ISSN 2186-7437

**NII Shonan Meeting Report**

No. 2012-11

# The NII Shonan Configurable Computing Workshop

Peter Athanas
Brad Hutchings
Kentaro Sano

November 12–15, 2012


National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-Ku, Tokyo, Japan

# The NII Shonan Configurable Computing Workshop

Organizers:
Peter Athanas (Virginia Tech, USA)
Brad Hutchings (Brigham Young University, USA)
Kentaro Sano (Tohoku University, Japan)

November 12–15, 2012

## 1 Overview of the workshop

### Description of the Workshop:

Configurable computing is an emerging technology proving to be capable of providing high computational performance on a diversity of applications, including 1-D and 2-D signal processing, simulation acceleration, computer graphics, and high-performance computing. High performance is achieved by rapidly reconfiguring the functionality and interconnectivity of the computing resources to match the computational requirements of specific applications. Rapid reconfiguration provides the illusion of having a much larger (virtual) hardware platform. With this approach, specific application properties, such as parallelism, execution profile, and data resolution can be exploited by creating custom operators, pipelines, and interconnection pathways.

In recent years a rapidly growing interest in using reconfigurable computing architectures for realizing and developing application-specific computer systems has been observed. The advances in reconfigurable technologies, in algorithms for implementation approaches and in automatic mapping methods of algorithms into hardware and processor spaces form a new computing paradigm of computing and programming, e.g. "Computing in Space AND in Time ". This requires different and novel approaches in engineering for developing reconfigurable systems and implementing complex algorithms, including theory, architecture structures, algorithms, design systems and industrial applications that demonstrate the benefits of this promising way of computing. The fast pace of development is not leaving industry enough time to develop the necessary theoretical foundation that underpins CAD tools, OS, designs, architectures and circuit technologies. Traditional hardware and software design processes and the tools to support them are not adequate for the design of run time reconfigurable systems. Therefore, the plan for this seminar is to focus on the issues relevant to the development of support for run-time reconfigurable systems that can be attractive to industry. A special focus will be given to dynamically run-time reconfigurable (RTR) solutions, since system adaptation and the advantages of this technology are highly visible.

Additional topics in this area of research include productivity and observability. Programmable logic devices continue to nearly double in size with each new generation. Although computational power used to grow at this same rate, this is no longer true. As such, compilation times for programmable technologies are getting longer, not shorter. Lengthy compilation time is one of the main drawbacks to this technology and it continues to limit its applicability to many problems. It

is very difficult to debug complex programs when compile times are measured in hours. Current devices continue to lack sufficient debugging capabilities. Programmers must wait for another lengthy compile each time they need to look at a new set of signals. Because of a general lack of observability simulation continues to be the debugging tool of choice even though it is at least 1,000,000 times slower than execution on a programmable chip.

The seminar covers: (1) architectures and techniques that support dynamic reconfiguration, productivity and debug, (2) circuit technologies, (3) system architectures, (4) physical CAD tools, (5) tools to aid in the design of RTR systems, domain-specific systems and related compilers and new (6) application domains, particularly those that can effectively exploit dynamically reconfigurable architectures. We are also interested in (7) novel approaches that enhance productivity and (8) improve observability in ways that aid in debug and verification and that hold the promise to dramatically reduce time-to-market for commercial endeavors. In addition, now that Configurable System-on-Chip devices are beginning to appear, we are interested in attracting research that (9) develops tools and design strategies to exploit the novel aspects of these devices and that can continue to reduce overall development time.

For this forum, we invited experts from these various research areas to present their work and opinions. In particular, we encouraged these experts to candidly describe those key research problems that, until solved, continue to hinder advancement. The outcome of this workshop should identify key problems along with proposed technical approaches to attack these problems. We believe that researchers who attended this workshop would be able to not only help solve these problems but would also help to advertise the importance of these problems to the larger research community. The seminar that we held was an excellent opportunity to discuss the results from the mentioned projects as well as research areas with researchers from Japan, Europe, United States, Canada, Asia, and Australia.

## Interdisciplinary Seminar:

Researchers and practicing engineers alike need to be able to operate in interdisciplinary environments. Not only have boundaries largely disappeared between Computer Science and Electrical Engineering, for example, but we also see that the boundaries between computing and many other fields such as computational biology, chemistry, etc., are continuing to soften. Actual surveys in the EU Project MORPHEUS describe the increased request of the industry for multidisciplinary skilled engineers. The summary of the feedback can be reported like this: software engineers need to understand new hardware paradigms (reconfigurable computing) and hardware engineers need to understand that software defines the product at the end. Supporting this trend, new workshops like the International Workshop on Reconfigurable Computing Education (RC-Education, http://www.fpl.uni-kl.de/RCeducation08/) and the AETHER - MORPHEUS Workshop - Autumn School (AMWAS, http://www.alari.ch/ AMWAS08/) were established in order to bring together people from different research areas. But there is more activity required to bridge these gaps. Apart from the example described above, interdisciplinary skills needs to be introduced also for physicists, chip designers and hardware architecture specialists in order to master the challenges coming up with future complex electronic systems. This seminar brought together researchers from industry and academics with an excellent reputation and the required wide base of disciplines which targets all areas of interest for future reconfigurable architectures.

# 2 Overview of Talks

## Can Coarse Grained Reconfigurable Architecture (CGRA) survive?

Hideharu AMANO, Keio University, Japan

Recently, a lot of projects for CGRA were canceled especially in Japan. CGRAs are competitive with SIMD architectures including GPU. Although current CGRAs are advantageous from the viewpoint of energy consumption, the performance is much lower. This talk tries to find the way that CGRAs can survive as an acclererator in future SoCs or SiPs.

## Disconnection between academic research priorities and industry needs

Neil BERGMANN, University of Queensland, Australia

Most research in configurable computing fails to make a significant impact into the commercial domain of FPGA-based system design. After more than 20 years of research into dynamic partial reconfiguration, there are still no mainstream commercial applications of the technology, at least that I am aware of. This is not a reflection on the technical quality of DPR research ? some of the brightest technical researchers have developed sophisticated frameworks which have simplified the task of building DPR systems. With such a depth and volume of research, if there was a simple method for designing such systems, and a " killer app " that could make use of this method, then I think we would have found them. Large reconfiguration times and file sizes, and complex CAD tools might explain why DPR is not commercially widespread, but it does not seem to be enough to explain why it is non-existent in real products.

Today, one needs to question whether even more effort to make DPR simpler/faster/smarter is providing an answer to a problem that commercially nobody is asking. One can make similar observations about FPGA-based supercomputing. Despite decades of research on building highly parallel, general-purpose, stream-based application accelerators, and some impressive speedups, there does not seem to be widespread use of the technology except by configurable-computing researchers, and then mostly the outcome is research papers. The situation is a little better than in DPR ? There are commercial vendors of FPGA accelerator boards, but the applications (I believe) tend to be single-use embedded products requiring massive computation (cryptography, software-defined radio, radar processing). The best recent attempt to build a general, programmable FPGA supercomputer (Maxwell, in Edinburgh) has run its course, with only limited use. On the other hand, FPGAs are becoming ubiquitous in high-end electronic design, and designers still struggle with adequate tools and frameworks to design, build and test hardware/software systems. To me, this is where there is still substantial, commercially relevant research to be done. Here 's some questions still to be answered to enable mixed hardware/software design to be easy:-

1. What are appropriate abstractions for cooperating hardware and software? (processes, tasks, VMs, instructions, stream-based tasks).
2. What are appropriate interprocess communications methods for hardware/software ? (pipes, messages, postboxes, shared memory, MPI$\cdots$) How to automate these interfaces?
3. How to describe HW/SW systems using some generic framework before making partitioning decisions (C-code, System-C, Petri-nets, state diagrams, $\cdots$)
4. How to make these frameworks accessible to students and engineers? Tools, boards, text-books, lecture notes.

## Advancing Reconfigurable Computing Requires a System-Level Perspective

Paul CHOW, University of Toronto, Canada

Reconfigurable computing technology is struggling to gain acceptance in the broader computing community. The main reason is that the tools used to design applications are not easily accessible to the application experts that are developing the applications. Researchers have been working on tools for about 20 years and the tools are still difficult to use. Why is this the case?

There is a large impedance mismatch at the architecture/hardware and tools boundary. Researchers need to take a system-level approach to the problem. The tools and the architecture are all parts of the problem and they will be parts of the solution. Completely new approaches to developing better tools and architectures must be taken.

A good example of the benefits of taking new approaches is the development of *RISC Architectures*, which has led to the significant computing performance we have available today. What is often forgotten, because of the term *RISC*, is that the reduced instruction set is not the reason for the performance gains. Instead, dramatic performance gains were made as a result of a system-level analysis of how the architecture, software, hardware and implementation technologies available interacted and affected performance. The understanding how to leverage the strengths of each aspect of the system and make the best trade offs across the system has taken us to the computing performance achievable today.

Current reconfigurable computing research is in an *architectural rut*. We have been trying to build reconfigurable computing systems on top of a fabric that was designed to represent gates and interconnect. While this fine-grain technology can be used to build any circuit, it is extremely low-level and therefore results in the difficulties we currently experience: one must be a hardware designer to use it and the tools take a long time, neither of which is conducive to a computing environment.

Can we take another approach to discover a more suitable architecture for *Reconfigurable Computing*? Might this lead to a new computer architecture? Will it still be called *Reconfigurable Computing*? Can we break free of the *von Neumann* architecture that has brought us to where we are today, but is now showing its limitations?

The *RISC* revolution started with the following questions and observations: What are compilers capable of doing and how should the hardware complement that? What can the technology provide in terms of hardware? Compilers work best when they can generate fundamental operations where the optimizations are straightforward and the data operations can be managed with good register allocation. Simple instructions can be pipelined and use less gates. Pipelines make it possible to increase clock frequencies. Less gates leaves more room for caches. Decisions about features are made by quantitative analysis.

Note that the early FPGA architecture studies and VPR research at the University of Toronto were motivated by the above strategy.

Can we apply the same strategy for *Reconfigurable Computing*? What are the appropriate questions and observations that can set *Reconfigurable Computing* research down a new and fruitful path? To gather data that can be used to inform our decisions, it would be helpful to have an "input" language. For RISC processor design, the language was principally C. Or, do we start with the capabilities of high-level synthesis (HLS) as the starting point, much like we started with optimizing compilers for RISC architectures? Is HLS mature enough to define an architecture? If so, what does HLS do well, and does that suggest the architecture that will be an efficient target for HLS? Any discussion must include the HLS experts.

The system-level perspective for *Reconfigurable Computing* is a much larger problem than it was for *RISC Computing*. We must find a way to make the problem easier to understand and

attack.

There have been very few fundamental leaps in computer architecture. The last was RISC technology. Making *Reconfigurable Computing* work will be the next great achievement in computer architecture.

## Towards high level design tools

Rene CUMPLIDO, INAOE,Mexico

The design complexity of current digital hardware systems has grown at the same pace as the scale of integration of modern integrated circuits and the demand of functionality. This complexity gave rise to different design approaches at different levels of abstractions, which has resulted in a large number of methodologies and tools that were designed to exploit specific characteristics of available technologies. Current design processes are not straightforward; design engineers deal with the problem of having to create processing systems using a large variety of tools and methodologies.

Recently, there have been some efforts to develop design methodologies that describe the functionality of digital hardware systems using languages and tools similar to the ones employed to build software systems. The aim is to reduce complexity during the design of digital hardware systems by means of high level tools that are based on UML-like modeling techniques. It is too early to say if these methodologies and tools will achieve their goal of " standardizing "the design of custom processing systems, but as a community we would greatly benefit as it would allow complete design space exploration in far less time than current approaches.

## It's been said before: Future design methods need AI, and AI could learn from us

Oliver DIESSEL, University of New South Wales,Australia

A current technology trend sees ever more switches being integrated on-chip. Consequently, we will see ever-tighter on-chip integration of reconfigurable logic, processors, networks and memory. The number of custom design starts will continue to diminish as fewer, more generic, yet customizable platforms remain. The devices of the future will be distinguished by the relative amounts of undifferentiated resources and customized hard IP provided.

To effectively utilize such devices, more effective programming models and CAD tools are needed. Far from being in a rut, our community faces a decisive challenge to create methodologies that will allow systems to be engineered from the devices of the future. The solutions will need to be efficient, powerful, energy aware, flexible, robust, scalable, and cheap. My guess is that we will need to bring Artificial Intelligence to bear on solving this problem. And if we do so, we might help to advance AI.

## Using Reconfigurable Architecture as a Universal Computing Engine

Tetsuo HIRONAKA, Hiroshima City University, Japan

Using a reconfigurable architecture as a universal computing engine is a big challenge. To reach the goal for these challenge two points must be achieved.

First point is to introduce a good parallel programming model. Reconfigurable Computing will succeed only when unnatural sequential programming 's are thrown away and switched to the

natural parallel programming model. To achieve this, we need to push more power in software engineering for parallel programming.

The second point is to have a good method to evaluate the performance of a reconfigurable architecture. Because a well written parallel program that extracts application's parallelism highly, the amount of parallel hardware you have scale up the performance you can achieve. So just comparing absolute performance is not academic. The performance of the Reconfigurable architecture must be evaluated by the efficiency of hardware usage and scalability. However, to evaluate the efficiency of hardware, we need a clear method to evaluate the amount of the hardware used to achieve the performance. Which method is comprehensible enough to be used for comparing difference architectures, including non-reconfigurable one. When the reconfigurable architecture succeeds to triumph over the efficiency of the non-reconfigurable ones, it will become the true next-generation computer.

## Perspectives on the use of Hard-CPUs together with FPGAs

Andreas KOCH, Technische Universitat Darmstadt, Germany

In recent years, the use of CPUs integrated with FPGAs has been a topic of interest both for academic study and industrial use. After some early commercial devices such as Altera Excalibur and Xilinx Virtex II Pro, intervening generations have not supported on-chip hard processors (e.g., Xilinx Virtex 6 and 7, Altera Arria/Cyclone up to IV ). These devices relied on soft-core processors instead, achieving improved flexibility, but requiring more silicon area and power due to the use configurable resources. Only recently have devices with integrated hard-CPUs appeared again (Xilinx Zynq series, Altera Arria/Cyclone V). In my contribution, I present three examples for the use of reconfigurable devices and examine their impact on the use of FPGA-integrated hard CPUs.

The first example is HaLoMote, a wireless sensor node optimized for very low-power operation (possibly supplied by energy harvesting). In contrast to many other approaches employing pure-software solutions running on ultra-low-power processors such as the TI MSP430, the key application of the HaLoMote is structural health monitoring (SHM) of bridges or other large structures, where the distributed algorithms require sufficient compute capacity even in the nodes. Here, a processor such as the MSP430 is simply not able to provide the required peak compute performance. As a solution, we have employed a low-power 8b 8051-compatible processor on the radio chip to handle just the very simple communication protocol, but provide the compute capacity by a low-power Actel IGLOO FPGA that supports active power management. It is mostly left in a deep-sleep mode where it requires just microwatts to maintain state, and only awakens when a structure vibration event of sufficient magnitude occurs. Then, a non-trivial analysis algorithm is very quickly executed in dedicated hardware to determine whether the event is sufficiently important to merit a (energy intensive) radio transmission for further processing. By using dedicated hardware, the FPGA can very quickly return to sleep again and achieves a significantly better energy balance than waking up a more powerful processor, but which would still be slower than dedicated hardware (e.g., a low-power TI C55 DSP). It it were possible to have such a low-power core present on the FPGA chip with separate power management, the energy balance of the system would improve significantly, as the external processor actually requires most of the system power when the radio is quiet and the FPGA is sleeping. An embedded soft-core would not help us, since the radio protocols need to run continuously (if at a low rate) and would prevent the entire FPGA from powering down (it is not possible to selectively power-down just parts of the device).

The second example is an architecture for general-purpose reconfigurable embedded computing. Here, a mix between a full-capability operating system (embedded Linux, in our case)

running on a sufficiently capable processor, and a larger reconfigurable compute capacity is required. Our ACE-M5 architecture achieves this by relying on a Xilinx Virtex 5 FX device. The embedded PowerPC 440 cores can easily handle all system management tasks. Furthermore, due to the on-chip nature of both the processor(s) and the reconfigurable area, both units can be integrated tightly for high bandwidth and low-latency signaling. For example, even with full-scale Linux running, an accelerator computing on the reconfigurable area has access to 89% of the bandwidth to shared memory, and accelerator-to-CPU signaling requires less than 10us. This allows demanding real-time applications (e.g., robotics and image processing) far exceeding the capabilities of much more powerful embedded processors. While soft-core processors such as the Xilinx MicroBlaze CPU could have been used here (energy and area efficiency was only a secondary target), they would been significantly slower than the 2x super-scalar PowerPC cores.

The final example considers a high-performance computing architecture. The Convey HC-1ex machine combines a quad-core Xeon CPU clocked at 2+ GHz with a reconfigurable subsystem consisting of four large FPGAs and a high-bandwidth (80 GB/s) memory system. Both units communicate over the Intel Front-Side Bus (FSB) which supports shared cache coherent virtual memory. For selected bioinformatics applications, it has achieved speedups of 50x over two-socket x86 servers. However, despite the apparent power of the architecture, two limitations affect its performance in practice: Due to the use of FSB, which has been deprecated by Intel since 2008, the processor models supported are no longer competitive with current-generation CPUs (that now use QuickPath Interconnect, QPI), and can thus actually cause slow-downs over other machines that execute software-only, but on current, much faster processors. Second, while the machine does support transparently shared use of memory attached to the CPU and the reconfigurable subsystem, the FSB with its bandwidth of just 8 GB/s and access latencies in the hundreds of nanoseconds actually becomes a bottleneck. Only by carefully copying (aided with hardware support in the machine) data between the two memory systems to the currently active compute element can this slow-down be avoided.

- From the preceding examples, it should be obvious that the combination of a CPU and reconfigurable processing can be very useful, even for wildly different application fields. As shown by the middle example, the tight on-chip integration of reconfigurable and software-programmable computing allowed efficiency gains that were not possible in either of the discrete approaches. Both in the first and third examples, having discrete CPUs and FPGAs led to a loss in efficiency. For the first case, the integration of a small low-performance low-power CPU with the FPGA, supporting fine-grained power management, would have led to higher board density, improved bandwidth/latency in communications, and better power consumption. For the third case, a tighter integration would have avoided the bottlenecks that appear even when linking the two elements with a relatively high-performance bus such as FSB.

On the flip-side, a tighter integration between CPUs and FPGAs also carries some very practical problems: As shown above, different application fields also impose wildly different requirements on their CPUs. For handling the wireless network protocols, even the slow 8051 core was already over-designed. However, in the HPC area, the 2+ GHz quad core could sometimes not keep up with the performance of the reconfigurable subsystem, and consequently led to decreased speedups. Thus, it is highly desirable to always pick the most suitable CPU / FPGA combination for the specific use-case. To actually fabricate and inventory a sufficient breadth of FPGA / CPU combinations will prove quite challenging, however. To some extent, these difficulties might be addressed by fabricating CPUs and FPGAs on separate dies and mixing-and-matching them using 2.5-D and 3-D stacking technology as required just before packaging.

## Partial Reconfiguration is Ready for Takeoff

Dirk KOCH, University of Oslo, Norway

For more than two decades, run-time reconfiguration of FPGAs was preliminary an academic niche but there is currently a strong trend that this technology will make its way into commercial products. For example, the 100ms setup time for PCIe can only be reached by using bootstrapping on large FPGAs. Here, the PCIe related logic will be configured in a time critical initial phase followed by the rest of the system in a second partial reconfiguration phase. Partial reconfiguration has also several opportunities for general purpose computing. For instance, like run-time libraries are commonly used in the software world, partial reconfiguration can be used to implement a similar concept in hardware by dynamically linking modules into a system with the help of partial reconfiguration.

The trend towards partial run-time reconfiguration can also be observed inside the FPGAs themselves, because all recent devices of the two major SRAM-based FPGAs are now fully supporting partial reconfiguration - and this holds for both, the high-end and the low cost devices.

The support for partial reconfiguration has not only improved on the hardware side, also design tools have significantly improved. This includes FPGA vendor tools such as PlanAhead, which provide basic functionality to implement reconfigurable systems, but also academic tools such as OpenPR or GoAhead. The latter, for example, features module relocation and allows implementing partial modules completely independent to any other part of the system.

Finally, the progress in silicon industry seams to continue and device capacity will soon exceed an order of magnitude more than what we see today. This will need new technologies which are strongly related to partial reconfiguration, such as FPGA virtualization or component-based design. Here, complex systems will be rapidly composed upon fully physically implemented modules (including place and route), which is basically the static version of partial reconfiguration. Today, partial reconfiguration is not flying, but the mentioned trends is giving it the right thrust.

## Foundations of Reconfigurable Computing

Wayne LUK, Imperial College,London, United Kingdom

Reconfigurable Computing can be regarded as a synergistic combination of two disciplines: Reconfigurable Technology and Custom Computing. Custom Computing provides the abstractions for optimizing computations in space and in time to meet given requirements; Reconfigurable Technology provides the physical means for realizing such abstractions effectively.

While there has been much research in Reconfigurable Technology and in application studies of Custom Computing, the scientific foundations of these disciplines are perhaps less obvious:

1. What are the guiding principles of Reconfigurable Computing that will stand the test of time? What are the origins of such principles, and how can they be exploited?

2. What are the limits of capabilities of Reconfigurable Computing? How would such limits be addressed, by Reconfigurable Technology and by Custom Computing?

3. How would Reconfigurable Computing be remembered in fifty years' time?

There is no doubt that Reconfigurable Computing is exciting. Addressing questions like the above ones would provide insight to the origins of such excitement, which would also help finding fundamental solutions to practical issues, such as:

1. How to make the best use of available devices, systems and tools by improving design efficiency and designer productivity?
2. How to explore novel devices, systems and tools that achieve given requirements in design efficiency and designer productivity?
3. What new theories and algorithms would be needed to support the above?

## Functional and constraint programming in combination with machine learning to increase productivity and performance

Nele MENTENS, Katholieke Universiteit Leuven, Belgium

Electronic Design Automation (EDA) plays a central role in bridging the productivity gap for designing complex electronic systems. Most existing tools generate hardware and embedded software that satisfy the required functional specifications. However, for many applications, it is also important that non-functional requirements are met, such as performance, resource utilization and power consumption, either in a strict way (e.g. a maximum surface is given) or in a tradeoff manner (e.g. low power consumption is more important than high performance). The concept is generic, but to obtain a competitive solution, application-specific low-level components are needed.

We propose to use functional programming to provide high-level descriptions of the components in the application-specific libraries and to easily integrate the generation of hardware and software and the use of back-end synthesis and compilation tools. Further, we propose to use constraint programming to take into account the non-functional requirements. In order to explore the design space in a smart manner, search algorithms and machine learning are applied to narrow down the useful part of the design space early in the generation process.

## Can FPGAs and/or reconfigurable devices be a common platform for high-performance computing?

Kentaro SANO, Tohoku University, Japan

FPGAs/Reconfigurable devices can be common platform of high-performance computing but they have to satisfy the three conditions related to device, applications, and programming, respectively. In general, device influences sustained performance (P), power consumption (W), cost (C). Super-computing prefers devices for better P, P/W, and P/C. Looking at present devices, FPGAs can achieve comparable P or better P/W in comparison with those of such accelerators as GPUs and Intel MICs, while they depend on applications. On the other hand, the accelerators provide much better P/C than FPGAs. However, the end of technology scaling and the dark silicon problem can be a tailwind for FPGAs.

Applications give operation (floating-point, fixed-point, integer, logical one), parallelism (spatial/coarse or temporal/fine), and performance characteristic (memory-bound or compute-bound) for computation. Today's super-computing mainly demands computation with floating-point operations, spatial/coarse parallelism, and memory-bound performance, and therefore present processors and accelerators are designed for them except memory-bound performance. For such computation, FPGAs also have comparable sustained performance, while they are not drastically good. However, flexible FPGAs are much better at non floating-point, temporal/fine-parallel and memory-bound performance than processors and accelerators. Some super-computing applications have such characteristics. Further advancement of FPGAs could give them more advantages for computing.

The biggest problem is an interface of programming for FPGAs to become a platform of super-computing. In standard super-computing, users directly write and change a code for computation, which is written in traditional languages (C or FORTRAN). Recently their extension (OpenMP, OpenCL, MPI) is also necessary for super-computing. Since these languages and extensions are common interfaces of super-computing, FPGAs also have to fit them. Recently, FPGAs already made a success for embedded super-computing as Maxeler's story shows, where users do not program codes, but just use softwares that are already programmed. For standard super-computing on FPGAs, we need seamless tool-chain or compiler for traditional languages, and acceptable compilation time in addition to efficiency bringing sufficient performance to FPGAs.

To meet these conditions, I think that the key is abstraction layer which lies between reconfigurable devices (FPGAs) and software. We should define programmable hardware layer, which can be based on application-domain architectures, such as architectures for each of OpenCL, OpenACC, OpenMP, MPI, etc. Open question is: How efficient is this approach for exploiting the performance? and How we efficiently provide compilers to these architectures?

## Programmable Architecture for Pattern Matching

Tsutomu SASAO, Kyushu Institute of Technology, Japan

Sasao invented a new architecture for pattern matching, that works great for limited applications. Some of results were presented at ICCAD 2008, DATE2012, ASPDAC2012, ARC2012, and the book "Memory-Based Logic Synthesis," Springer 2011. Applications to IPV6 router will be presented at ARC2013. Sasao is now improving the optimization algorithm.

Questions and answers are as follows: Q) Do we need fundamentally new architecture? A) Yes. We need it. To continue to progress, to write papers, and to get research grant, we need fundamentally new architecture.

## Bringing Reconfigurable Computing to the Mainstream

Hayden SO, University of Hong Kong, China

Despite years of demonstrated success in accelerating application across many demanding domains, reconfigurable computing remains a niche that has limited reception in the mainstream computing arena. There are two areas that deserve particular attention from the community in order to bring reconfigurable computing to the mainstream: (1) improving design productivity and (2) integrate with the mainstream computer architecture.

Design productivity is a difficult to define quantity. It can generally be thought as the time required to complete a design-test-debug iteration for application development. The shorter time each step takes, the more iteration may be completed per day, and the more likely that a deliverable design can be developed faster. To that end, the conventional software development methodology is well-established after decades of evolution. Today, compilation of even the most complex software application is swift; the runtime operating system is well designed, and there are plenty of debugging tools and methodology for testing the resulting system. On the other hand, there remains very little established industrial standard on application development for reconfigurable computers. Most system relies on low-level hardware design methodology that was originally designed for VLSI designs. Also, compiling such designs may take hours to days, limiting the number of compile-debug cycle to close to one per day. Then, operating system support for non-conventional computation model remains limited. Finally, debugging tools for reconfigurable computers are generally lacking. Without a high-productivity design methodology, it is difficult to attract novel users to appreciate the benefits of reconfigurable computers.

Application designers who demand the highest performance may instead turn to other forms of compute accelerators, such as the use of GPU for the better development environment.

Apart from improving design productivity, it is also important to integrate reconfigurable computers to the mainstream computer architecture. Most research projects on reconfigurable computers, while achieving superior performance under a lab environment, rely on non-compatible interfaces and techniques that are difficult to apply to other scenarios. As such, the results from one research group can rarely be reused in another setting, forcing many researchers to "reinvent the wheel" over and over. In order for reconfigurable computers to become mainstream, more efforts must be devoted to integrating such non-conventional computer model with the mainstream computer architecture. Finally, for sake of standardization and improving backward compatibility, it is imperative for the industry and the academia to pay more attention to developing abstraction layers for reconfigurable computers, similar to the ISA layer of standard computers.

## Requirement of a standard programming model for reconfigurable processors

Kazuya TANIGAWA, Hiroshima City University, Japan

A standard programming model for CPUs has been developed, and hence, any high-level language and CPU configuration can be chosen. For example, a program written in a high-level language is executable on various CPUs. Moreover, various high-level languages are available for developing a program that is executable on a target CPU. To develop such an environment, CPUs and high-level languages need to be developed based on a standard programming model.

However, there is no standard programming model for Field Programmable Gate Arrays (FPGAs) or reconfigurable processors, and hence, in many cases, it is difficult to port a program written for a reconfigurable processor to other reconfigurable processors (here, a program constitutes circuit information executable on a target reconfigurable processor). Modifying a program according to the target reconfigurable processor and high-level language is problematic.

Therefore, in my opinion, a standard programming model for reconfigurable processors is required for the widespread use of such processors.

## Creating Heterogeneous Design Tools; the missing link

John WATSON, Data IO Corporation, USA

Our goal should be that all hardware architectures are adaptable to match their applications; first at the application level, then at the applet/subroutine level, and then at the algorithm level. In moving down each of these, the reconfiguration times get smaller and happen more often, until in the end, the hardware is constantly being adapted "in real time" with the software program becoming more of a hardware scheduler as "software processes" are called into action. These types of ideas have not caught on for two reasons, a) the lack of industrial understanding of data flow/data driven parallel processing efficiency and b) Moore's law, whose historic price drops have masked the need for efficiency. Since low cost is the market driver for technology, as long as Moore's Law could schedule cost decreases no one looked for a better solution; resulting in our current architectural and processing rut. Hence, our industry traded low cost and software "ease of use" for everything else. The focus became the short term solution and not necessarily the right solution. We must get back on the right track and that won't happen via Wall Street, Venture Capitalists or Silicon Valley.

I feel two things are needed:

1. Universities, community colleges and high schools that teach HW and SW efficiency.
   Students need to realize that parallel is the natural way of everyday life, and that serial processor structures are artificial, unnatural and the old way of thinking. All of the emerging fields; big data, cloud computing, informatics, biotech, robotics, autonomous vehicles, mobile, etc. are worlds in which parallel is the dominant theme?

2. A Moore's Law equivalent for FPGA/RC design productivity tools.
   FPGAs ICs are "designed" while processor ICs are "programmed". The line between design and program needs to be removed; the line between "IP" and "applications", between "PAR" and "compiler", "timing closure" and "cycle count", "design verification" and "application testing," etc. Furthermore, FPGA/RC/heterogeneous design tools must scale across all applications and IC architectures, work across time and space, compile quickly and provide a quantum leap in design reuse. A fab process shrink and the next larger wafer size work across all CMOS chips. Where is the design tool equivalent? We should discuss how to do this.

The battle to eradicate Fortran and Cobol as dominant languages took a decade. But the move from those old languages enabled the first minicomputers, then PCs and then the mobile market. Without moving away from those languages none of this would have been possible.

# 3 Discussion on a Grand Challenge Vision for Computing

The objective of this discussion was to determine if there was a foundation for establishing a community-wide vision for configurable computing. The participants structured the discussion progressing from the basic understanding of contemporary issues to a plan for future action.

The first topic was an assessment on the current state of the computing world. The participants noted that there are many disparate uses for computing, many different architectures, and many scales of operation such as:

1. multi-core embedded processors
2. sensor networks
3. mobile computing
4. desktop computing
5. cloud computing
6. high performance computing
7. Internet infrastructure

From this point, the participants discussed what are the common links, and the distinguishing links. These were identified as follows:

1. all do computing
2. parallel computing
3. heading towards heterogeneous computing elements, if not already
4. power/energy issues
5. communications issues
6. data
7. memory
8. I/O

The issue examined was that the broad spectrum of computing systems all look the same at this level, yet each community has it's own approach to solving problems. There are inefficiencies as each struggles with similar problems independently.

The participants then examined the computing fundamentals as a basis for finding common threads between the communities:

1. control
2. dataflow
3. computation
4. inter-{process, thread, processor, node} communication
5. I/O

All of these use the realm of "scale" differently. For example, solutions at a smaller scale may be totally irrelevant to those working at a much broader, higher scale. This is something that we must all deal with.

The participants then discussed where this technology is taking us. There are many notable advances that need consideration. In the long run, one must consider the possibilities well beyond CMOS when the possibility of molecular/nano computing is realized. The discussion centered on if the fundamentals of computing concepts would change.

The session concluded with a discussion on what is needed for the community. This was a lengthy discussion that focused on the following topics:

1. a common (unifying?) architecture model
2. build DSL's on top of this
3. tools that take the application as an input constraint
4. optimize inter-{process,thread,processor,node} communication according to whether it is a sensor network or a million-node HPC system
5. optimize power use accordingly
6. provide scalability and portability through the spectrum of targets
7. common design flow and debugging framework
8. must account for heterogeneity

The participants then concluded with a plan to create a roadmap document with the support of the configurable computing community as a whole.