

Design Aspects of Self-Organizing Heterogeneous Multi-Core Architectures

Rainer Buchty, Wolfgang Karl
Universität Karlsruhe (TH) – Institut für Technische Informatik (ITEC)
Lehrstuhl für Rechnerarchitektur
Zirkel 2, 76131 Karlsruhe, Germany
`{buchty|karl}@ira.uka.de`

Abstract

Already today we face architectures featuring up to several hundreds of processors, being able to manage several thousand concurrent threads. Future architectures, however, will not only see an increase in parallelism but also feature an increase in heterogeneity and reconfigurability. Judging from current production and prototype architectures, we also see that such systems will be tiled, i.e. individual cores with local memory interconnected through some means of on-chip communication.

Current discussions show that existing approaches to application mapping, parallelization, data locality optimization, and system management do not match these upcoming architectures well, thus rather hampering than harnessing the power of future systems. We will therefore outline the requirements of upcoming architectures and demonstrate how self-organization, including bio-inspired, techniques may help to manage system complexity. Key to these techniques is a sophisticated decentralized, hierarchical monitoring approach suitable for sustained real-time monitoring and event correlation for current and future high-performance architectures.

1 Introduction & Motivation

Following current forecasts [8, 9], computing systems of the already near future will be massively paral-

lel. Graphics Processing Units (GPUs) may serve as a role model for this kind of development: here, already today, we see architectures featuring up to several hundreds of processors [7, 2], offering hardware support for several 1000 concurrent threads and according programming environments [14, 1].

The vendors' road-maps also indicate an increase in heterogeneity, leading to not only massively parallel architectures, but architectures where the individual cores might show different characteristics. Starting with fixed architectures like IBM's Cell BE [11], featuring 8 vector engines (Synergistic Processing Elements, SPEs) and a general-purpose main processor (Power Processing Element, PPE) or hybrid approaches outlined in the context of terascale computing [13], future architectures will not only feature heterogeneity but also make use of reconfigurability.

A common property of all these architectures is their inherent tiled structure, i.e. individual computation nodes connected using some arbitrary interconnection structure, e.g. a peer-to-peer-like grid, FIFO-based rings, or some other, eventually combined approach. Each tile consists of an individual processor core and an associated local or locally shared, i.e. shared by a set of adjacent processors, memory. This memory is typically of cache-like size such as e.g. the 128kB scratch-pad memories of each SPE within the Cell BE. It is the responsibility of the programmer to find a best-fitting distribution and mapping of program and data to computation nodes and their associated memories. Furthermore,

the programmer needs to take care about data movement from an external, global shared memory into the individual local memories. In some cases, such as e.g. Nvidia’s CUDA, this is accounted for by the programming model and the compiler; in any case, the given restraints of memory requirements of an individual node’s computation together with interconnect restrictions (access time & latency) typically leads to a significantly impaired amount of parallelism. For instance, of the 12,288 potential threads within the Nvidia G80 GPU architecture typically only 4000-6000 can be assigned due to memory requirements.

The second property and possibly one key feature of future architectures, especially in the embedded area, is their reconfigurability. With the increasing density and resulting functionality of FPGA technology, the use of such is no longer limited to very delicate applications, but also makes it sensible for being incorporated into more general systems.

An example of such use is the successful introduction of so-called platform FPGAs [18] which basically combine reconfigurable FPGA technology with conventional processor cores and dedicated communication hardware.

Therefore, architectures would ideally not only supply a high degree of parallelism, but also sufficient reconfiguration capabilities to enable tuning the architecture for the specific requirements of a running application, e.g. adjusting the interconnect structure or partitioning a shared memory resource other than using a fixed scheme. An academic example of such a parallel, dynamically reconfigurable tiled architecture is the Digital On-demand Computing Organism (DodOrg)¹ [3]. It comprises an array of individual processing cells, connected using peer-to-peer networking. Each processing cell is reconfigurable in terms of function and connectivity, therefore not only supporting HW-based application acceleration, but

also required grouping and communication of cooperating processing cells.

While computer science already knows about and uses application-mapping and -deploying techniques for parallel systems and is also able to deal with heterogeneous systems, these methods, however, are hardly able to cope with reconfigurable systems where both, functionality and connectivity of individual nodes, may be changed during run-time.

Dealing with run-time reconfigurable architectures, however, imposes a new system quality which cannot be easily harnessed by traditional techniques: software-based approaches dealing with task/thread deployment and dynamic parallelization such as e.g. OpenMP require an underlying run-time framework for each possible configuration of the present computing nodes, therefore limiting the amount of reconfigurability. This problem is especially visible with respect to dedicated hardware accelerators which typically are realized in a way to purely accelerate one or more computations of an application, but do not necessarily provide a general-purpose processing layer.

Hardware-based approaches such as the Hardware Thread Manager present in Nvidia’s current GPUs dealing with the distribution of individual threads, in term, are inherently fixed and will not cope with a changing system. While it is of course possible to implement some sort of configurability, such approaches will naturally only deal with a changing number of uniform nodes, but not individually reconfigurable computation nodes. Similarly, explicit communication such as offered by MPI is does not cope with a changing system; the user-programmed processing and communication pattern will stay the same and not reflect the changed system.

Therefore, reconfigurable architectures require novel approaches to programming and run-time environments; existing approaches to application mapping, parallelization, and system – especially memory – management rather hamper than harness the power of such upcoming architectures.

A way to overcome such memory issues with respect to memory resource assignment and communication is to not only introduce reconfigurability into the computing resources but also making the

¹The DodOrg Project is a cooperation of five partners at Universität Karlsruhe (TH): Prof. Dr.-Ing. Jürgen Becker, Prof. Dr. Uwe Brinkschulte, Prof. Dr.-Ing. Jörg Henkel, Prof. Dr. Wolfgang Karl, and Prof. Dr.-Ing. Heinz Wörn. It was funded by the German Science Foundation (DFG) within the Special Priority Program 1183 “Organic Computing” and is now funded as an individual project.

memory system reconfigurable and introducing self-management.

We therefore like to treat reconfigurability as a more generalized approach to parallelism and heterogeneity: not only the number of nodes may change, but also their type and capabilities. Traditional techniques rely on either the number of nodes or their type being fixed. For instance, mapping an application to the Cell BE requires careful parallelization and modeling the data flow to achieve the desired speedup [12]. Likewise, deploying applications on general-purpose computation machines such as supercomputers or grid-computing systems requires adhering to given programming models easing the on-demand distribution of workload and data on the computing infrastructure.

Reconfigurability delivers the required flexibility to adopt the architecture to computation problems, but significantly increases system complexity. The programmer now also has to consider a potentially changing system configuration. Therefore, we see self-adaptation as one potential solution to the problem of dealing with highly parallel, heterogeneous, and reconfigurable systems. While reconfigurability provides the required hardware features for dynamical systems, self-adaptation – driven by sophisticated real-time monitoring – delivers a method to deal with the increased system complexity.

Hence, we like to structure the remains of this paper as follows: first, we will outline the DodOrg architecture as a role-model of upcoming so-called Self-X systems, i.e. parallel, heterogeneous and reconfigurable architectures. DodOrg explicitly features the use of bio-inspired techniques to manage system complexity. Key to this management is the use of decentralized, hierarchical monitoring and run-time monitor data evaluation. We will outline the used mechanisms and demonstrate how these are generally applicable with respect to parallel, distributed, heterogeneous, and reconfigurable systems. These mechanisms are key to our Self-aware Memory (SaM) [6] architecture, providing flexible assignment and management of memory resources as a way to leverage memory and communication aspects in parallel, heterogeneous, and potentially dynamically changing systems.

2 Challenges and Responses

Managing dynamically changing systems is a complex task, commonly referred to as the MAPE cycle which divides into four distinct phases: monitoring, analysis, planning, and execute; the system state is derived from monitored data, analyzed with respect to objective functions, leading to a changed system configuration which finally is applied, and the next MAPE cycle starts. Hence, it is obvious that not only sufficient monitoring of the current system state is required, but also according real-time analysis of the collected monitoring data, driving corresponding system adaptation.

We therefore like to introduce the Digital On-demand Computing Organism for Real-time Systems (DodOrg) [3] as a role-model for upcoming parallel, heterogeneous, and dynamically changing systems. The DodOrg hardware comprises a tiled architecture consisting of reconfigurable hardware units called Organic Processing Cells (OPC), providing processing and memory resources, interconnected by a dynamically adaptable communication infrastructure [15]. Using an artificial hormone system [5, 4] workload is assigned to individual OPCs in a way that cooperating tasks are processed by neighbored cells, the so-called organ formation. A DNA-inspired process takes care of respective reconfiguration of affected OPCs and communication infrastructure.

System state signalling and interpretation is implemented akin to biological organisms where the concentration level of hormones is measured: if a certain threshold is reached, some action is triggered such as e.g. rising blood pressure and heart beat rate. It is also possible, that based on the threshold levels of one or more individual hormones a so-called second messenger, i.e. a new, derived event type, is generated.

Any organic self-configuring architecture therefore requires a comprehensive, flexible, and adaptive monitoring approach, as run-time configurable and self-adapting architectures impose several problems which cannot sufficiently be addressed by current programming models: neither does the programmer know up-front how many cells will process an application task, nor does he know where the resulting

organ will be formed, how it will communicate, or where input and output data will be stored.

Hence, gathering and processing system status data in real-time is the key to successful control of the adaptive processes and therefore successful application processing. The used concepts will be explained in Section 2.1.

This necessity we want to illustrate with a second, more general example: in parallel systems, we typically face a fixed assignment of memory to computation nodes; likewise, the access media is fixed. These fixed resources require the programmer to find a well-fitting mapping of the application computation onto a given architecture. Once the system changes, this mapping has to be re-done to fit the current architecture requirements.

Interestingly, not so much the decomposition of a program takes the main effort when parallelizing an application, but especially data distribution and data-related communication. We therefore like to introduce Self-aware Memory in Section 2.2 as a more general approach to manage complexity in parallel, heterogeneous, and potentially reconfigurable systems and easing programming by making the memory subsystem reconfigurable, being able to autonomously assign memory and communication resources of a given resource pool.

2.1 System Introspection and Evaluation

Addressing the specific requirements of reconfigurable architectures in general and steering configuration cycles in particular, we propose two basic mechanisms for next-generation monitoring infrastructures: the limitation of monitoring resources we address by using so-called associative counters triggering to arbitrary events instead of being hard-wired to a small preselection of individual events. This obviously requires self-identifying event coding.

While developed within the DodOrg context, these approaches never the less are generally suitable for high-performance monitoring in parallel, distributed, adaptive, and self-adaptive systems.

2.1.1 Associative Counter Arrays

So-called performance counters enable real-time monitoring of event data without requiring to trigger an external monitoring instance with every event occurrence. Instead, semantic compression is archived by only forwarding an accumulated number of event occurrences instead of individual events is forwarded to a memory buffer where it then can be processed by a higher monitoring instance, e.g. for correlation or visualization.

In contrast to conventional systems, where both, counter size and event association are typically fixed, our monitoring approach features the use of associative counter arrays. Hence, no hard-wired connection between one or more predefined events and a single counter exists; instead, the counters are self-triggering to any event. They also feature counter overflow control, aiding histogram generation for higher-level monitoring.

Associative counters basically resemble cache-like structures: upon event occurrence, the event, identified by a unique ID, is caught by the counter array and assigned to the first spare, i.e. unassigned, counter. Subsequent occurrences of this very event will trigger the assigned counter. If no spare counter is available, an assigned counter will be re-assigned: in this case, the existing counter value and the event identification are evicted from the counter array, and the referring counter is initialized and assigned to the new event. In contrast to caches, only spills to higher monitoring instances are required which then perform further processing of the sent monitoring information.

To make this concept work, a unique and self-defining encoding of system events as shown in the upper part of Figure 1 is required. This encoding consists of two parts, a local part denoting the event itself (e.g. a memory access) and associated data (e.g. the address and read/write flag), and an additional part containing the source of this event, i.e. which OPC performed this operation.

In the following, we will show that such a unique event tag does not only ease event monitoring using associative counter arrays, but also correlation and evaluation of such monitored event data.

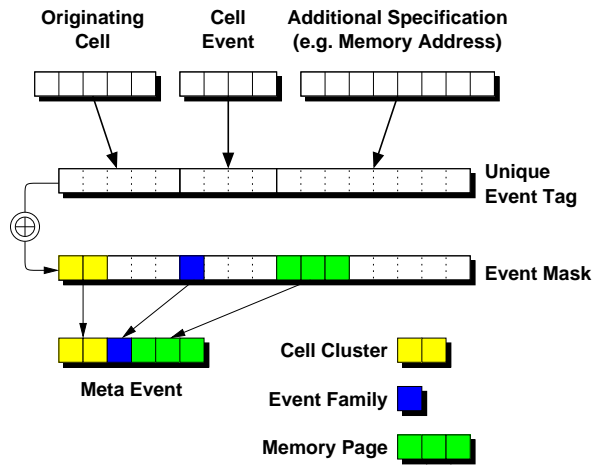


Figure 1: Unique Event ID Construction and Mask-based Event ID Evaluation

2.1.2 Biologically inspired Event Communication and Evaluation: The Hormone Concept

In biological systems, certain events are communicated using so-called messengers, or hormones. The amount of messengers – i.e. events – generated is solely defined by its respective producers, i.e. no central instance commands generation of events, neither does a central instance command event encoding. Likewise, whether to react or not to one or more hormones is solely decided by the receiver. The absence of central production and evaluation instances makes this signalling system very flexible and scalable.

If a certain hormone threshold is reached, i.e. if a sufficient amount of hormones is sent out during a specific time window, some action is triggered, such as e.g. rising blood pressure and heart beat rate. It is also possible that based on the threshold levels of one or more individual hormones a so-called second messenger, i.e. a new, derived event type, is generated. Messengers are inherently self-defining by their chemical structure; hence, using an encoding mechanism and a method of accumulation as described in Section 2.1.1, both, hormone concept in general, and concentration triggering in particular can be directly applied to digital systems as shown in [3].

Associative counter arrays already provide sufficient flexibility for event detection; in addition to that, also a flexible method of event correlation is required, including the possibility of focus adjustment, i.e. changing the event granularity or “monitoring resolution”. Further amount of flexibility is introduced by not using event IDs directly, but applying a simple identification mask and therefore being able to narrow and widen the focus of event accumulation: with this mask, mandatory and “don’t care”-fields of the event ID are specified. This mask is then applied to incoming event IDs using a simple Boolean function. In case of a match, either a counter may be triggered or, mimicking the so-called second-messenger principle, a new event may be sent into the system. Figure 1 shows such an evaluation process.

Since only the interpretation of already generated monitoring data is changed, different monitoring instances might coexist, each interpreting the present monitoring data differently.

The drawback of such an approach is a potentially high communication load imposed by transportation of monitoring data. However, [17] showed that for typical setups the amount of monitoring data can be easily transported in modern communication infrastructures, including networks on chips (NoCs).

2.2 Self-aware Memory (SaM)

One of the biggest challenge of upcoming architectures are programmability and efficient use. As of now, these two seem to be contradicting: either a huge effort is put into manual optimization for a given architecture, e.g. the Cell BE, leading to a best possible application mapping but completely architecture-centric code, or a higher level approach is used such as Nvidia’s CUDA for GPGPU programming. The latter leads to a very readable and (among supported architectures) portable code, but relies on a static, compiler-based resource-partitioning scheme combined with a hardware-assisted thread scheduler.

As we can see from this example, one major problem therefore is not parallelization per se but efficient use of memory and communication resources. Hence, we propose a technique called Self-aware Memory (SaM) [6]. SaM has the potential to leverage sev-

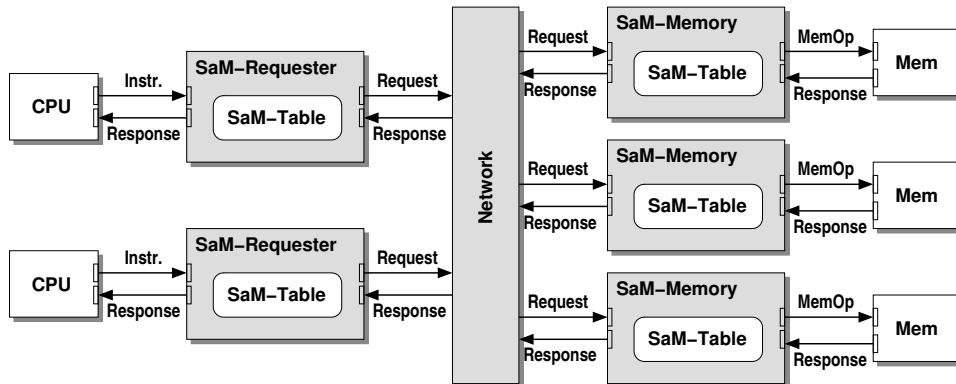


Figure 2: Self-aware Memory [6]

eral problems related to parallel, heterogeneous, and reconfigurable architectures.

With SaM, the memory-modules are no longer directly assigned to a single processor but are part of a memory resource network. This network may be an exclusive memory network or embedded into existing connection resources, forming a virtual memory network. Communication is done using a light-weight, non-overhead protocol; for ease of integration, we chose a hormone-inspired protocol tightly interfacing with the event ID concept explained in the previous two sections.

As illustrated by Figure 2, every memory entity is connected to a management component (SaM-Memory) which also serves as an interface to the memory network. Several different memories may co-exist in the system, each of them connected through an own SaM-Memory component to the network. Processors (or any other memory-accessing entity) are likewise attached to the memory network through a similar component (SaM-Requester), managing the memory accesses of a processor and handles the overall access protocol of SaM. It also generates request messages to the memory subsystem and processes the answers of the affected memory components. Both units employ dedicated tables (SaM-Table) for storing internal information regarding memory allocation and ownership, i.e. address translation plus ad-

ditional housekeeping data for memory management and access security.

What makes SaM self-aware is the monitoring-driven self-evaluation, enabling memory access interpretation and therefore leading to autonomous data locality optimization and self-steered prefetching. To support this, data and management information are stored together so that not only the data but also the managing information, i.e. which memory region belongs to what processing entity (along with according access rights) is kept within the individual memory node.

This already addresses two major problems in application mapping. Combined with hardware-based reconfigurability, also the remaining memory-related problems can be addressed. Using fine-grained reconfigurability and resource assignment, we are not only able to partition and assign existing memory and communication resources, but moreover employ on-demand synthesis, i.e. reconfiguration of existing chip area, making it possible to transform logic resources from processing into storage or communication.

2.3 Prototype Implementations

To fully elaborate the proposed concepts, we developed two prototypes: the test case for our monitoring software prototype is the control infrastructure for SaM. To account for the specific require-

ments of or DodOrg architecture, the memory system uses a lightweight, hormone-inspired protocol where any kind of memory access (allocation request, resource grant, read/write access) is encoded as a unique event.

We applied the monitoring approach outlined in the previous section to an existing UNISIM[10]-based simulation environment, introducing associative counter arrays and hormone-inspired event messaging providing the required information-gathering and -processing entity is required to analyze and optimize memory accesses in a distributed, localized fashion.

Despite its simplicity, this monitoring approach proved very powerful. By simply triggering to event IDs of individual memory accesses, we were easily able to refine and optimize the used communication protocol and extract access data required for automated prefetching and data locality optimization. The approach therefore proved being a vital building block for upcoming fully autonomous, self-optimizing systems.

Targeting a Xilinx Virtex4-FX100 FPGA, a monitor configuration of 64 individual 8-bit counters with a tag size of 40 bits was implemented, accounting for about 4% of logic use (slices), mostly holding the access logic, and 6% of on-chip memory storage (RAMB16) for the associative counter array; implemented into an existing high-performance bus interface core [16], the monitor-equipped core shows an increase of 36% in logic and 33% in memory, mainly accounting for the tag-test and storage infrastructure.

This configuration provides sufficient resources for current and future bus systems, not only giving a realistic scenario for a hardware-based monitoring unit but more specifically demonstrating the suitability for high-performance communication monitoring as required for on-chip multi-core connectivity.

3 Conclusion

Already today, we see massively parallel systems dealing with dedicated applications; architectures of the future, however, will not only be massively parallel but introduce heterogeneity and reconfigurability

as a means to fully exploit system resources by providing maximum flexibility.

The complexity of such systems is nearly impossible to handle using traditional approaches of both, systems engineering and application programming. Therefore, reconfigurability as a hardware property shall be accompanied by self-management, not only releasing pressure from the programmer but actually making it possible to fully exploit such architectures.

Demonstrated by the DodOrg architecture, apart from providing reconfigurable hardware, key to such reactive, self-managing behavior is sophisticated monitoring. Unlike conventional monitoring, which typically relies on off-line analysis and optimization, a method is required to provide high-performance on-line monitoring and analysis. Such methods do not only serve specific architectures, but are generally applicable to self-management strategies as demonstrated with the Self-aware Memory concept, addressing the problem of data distribution and communication using self-management and self-adaption.

The complexity of computer systems has already risen to an amount where established techniques can hardly deal with this very complexity. Using self-managing methods leading to self-adapting systems, however, gives us a means to even harness the power of forecast massively parallel, heterogeneous, and reconfigurable systems.

References

- [1] Advanced Micro Devices, Inc. AMD Stream Computing Software Stack. Nov 2007. <http://ati.amd.com/technology/streamcomputing/firestream-sdk-whitepaper.pdf>.
- [2] Advanced Micro Devices, Inc. ATI RadeonTM Desktop Graphics. 2008. <http://ati.amd.com/products/home-office.html>.
- [3] Jürgen Becker, Kurt Brändle, Uwe Brinkschulte, Jörg Henkel, Wolfgang Karl, Thorsten Köster, Michael Wenz, and Heinz Wörn. Digital On-Demand Computing Organism for Real-Time Systems. In Wolfgang Karl, Jürgen Becker, Karl-Erwin Großpietsch, Christian Hochberger, and

- Erik Maehle, editors, *Workshop Proceedings of the 19th International Conference on Architecture of Computing Systems (LNI P81)*, pages 230–245, March 2006.
- [4] Uwe Brinkschulte, Mathias Pacher, and Alexander von Renteln. An Artificial Hormone System for Self-Organizing Real-Time Task All in Organic Middleware. In *Organic Computing*. Springer, 2007.
- [5] Uwe Brinkschulte, Mathias Pacher, and Alexander von Renteln. Towards an Artificial Hormone System for Self-Organizing Real-Time Task Allocation. In *5th IFIP Workshop on Software Technologies for Future Embedded Ubiquitous Systems (SEUS 2007)*, 2007.
- [6] Rainer Buchty, Oliver Mattes, and Wolfgang Karl. Self-aware Memory: Managing Distributed Memory in an Autonomous Multi-Master Environment. In *The 2008 International Conference on Architecture of Computing Systems (ARCS 2008)*, Dresden, Germany, February 25-28 2008.
- [7] Nvidia Corp. NVIDIA GeForce 9 Series. 2008. <http://www.nvidia.com/object/geforce9.html>.
- [8] Koen De Bosschere, Wayne Luk, Xavier Martorell, Nacho Navarro, Mike O’Boyle, Dionisios Pnevmatikatos, Alex Ramirez, Pascal Sainrat, André Sez nec, Per Stenström, and Olivier Temam. High-Performance Embedded Architecture and Compilation Roadmap. In *Transactions on High-Performance Embedded Architectures and Compilers I (LNCS 4050)*, pages 5–29. Springer Berlin / Heidelberg, 2007. ISBN 978-3-540-71527-6.
- [9] Kunle Olukotun et al. Towards Pervasive Parallelism. In *Barcelona Multicore Workshop (BMW2008)*, Jun 2008. <http://ppl.stanford.edu/wiki/images/9/93/PPL.pdf>.
- [10] European Network of Excellence on High-Performance and Embedded Architectures and Compilation (HiPEAC). UNISIM: UNITed SIMulation Environment. 2008. <http://unisim.org>.
- [11] M. Gschwind, H.P. Hofstee, B. Flachs, M. Hopkins, Y. Watanabe, and T. Yamazaki. Synergistic Processing in Cell’s Multicore Architecture. *IEEE Micro*, 26(2):10–24, March-April 2006.
- [12] Hans Vandierendonck and Sean Rul and Michiel Questier and Koen De Bosschere. Experiences with Parallelizing a Bio-informatics Program on the Cell BE. In *High Performance Embedded Architectures and Compilers: Third International Conference, HiPEAC 2008 (LNCS4917)*, pages 161–175. Springer Berlin Heidelberg New York, Jan 2008. ISBN 3-540-77559-5.
- [13] Jim Held, Jerry Bautista, and Sean Koehl. From a Few Cores to Many: A Tera-scale Computing Research Overview. *Research at Intel White Paper*, 2006. http://download.intel.com/research/platform/terascale/terascale_overview_paper.pdf.
- [14] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with CUDA. *ACM Queue*, 6(2):40–53, 2008.
- [15] Christian Schuck, Stefan Lamparth, and Jürgen Becker. artNoC - a novel multi-functional router architecture for Organic Computing. In *17th International Conference On Field Programmable Logic and Applications (FPL2007)*, August 2007.
- [16] Ulrich Brüning, Holger Fröning, et al. HTX Board Universal Reference Design. 2008. <http://www.hypertransport.org/products/productdetail.cfm?RecordID=75>.
- [17] Alexander von Renteln and Uwe Brinkschulte. Reliability of an Artificial Hormone System with Self-X Properties. In *Parallel and Distributed Computing and Systems*, Cambridge, Massachusetts, USA, November 19-21 2007.
- [18] Xilinx, Inc. VirtexTM Family FPGAs. 2008. http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/index.htm.