supercomputers before they became available in commodity computing (vector processing, high speed interconnects, parallel and distributed processing).

Computational science definitely helps in solving the energy and health challenges.

Smart camera networks

Right now, camera networks involving dozens or even hundreds of cameras are being installed for improving security and safety in public spaces and buildings and for monitoring traffic. Companies are already envisaging "general purpose" home camera networks that could be used for a variety of purposes such as elderly care, home automation and of course security and safety. At the European level, there is a strong push to introduce cameras and other sensors into cars, for improving traffic safety through assisted driving. Finally, camera technology is introduced in a wide range of specialized applications, such as precision agriculture, where crops are monitored to limit the use of pesticides.

In many of these emerging applications, it is impossible for a human to inspect or interpret all available video streams. Instead, in the future computers will analyze the streams and present only relevant information to the operator or take appropriate actions autonomously.

When camera networks grow to hundreds of cameras, the classical paradigm of processing video streams on centralized dedicated servers will break down because the communication and processing bandwidth does not scale sufficiently with the size of the camera networks. Smart cameras cooperating in so-called distributed camera systems are the emerging solution to these problems. They analyze the video data and send condensed meta-data streams to servers and to each other, possibly along with a selection of useful video streams. This solution scales better because each new camera adds additional distributed processing power to the network. However, several challenges remain, e.g., the development of mechanisms for privacy protection, as well as the development of hardware/software platforms that enable both productive programming and power-efficient execution.

The latter is particularly important for wireless cameras that offer many advantages such as easier ad hoc installation.

Just like video processing in future cars and in future domestic robots will have to adapt to changing circumstances, so will the software that analyses video streams. An example is when the operation mode of a camera network monitoring a large crowd has to switch from statistical crowd motion detection to following individual suspects.

Clearly smart camera networks can help with societal challenges, including safety, productivity and the environment.

Realistic games

According to the European Software Association [ESA], the computer and video game industry's revenue topped $22 billion in 2008. Gaming is a quickly growing industry and it is currently a huge driver for innovations in computing systems. GPUs now belong to the most powerful computing engines, already taking full advantage of the many-core roadmap. Gaming will definitely be one of the future "killer applications" for high-end multi-core processors, and we expect gaming to remain one of the driving forces for our industry.

It can be expected that at least some games will bridge the gap between virtual worlds and the real world. For example, at some point a player might be playing in front of his PC display, but at another point in the same game he might go searching for other players in this hometown, continuing some mode of the game on his PDA with Bluetooth and GPS support. Such games will need to support a very wide range of devices. This contrasts with existing games for which a large fraction of the implementation effort is spent on implementing device-specific features and optimizations.

Gaming does not directly address one of the societal challenges, but together with the entertainment industry it contributes to the cultural evolution of our society. It also helps people to enjoy their leisure time and improves their well-being.

Business trends

Current business trends, independent of the economic downturn, have a deep impact on ICT. The economic downturn only speeds up those trends, deepening the short-term and middle-term impact. This section describes the most important recent business trends in ICT.

Industry de-verticalization

The semiconductor industry is slowly changing from a high-tech into a commodity industry: chips and circuits are everywhere and need to be low cost. This will have wide raning implications, and what happened to the steel industry could repeat itself for the silicon industry. We observe industrial restructuring or "de-verticalization": instead of having companies controlling the complete product value chain, the trend is to split big conglomerates into smaller companies, each of them more specialized in their competence domain. For example, big companies are spinning off their semiconductor divisions, and the semiconductor divisions spin off the IP creation, integration and foundry, thus becoming "fabless" or "fablight". Examples are Siemens, Philips, and, in the past, Thomson.

Consolidation by merging and acquisition also allows companies to gain critical mass in their competence area, sometimes leading to quasi monopolies. One of the reasons is cost pressure: only the leader or the second in a market can really break even.

A horizontal market implies more exchanges between companies and more cost pressure for each of them. An ecosystem is required to come to a product. Sharing of IP, tools, software and foundries are driving an economy of scale. Standardization and cooperation on defining common interfaces is mandatory, such that different pieces can be integrated smoothly when building a final product.

At least two side effects can result from this approach: higher cost pressure offsets the advantages of the economy of scale, and final products are less optimized. Both side effects are caused by the same fundamental reason: each design level in a system is optimized to maximize benefits for all of its target uses, but not for any particular end product. In other words, all design levels are optimized locally rather than globally. In an integrated approach, not applying a local optimization to an isolated level or applying that optimization differently could lead to a better global optimization. Furthermore, interoperability and communication add extra layers, and therefore costs. Those costs can be of a financial nature, or they may come in the form of lower performance or lower power efficiency.

More than Moore

Moore's Law has driven the semiconductor industry for decades, resulting in extremely fast processors, huge memory sizes and increasing communication bandwidth. During those decades, ever more demanding applications exploited these growing resources almost as soon as they arrived on the market. These applications were developed to do so because the International Technology Roadmap for Semiconductors (ITRS) and Moore's Law told them when those resources would become available. So during the last decades, computing systems were designed that reflected the CMOS technology push resulting from Moore's Law, as well as the application pull from ever more demanding applications. A major paradigm shift is taking place now, however, both in the technology push and in the application pull. The result of this paradigm shift has been called the "More than Moore" era by many authors; see for example [MtM].

From the point of view of the technology push, two observations have to be made. First of all, the cost levels for system-on-chip development in advanced CMOS technology are going through the roof, for reasons described in more detail in later sections. Secondly, the continuing miniaturization will have to end Moore's Law one day in the not so distant future.

From the application pull perspective, it has become clear that consumers and society have by and large lost interest in new generations of applications and devices that only feature more computational power than their previous generation. For improving the consumer experience, and for solving the societal challenges, radically new devices are needed that are more closely integrated in every-day life, and these require sensors, mechatronics, analog- and mixed-signal electronics, ultra-low-power or high-voltage technologies to be integrated with CMOS technology.

Devices that embed multiple technologies are instances of the "More than Moore" approach: combining generic CMOS-technology with new technologies for building more innovative, dedicated, smarter and customer-tailored solutions. This new era of added-value systems will certainly trigger innovation, including new methodologies for architecting, modeling, designing, characterizing, and collaborating between the domains required for the various technologies combined in a "More than Moore" system.

The "More Moore" race towards ever-larger numbers of transistors per chip and the "More than Moore" trend to integrate multiple technologies on silicon are complementary to achieve common goals such as application-driven solutions, better system integration, cost optimization, and time to market. Some companies will continue to follow the "More Moore" approach, while others will shift towards the "More than Moore" approach. This will drive industry into a direction of more diversity and wider ecosystems.

Less is Moore

Together with the "More than Moore" trend, we observe another trend fueled by Moore's law: people no longer only want more features and better performance, but are increasingly interested in devices with the same performance level at a lower price. This is particularly true for personal computers. The sudden boom of netbooks, based on low cost and lower performance processors such as Intel Atom or ARM processors, is an example of this new direction. People notice that these devices offer enough performance for everyday tasks such as editing documents, listening to music and watching movies on the go.

The limited processor performance also reduces power consumption and therefore improves mobility. For example, netbooks have up to 12h autonomy, much better than laptops. Due to their lower price, they also open new markets, allowing better access to ICT for developing countries as was tried in the One Laptop Per Child project.

This trend also has an impact on software, as it now needs to be optimized to run smoothly on devices with less hardware resources. Contrary to previous versions, new operating system releases seem to be less compute-intensive. This can be seen in comparing the minimum requirements of Microsoft's Windows 7 to those of Microsoft's Vista, and Apple's Snow Leopard OS also claims improvements in the OS internals rather than new features. This trend extended the lifetime of Windows XP, and gave rise to a wider introduction of Linux on consumer notebooks.

This trend is also leading to computers specifically designed to have extreme low power consumption. The appearance of ARM-based netbooks on the market demonstrates that even the once sacred ISA compatibility is sacrificed now. This creates excellent opportunities for Europe.

Convergence

Another business trend is convergence: TVs and set-top-boxes share more and more functionality with PCs and even have access to the Internet and Web 2.0 content. Telecom companies are proposing IP access to their customers, and broadcast companies are challenged by IP providers who deliver TV programs over IP (ADSL). End users want to restrict the number of different providers, and expect to have voice, data, TV and movies accessible both on their wired and wireless devices.

When using devices compliant with Internet standards, TV viewers can now have full access to all of its contents and to cloud computing. TV shows can be recorded on a Network Attached Storage or NAS device, or be displayed from YouTube. The TV and other devices such as mobile phones, can also access all the user's pictures and music.

The convergence mainly relies on common standards and protocols such as DLNA, Web standards, Web 2.0, and scripting languages, and not so much on closed proprietary software. As a result, the hardware platform on which applications run is becoming irrelevant: commonly used ISAs like x86 are not compulsory anymore, so other ISAs like ARM can also be used where beneficial. End users care more about their user experience, including access to the web, email, their pictures and movies, etc., than they care about a platform supporting all these services.

Today, most desktop and laptop computers are based on the x86 architecture, while mobile phones use the ARM architecture, and high end game consoles use the PowerPC architecture. The main factor preventing architectures other than x86 to be used for desktops and laptops is the operating system. If Microsoft Windows were ported to different processor architectures such as the ARM architecture, the market could change. Other OSes, like Apple's Mac OS X and Google's Android, could also challenge the desktop market, thanks to their support for the ARM architecture in the mobile domains.

Legacy software for the desktop and laptop can be an important roadblock for the adoption of different ISAs and OSes. Emulation of another ISA is still costly in terms of performance, but has now reached a level of practical usability. For example, Apple's Mac OS X running on the Intel architecture can execute native PowerPC binaries with no significant user hurdle.

Another convergence is optimally making use of the hardware's heterogeneous processing resources, for example by better dividing tasks between the CPU and the GPU where the GPU is the number cruncher, and the CPU serves as the orchestrator. Common software development in OpenCL [OpenCL] and GrandCentral [Grandcentral] tool flows will help to define applications that can efficiently use all the hardware resources available on the device, including multi-core CPUs and GPUs.

The economics of collaboration

The Internet has boosted the appearance of new communities and collaborative work. People are contributing their time and sharing knowledge and expertise with others like never before. This phenomenon is increasingly visible in all ICT domains:

- In the software field, Linux and gcc are two prominent examples. A state-of-the-art operating system and compiler have been built, and are offered under free licenses as the result of tremendous work by hundreds of specialists. The developer community groups a diverse crowd of independent contributors, company employees, and students. Apart from financial advantages, contributors are motivated by factors such as reputation, social visibility, ethics, the raw technical challenge, and the eventual technical advantage.
- In terms of expert knowledge, Wikipedia has caused the disappearance of Microsoft Encyclopaedia (Encarta). The Web 2.0 evolution has brought about a boom in terms of content creation by end users. Free, community-built content-management software such as Drupal also plays an important role in this development.
- Regarding social culture, YouTube and other portals make available video and music offered by their authors under so-called Copyleft licenses, which allow freedom to use and redistribute contents.

All this community-generated content has grown thanks to the use of novel licensing terms such as the GNU General Public License (GPL) and the Creative-Commons Copyleft license. These licenses focus on the protection of the freedom to use, modify and redistribute content rather than on limiting their exploitation rights.

This has led to increased competition both in the software and in the content generation markets. At the same time it enables more reuse and stimulates investing resources in opening niche markets that would otherwise be too unprofitable to enter. Moreover, people want to share and exchange their creations, resulting in more demand for interoperability.

User-generated content and independent publishers represent an increasingly important share of the media available on the Internet, resulting in increased competition for publishing houses. This trend also redefines the communication, storage and computation balance over the network.

Infrastructure as a service – cloud computing

Another business trend is the evolution towards providing services instead of only hardware. The main financial advantage is to have continuous revenue, instead of "one shot" at the sale of the product. After-sales revenue has also decreased because nowadays most consumer devices are designed to be discarded rather than repaired, and product lifetime has also been reduced to continuously follow the latest fashion trends for, e.g., mobile phones. The fact that most modern consumer devices are not really repairable has a bad impact on the environment, but it also fuels the recycling business.

The infrastructure service model requires the provider to have a large ICT infrastructure that enables simultaneously serving a large number of customers. If the service is offering processing power, the large scale is also a way to reduce peak load. This can be done by exploiting the fact that not all users will require peak performance at the same time, if necessary by providing dedicated billing policies that encourages users to adapt their usage profile so as to spread peak consumption. It is then better to have a shared and common infrastructure that is dimensioned for average load, as opposed to having many unused resources at the customer side due to over-dimensioning to cope with sparse peak requests.

Processing power and storage services, such as for indexing the web or administrating sales, are also increasingly offered to end-users. Google first provided storage with Gmail and later on for applications, Amazon now provides computing power, and there are many other recent examples. Together with ubiquitous connectivity, this leads to "cloud computing": data and resources from the end user will be stored somewhere on the cloud of servers of a company providing services.

When the cloud provides storage and processing power, the end-user terminal device can be reduced to input, output and connectivity functionality and can therefore become inexpensive. This model has already started with mobile phones, where the cost for the user is primarily in the subscription and not in the hardware of the terminal itself.

We even see this model being considered for high-end gaming [AMD], where a set of servers generates high-end graphics and delivers them to a rather low-cost terminal. This model could also be an answer to unlicensed software use and multimedia content: the game software will run on the server and will never be downloaded to the client. For media, streaming-only could deliver similar benefits.

However, this model has several implications:

- The client should always be connected to the cloud's servers.
- Compression techniques or high-bandwidth connections are required (mainly for high-definition video and gaming)
- The customer should trust the provider if he/she stores private data on the provider's cloud.
- The cloud should be reliable 24/24, 7/7, 365/365.

As of 2009, companies like Google, Microsoft and Amazon still face problems in this regard with, for example, web services going down.

The necessity to be constantly connected accompanied by privacy concerns may hamper the success of this approach: "computing centres" were inevitable in the 80's, but the personal computer restored the individual users' freedom. These two opposites consisting of resources centralized at the provider with a dumb client, versus a provider only providing the pipes and other computing and storage resources belonging to the customer, still have to be considered.

Therefore, companies are looking more and more into providing services. IBM is a good example for the professional market, while Apple is an example for the consumer market with its online store integrated in iTunes. Console providers also add connectivity to their hardware devices to allow online services. Connectivity also allows upgrading the device's software, thereby providing the user with a "new" device without changing the hardware.

Technological constraints

This section gives an overview of the key technological evolutions and limitations that we need to overcome in order to realize the applications of the future in an economically feasible manner.

Hardware has become more flexible than software

This trend is also called the hardware-software paradox. It is a consequence of the fact that the economic lifetime of software is much longer than the economic lifetime of hardware. Rather than looking for software to run on a given hardware platform, end users are now looking for hardware that can run their existing and extremely complex software systems. Porting software to a completely new hardware platform is often very expensive, can lead to instability, and in some cases requires re-certification of the software.

At the same time, hardware is evolving at an unprecedented pace. The number of cores and instruction set extensions increases with every new generation, requiring changes in the software to effectively exploit the new features. Only the latest software is able to take full advantage of the latest hardware improvements, while older software benefits much less from them.

As a result, customers are increasingly less inclined to buy systems based on the latest processors, as these provide little or no benefit when running their existing applications. This is particularly true for the latest multi-core processors given the many existing single-threaded applications.

Power defines performance

Moore's law and the associated doubling of the number of transistors per IC every process generation, has until recently always been accompanied by a corresponding reduction in supply voltage, keeping the power envelope fairly stable. Unfortunately, voltage scaling is becoming less and less effective, because further reducing the supply voltage leads to increased leakage power, offsetting the savings in switching power. At the same time, the ITRS projects that integration will continue due to smaller feature sizes for at least another five generations [ITRS]. Therefore, while future chips are likely to feature many cores, only a fraction of the chip will likely be active at any given time to maintain a reasonable power envelope.

Since it will not be possible to use all cores at once, it makes little sense to make them all identical. As a result, functional and micro-architectural heterogeneity is becoming a promising direction for both embedded and server chips to meet demands in terms of power, performance, and reliability. This approach enables taking full advantage of the additional transistors that become available thanks to Moore's Law.

Heterogeneous processors are already widely used in embedded applications for power and chip real-estate reasons. In the future, heterogeneity may be the only approach to mitigate power-related challenges, even if real-estate no longer poses any significant problems. For example, Intel's TCP/IP processor is two orders of magnitude more power-efficient when running a TCP/IP stack at the same performance as a Pentium-based processor [Borkar2004].

Energy efficiency is a major issue for mobile terminals because it determines autonomy, but it is also very important in other domains: national telecom providers are typically the second largest electricity consumers after railway operators, and the CO_2 impact of data centers is increasing continuously.

Communication defines performance

Communication and computation go hand in hand. Communication — or, in other words, data transfers — is essential at three levels: between a processor and its memory; among multiple processors in a system; and between processing systems and input/output (I/O) devices. As transistors and processors become smaller, the relative distance of communication increases, and hence so does its relative cost. At the first level, as the number of megabytes of memory per processor increases, so does memory access time measured in processor clock cycles. Caches mitigate this problem to some extent, but at a complexity cost. At the second level, with more processors on a chip or in a system, traditional buses no longer suffice. Switches and interconnection networks are needed, and they come at a non-negligible cost. At the third level, chip and system I/O is a primary component of system cost, both in terms of power dissipation and of wiring area or system volume.

Because of the high cost of communication, locality becomes essential. However, communication and locality management are expensive in terms of programmer time. Programmers prefer the shared memory programming models, whereby they view all data as readily available and accessible by address at a constant cost, independent of its current location. Real multiprocessor memory however has to be distributed for performance reasons. Yet, we prefer not to burden programmers with managing locality and transfers: in case of coherent caches hardware is responsible for these tasks, and modern research into run-time software enables implementing more sophisticated locality algorithms than those available when relying on hardware alone.

The system not only has to communicate with various memory hierarchies, but also has to exchange data with the outside world. This external communication also requires large amounts of bandwidth for most applications. For example, a stream of High Definition images at 120 fps leads to bandwidth requirements of about 740 MB/s. This is more than transferring the content of a CD in one second.

ASICs are becoming unaffordable

The non-recurring engineering (NRE) costs of complex application-specific integrated circuits (ASICs) and Systems on a Chip (SoCs) are rising dramatically. This development is primarily caused by the exponential growth of requirements and use cases they have to support, and the climbing costs of creating masks for new manufacturing technologies. The ESIA 2008 Competitiveness Report [ESIA2008] illustrates this trend. In addition to the cost of managing the complexity of the design itself, verification and validation are also becoming increasingly expensive. Finally, the integration and software development costs also have to be taken into account.

These costs have to be recuperated via income earned by selling chips. However, the price per unit cannot be raised due to strong competition and pressure from customers. As a result, the development costs can only be recovered by selling large quantities of these complex ASICs. ASICs are by definition, however, application-specific and are often tuned to the requirements of a few big customers. Therefore, they cannot be used "as is" for multiple applications or customers. This leads to a deadlock: the market for these chips may not be large enough to amortize the NRE costs. That is, of course, unless newer technologies help to drastically reduce these costs.

Fortunately, every cloud has a silver lining. As it happens, the multi-core roadmap is creating new opportunities for specialized accelerators. In the past, general-purpose processor speed increased exponentially, so an ASIC would quickly lose its performance advantage. Recently, however, this processor trend has considerably slowed down. As a result, the performance benefits offered by ASICs can now be amortized over a longer period of time [Pfister2007].

Worst-case design for ASICs leads to bankruptcy

Current chips for consumer applications are designed to function even in the worst-case scenario: at the lowest voltage, the worst process technology corner and the highest temperature. Chip binning, i.e., sorting chips after fabrication according to capabilities, is usually not performed because the testing costs outweigh the income from selling the chips. Microprocessors are an exception to this rule, as the selling price of these chips is so high that the binning cost is relatively low. Nevertheless, even for microprocessors chip binning is only applied for a few parameters, such as stable clock frequency, and not yet for others, such as correctly functioning cache size.

The practical upshot is that most consumer chips are overdimensioned. In most realistic cases typical use is far from the worst case, and this gap is even widening with the use of very dense technologies at 45 nm and below, because of process variability. The increasing complexity of SoCs is also a factor that widens the gap due to the composition of margins. If the architecture and design methodologies do not change, we will eventually end up with such large overheads that it will become economically infeasible to produce any more chips.

New design methodologies and architectures will be required to cope with this problem. For example, the "Razor" concept [Ernst2004, Blaauw2008] is one solution. In this case errors are allowed to occur from time to time when typical conditions are not met, but they are detected and subsequently corrected. Alternative methods are using active feedback and quality of service assessments in the SoC. One very important issue is that most of the techniques currently under development decrease the system's predictability, and thereby also any hard real-time characteristics it may have had.

Systems will rely on unreliable components

The extremely small feature sizes mean that transistors and wires are no longer going to behave in the way we are used to. Projections for transistor characteristics in future fabrication processes indicate that scaling will lead to dramatically reduced transistor and wire reliability. Radiation-induced soft errors in latches and SRAMs, gate-oxide wear-out and electromigration with smaller feature sizes, device performance variability due to limitations in lithography, and voltage and temperature fluctuation are all likely to affect future scaling.

An important consequence is that the variability of different parameters such as speed and leakage is quite high and changing over time. Sporadic errors, a.k.a. soft errors and aging problems, are consequently becoming so common that new techniques need to be developed to handle them. This development has only just started; in the near future, reliable systems will have to be designed using unreliable components [Borkar2005].

For Europe, this evolution is an opportunity since it can apply its extensive knowledge of high-availability systems in the commodity market.

Time is relevant

Many innovations in computing systems have only focused on overall or peak performance, while ignoring any form of timing guarantees. In the best case, an abstract time notion was used in the time complexity analysis of an algorithm. Common computing languages today do not even expose the notion of time, and most hardware innovations have been targeting best-effort performance. Examples are the introduction of caches, various kinds of predictors, out-of-order processing and lately multi-core processors [Lee2006]. Classic optimizations in compilers also go for best-effort optimizations, not for on-time computations.

While this is not a problem for scientific applications, it poses a major hurdle for systems that have to interact with the physical world. Examples are embedded systems, consumer systems such as video processing in TV sets, and games.

Embedded systems are generally interfacing with the real world, where time is often a crucial factor, either to sample the environment or to react to it as in, e.g., a car ABS system. This is different from most computer systems that have a keyboard and displays as interfaces, where users are used to small periods of unresponsiveness. Nevertheless, even in this latter situation, explicitly taking time into account will improve the user experience.

The time factor is also of paramount importance for the "disappearing computer", a.k.a. ambient intelligence. In this case the computer has to completely blend in with the physical world, and therefore must fully operate in real time.

Even for scientific applications time starts to matter. Parallel tasks should ideally have the same execution time in order to minimize synchronization delays and maximize throughput. Execution time estimates for a variety of cores and algorithms are indispensible metrics for this optimization process.

Many other trends and constraints also directly affect this topic. Ubiquitous parallelism challenges the design flows for a whole class of systems where design-time predictability is the default assumption. Process variations and transient errors are interfering with real-time behavior.

Operating systems, run-time systems, compilation flows and programming languages have been designed to harness the complexity of concurrent reactive systems while preserving real-time and safety guarantees, for example through the use of synchronous languages. Current evolutions however require that predictability and performance be reconciled with the architecture and hardware sides as well. In turn, this will likely trigger cross-cutting changes in the design of software stacks for predictable systems.

Computing systems are continuously under attack

As is clear from the application trends, private data will be stored on devices that are also used to access public data and to run third-party software. This data includes truly private information, like banking accounts, agendas, address books, and health records, as well as personally licensed data. Such data can be stored on personal or on third-party devices. An example of the latter case could be a company that rents out CPU time or storage space as part of a cloud. As such, the private data can also include code with sensitive IP embedded in it.

As a result, many types of sensitive data will be present simultaneously on multiple, worldwide interconnected devices. The need for security and protection is therefore larger than ever. Two broad categories of protection need to be provided. First, private data stored or handled on a private device needs to be protected from inspection, copying or tampering by malicious third-party applications running on the same device. For such cases, the protection is commonly known as protection against malicious code: the device is private and hence trusted, but the third-party code running on it is not.

Secondly, private data stored or handled on third-party devices needs to be protected from inspection, copying or tampering by those third-party devices or by third-party software running on them. This case is commonly referred to as the malicious host case, in which a user entrusts his own private code and data to an un-trusted third-party host environment.

Parallelism seems to be too complex for humans

Unmanaged parallelism is the root of all evil in distributed systems. Programming parallel applications with basic concurrency primitives, be it on shared or distributed memory models, breaks all rules of software composition. This leads to non-determinism, debugging and testing nightmares, and does not allow for architectural optimizations. Even specialists struggle to comprehend the behavior of parallel systems with formal models and dynamic analysis tools. Alternative recent concurrency primitives, such as transactional memory, suffer from other problems such as immaturity and a lack of scalability.

Hence, most programmers should not be required to directly care about the details of parallelism, but should merely have to specify the partitioning of their sub-problems into independent tasks, along with their causal relations. Composable formalisms and language abstractions already exist that offer exactly this functionality. Some of these techniques are very expressive; some lead to inefficiencies in mapping the exposed concurrency to individual targets. There are huge challenges and difficult tradeoffs to be explored in the design of such abstractions, and in the associated architectures, compilation, and run-time support to make them scalable and efficient.

Effective software engineering practices cannot and should not let the programmers worry about the details of parallelism. They should only focus on correctness and programmer productivity. Performance optimizations, including the exploitation of concurrency on a parallel or distributed platform, should be done by automatic tools. David Patterson talks in this context about the productivity layer that is used by 90% of the programmers and the efficiency layer that is used by 10% of the programmers [Patterson2008].

Except for specific high-performance computing applications — where a small fraction of the programmers are experts in parallel computing and the applications are fairly small — and for the design-space exploration of special-purpose systems, the quest for efficiency and scalability should never limit design productivity.

One day, Moore's law will end

The dissipation bottleneck, which slowed the progress of clock frequency scaling and shifted computing systems towards multi-core processors, was a reminder that the smooth evolution of technology we have enjoyed for decades may not last forever. Therefore, investigating alternative architectures, programming models and technologies, stems from a very practical, if not industrial, concern to anticipate drastic changes in order to be ready when needs be. For instance, research on parallelizing compilers and parallel programming models has intensified only when multi-core processors became mainstream, and it is not yet mature in spite of strong industry needs.

The original Von Neumann model has been a relatively nice fit for the technology evolutions of the past four decades. However, it is hard to neglect the fact that this model is under growing pressure. The memory bottleneck occurred first, followed by the instruction flow bottleneck (branches), and more recently by the power dissipation bottleneck. As a result of the power dissipation bottleneck, processors hit the frequency wall and architects shifted their focus to multi-core architectures. The programming bottleneck of multi-core architectures raises doubts on our ability to take advantage of many-core architectures, and it is not even clear that power dissipation limitations will allow the usage of all transistors and thus all the cores available on a chip at the same time. More recently, the reliability bottleneck involving defects and faults brings on a whole new set of challenges. It is also unclear whether it will still be possible to precisely lay out billions of transistors, possibly forcing chip designers to contemplate more regular structures or learn to tolerate structural irregularities.

Architects have attempted to meet all these challenges and preserve the appearance of a Von Neumann-like architecture. However, the proposed solutions progressively erode performance scalability up to the point that it may now make sense to investigate alternative architectures and programming models better suited to cope with technology evolution, and which intrinsically embrace all these properties/constraints rather than attempt to hide them.

For instance probabilistic-based transistors that leverage rather than attempt to hide the unreliability of ultra small ultra-low-power devices, promise very significant gains in power, but require to completely revisit even the algorithmic foundation of a large range of tasks [Palem05].

Similarly, neuromorphic architectures, pioneered by Carver Mead [Mead89], promise special-purpose architectures that are intrinsically tolerant to defects and faults.

Even if alternative architectures and programming models can cope with increasingly constrained CMOS or even silicon-based circuits for some time, we know that there are physical limits to the reduction of transistor size. Therefore, there is a need for investigating not only alternative architectures and programming models, but also alternative technologies.

There is a vast range of possible alternative technologies. A non-exhaustive list includes nanotubes, molecular computing, spintronics, quantum computing, chemical computing, biological cells or neurons for computing [Vas97]. A distinct possibility is that not one particular technology will prevail, but that several will co-exist for the different tasks they are best suited for. One can for instance envision a future in which quantum computing is used for cryptography and for solving a few NP-hard problems, while neuron-based architectures are used for machine-learning based tasks.

Another possibility is that a particular technology will prevail, but it would be extremely difficult to anticipate the winning technology. As a result, it is difficult to start investigating novel architectures and programming models capable to cope with the properties of this novel technology. One way to proceed is to abstract several common properties among a large range of technologies. That enables shielding the architecture and programming language researcher from the speculative nature of technology evolution.

For instance, one can note that, whether future technologies will be ultra-small CMOS transistors, nanotubes, or even individual molecules or biological cells, these elementary components all share several common properties: they come in great numbers, they won't be much faster or may even be way slower than current transistors, long connections will be slower than short ones, they may be hard to precisely lay out and connect, and they may be faulty.

Once one starts going down that path, it is almost irresistible to observe that nature has found, with the brain, a way to leverage billions of components with similar properties to successfully implement many complex information processing tasks. Similarly, organic computing stems form the self-organization and autonomic properties of biological organisms [Schmeck2005].

Technical challenges

In order to meet the requirements of future applications, the identified technical constraints mandate drastic changes in computer architecture, compiler and run-time technology.

Architectures need to address the constraint that power defines performance. The most power-efficient architectures are a combination of complex, simple and specialized cores, but the optimal combinations and their processing elements and interconnect architectures still remain to be determined. Moreover, this design space heavily depends on the target applications. To achieve higher performance, system developers cannot rely on technology scaling any longer and will have to exploit multi-core architectures instead. However, as mentioned earlier, handling concurrency only at the software layer is a very difficult task. To facilitate this, adequate architectural and run-time system support still needs to be developed in addition to advanced tools.

Moreover, system-level solutions for optimizing power efficiency make it significantly more difficult to meet the predictability and composability requirements. These requirements are very important for many existing and future multi-threaded applications, but the currently used worst-case execution time (WCET) analyses do not deliver anymore in these situations. A new generation of approaches, models and tools will have to be designed to support and meet the requirements of multi-core programming, predictability and composability. Again a holistic hardware/software scenario is envisioned. More precisely, future, power-aware architectures shall make the necessary information available and expose the right set of hooks to the compiler and the run-time system. With these means at hand and adherence to compile-time guidelines, novel run-time systems will be able to take the correct decisions in terms of power optimization.

Just like we will need system-level solutions to obtain acceptable power efficiency, we will also need system-level solutions to ensure reliable execution. Hardware should detect soft errors and provide support for bypassing or re-execution. Because the number of hard defects will be relatively high and will possibly increase during the system's lifetime, simply abandoning or replacing coarse-grain defective parts will not work anymore. Instead, more flexible solutions are required that enable adapting running software to evolving hardware properties.

With respect to productivity, which can be improved through reuse and portability, the fact that software is now more expensive than hardware requires software developers to stop targeting specific hardware. This is, however, very hard in practice because existing compilers have a hard time taking full advantage of recent architectures. To overcome this difficulty, new tool flows have to be designed that can automatically exploit all available resources offered by any target hardware while still allowing the programmer to code for a given platform, leading to true portable performance.

Failure in pushing the state of the art in these areas may lead to stagnation or decreasing market opportunities, even in the short term. The seven challenges that we identified are the following:

1. Performance;
2. Performance/€ and performance/Watt/€;
3. Power and energy;
4. Manageable system complexity;
5. Security;
6. Reliability;
7. Timing predictability.

Performance

Throughout the history of computing systems, applications have been developed that demanded ever more performance, and this will not change in the foreseeable future. All of the earlier described applications require extremely large amounts of processing power.

Until recently, the hardware side has provided us with constant performance increases via successive generations of processor cores delivering ever higher single-thread performance in accordance with Moore's law. Thanks to increasing clock frequencies, improved micro-architectural features, and improved compiler techniques, successive generations of cores and their compilers have always been able to keep up with the performance requirements of applications. This happened even for applications that were mostly single-threaded, albeit at the expense of huge amounts of transistors and increasing power consumption to deliver the required instruction-level and data-level parallelism.

Hence, until recently meeting these requirements did not mandate major changes with respect to software development. Instead, it sufficed to wait for newer generations of processors and compilers that provided programmers with the required performance improvements on a silver platter. Unfortunately this performance scaling trend has come to an end. Single-core performance increases at a much slower pace now, and the use of parallelism is the only way forward. Existing research into efficient and high performance architectures and infrastructures, which has (except for the last years) always relied on the old scaling trend, has not yet provided us with appropriate solutions for the performance problems we are currently facing. In particular, hardware research has to be linked closer with research in compilers and other tools to enable the actual harnessing of potential performance gains offered by improved parallel hardware.

Performance/€, performance/Watt/€

Due to the current downturn of economy, the constraint of cost becomes more critical than ever. In tethered devices, performance per Euro is key, as demonstrated by the emergence of low-cost computers such as Atom-based or ARM-based netbooks. For mobile devices, the criterion of choice is performance per Watt per Euro: enough performance to run most applications, but with a long autonomy and at a low price.

Due to the rising operational costs of energy and cooling, and because chip packaging costs contribute significantly to the final costs of hot-running chips, the criterion of performance per Watt per Euro has also become key for cloud computing clusters. As previously pointed out, more and more consumers prefer the right price for reasonable performance, rather than the best performance at all costs. Companies are also looking to reduce their ICT infrastructure costs, potentially leading to new business models based on renting out computing power and storage space.

Power and energy

All of the described future applications require high energy efficiency, either because they run on batteries and require good autonomy or because of energy, packaging and cooling costs, or both. In cars, for example, processors are packaged in rubber coatings through which it is difficult to dissipate much heat. Moreover a number of digital processes in future cars will continue to run when the engine is turned off; hence they should consume minimal energy. Body implants obviously cannot generate a lot of heat either, and require a longer autonomy. Domestic robots also entail high autonomy, both to avoid day-time recharging and to survive power outages.

In the past, energy efficiency improvements were obtained through shrinking transistor sizes, through coarse-grain runtime system techniques such as dynamic frequency scaling and the corresponding voltage scaling, and through fine-grained circuit techniques such as clock and power gating. Furthermore, where no adequate programmable alternatives were available, ASIC and ASIP designs were developed to obtain satisfactory power efficiency. Today, power scaling offers diminishing returns, leakage power is increasing at a rapid pace, and the NRE costs of ASICs and ASIPs are making them economically unviable.

Managing system complexity

Besides performance increases, we also see significant increases in system complexity. The reason is not only that systems are composed of more and more hardware and software components of various origins, but also that they are interconnected with other systems. The impact of a local modification can be drastic at system level, and understanding all implications of a modification becomes increasingly hard for humans. We enter an era where the number of parallel threads in a data center will be in the millions. This matches the number of transistors in a core.

Some of the major technical aspects of managing system complexity relate to composability, portability, reuse and productivity.

Composability in this context refers to whether separately designed and developed applications and components can be combined into systems that operate, for all of the applications, as expected. For example, in future cars, manufacturers would like to combine many applications on as few processors as possible, while still keeping all the above requirements in mind. Ideally, manufactures would like to be able to plug in a large variety of services using a limited range of shared components. That would enable them to differentiate their products more easily between different service and luxury levels. Similar reasoning holds for many other future applications. One of the main challenges related to composability is the fact that physical time is not composable, and that the existing models to deal with parallelism are mostly non-composable either. Recent techniques that try to deal with this issue, such as transactional memory, are far from being mature enough at this point in time.

Many concrete instances of the aforementioned applications are niche products. In order to enable their development, design, and manufacturing in economically feasible ways, it is key to increase productivity during all these phases. Two requirements to achieve higher productivity are portability and reuse. Enabling the reuse of hardware and software components in multiple applications will open up much larger markets for the individual components, as will the possibility to run software components on diverse ranges of hardware components. The latter implies that software should be portable and also composable.

Recent techniques to obtain higher productivity include the use of bytecode and process virtual machines, such as Java bytecode and Java Virtual Machines. Their use in heterogeneous systems has been limited, however.

Security

All described future applications will make use of wireless communications. Hence they all are possible targets of remote attacks. In Human++ body implants and domestic robots, security is critical to defend against direct attacks on a person's well being and against privacy invasions. Privacy is also a concern in telepresence applications, as is intrusion. It is not hard to imagine how fraud can take place in a telepresence setting in which virtual reality image synthesis recreates images of participants rather than showing plain video images of the real persons that are believed to participate.

In applications such as the autonomous vehicle and in many wireless consumer electronics, security is also needed to protect safety-critical and operation-critical parts of the systems from user-controlled applications.

In these contexts and in the context of offloaded computing, protection against malicious host and malicious code attacks still poses significant challenges, in part because this protection has to work in the context of other constraints and trends. For example, it is currently still an open question what is the best way to distribute applications. The distribution format should be abstract enough to provide portable performance and it should at the same time provide enough protection to defend against a wide range of attacks. On the one hand performance portability, i.e., the capability to efficiently exploit very different types of hardware without requiring target-dependent programming, necessitates applications to be programmed on top of abstract interfaces with high-level, easy-to-interpret semantics, and to be distributed in the format of those interfaces. Protection, on the other hand, requires the distributed code to contain a minimum amount of information that may be exploited by attackers. Additionally, all techniques developed and supported to meet these requirements in the malicious host context can also be abused by malicious code to remain undetected. As such, providing adequate software and data protection is a daunting challenge.

Modern network security systems should adapt in real time and provide the adequate level of security services on-demand. A system should support plenty of network security perimeters and their highly dynamic nature caused by actors such as mobile users, network guests, or external partners with whom data is shared.

Until today, the above security challenges have largely been met by isolating processes from each other. By running the most critical processes on separate devices, they are effectively shielded from less secure software running on other system. Given the aforementioned challenges and trends, the principle of isolating applications by isolating the devices on which they run cannot be maintained. Instead, new solutions have to be developed.

Reliability

To safeguard users, future applications have to be absolutely reliable. For example, safety features in cars, airplanes or rockets need to behave as expected. The same holds for body implants and clearly for, e.g., telesurgery as an application of telepresence.

Several techniques are used today to guarantee that systems behave reliably. Hardware components have their design validated before going into production, and they are tested when they leave the factory and during deployment. This testing is performed using built-in tests of various kinds. When specific components fail, they or the total system are replaced by new ones. Some components include reliability-improving features such as larger noise margins, error-correcting/error-detecting codes, and temperature monitoring combined with dynamic voltage and frequency scaling. Most if not all of these features operate within specific layers of a system design, such as the process technology level, the circuit level or the OS level.

These solutions of detecting and replacing failing components or systems, and of improving reliability within isolated layers, works because the number of faults to be expected and the number of parameters to be monitored at deployment time are relatively low, and because fabrication and design costs as well as and run-time overheads are affordable. Obviously, the latter depends on the context: many existing reliability techniques have only been applied in the context of mainframe supercomputers, because that is the only context in which they make economic sense. However, as technology scales, variability and degradation in transistor performance will make systems less reliable. Building reliable systems using existing techniques is hence becoming increasingly complex and costly; the price of system power consumption and performance is getting higher, while the costs for designing, manufacturing, and testing also increase dramatically. Consequently, we need to develop new hardware and software techniques for reliability if we want to address and alleviate the above costs.

For safety-critical hardware and software verification and diagnostic tools are used, but to a large extent verification is still a manual and extremely expensive process.

Timing predictability

Most future applications require hard real-time behavior for at least part of their operation. For domestic robots, cars, planes, telesurgery, and Human++ implants, it is clearly necessary to impose limitations on the delay between sensing and giving the appropriate response.

Today, many tools exist for worst-case execution time analysis. They are used to estimate upper bounds on the execution time of rather simple software components. These methods currently work rather well because they can deal with largely deterministic, small, usually single-threaded software components that are isolated from each other. In future multi-threaded and multi-core platforms, accurately predicting execution time becomes an even harder challenge, both for real-time and for high-performance computing systems.

In addition, execution time estimates are becoming increasingly important outside the real-time domain too. For parallel applications, it is important that all processes running in parallel have the same execution time in order to maximally exploit the parallel resources of the platform, and limit the synchronization overhead. Especially on heterogeneous multi-cores, being able to accurately estimate execution times is crucial for performance optimization.

2. HiPEAC Vision

This chapter provides an overview of technical directions in which research should move to enable the realization of the Future Applications required for dealing with grand societal challenges, taking into account the technological constraints listed above.

Our approach starts from the observation that the design space, and hence the complexity, keeps expanding while the requirements become increasingly stringent. This holds for both the hardware and the software fields. Therefore, we are reaching a level that is nearly unmanageable for humans. If we want to continue designing ever more complex systems, we have to minimize the burden imposed on the humans involved in this process, and delegate as much as possible to automated aids.

We have opted for a vision that can be summarized as keep it simple for humans, and let the computer do the hard work.

Furthermore, we also have to think out of the box by inventing and investigating new directions to start preparing for the post-Moore era by considering non-traditional approaches such as radically different new programming models, new technologies, More-than-Moore techniques or non-CMOS based computational structures.

Keep it simple for humans

To enable humans to drive the process and to manage the complexity, we primarily have to increase the abstraction level of the manipulated hardware and software objects. However, we propose domain-specific objects rather than very generic objects, because they are more concrete and understandable and also easier to instantiate and optimize by computers. In order to do so, two main developments are required:

1. Simplify system complexity such that the systems become understandable and manageable by human programmers, developers, designers, and architects.
2. Use human intellect for those purposes it is best suited for, including reasoning about the application, the algorithms, and the system itself, and have it provide the most relevant information to the compiler/system.

We now discuss three profiles of humans involved in the design and maintenance of computing systems: the software developers, the hardware developers, and the system people, i.e., the professionals building and maintaining the systems.

Keep it simple for the software developer

One of the grand challenges facing IT according to Gartner is to increase programmer productivity 100-fold [Gartner08]. It is immediately clear that traditional parallel programming models are not going to be very helpful in reaching that goal. Parallel programming languages aim at increasing the performance of code, not the productivity of the programmer. What is needed are ways to raise the programming abstraction level dramatically, such that the complexity becomes easier to manage.

Traditional parallel programming languages should be considered as the machine language of the multi-core computing systems. In this day and age, most programmers do not know the assembly programming of the machine they are programming thanks to the abstractions offered by high-level languages. Similarly, explicit parallelism expressions should be invisible to most programmers. Traditional parallel programming languages therefore cannot be the ultimate solution for the multi-core programming problem. At best they can be a stopgap solution until we find better ways to program multi-core systems.

The programming paradigm should provide programmers with a high-level, simple but complete set of means to express the applications they wish to write in a natural manner, possibly also expressing their concurrency. The compiler and the runtime system will then be able to schedule the code and to exploit every bit of the available parallelism based on the software developer's directives, the targeted architecture and the current status of the underlying parallel hardware.

High-level domain-specific tools and languages will be key to increasing programmer productivity. Existing examples are databases, MATLAB, scripting languages, and more. All these approaches enable raising the level of abstraction even further when compared to one-language-to-rule-them-all-approaches. The above languages are becoming increasingly popular, and not only as scripting languages for web applications: more and more scientists and engineers evaluate their ideas using dynamic, (conceptually) interpreted languages such as Python, Ruby and Perl instead of writing their applications in C/C++ and compiling them.

Visual development environments, where applications are defined and programmed mainly by composing elements with mouse clicks and with very little textual input, are maturing rapidly. Such environments allow even the casual developers to create complex applications quite easily without writing long textual programs.

In line with this vision, we believe that it is important to make a clear distinction between end users, productivity programmers and efficiency programmers as shown in Figure 1.

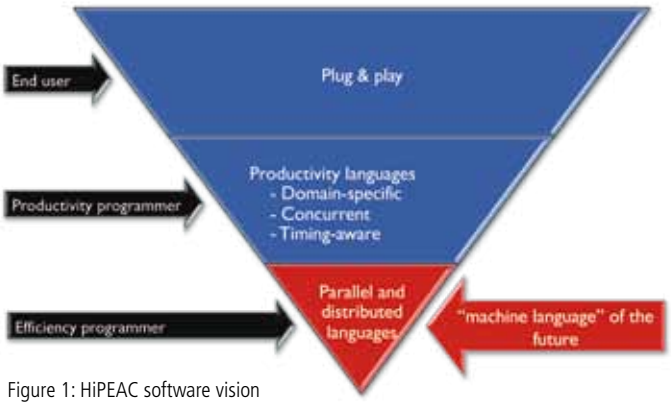


Figure 1: HiPEAC software vision

End users should never be confronted with the technical details of a platform. They are only interested in solving their everyday problems by means of applications they buy in a software store. For them it is irrelevant if the real execution platform consists of a single-core or a multi-core processor. They are generally not trained computer scientists.

Among the trained computer scientists, about 90% are developing applications using high-level tools and languages. They are called productivity programmers. Time to market and correctness are their primary concerns.

We believe that the programming languages and tools will have to have the following three characteristics.

1. Domain-specific languages and tools will be designed specifically for particular application domains, and will support the programmer during the programming process. General-purpose languages will always require more programmer effort than domain-specific languages to solve a problem in a particular domain. Examples of such languages are SQL for data storage and retrieval, and MATLAB for signal processing algorithms.

2. Express concurrency rather than parallelism. Parallelism is the result of exploiting concurrency on a parallel platform, just like IPC (instructions per cycle) is the result of the exploitation of ILP (instruction-level parallelism) on a given platform. A concurrent algorithm can perfectly well execute on a single core, but in that case will not exploit any parallelism. The goal of a programming model should be to express concurrency in a platform-independent way, but not to express parallelism. The compiler and the run-time system should then decide on how to exploit this concurrency in a parallel execution.

   Automatic extraction of concurrency from legacy code is a very difficult task that has led to many disappointing results. Maybe dynamic analysis and speculative multi-threading could still offer some promising solutions in this area. In the predictable future, we expect that automatic parallelization will not be able to extract many kinds of concurrency from legacy code. We therefore conclude that future applications should not be specified anymore in hard-to-parallelize sequential programming languages such as C.

It is generally considered more pragmatic to abandon the hard-to-parallelize sequential languages and to let the parallelizing compiler operate on a concurrent specification. An example of such a specification is the expression of functional semantics using abstract data types and structures with higher-level algorithms or skeletons, such as the popular map-reduce model [Dean2004]. Dataflow languages such as Kahn process networks have the most classical form of deterministic concurrent semantics. They are valued for this property in major application domains such as safety-critical embedded systems, signal processing and stream-computing, and coordination and scripting languages.

Therefore, domain-specific languages or language extensions need to be developed that allow the programmer to express what he knows about the application in a declarative way in order to provide a relevant description for the compiler and the run-time system that will map the application description to the parallel hardware and manage it during execution. Raising the abstraction level makes extracting semantic information, such as concurrency information, from the programs easier. This information will be passed on to compilers, run-time systems and hardware in order to map the program to parallel activities, select appropriate cores, validate timing constraints, perform optimizations, etc.

A very important characteristic of future programming languages is that they should be able to provide portable performance, meaning that the same code should run efficiently on a large variety of computing platforms, while optimally exploiting the available hardware resources. Obviously, the type of concurrency must match the resources of the target architecture with respect to connectivity and locality parameters; if this is not the case, the mapping will be sub-optimal.

It is clear that an approach in which code is tuned to run on a particular platform is by definition not portable and therefore not viable in the long term, since the cost of porting it to new hardware generations becomes prohibitively high. It is important to realize that programming models do not only have an entry cost in the form of the effort needed to port an application to a particular programming model, but also an exit cost that includes the cost to undo all the changes, and to port the application to a different programming model.

In this respect, future tool chains will support the programmer by giving feedback about the (lack of) concurrency that it is able to extract from the software. This feedback will be hardware-independent, but it might be structured along the different types of concurrency at the instruction level, data level, or thread level, and it might be limited to specific types of corresponding parallelism support in which the programmer has expressed interest. This expression of interest can be explicit but should not be so. For example, the simple fact that compiler backends are being employed for specific kinds of parallel hardware only, can inform the compiler front-end of the types of concurrency it should try to extract and give feedback on.

Getting early feedback on available concurrency, rather than on available parallelism, will allow a programmer to increase his or her productivity. Since the feedback is not based on actual executions of software on parallel hardware, it will be easier to interpret by the programmer, and it will be available even before the software is finished, i.e., before a fully functional program has been written. This is similar to the feedback a programmer can get on-the-fly from integrated development environments such as Eclipse about syntactical and semantic errors or in the code he or she is typing. That feedback is currently limited to relatively simple things such as the use of dangling pointers, the lack of necessary exception handlers, unused data objects, and uninitialized variables. In the HiPEAC vision, the amount of feedback should be extended to also include information about the available concurrency or the lack thereof.

3. The time parameter has to be present very early on in the system definition, so as to allow for improved behavior. For example, instead of optimizing for best effort, optimizing for "on-time" could lead to lower power consumption, less storage, etc. For real-time systems, having time as a first class citizen both in the design of the hardware and software will ease verification and validation.

   A practical approach in this case could be to develop new computational models, in which execution time can be specified as a constraint on the code. E.g., function foo should be executed in 10 ms. It is then up to the run-time system to use hardware resources such as parallel, previously idle, accelerators in such a way that this constraint is met. If this turns out to be impossible, an exception should be raised. Being able to specify time seems to be an essential requirement to realize portable performance on a variety of heterogeneous multi-core systems.

Finally, the remaining 10% of trained computer scientists will be concerned with performance, power, run-time management, security, reliability and meeting the real-time requirements, i.e., with the challenges presented earlier on. They are called the efficiency programmers and they are at the heart of the computing systems software community. They will develop the compilers, tools and programming languages, and they can only do so by working together intimately with computer architects and system developers. HiPEAC programmers represent such a community and have to come up with efficient parallel and distributed programming languages.

Given the large number of sequential programming languages, we believe that there are no reasons to assume that there will eventually be one single parallel programming model or language in the future. We rather believe that there is room for several such languages: parallel languages, distributed languages, coordination languages, …

The approach of this section can help with addressing the constraints Parallelism seems to be too complex for humans and hardware has become more flexible than software.

Keep it simple for the hardware developer

Just as it will be necessary to increase the abstraction level for programmers in order to cope with the complexity of modern information processing systems, hardware designers will also have to cope with additional complexity. Future systems will therefore be built from standard reusable components like cores, memories, and interconnects as shown in Figure 2. This component-based design methodology will be applicable at different levels, from the gate level to the rack level.

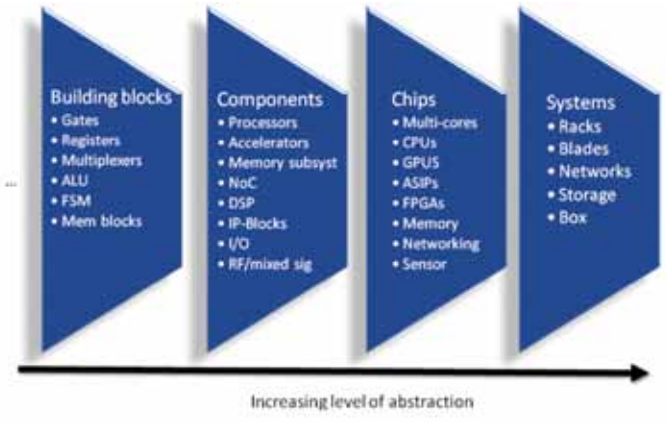


Figure 2: Component-based hardware design at different levels

Similar to high-level software design, most computing systems will be designed using high-level design tools and visual development environments. Computing systems will be built from modules with well-defined interfaces, individually validated and tested. Building complex systems is simplified by selecting hardware or software library components and letting the tools take care of the mapping and potential optimizations. Standard interfaces introduce overheads in the system design, in terms of performance loss, or power/area increase. Therefore, before finalizing a design, dedicated tools might break down the interfaces between modules in order to improve performance through global optimization, rather than only focusing on local optimizations. For example, for certain application domains, caches, and even floating-point units, can be shared by several cores. The synthesis tools and design space exploration systems could perform such optimizations. The applied transformations will lead to the blurring of processors, which will be less and less individually distinguishable. As such, full-system optimization will overcome many of the inefficiencies that were introduced by the component-based design methodologies.

The increasing non-recurring engineering (NRE) cost of Systems-on-Chip (SoC) requires that they be sold in larger quantities so this additional cost can be amortized. This can lead to a decrease of the diversity of designed chips, while the market still requires different kinds of SoCs, specialized for various application domains. A technology called System in Package (SiP) can help to solve this dilemma. In a SiP, each die uses the technology most suited to its functionality such as analog, digital, and is interconnected either in two or in three dimensions. The latter is called 3D stacking, allowing for higher density of integration than with standard chips.

Research challenges in this domain are reducing costs, and exploring new technology for interconnects, for example in the form of a wireless Network-on-Chip (RF-NoC). The flexible composition of various components while avoiding the high cost of making new masks for IC fabrication is a potential answer to ASICs becoming unaffordable. The ESIA 2008 competitiveness report also explains this trend on page 42 ("D4 The increasing importance of multi-layer, multi-chip solutions") [ESIA2008]. Besides the potential use in SiPs, the module approach is already used in several systems at the chip level such as the Nota proposal from Nokia [Nota] but not at the die level.

We again encounter an inverted pyramid, depicted in Figure 3.

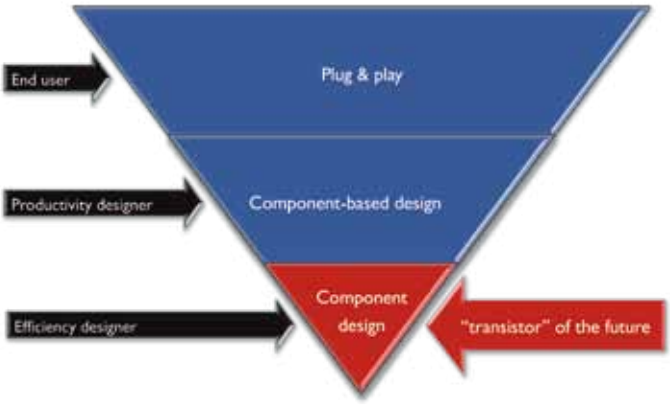


Figure 3: HiPEAC hardware vision

End users represent the vast majority of the population coming into contact with computing systems, and they do not need to know anything about the complexity of the underlying system. All they want (and need) is for the system to work. Next up are the high-level designers, whose main concern is productivity, combining predefined blocks such as processors, IP blocks, interconnects, chips, and boards. Many of these designers do not know the architectural nor micro-architectural details of the components they are integrating, and cannot spend their time optimizing them for performance, power, or cost. Instead they rely on automated tools to approximate these goals as much as possible.

One particular case is embedded systems integration where real-time guarantees are required for the total system design while the critical and less critical components are sharing resources. This type of "mixed criticality systems" needs new design veri-

fication technologies that must adhere to rigid verification and certification standards that apply to, e.g., transport or medical systems.

Finally, we have a small set of people who make the productivity layer possible by designing the different components, and by developing the high-end tools that automatically do most of the job. Architects and efficiency designers are primarily concerned with the definition and shaping of libraries of components, their interconnection methods, their combination and placement, and the overall system organization and efficient interfacing with the rest of the system. Architects can only do so while working closely with developers of programming languages, compilers, run-time systems, and automated tools, and require assistance themselves from advanced software and tools. They have to come up with efficient, technology-aware processing elements, memory organizations, interconnect infrastructures, and novel I/O components.

Every one of these library components faces a number of unresolved challenges in the foreseeable future:

- General-purpose processor architecture: a range of such cores will be needed, from simple ones for power and area efficiency to complex ones for sequential code performance improvements required by Amdahl's law, and from scalar to wide vectors for varying amounts of data parallelism. Optimization for power and reliability are whole new games, as opposed to optimization for performance as seen in previous decades.

- Domain-specific accelerators: a large spectrum of such cores will be required, including vector, graphics, digital signal processing (DSP), encryption, networking, pattern matching, and other accelerators. Each domain can benefit from its own hardware optimizations, with power, performance, and reliability or combinations thereof being primary concerns. The extensive use of accelerators automatically leads to heterogeneous domain-specific architectures.

- Memory architecture: as discussed earlier, communication, including processor-memory communication, is expensive. Consequently, a central concern in all parallel systems is improving locality, through all means possible. Caches are one method to do so, but there is still significant room for improvements in coherence, placement, update, and prefetching protocols and techniques. Directly addressable local memory, so-called scratchpad memory, with explicit communication through remote DMA control is another method for managing locality. Memory consistency, synchronization and timing support are other critical dimensions where hardware support can improve performance.

- Component interconnection: the more components there are in a system, the higher the importance of the interconnect characteristics. Chip-to-chip connections already account for a major portion of system cost in terms of pins, wires, board area, and power consumption to drive them. Intra-chip communication is quickly turning to Networks-on-Chip (NoC) for solutions; however, NoCs still require large areas and a lot of power, while exhibiting deficiencies in quality of service, latency, guarantees, etc. Glueless interfacing between cores, memories and interconnects is another open problem.

- Reconfigurable architectures: Reconfigurable multi-core architectures can help with solving the problem of hardware flexibility without excessive NRE and process mask costs; in addition, they can be very useful for reliability in the presence of dynamic faults. The current state of the art barely scratches the surface of the potential offered by such flexible systems.

Future systems will be heterogeneous. Paradoxically, the 'keep it simple for humans' vision naturally leads to heterogeneous systems. Component-based hardware design naturally invites the hardware designer to design heterogeneous systems. On top of this designed heterogeneity, increasing process variability will introduce additional heterogeneity in the chip fabrication process. As a result of this variability, fabricated systems and components will operate at different performance/power points according to probabilistic laws, including even some completely dysfunctional components. Furthermore, the appearance of multiple domain-specific languages will lead to applications that are built from differently expressed software components. At first sight, this increase in complexity might look like a step backward, but this is not necessarily the case.

As power and power efficiency become the issue in designing future systems, new computational concepts start to emerge. It is well known that using special-purpose hardware to solve domain-specific problems can be much more efficient. Due to the increasing NRE costs, it is desirable to design systems for domains of applications rather than for single applications. The relative low volume of ASICs and the high cost to prototype and validate such systems suggests designing custom processors or accelerators that address specific domain requirements rather than specific requirements of individual applications. Typically, the tradeoff between the degree of programmability and the efficiency of the accelerators is at the heart of this challenge, with general-purpose processors lying at one end of the spectrum, and non-programmable accelerators at the other. GPUs are in the middle of the spectrum, providing an order of magnitude better performance than general-purpose hardware for the same use while still being useful for solving non-graphical computation tasks when they fit the provided hardware [Cuda, OpenCL].

Integrating different types of architectures on the same die seems to be a very attractive way for achieving significantly better performance for a given power budget, assuming we understand the class of applications that may run on that die. To cope with Amdahl's law, at least two types of cores are required: cores for fast sequential processing that cannot be parallelized, and cores optimized for exploiting parallelism. Generic coprocessors, helping with memory management, task dispatching and activation, data access, and system control can significantly improve global performance. Generic tasks, such as data decoding/encoding, can be mapped onto more specialized cores, increasing the efficiency without compromising the general-purpose nature of the system. All of this comes to no surprise: nature has discovered millions of years ago that heterogeneity leads to a more stable and energy-efficient ecosystem.

Keep it simple for the system engineer

Given the growing heterogeneity of multi-core processors both in the number of cores and in the number of ISAs, it is clear that the statically optimized binary executable will have a hard time providing optimal performance on a wide variety of systems. Instead, run-time systems will need to adapt software to the available number of cores and accelerators, to failed components and other applications competing for resources, etc. Since such adaptations are done at run time they must be done efficiently, preferably with assistance from the compiler.

In order to keep all this complexity manageable for the software developers and system people, and to give hardware designers the freedom to continue innovating in diverging ways, we need an isolation layer between the software and hardware, i.e., a virtualization layer as shown in Figure 4. Depending on whether this virtualization layer sits above or below the operating system, we talk about process virtualization or system virtualization, respectively. In this vision, the use of binary executables as distribution format for applications should be abandoned and replaced with an intermediate code enriched with meta-data. This code format should be flexible enough to allow for:

1. efficient translation into a number of physical ISAs;
2. efficient exploitation of parallelism;
3. easy extensibility with extra features.

Virtualization serves two purposes: on the one hand, the virtualization layer can be seen as a separate platform to develop code for. A well-designed virtual platform will take advantage of the features of the underlying hardware/software, even if these features change throughout the execution or were unknown at the time an application was developed. On the other hand, virtualization can be used to emulate one platform on top of another. This ensures compatibility for legacy applications, and can also add extra functionality such as resource isolation by running different applications inside isolated virtualized environments.

In both cases, the key complexity issues are limited to a single component, the virtualization layer. These issues therefore become easier to manage. The design of the virtualization layer will, however, include many challenges of its own, such as the choice of appropriate abstractions, the communication channels between the virtual machine and the software running on top of it, and the kinds of meta-information to include in clients of the virtual machine.

The timing requirement can then be realized during system integration, when software is mapped onto hardware. For real-time systems, considering time as a core property in the design of both the hardware and the software will ease the verification and validation, and hence simplify the work of the system integrator. For software, traditional programming languages do not embed a notion of time. Timing information is only an afterthought, dealt with by real-time kernels, leading to a nightmare for system developers and for validation. Adding time requirements early on in the software development cycle will enable tools to optimize for it, and to choose the right hardware implementation. For example, most systems are optimized for best effort, while the optimum could be on-time scheduling, resulting in fewer hardware resources. A time-aware virtualization layer will ensure that the requirements are fulfilled at run time, avoiding increased complexity for system developers and during validation.

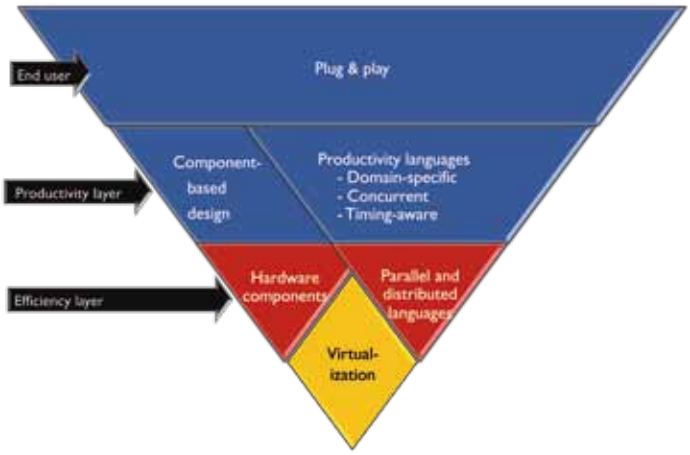


Figure 4: Role of virtualization in the HiPEAC vision

Let the computer do the hard work

This section gives an overview of the ways in which the computer can help humans with the hard work. More than ever, the computing system industry is facing the conflicting challenges of achieving computing efficiency, of adapting features to markets and various customers, and of reducing time to market and development costs. By adapting, modifying or adding specific features to generic architectures, customized systems allow savings in silicon area and power efficiency, and they enable us to meet high performance requirements and constraints. If the future will be heterogeneous, it is paramount that the different components of such heterogeneous systems can be designed and produced efficiently.

"Letting the computer do the hard work" might be considered dangerous by some: we might give up on the fine understanding of how systems work because they will be too complex and will be built by computers. While it is debatable whether this will be problematic or not, it does not even need to be the case. Computers can also be limited to assisting with the logical steps required to reach the final system, for example formal verification can prove the correctness of a process and explicitly list the steps of the required proof.

From the hardware point of view, SoCs have hundreds of millions of transistors, and a complete system integrates several chips. Up to now, complexity management consists of increasing the number of abstraction levels: after manipulating transistor parameters, tools enable designers to manipulate sets of transistors or gates, and so on until the building elements become the processor itself with its memories and peripherals. By increasing the abstraction level from transistors to processors, the process of building complex devices is kept manageable for a human designer, even if the size of teams to build SoCs increased over time. However, each level of abstraction decreases the overall efficiency of the system due to complex dependencies between abstraction layers that are not taken into account during intra-layer optimizations.

As the performance improvements of individual cores have become much smaller during the past years, the overhead, not only in terms of performance, but also in terms of power and predictability, is not compensated anymore. So the method of solving all problems by simply adding additional abstraction layers is no longer feasible. Moreover, when designing and optimizing an architecture in terms of power, area or other criteria, the number of parameters is so high and the design space so large, complex and irregular, that it is almost impossible to find an optimal solution manually. Hence, techniques and tools to automate architectural design space exploration (DSE) have been introduced to find optimized designs in complex design spaces. In a sense, DSE automates the design of systems.

From the software point of view, the abstraction level has also been increased: assembly programming is rarely used anymore compared to the vast amounts of compiled code. Nowadays optimizing compilers are the primary means to produce executable code from high-level languages quickly and automatically while satisfying multiple requirements such as correctness, performance and code size for a broad range of programs and architectures. However, even state-of-the-art static compilers sometimes fail to produce high-quality code due to large irregular optimization spaces, complex interactions with underlying hardware, lack of run-time information and inability to dynamically adapt to varying program and system behavior. Hence, iterative feedback-directed compilation has been introduced to automate program optimization and the development of retargetable optimizing compilers. At the system level, it is important that hardware and software optimizations are not performed in isolation but that full system optimization is aimed at and combined with the adaptive self-healing, self-organizing and self-optimizing mechanisms.

Figure 5 shows the different hard tasks that can be delegated to a computer. The ultimate goal of all the tasks is to optimize the non-functional metrics of the list of challenges that we have identified.

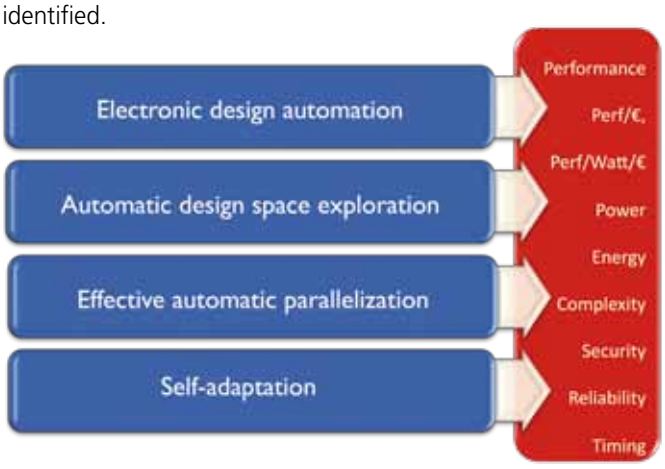


Figure 5: Hard tasks that can be delegated to the computer

Electronic Design Automation

Electronic design automation (EDA) methodologies and tools are key enablers for improved design efficiency concerning computing systems. In the light of moving towards higher density technology nodes in the time frame of this vision, there is an urgent need for higher design productivity.

EDA is currently aiming at a new abstraction level: Electronic System Level (ESL). ESL focuses on system design aspects beyond RTL such as efficient HW/SW modeling and partitioning, mapping applications to MPSoC (Multi-Processor System-on-Chip) architectures, and ASIP design. While ESL is currently driven by the embedded systems design community, there are numerous opportunities for cross-fertilization with techniques that originate from within the high-performance community, such as fast simulation and efficient compilation techniques. Similarly, the high-performance community could benefit from the advanced design techniques that were developed for the embedded world.

EDA definitely helps to solve the problem of ASICs becoming unaffordable.

Automatic Design Space Exploration

In order to explore the immense computer architecture and compiler design spaces, intuition and experience may not be good enough to quickly reach good enough/optimal designs. Automated DSE can support the designer in this task by automatically exploring and pointing to good designs, both with respect to architecture features and compiler techniques such as code transformations and the order in which they are applied. For modern computing systems, the combined architecture and compiler space is immense — with 10100 design points being no exception — and the evaluation of a single design point takes a lot of time because in theory it encompasses the simulation of an entire application on a given system.

Challenges in the DSE area are:

1. Since the total design space is now so huge, improved heuristics are needed to efficiently cull the design space in search for a good solution. The challenge is to find efficient search strategies in combinatorial optimization spaces, determining how to characterize such spaces and how to enable the reuse of design and optimization knowledge among different architectures, compilers, programs, and run-time behaviors.
2. Besides parametric design space exploration by which an optimal solution is searched in a parameter space, heterogeneous multi-core systems also require structural design space exploration where complete structures such as interconnects, memory hierarchies, and accelerators are replaced and evaluated. Changing the structure of the system also requires changes to the complete tool chain in order to generate optimized code for the next system architecture. One of the challenges is to solve all compatibility, modularity, and concurrency issues so as to allow all architectural options to be explored fully automatically.
3. Identifying correlations between architectures, run-time systems and compilers in relation to how they interact and influence performance. Automatic exploration should provide feedback to help understand why certain designs perform better than others, and predictive models need to be built to accelerate further explorations.

DSE directly contributes to addressing most of the technical challenges.

Effective automatic parallelization

Since we believe that the application programmer should mostly be concerned with correctness and productivity, and the computer should take care of the non-functional aspects of code such as performance, power, reliable and secure execution, the mostly non-functional task of parallelization should also be taken care of by the compiler rather than the programmer. For this purpose, automatic parallelization for domain-specific languages is indispensable.

Identifying concurrency in legacy code, either manually or automatically, is extremely cumbersome. Besides, for many legacy applications it is a non-issue as these applications already run fine as sequential processes on existing hardware. For new applications, the choice of the development environment is crucial. Domain-specific languages should be seen as an opportunity to provide the software and compiler development community with appropriate means to express concurrency and to automatically or semi-automatically extract parallelism.

After identifying the concurrency, it has to be exploited as parallelism. A very important aspect here is the level at which concurrency manifests itself, as this determines the granularity of parallelism. For example, it can be quite impossible to obtain performance benefits from mapping a certain fine-grained data-parallel kernel onto thread-level parallelism of a multi-core processor, while the same fine-grained parallelism can yield huge speedups on single-instruction-multiple-data (SIMD) architectures such as graphics processors.

The automatic extraction of concurrency and mapping it onto parallel hardware will be a two-phase approach, a.k.a. split compilation, where at least some time-consuming hardware-independent code analyses will be performed by a static compiler to extract concurrency. Subsequently, a dynamic compiler will perform the hardware-dependent transformations required to exploit the available parallelism based on the results of these concurrency analyses.

In such an approach, the first phase might be hardware-independent, but is not necessarily independent of the second phase. Depending on which tools will be used in the second phase, the first phase might need to extract different kinds of information. It will then be the responsibility of the first phase to produce the necessary meta-data in byte code or native code for the second phase, and to present the programmer with feedback on the available concurrency or the lack thereof.

Automatic parallelization definitely contributes to resolve the constraint that parallelism seems to be too complex for humans.

Self-adaptation

Ever more diversified and dynamic execution environments require applications, run-time environments, operating systems and reconfigurable architectures to continuously adjust their behavior based on changing circumstances. These changes may relate to platform capabilities, hardware variability, energy availability, security considerations, network availability, environmental conditions such as temperature, and many other issues.

For example, think of a cell phone that was left in a car in the summer, and heated up to 60°C. For this type of situation, run-time solutions should be embedded to cope with extreme conditions, and help the system to provide minimal basic functionality, even in the presence of failing high-performance components, all the while maintaining real-time guarantees.

With respect to protection against attacks, a system that is capable of detecting that it is not being observed by potential intruders can choose to run unprotected code rather than code that includes a lot of obfuscation overhead. When the system detects potential intrusion, it can defend itself by switching to obfuscated code.

This level of adaptability is only possible if the appropriate semantic information is made available at run time at all levels. This ranges from the software level, where opportunities for concurrency have to be specified, over to the system level where information about attacks and workload are being produced, to the physical hardware, where information about the reliability of the hardware and about operating temperature needs to be available. All this information has to be made available through a transparent monitoring framework. Such a framework has to be vertically integrated into the system, collecting information at each level and bringing it all together. This information can then be used by clients to adjust their behavior, to verify other components, to collect statistics and to trace errors.

Radically new approaches based on collective optimization, statistical analysis, machine learning, continuous profiling, run-time adaptation, self-tuning and optimization are needed to tackle this challenge.

Self-adaptivity helps dealing with the constraint that hardware has become more flexible than software, that systems are continuously under attack, and that worst case design leads to bankruptcy.

If all above is not enough it is probably time to start thinking differently

The previous directions for solving the challenges are mainly extrapolations of existing methods, still relying on architectures with processors, interconnect and memories organized as conceptual Von Neumann systems, even if under the hood most of them are not Von Neumann architectures anymore. Moreover, in those solutions, the architectures were programmed explicitly with languages that more or less describe the succession of operations to be performed. However, to solve future challenges it might also be possible to start thinking more out-of-the-box. In nature, there are plenty of data processing systems that do not follow the structure of a computer, even a parallel one. Trying to understand how they process data and how their approach can be implemented in silicon-based systems can open new horizons.

For example, to solve the power issue, reversible computing offers the theoretically ultimate answer. Neural systems are highly parallel systems but they do not require a parallel computer language to perform useful tasks. Similarly, drastic technology constraints for CMOS architectures are often seen as a difficult if not deadly issue for the computing community. However, they should also be considered as a tremendous opportunity to imagine drastically different architectures, to shift to alternative technologies, and to start designing systems for radically different purposes than just computing.

Alternative reasoning need not be restricted to the elementary computing elements; it can also apply to the systems themselves.

On the one hand, researchers from the architecture/programming domain are too often solely focused on performance, and they often miss application opportunities where they could leverage their knowledge for novel applications. For instance, architects could have anticipated way in advance when cost-effective hardware would be capable of performing real-time MPEG encoding, leading to hardware-based video recorders. There probably exist countless further applications that researchers from our or other domains could anticipate.

On the other hand, systems can do far more than compute tasks. Distributed control and collective behavior could breed self-organizing and self-healing properties. Such systems can be used for surveillance applications, as in so-called smart dust or smart sensors, for improving the quality of life or work in smart spaces - smart town, building, room - or for 3D rendering (e.g., Claytronics) and a vast range of yet unforeseen applications, and propose an entirely different approach for system design, management and application.

We also need to think differently about synergies between different technologies, and interfaces between them. For example, the Human++ could pave the way of interfacing biological carbon-based systems with silicon-based sensors or processing modules.

Impact on the applications

In this section, we discuss the potential impact of the directions and paradigms presented in the HiPEAC vision on the future applications, to determine how this vision can help to enable said applications.

Domestic robots

As discussed before, domestic robots will perform a myriad of tasks, which will differ from user to user, from room to room, from time to time. Important parts of the tasks will likely be artificial intelligence and camera image processing. These have to happen in real time for safety and for quality of service reasons. This requires very high performance systems. Furthermore, to increase the autonomy of the robot, the processing needs to be power-efficient. That will imply, amongst others, that depending on the particular situation and task of the robot, less or more complex image processing has to be performed. As indicated before, such power-efficient processing capabilities can only be delivered through heterogeneous, many-core computing devices. The proposed vision makes this possible as follows:

1. Domain-specific programming languages enable the AI developers and the image processing developers to operate most efficiently within their own domain without requiring them to have a deep understanding of the underlying hardware and the underlying design-time or run-time software support.
2. Having time-aware languages that support the notion of concurrency rather than parallelism will further increase their efficiency. Improving the tool chain's ability to specialize the program to each target and execution context will also help.
3. The use of virtualization will enable programmers to develop independently of specific hardware targets, thus enlarging the market for the developed software.
4. As such, the development of domestic robot software becomes more efficient, up to the point where the development of niche applications for very specific circumstances (that would otherwise imply too small markets) becomes economically viable.
5. By enabling the design of programmable computing components that support the same virtual bytecode interface, these components can easily be composed into many-core distributed robot processing systems. The result is that a de-verticalized market for robots is created in which robot designers can easily combine components, up to the point where robot extensions become available that are add-ons to basic robot frameworks.
6. This creates a larger market for robot components, and allows specific robots to (1) be designed for specific environments, (2) to be adapted cheaply to changing environments such as people that move to different locations or live longer.

7. The availability of multiple components that support the same interface, albeit at different performance levels for different applications or application kernels, enables the run-time management to migrate critical tasks from failing components to correctly operating components, thus increasing the reliability of the device and offering a graceful degradation period in which the luxury functionality of the devices might be disabled, but in which life-saving functionality is still operating correctly.
8. With the run-time techniques proposed in this vision, the robot will be able to optimize, at any point in time, its computing resource usage for the particular situation at hand. Because of virtualization and run-time load balancing techniques, a minimal design can be built that switches dynamically between different operating modes in time (time-multiplexing so to speak) without needing to be designed as the sum of all possible modes. Moreover, adaptive self-learning techniques in the robot can optimize its operation over time as it learns the habits of the people it is assisting.

As a result, software designers, hardware designers and robot integrators can achieve higher productivity in designing and building robots as well as being able to target and operate in larger markets. At the same time the resulting designs will be cheaper for end users, both in terms of buying cost and in terms of total cost of ownership, and they will provide longer autonomy and higher reliability without sacrificing quality of service. Without the directions and paradigms proposed in this vision, it is hard to imagine such an evolution.

The car of the future

Today's cars already contain numerous processors to run numerous applications. Top-end cars contain processors for engine control and normal driving control, processors for active safety mechanisms such as ABS (anti-lock braking systems) or ESC (electronic stability control), processors for car features such as controlling air-conditioning, parking aids, night vision, windows and doors, processors for the multimedia system including GPS, digital radio, DVD players, ... In current designs, these applications are isolated from each other by running them on separate processors. Clearly, this is a very expensive, inflexible solution, which does not scale.

When more and more electronic features will be added in the future, the software of those applications will be executed on much fewer processors, each running multiple applications. Some of these processors will run safety-critical software in hard real time, while others will run non-critical, soft real-time software.

Both the design of these processors and the design of the software running on top of them will benefit from the technical paradigms presented in this vision. As with domestic robots, hardware and software reuse will be improved, as will the productivity with which they are designed and implemented, for example by allowing domain experts to use their own domain-specific programming languages. We expect that open platforms will be created based on different aspects of this vision, that will result in multiple cars with a wide range of supported (luxury) features.

Such platforms that facilitate the combination of different software components for design-time differentiation of built cars will also facilitate updates to the software during a car's lifetime. It can be expected that during a car's lifetime, developments in software-controlled applications such as engine efficiency or automatic traffic sign recognition will occur. As an example of this, consider the optimization of the Toyota Prius engine control by means of recurrent neural networks developed by Prokhorov [Prokhorov]. This improved the fuel efficiency of the Prius with 17%, using a simple software update.

The different design-time and run-time tools outlined in this vision will enable maintainers to perform updates fully automatically or semi-automatically. In the latter case, driver input can be taken into account, e.g., to prioritize the non-critical applications that are available but cannot be installed together.

Another step is to combine safety-critical real-time applications and non-critical applications on the same processors. Virtualization can play an important role here, to isolate different applications from each other and to guarantee real-time performance for those applications that need it.

Telepresence

Many questions about how telepresence systems will operate in the future are currently unanswered. Will systems be based on thin clients with very limited processing power or on more expensive and powerful fat clients? How much processing will be carried out on centralized servers? Maybe the market will slowly evolve between different systems. Maybe multiple systems will co-exist, for example with one system for the consumer market and another for the professional market, which has different quality requirements. Alternatively, service providers could provide different quality levels to different consumers, which require different types of client devices and different amounts of centralized processing. In short, many different approaches are likely to co-exist over time.

Developing the necessary hardware components and devices that can handle the processing demands of telepresence systems, as well as the necessary software that runs on top of them will be too expensive if that hardware and software can only be used in specific systems with specific setups and operation modes.

The HiPEAC vision provides adequate means to avoid this problem, as it proposes strategies that enable developing software independently of the specific hardware setup, and provides the means to develop components that can be used in a wide range of systems. Furthermore, the run-time techniques for managing software running on hardware components such as virtualization, self-observation / adaption / checking / monitoring, etc. will enable load-balancing between client-side computing and centralized computing on servers, thus easing the support for a multitude of business models and service levels for different users.

Aerospace and avionics

Postponing many decisions to flight-time in order to optimize the efficiency of routes and procedures, seems to make it harder to validate the decision making process and to prove it correct and safe, and hence it will make it harder to certify new designs.

However, by allowing the developers of that decision process to work with domain-specific tools and by allowing them to develop for a virtual platform, that does not change over time and remains the same for all plane designs, the validation and certification will become simpler and more cost-effective. Moreover, this might allow for simpler decision processes to be validated and certified early on during the lifetime of an airplane, and more complex ones later on. This is fundamentally not all that different from the engine control of the Toyota Prius being updated when it enters the dealer's garage for maintenance, albeit the safety criteria being stricter for aerospace and avionics. Also, giving the developers a means to express the time parameter in the description of their systems will further enhance the predictability and safety of the system when used in combination with appropriate validation and mapping tools.

Furthermore, it might also allow airplane designers and builders to replace individual components by other, improved ones during the airplane's lifetime, which would then save large amounts of money, as no large stacks of original components need to be stocked for long periods of time.

For space missions and devices that get launched into space, the vision supports the assembly of devices from components that can more easily be reprogrammed and reconfigured. As such, the individual hardware components can serve to some extent as backups for each other, and redundancy can be implemented at the system level, where it can be done more efficiently than at the individual component level. The whole-system EDA tools that perform the vertical integration and whole-system optimization will take care of this.

Human++

As with domestic robots, implants in human bodies and extensions to those bodies will have to operate under a variety of circumstances, performing a wide range of tasks. Those circumstances and tasks depend on the patient at hand, on his or her disease, handicap, job, etc.

Developing specific solutions from scratch for each patient is not economically feasible. Still, all solutions have to be very energy efficient in order to increase their autonomy and limit heat emission. In advanced uses, one may design systems capable of simulating the behavior of millions of neurons in real time under tight resource constraints. Such challenges will feed a never-ending quest for performance/Watt and performance/Joule, leveraging very specific and multi-disciplinary domain knowledge.

Reuse and customization, both of hardware and software designs, and optimizations late during the design, i.e., when specific combinations of hardware and software have been created for specific patients, are therefore paramount. Clearly the HiPEAC vision supports such productive designs and assembly of components into customized systems. Furthermore, adaptive components, either in hardware or in software, will enable adapting to changing patient conditions, e.g., to learn patient-specific brain functioning and the appropriate responses to patient-specific inputs.

Computational science

Just like datacenters, supercomputers are composed of components (containers, racks, blades, interconnects, storage, cooling units, etc.). At this point there is not much difference to traditional datacenters. The biggest difference is in the workload, which is a single application in case of a supercomputer.

Given the nature of these workloads, most programmers are currently working at the efficiency layer as performance is the only metric that really counts in supercomputing. However, also in this area, there is a clear need to look for more abstract domain-specific frameworks and toolboxes for expressing the algorithms that need to be executed. Such toolboxes make the algorithms more portable between different systems, they speed up program development, and they hide the intricacies of parallelizing computational kernels. Current models such as MPI are too low level, and therefore inadequate to deal with future exascale systems with millions of cores, especially when several of them fail during the execution of an application.

We expect that, according to the principles and paradigms of this HiPEAC vision, future domain experts will be able to practice computational science within their own domain. Today's scientists either need to become domain experts in parallel programming languages themselves or they need to rely on the limited capabilities of software toolboxes that were programmed by their colleagues to solve particular problems on particular hardware platforms. In the future, they will instead be able to write new applications in their own domain-specific language. Next, tools developed by the HiPEAC community will make sure these applications run well on the exascale computers that this community will also develop.

As a result, computational science will have a much more gentle learning curve for scientists in many other disciplines. Consequently, this domain will open up to many more scientists and it will be able to evolve at a much faster rate, not being slowed down by the huge efforts it currently takes to port existing scientific code bases to new platforms or new applications. An example of a relatively novel new application is financial risk analysis. Many other new applications will follow. That way, this vision will help growing the field of computational science.

Smart camera networks

Smart camera networks can be used for a large variety of monitoring tasks being performed under varying conditions. Also, the tasks and hence the applications running on the individual cameras might change at deployment time.

It is likely that different applications will feature different sub-algorithms, so-called software kernels, featuring different kinds of concurrency. Hence different hardware designs are optimal for different applications. However, designing hardware components such as individual cores and accelerators that will only be used for one (niche) smart camera application is economically infeasible. Likewise, writing software kernels that will only be used in one application is very expensive, in particular if this has to be redone for each possible accelerator design.

The HiPEAC vision of using virtualization will increase both the market for developed software and the market for developed hardware components. It will also make life easier for the smart camera network maintainer, as it will allow him to add new cameras to a network of different manufacturers and with different features, as long as they support the same virtual interface.

Moreover, the reconfiguration, customization and run-time adaptation techniques will facilitate the switching between tasks during the deployment of smart camera networks.

Realistic games

At least some future games will involve multiple devices, with differing computational power and different functionalities. These devices might also be running other applications that have to be kept isolated from games, for example because of security reasons. Consider, e.g., devices accessing mobile communication networks and running downloaded game software. Obviously, the network operator does not want his network to be vulnerable to incursions by the downloaded software.

Moreover, games will have to run on a much wider range of hardware devices. Whereas today's games are programmed for a single platform such as Microsoft's Xbox, Sony's Playstation 3, or the Nintendo DS, or where their implementation involves a very large porting effort to target multiple platforms, the HiPEAC vision supports more productive programming with portable performance. Virtualization, domain-specific programming languages, and component-based hardware design. Consequently it will help to create a larger, more competitive market for gaming devices and games.

As entertainment in general and gaming in particular has always been a technology driver, we expect this larger, more competitive market to benefit other markets and technology progress as well.

3. Recommendations

Before indicating research objectives, we present a SWOT (Strengths, Weaknesses, Opportunities, and Threats) analysis of Europe's ICT industry and research. The results from this analysis, will assist in shaping future research objectives.

Strengths

During the past decades, the European ICT industry has created a strong embedded ecosystem, which spans the entire spectrum from low power VLSI technologies to consumer products.

In the semiconductor and processing elements field, companies such as ARM, ST, NXP, Infineon, etc. are leading companies in the domain of embedded systems, and have very strong presence in the European and worldwide embedded market. Validation and real-time processing are aspects in which the European industry has particularly excelled.

At the end of the value chain of this ecosystem, large end-user European companies have a strong market presence in different domains such as in the automotive industry (Volkswagen, Renault-Nissan, Peugeot-Citroën, Fiat, Daimler, ...), the aerospace and defense industry (Airbus, Dassault, Thales, ..) and the telecommunication industry (Orange, Vodafone, Nokia, Sony Ericsson, …). These large industries heavily depend on and influence the technologies produced by the semiconductor and associated tools industries. They also rely on a strong portfolio of SMEs that strengthen the technical and innovative offers in the market.

Weaknesses

European Computing Research is characterized by a weak link between academia and industry, especially at the graduate level. Companies in the United States value PhD degrees much more than European companies which often favor newly graduated engineers over PhD graduates. This leads to a brain drain of excellent computing systems researchers and PhD graduates trained in Europe to other countries where their skills are valued more, or to different economic sectors like banking. As a consequence, some of the successful research conducted in Europe ends up in non-EU products or does not make it into a product at all.

From an industrial point of view, Europe lacks very visible truly pan-European industrial players in the HiPEAC domain, especially compared to the USA. This severely reduces the potential synergies and impact of these industries. Furthermore, the European ICT industry misses a major high-performance worldwide general-purpose computing company such as HP, Intel or IBM in the USA. Main components for general-purpose computers, such as microprocessors, GPUs, and memories are also produced outside Europe.

At the research level, European research in computing systems is lacking international visibility due to the absence of a sufficient number of highly visible computer engineering departments. Furthermore, several major and competitive computing systems conferences are mainly controlled by American universities who use them as a tenuring mechanism for their own graduates, making it more difficult for Europeans to get their work published there.

The lack of open source tools in the computing systems domain (for example synthesis tools) is a weakness of European research in the HiPEAC domain. Hardware development is missing the same kind of ecosystem that exists for the open source software, which allows small groups, start-ups, universities and individuals to have a significant contribution to the innovation in the hardware domain: open source CAD tools are not widely usable, FPGA validation platforms are expensive and not easily available and testing ideas on real silicon is still a marathon that also requires solid financial background.

All these weaknesses are linked together: perhaps because computing systems is not considered as a strategic domain, no truely pan-European company in this field has emerged. This may explain the lack of European industrialization of European research results and the weak links between industry and universities. Consequently, Europe lacks internationally visible computer engineering departments.

It is also worth noting that the language diversity in Europe is a handicap to attract bright international students to graduate programs. Furthermore, the lack of command of the English language by graduates in some countries is also hampering international networking and collaboration.

Opportunities

As paradoxical as it may appear, several challenges that society is facing are at the same time also huge opportunities for the research and industry in ICT. For example, the aging population challenge will require the development of integrated health management systems and of support systems that allow people to stay longer in their home. The European expertise in low-power and embedded systems and its SME ecosystem is an asset for tackling other grand challenges like environment, energy and mobility.

Disruptive technologies such as cloud computing and convergence of HPC and embedded computing represent huge opportunities for Europe. The trend of more distributed systems, integrated in the environment using a mix of technologies such as the "More than Moore" approach, could be beneficial to the European semiconductor industry, which has a lot of expertise in the wide range of required technologies.

The cultural diversity of Europe creates opportunities for Europe in a global world that will not necessarily be dominated by non-European companies and institutions anymore. European companies are more sensitive to cultural differences that might become important in developing new markets all over the world.

From an educational perspective, it is worth noting that, as of 2008, 210 European universities are rated among the top 500 universities in the Shanghai Jiao Tong University ranking [ARWU], this is more than the United States of America (190 universities). The European university system thus benefits from a very strong educational taskforce and a highly competitive undergraduate and graduate educational system. Additionally, European research traditions and different educational policies installed at national levels and at the European level help with establishing longer-term research as well as a stronger analytical approach in the ICT research area. The ongoing bachelor-master transformation will hopefully further strengthen the European educational system.

Finally it is worth noting that the proximity of Europe to the Middle East, the Russian Federation and Africa represents a huge market opportunity and should not be neglected.

Threats

The labor cost as well as the inertia caused by administrative overhead and IP regulations significantly hampers the European industry.

Currently most, if not all, high-end and middle-end general-purpose processor technology is developed in the USA. China is also developing its own hardware, of which the Loongson processor is the best-known example. With the development of low-power processors such as the Intel Atom in the USA, Europe risks ending up without any semiconductor industry left, neither in the high-performance nor in the embedded domain.

At the political level, Europe does not consider computing systems a strategic technology, unlike other technologies such as energy, aerospace and automotive technology. We should not forget that most other major economies treat computing systems as a strategic technology, even under control of national security agencies as in the USA. Computing systems technology is at the basis of almost all other strategic areas, including defense equipment and satellite control. Export restrictions could one day limit European ambitions in these areas, especially if Europe would become completely fabless.

The lack of venture capitalist culture and policy contributes to the brain drain: it is much harder for a PhD graduate in Europe to attempt to build his own startup to industrialize the results of his research. More generally, bureaucracy and administrative procedures in some countries are preventing or killing several new initiatives. As a result, Europe's big industry tends to follow rather than to lead as far as new opportunities are concerned.

The language diversity in Europe is a handicap to attract bright international students. Of those that come, many will return to their home country after graduation. As European students increasingly lack interest in computing, the European companies will have more difficulties to hire top talents. Furthermore, the lack of command of the English language by graduates in some countries is also hampering international networking and collaboration.

Research objectives

The HiPEAC vision is summarized in Figure 6 and Figure 7.

We believe that in order to manage the complexity of future computing systems consisting of hundreds of heterogeneous cores, we should make a distinction between three groups of stakeholders. End users who are buying hardware and software for example in a store or on the Internet are by far the largest group. For them, installing and using hardware and software should be just plug-and-play, completely hassle-free. They should be completely oblivious of the kind of hardware and software they are using. This should be comparable to the type of alloy used in the engine of a car, undeniably very important for the car manufacturer, but infinitely less important for the end-user than the features of the in-car entertainment system. For the end user, there is no distinction between hardware and software, there is only the system.

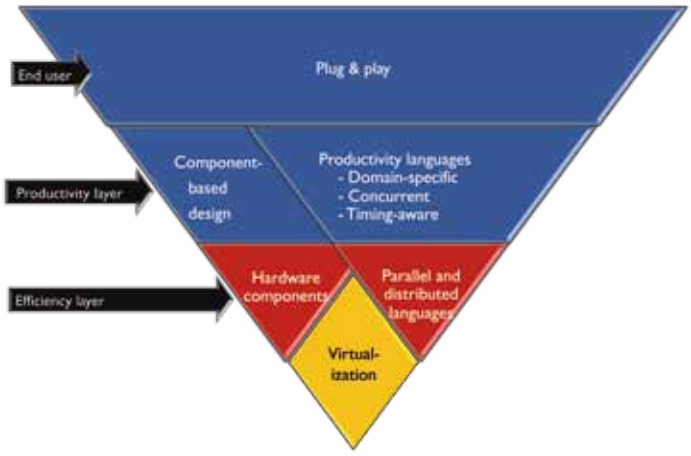


Figure 6 Productivity and efficiency layers in hardware and software design

The second group is working at the productivity layer; these are the product designers who mostly care about correctness, but less about the non-functional properties of a system. For this group, design time and time to market are the most important criteria once design constraints (e.g. power, real-time) have been met. The faster a correctly working system can be built, the better. The magic word at this level is abstraction. The more we can abstract the low level details of the implementation, the better. At the software level, we radically propose the use of domain-specific languages that enable expressing concurrency and timing in a way that is familiar to the designer. At the hardware level we propose the use of component-based hardware design, from the transistor level to the rack level. This will lead to less optimized systems, but it will dramatically reduce the complexity of the design, and therefore improve the time-to-market of the product.

Finally, there are the engineers working at the efficiency layer. At the hardware level, they are implementing the (optimized) building blocks for the component-based design. This hardware

will be able to adapt itself, for example by switching off unused parts and by migrating activity across the systems to avoid hot spots or to deal with failing components. At the software level, the engineers are designing parallel and distributed programming languages that are to be considered the machine language in the multi-core era. They also take care of the runtime systems and virtual machines. One of the major challenges for software is portable performance, meaning that platform-neutral software adapts itself to the hardware resources available on a given platform.

The main research focus of the HiPEAC community is on the efficiency layer. It also produces some of the tools for the productivity layer. Of course, it also uses its own productivity tools when working on the basic components of the efficiency layer.

This HiPEAC vision can be realized by the use of domain-specific, concurrent, and timing-aware systems, component-based hardware and software design, self-adaptation and portable performance. The use of these techniques leads to shorter design cycles but this does not come for free: the resulting systems may be less-than-optimal. To compensate for this, we propose to use global optimization techniques that eliminate the overhead from the extra abstraction layers and from additional interfaces.

In order to realize the HiPEAC vision, we propose six research objectives. They all take the technology trends into account, and support the HiPEAC vision. They are described in more detail below.

Design space exploration

Design space exploration is about automatically optimizing a system for non-functional metrics as listed under the challenges. Design space exploration searches for the best design point in a high-dimensional design space. The dimensions of the design space can be either parametric (such as cache size), or structural (such as the number and types of cores). Design space exploration is a global optimization technique that can automatically generate optimized domain-specific solutions. Effective design space exploration should not only explore the hardware design space, but also the software design space (a different hardware architecture might require a different algorithmic solution, or different compiler optimizations).

Key issues are:

- Design space exploration for massively heterogeneous multi-core designs, i.e. selecting the optimal heterogeneous multi-core system for a given workload. This requires modular simulators, and a parametric and structural design space.
- The development of efficient search strategies in combinatorial optimization spaces, and the building of predictive models to guide the search.
- Combined hardware/software exploration, i.e. support for co-evolution of hardware and software. Identifying the appropriate software design space, and the development of tunable compilers.
- Multi-objective optimization for two or more of the technical challenges, e.g., not only for best-effort performance but also for on-time performance.

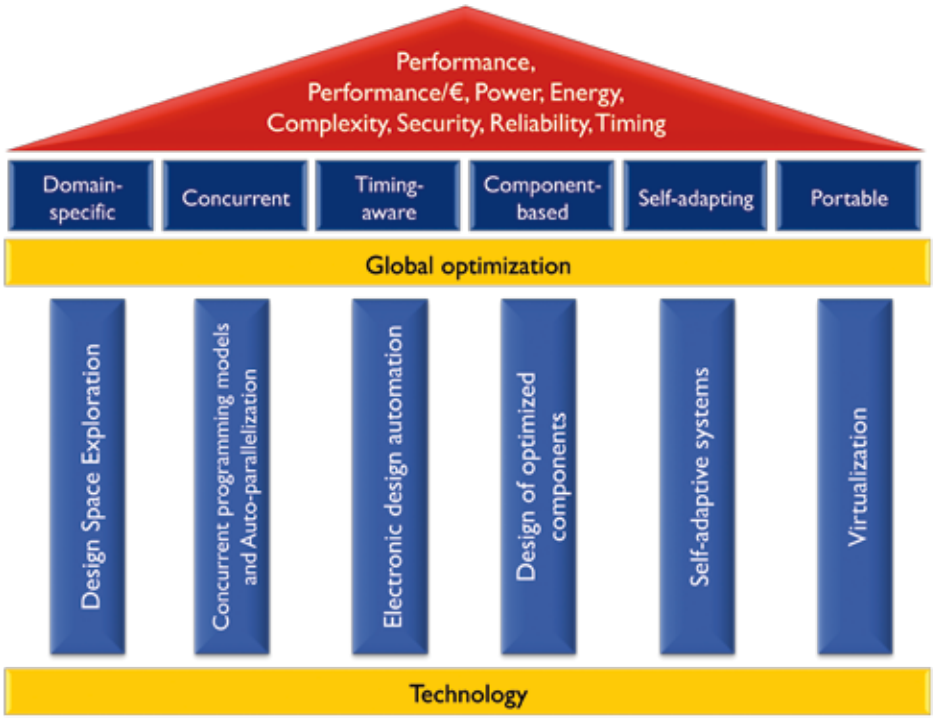


Figure 7: General recommendations and their relations

Concurrent programming models and auto-parallelization

The holy grail of the multi-core era is automatic parallelization of code. Rather than starting from legacy C code, we propose to start from platform-neutral domain-specific, timing-aware and concurrent languages. The auto-parallelizer must be able to convert concurrency into parallelism, and exploit the parallel resources that are available in a given hardware platform, effectively realizing portable performance.

The automatic mapping will be a two-phase approach, a.k.a. split compilation. The first, static, hardware-independent phase will extract concurrency information from the code and give feedback to the programmer about the available concurrency or lack thereof. The second, possibly dynamic, hardware-dependent phase, will then map that concurrency on the available parallel hardware. In this approach, the first phase is hardware-independent, but is not necessarily independent of the second phase. Depending on the tools or mapping techniques that will be used in the second phase, the first phase might need to extract different kinds of information.

Key issues are:

- The design of truly platform-neutral concurrent, domain-specific, timing-aware languages. Although not per se a HiPEAC activity, language designers might need our help to come up with concepts that are amenable to parallelization.
- The design of a tool flow that allows the extraction of all necessary concurrency information to exploit all possible parallelism. The static first phase of the split compilation needs to be made retargetable to the dynamic second phase.
- How to give to programmers the most useful feedback concerning the concurrency in their applications.
- The development of second-phase techniques for automatically mapping concurrency to a multitude of parallel hardware structures, including reconfigurable fabrics, graphical processing units, and accelerators of all kinds. Portable performance.

Electronic Design Automation

Component-based design requires tools that enable productivity designers to compose their design starting from a high level functional description. EDA technology is a key factor in reaching higher design productivity of future heterogeneous multi-core systems.

EDA is currently aiming at a new abstraction level: Electronic System Level (ESL). ESL focuses on system design aspects beyond RTL such as efficient HW/SW modeling and partitioning, mapping applications to MPSoC architectures, and ASIP design.

Key issues are:

- Component-based design, from the basic building blocks up to the complete datacenter.
- Accurate and fast evaluation of performance, power consumption and temperature of the resulting system.
- Manageable simulation, validation and certification time.
- Automatic generation of hardware accelerators from high-level specifications.
- The design of self-adaptive systems.

Design of optimized components

Component-based design can only be productive if it can build upon an extensive set of well-designed and fully-debugged components. In the hardware domain, they are called IP-blocks; in the software domain, we call them libraries. These components should on the one hand be optimized for the function they were designed for, and on the other hand they should be general enough to be applicable in a wide range of applications. This dilemma might lead to suboptimal solutions, which is the price one has to pay for a faster time to market.

Key issues are:

- General-purpose processor architecture: optimization for power and reliability.
- Correct selection and architecture of domain-specific accelerators.
- Improvements of the memory architecture.
- New components interconnection systems.
- Efficient reconfigurable architectures.

Self-adaptive systems

Three aspects of future computing systems will show variability over time and space. The available hardware will vary because of wear-out, process variability, reconfiguration and monitoring local heat production. Furthermore, the environment in which the system operates will change. Physical properties, such as temperature, will change and affect the operation of the devices, as well as other properties that form inputs to the applications running on the devices, such as changing light conditions around a smart camera. More virtual changes will also occur, such as when previously undisturbed systems become the target of a security invasion. Furthermore, we have seen many applications where the applications themselves, i.e., the software running on the devices, changes because different functionality is needed at different points in time.

Since optimizing these computing systems for all worst-case scenarios of the three aspects is not feasible, we have to start developing systems that adapt dynamically to changing conditions. This requires a large investment in methodologies and tools.

Key issues for these methodologies are that they should support

- An integrated approach for all three kinds (hardware, software, environment) of changing variables.
- System-wide approaches for global adaptation and optimizations rather than local adaptation and optimization.
- Appropriate split between static compilation phases and dynamic, adaptive phases.

Virtualization

Virtualization is a basic technique that separates workloads from the physical hardware. It allows for running legacy software on new hardware, for dynamically adapting applications to changing hardware resources, and for isolating software domains (to do dedicated resource provisioning, or for security).

Key issues are:

- Efficient virtualization of heterogeneous multi-core systems, or how to create a virtual architecture for a multitude of heterogeneous platforms, including accelerators. Modular virtualization frameworks.
- Performance models for virtualized workloads, essential for, a.o., scheduling virtualized workloads. Hardware/software support for dynamic instrumentation, monitoring and optimization.
- Real-time guarantees in virtual environments, validation, certification.

Conclusion

This document describes the HiPEAC vision. It starts by listing the grand societal challenges, the application and business trends, and the ten technical constraints ahead of us:

1. Hardware has become more flexible than software;
2. Power defines performance;
3. Communication defines performance;
4. ASICs are becoming unaffordable;
5. Worst-case design for ASICs leads to bankruptcy;
6. Systems will rely on unreliable components;
7. Time is relevant;
8. Computing systems are continuously under attack;
9. Parallelism seems to be too complex for humans;
10. One day, Moore's law will end.

These lead to technical challenges that can be summarized as improvements in seven areas: Performance, Performance/€ and performance/Watt/€, Power and energy, Managing system complexity, Security, Reliability, and Timing predictability.

From these challenges, trends and constraints follows the HiPEAC vision: keep it simple for humans, and let the computer do the hard work. This leads to a world in which end users do not have to worry about technicalities of platforms, where 90% of the programmers and hardware designers only care about productivity in designing software and hardware, and were only 10% of the trained computer scientists have to worry about efficiency and performance.

Systems will be heterogeneous for performance and power reasons, and computers will be used to specialize and optimize the system beyond the component level.

Besides the tasks for the humans, computers will do the hard work of searching for a good enough system architecture through design space exploration, generating it automatically using EDA tools, automatically parallelizing applications written in domain-specific languages, and make sure the system can automatically adapt to varying operating conditions.

Finally, the vision also reminds us that one day scaling will end, and that we should be ready by then to continue advancing the computing systems domain. Therefore it is suggested to start looking into upcoming alternatives, and to start building systems with them, in order to be ready when needed.

The vision concludes with a set of recommendations, areas in which research is needed to support the HiPEAC vision. These areas are, in no particular order: adaptive systems, concurrent programming models and auto-parallelization, the design of optimized components, design space exploration, electronic design automation, and virtualization.

This document does definitely not offer "silver bullet" solutions for the identified problems and challenges, but it does offer a number of directions in which European computing systems research can progress.

The described vision has been created by and for the HiPEAC community. By working in accordance with this common vision, European collaboration will become the most natural option for computing systems research. This vision can also focus the European research capacity to a smaller number of research objectives, thereby creating communities with enough critical mass to force real breakthroughs in the different areas.

References

[AMD] AMD Supercomputer To Deliver Next-Generation Games and Applications Entirely Through the Cloud available at http://www.amd.com/us-en/Corporate/VirtualPressRoom/0,,51_104_543~129743,00.html

[Asanovic2006] Asanovic, Krste and Bodik, Ras and Catanzaro, Bryan Christopher and Gebis, Joseph James and Husbands, Parry and Keutzer, Kurt and Patterson, David A. and Plishker, William Lester and Shalf, John and Williams, Samuel Webb and Yelick, Katherine A. The Landscape of Parallel Computing Research: A View from Berkeley, EECS Department, University of California, Berkeley, 2006.

[Bekey2008] The Status of Robotics, Bekey, G.; Junku Yuh; Robotics & Automation Magazine, IEEE Volume 15, Issue 1, March 2008 Page(s):80 - 86

[Blaauw2008] David Blaauw, Sudherssen Kalaiselvan, Kevin Lai, Wei-Hsiang Ma, Sanjay Pant, Carlos Tokunaga, Shidhartha Das, David Bull, "RazorII: In-Situ Error Detection and Correction for PVT and SER tolerance," IEEE International Solid-State Circuits Conference (ISSCC), February 2008

[Borkar2004] Shekhar Y. Borkar: Microarchitecture and Design Challenges for Gigascale Integration.37th Annual International Symposium on Microarchitecture (MICRO-37 2004), 4-8 December 2004, Portland, OR, USA. IEEE Computer Society 2004, ISBN 0-7695-2126-6

[Borkar2005] Shekhar Y. Borkar: Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. IEEE Micro, 25(6):10–16, 2005.

[CEATEC2008] Richard Bergman, AMD HD graphics technology accelerates the convergence of Digital Consumer Electronics and PCs, CEATEC 2008, October 2008. http://gl.ict.usc.edu/Research/DigitalEmily/ or http://technology.timesonline.co.uk/tol/news/tech_and_web/article4557935.ece

[Cisco] http://www.cisco.com/en/US/netsol/ns340/ns394/ns430/index.html

[Cuda] http://www.nvidia.com/object/cuda_develop.html and http://www.khronos.org/opencl/

[Dean2004] Jeffrey Dean, Sanjay Ghemawat: MapReduce: Simplified Data Processing on Large Clusters. OSDI 2004: 137-150

[Ernst2004] Daniel Ernst, Shidhartha Das, Seokwoo Lee, David Blaauw, Todd Austin, Trevor Mudge, Nam Sung Kim, Krisztian Flautner. "Razor: Circuit-Level Correction of Timing Errors for Low-Power Operation". IEEE Micro, 24(6):10-20, November 2004.

[ESA] http://www.theesa.com/newsroom/release_detail.asp?releaseID=44

[ESIA2008] Mastering Innovation Shaping the Future, ESIA 2008 Competitiveness Report, ESIA European Semiconductor Industry Association, 2008.

[FCOT05] Grigori Fursin and Albert Cohen and Michael O'Boyle and Oliver Temam, A Practical Method For Quickly Evaluating Program Optimizations, Proceedings of the 1st International Conference on High Performance Embedded Architectures & Compilers (HiPEAC 2005), LNCS 3793, pages 29-46, 2005.

[Gartner08] Gartner, Inc, Gartner Identifies Seven Grand Challenges Facing IT, April 2008.

[GC3] GC3 in Grand Challenges in Computing Research 2008, available at http://www.ukcrc.org.uk/grand_challenges/index.cfm

[Grandcentral] http://www.apple.com/macosx/snowleopard/

[ISTAG] Shaping Europe's Future through ICT, ISTAG, March 2006.

[ITRS] International Technology Roadmap for Semiconductors, http://www.itrs.net/Links/2007ITRS/LinkedFiles/AP/AP_Paper.pdf

[Katz2009] Randy H. Katz, Tech Titans Building Boom, IEEE Spectrum, 46(2):40-54, Feb 2009.

[Lee2006] Edward A. Lee, The Future of Embedded Software (Powerpoint presentation) May 22-24, 2006, Artemis Annual Conference, Graz, Austria. Available at http://ptolemy.berkeley.edu/presentations/index.htm

[Mead89] Mead, C. 1989 Analog VLSI and Neural Systems. Addison-Wesley Longman Publishing Co., Inc.[MtM] Innovations in the 'More than Moore' era, René Penning de Vries, EE Times Europe, 06/30/2009. http://www.eetimes.eu/218102043

[Muller2004] C. Müller-Schloer, C. von der Malsburg, R. P. Würtz: Organic computing. Informatik Spektrum, 27(4):332–336, 2004.

[Nota] http://www.notaworld.org

[OpenCL] http://www.khronos.org/opencl/

[Palem05] Palem, K. V. 2005. Energy Aware Computing through Probabilistic Switching: A Study of Limits. IEEE Trans. Comput. 54, 9 (Sep. 2005), 1123-1137.

[Patterson2008] David Patterson, Parallel Computing Landscape: A View from Berkeley, keynote at SC08, November 2008.

[Pfister2007] Gregory Pfister, IPDPS 2007 Panel Position: Is the Multi-Core Roadmap going to Live Up to its Promises? IDPDS, 2007.

[Prokhorov2008] D. Prokhorov, Toyota Prius HEV neurocontrol. In proceedings of the International Joint Conference on Neural Networks, 2007, p. 2129 - 2134.

[Schmeck2005] H. Schmeck: Organic computing – A new vision for distributed embedded systems. Proc. of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2005), IEEE CS Press, 201–203, 2005.

[Streit2005] Norbert Streit, Paddy Nixon, The disappearing computer, Communications of the ACM March 2005/Vol. 48, No. 3 33-35.

[Vas97] Cotofana, S., Vassiliadis, S. 1997. Low Weight and Fan-In Neural Networks for Basic Arithmetic Operations. In 15th IMACS World Congress 1997 on Scientific Computation, Modelling and Applied Mathematics, volume 4 Artificial Intelligence and Computer Science, 227—232

[Velliste2008] Meel Velliste, Sagi Perel, M. Chance Spalding, Andrew S. Whitford and Andrew B. Schwartz, Cortical control of a prosthetic arm for self-feeding, Nature, 2008

[Vocaloid] http://en.wikipedia.org/wiki/Vocaloid

[Whener2008] Michael Wehner, Leonid Oliker, and John Shalf Towards Ultra-High Resolution Models of Climate and Weather International Journal of High Performance Computing Applications 2008 22: 149-165. or http://www.lbl.gov/Science-Articles/Archive/NE-climate-predictions.html

Acknowledgements

The authors are indebted to several people who contributed to this document over the last year:

- As reviewers: Mladen Berekovic, Christian Bertin, Angelos Bilas, Attila Bilgic, Grigori Fursin, Avi Mendelson, Aly Syed, Alasdair Rawsthorne.
- All HiPEAC clusters and task forces.
- The teachers and company delegates at the ACACES 2008 and 2009 summer schools.
- The whole HiPEAC community.
- And last but not least, the European Commission, which triggered and sponsored this work through the HiPEAC2 project (Grant agreement no: ICT- 217068).

info@HiPEAC.net
http://www.HiPEAC.net/roadmap