An Organic Computing Approach to Sustained Real-time Monitoring

Rainer Buchty, David Kramer, Wolfgang Karl

Abstract

System monitoring is not only the key to system and application optimization, but also provides the fundamental functionality for adaptive and self-configuring systems. In this paper we describe a novel approach to monitoring based on biologically inspired methods, which not only suits traditional requirements in generating a detailed and pristine system state image, but also complies with the dedicated needs of self-configuring systems. As a beneficial side-effect, the proposed monitoring approach is inherently fault-tolerant and scalable with system size.

Key words: Monitoring, Bio-inspired, Hormone System, Adaptive Computing, Self-X

1 Introduction

Traditional use of monitoring techniques range from basic system introspection required for sanity checks or load balancing to detailed system traces used for application and architecture optimization. Especially the latter requires detailed and pristine recording of the system state to enable correlation of program code and monitored effects, such as e.g. cache hits and misses.

For these topics, a plethora of techniques and tools has been developed such as hardware counter registers triggering to individual events, profilers, or simulation infrastructures. Each of these methods, however, comes with certain drawbacks. Simulation is only as precise as the underlying simulation model, and typically several magnitudes slower than execution on a real system. It does, however, provide

Universität Karlsruhe (TH)
Institut für Technische Informatik
76128 Karlsruhe, Germany
`{buchty|kramer|karl}@ira.uka.de`

the possibility of closest possible introspection and is able to deliver a pristine system view, unaltered by monitoring side-effects.

Simulation is especially hampered by the problem of data size: even short program runs of few milliseconds are able to generate several megabytes of monitoring (trace) data at single-step resolution. Even at a far more coarse-grained monitoring resolution, a program run will easily generate traces in the gigabyte range.

All approaches typically share the way how monitoring data is collected and transported: collection is usually done using fixed, assigned counter registers which are subsequently polled by a monitoring framework. Hence, in real systems, side effects caused by monitoring occur such as interrupt triggering or data transport. When analyzing trace data, these side effects have to be carefully taken into account.

1.1 Adaptive and Self-configuring Systems

Conventional optimization relies on trace generation, off-line analysis of the generated trace data, and manual or semi-automated optimization. For trace generation, monitoring is solely based on pre-defined rules, i.e. the programmer defines upfront which events need to be generated or traced.

This is in contrast to adaptive, self-configuring systems: here, only a start configuration can be provided and it is unknown whether this will suit future system configurations or generate a sufficient amount of monitoring data to enable proper reaction to events. Furthermore, monitoring data must not be collected for off-line introspection, but rather be analyzed and evaluated on-line, adhering to application- and system-defined real-time constraints.

Hence, adaptive systems consist of a closed control loop as illustrated by Figure 1: using monitoring components, the system's current state is derived and evaluated against a destination state defined by so-called objective functions. Using these functions and the adaptive capabilities of the system provided by adaptive components, a refined system configuration is derived and a reconfiguration performed here-on.
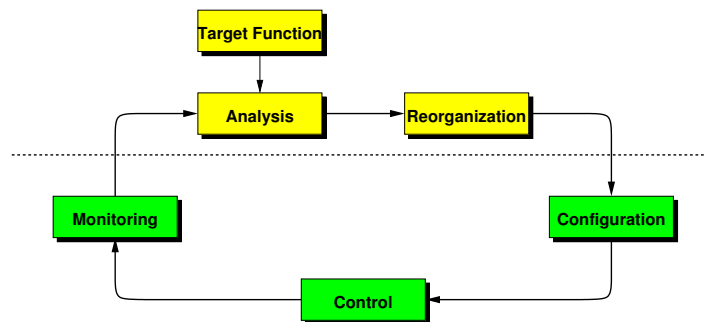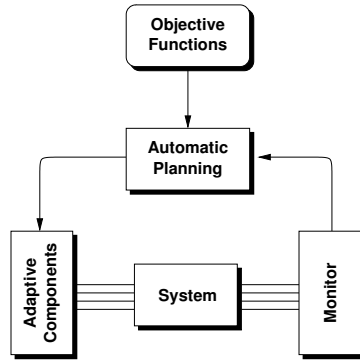

Fig. 1 Adaptive System: Control Loop
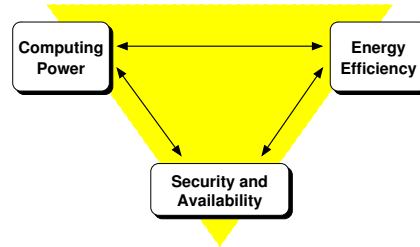
Fig. 2 Adaptive System: Architecture

Fig. 3 Adaptive System: Objective Functions

Objective functions, however, are usually contradictory. Power vs. performance may serve as an example: almost all dynamically applicable methods to lower power consumption will also decrease performance – and vice versa. Hence, data must be collected and evaluated in real-time to ensure that application demands, as defined through the provided objective functions, are met.

As said before, this entire process is unlike conventional, narrowly focused optimization of e.g. data locality, where only limited information needs to be gathered and correlated, and where correlation and optimization are typically done offline. Instead, an approach not hampered by a fixed corset of pre-defined events and rules is required which instead is tailored towards the needs of adaptive and self-configuring systems.

1.2 Outline

In this paper, we therefore propose a novel monitoring system approach. This approach employs a uniform, flexible method of associative counter registers, subsequently enabling real-time analysis of monitoring data, and minimizing side-effects caused by monitoring data transfer. The approach does not require up-front definition of monitoring rules; instead, a bio-inspired method is used, enabling fault-tolerant event generation, distribution, and evaluation. Therefore, within the scope of an adaptive system, also monitoring itself becomes adaptive and – as a beneficial side-effect of the used method, – easily scalable with system size.

This paper is organized as follows: we will first give an introduction to the specific set-up and needs of adaptive, self-configuring systems illustrated by a novel bio-inspired architecture. We then discuss existing monitoring approaches and their suitability with respect to real-time capabilities, system influence, and use within adaptive, self-configuring systems, followed by detailed presentation of our monitoring approach, its prototype implementation and results. The paper is closed with the conclusion and outlook.

2 Related Work

Trace generation for optimization purposes and feedback systems as required for self-configuring, adaptive systems require techniques to gather and process system parameters to be able to create a certain sense of self-awareness. These parameters can be collected on various system levels such as lowest hardware level, driver level, OS level, or application level.

On lowest hardware level, performance counters offer some rudimentary monitoring support. They are typically used to profile an application and investigate possible application optimizations, such as enhanced data layout in memory to improve cache use. Modern processor architectures offer so-called event or performance counter registers [2, 6, 11, 12, 18, 10, 17]. Number and use of these registers are dependent on the individual architecture: counter registers are either bound to certain events or can be more or less freely assigned [17, 11]. The majority of existing analysis tools is based on these counter registers.

Counter-based methods typically suffer from four basic limitations [21] which are number of registers, sampling delay, and lack of address profiling. Furthermore, it is not possible to differentiate between events being triggered by speculative and non-speculative execution. False counts from speculation are addressed with the precise event-based sampling (PEBS) of Intel's Pentium 4 architecture [12, 20]. The drawback of this method, however, is increased chip size and influencing the normal system behavior resulting from concurrent regular memory accesses of the currently running program. Similar, but less complex methods are implemented in the IBM's Power architecture [18, 10].

It is a general problem of sampling methods described above, that only system snap-shots are created. Thus, these methods serve only for creation of aggregated statistics. It is usually not possible to selectively pre-compute monitored data already on the monitoring tier, leading to impact resulting from reading and post-processing the counter registers.

While counter registers are well-suited for processor audit, they are not sufficient for flexible, system-wide, and generic monitoring concepts as required for autonomic computing systems. They are fixed and furthermore lack the possibility of pre-processing.

For architectures without such hardware support, monitoring can be achieved using plain software based on profilers inserting function prologues to collect statistical information such as `gprof` [7]. It is also possible to embed monitoring routines on driver level as demonstrated by the Myrinet-based Shrimp Cluster [13]. A combined hardware/software method was used on the SMiLE monitor for SCI networks [8].

To configure the monitoring system and extract and process monitoring data, higher-level monitoring APIs may be used such as [19], [3], or [14, 15]. Task of these APIs is to decouple monitoring devices from post-processing software by offering an abstract programming interface rather than directly accessing the monitoring hardware.

From the above examples several conclusions can be drawn: *Existing infrastructures in hardware are fixed.* While suitable for their designed task, monitoring infrastructures are too limited for generic use within a self-optimizing system. It is not possible to replace or adapt existing resources. We address this topic by using associative counter arrays instead of fixed event counters as outlined in Section 4.1.

Monitoring systems are application-specific. Powerful monitoring tools exist for various applications, attaching monitoring techniques to various system layers. What is missing, though, is generic support for plugging dedicated monitoring devices into the running system as required. Ideally, monitoring modules can be applied to all system layers through a defined and standardized interface. This is addressed by a uniform event specification as outlined in Section 4.2, enabling unique event identification where this identification also includes where and how an event was generated.

No standardized API exists. So far, several approaches for monitoring APIs exist, However, these are typically bound to certain applications. No uniform, application-independent, and standardized monitoring API being able to report existing monitoring resources, permitting to access these resources, and – with respect to self-organizing systems – enabling reconfiguration exists. We account for this by using bio-inspired mechanisms for event evaluation also discussed in Section 4.2.

3 DodOrg: A Self-configuring Bio-inspired Architecture

For upcoming dynamically changing and self-adjusting systems the conventional method of monitor-data processing is not suitable anymore. Such systems internally feature not one single observer/controller architecture, instead a multitude of individual control loops may coexist creating a hierarchy of de-central, decoupled control loops.

We want to illustrate this with the Digital on-demand Computing Organism for Real-time Systems (DodOrg) [4], a novel computer architecture inspired by biology.

Like biological organisms, DodOrg is hierarchically structured: its hardware is provided by so-called organic processing cells (OPCs). The OPCs provide different capabilities ranging from general-purpose processing to DSP and reconfigurable FPGA blocks, memories, and dedicated I/O cells. All cells are arranged in a grid featuring peer-to-peer connection of the cells.

Mapping an application's task to these cells follows the idea of organ formation in biological organisms: individual OPCs are grouped into work clusters, or organs, using a de-central hormone-inspired middleware. This grouping is based on an individual cell's suitability for application tasks, i.e. its architecture (CPU vs. DSP vs. FPGA), computing power, already assigned work load, and energy demands such as provided vs. required energy.

This is 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 architecture therefore requires a comprehensive, flexible, and adaptive monitoring approach, hence system monitoring is the important issue with self-organizing systems. In particular, the entire system must be constantly evaluated, therefore monitoring within DodOrg spans all system levels, i.e. individual monitors are distributed across the system and are connected to different components and layers.

4 The Monitoring Approach

As can be seen from the previous discussion, monitoring faces several challenges: upcoming systems not only demand a high degree of flexibility with respect to event detection and accumulation, but also require real-time correlation and interpretation of such data. Since the system itself might constantly change, no infrastructure relying on pre-defined events and monitor rules is able to provide necessary adaptivity.

We therefore propose two basic mechanisms for next-generation monitoring infrastructures suitable for both, scalable systems and dynamically, self-adapting systems: 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 pre-selection of individual events. This obviously requires self-identifying event coding.

In Section 4.1, we therefore show our proposed encoding scheme, the associative counter design, and beneficial side effects resulting from this approach.

Subsequently, in Section 4.2, we show how this unique event coding matches the biological model of hormones or messengers and how derived mechanisms overcome the necessity for pre-defined monitoring rules, resulting in an inherently flexible and adaptive nature of monitoring data evaluation.

4.1 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, a certain amount of events is counted and later an accumulated number 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. Introducing a programmable modulo – potentially useful for more infrequent

or rare events – enables control of the counter overflow, aiding histogram generation for higher-level monitoring.

Figure 4 illustrates the general construction and work principle: upon event occurrence, the event 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 reassigned: 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.
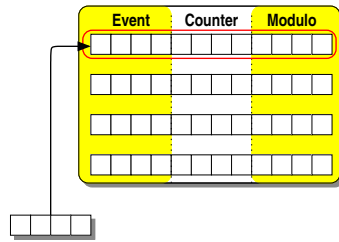
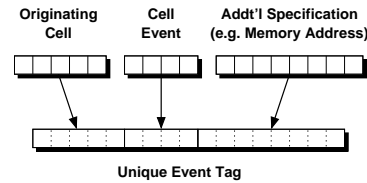**Fig. 4** Associative Counter Array: Principle of Operation

Fig. 5 Unique Event ID Construction

From this explanation it becomes obvious that the associative counter array is a cache memory by nature where the event ID becomes the cache tag and the the event-associated counter resembles the cached data. Hence, likewise replacement strategies including, but not limited to, FIFO, LRU, and LFU approaches.

To make this concept work, a unique and self-defining encoding of system events as shown in Figure 5 is required. This encoding consists of two parts, a local part denoting the event itself (i.e. a memory access) and associated data (the address and read/write flag), and an additional part containing the source of this event, i.e. which CPU 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 event monitor data.

4.2 Biologically inspired Event Communication and Evaluation: The Hormone Concept

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".

A suitable technique can, again, be found in biological organisms: here, 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.

In biological organisms, 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. Messengers are inherently self-defining by their chemical structure; hence, using an encoding mechanism and a method of accumulation as described in Section 4.1, both, hormone concept in general, and concentration triggering in particular can be directly applied to digital systems as illustrated by Figure 6.
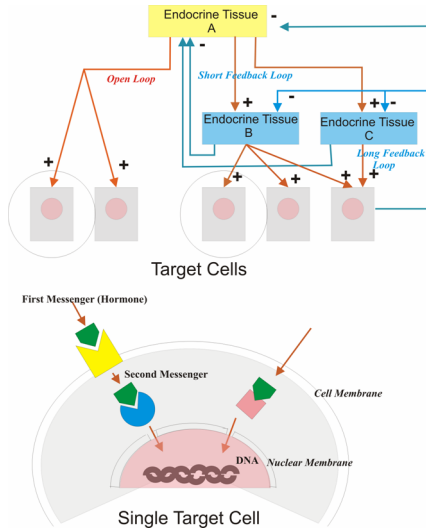

Fig. 6 Hormone Reception, Processing, and Second Messenger Principle
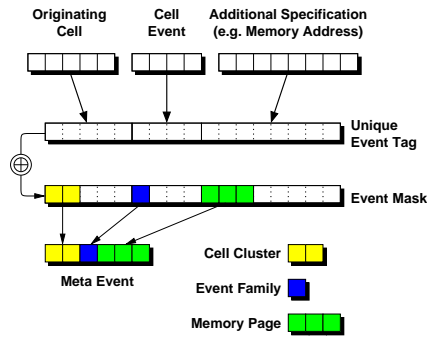
Fig. 7 Mask-based Event ID Evaluation

In such a system, whether to react or not to one or more hormones is solely decided by the receiver. With the absence of central instances therefore the the overall system becomes very flexible and scalable. Further amount of flexibility is introduced by not using event IDs directly, but to apply a simple identification mask and therefore be 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 7 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, [1] showed that for typical setups the amount of monitoring data can be easily transported in modern communication infrastructures, including networks on chips (NoCs).

5 Prototype Implementation and Results

To fully elaborate the concept, we developed two prototypes: a software prototype is used to demonstrate the suitability of the overall concept, and serves also as a case study for using the proposed concept to enhance an existing communication infrastructure.

In addition, we developed a hardware prototype aimed at monitoring traffic in contemporary high-performance bus-systems employed in current and future multi-core systems for core interconnection.

Our **software prototype** targets a sub-problem of distributed and heterogeneous architectures such as DodOrg: in such architectures, memory allocation, access, and access right management becomes crucial. Hence, we developed the concept of so-called Self-aware Memory [5], providing a more intelligent, scalable, and de-central memory management suitable for highly heterogeneous parallel systems with special focus on dynamic, adaptive systems.

Communication within SaM is based on a lightweight, hormone-inspired protocol, where each memory access (allocation request, grant, and read/write access) is therefore encoded as a unique event.

The monitoring approach proposed in this paper was applied to an existing SaM simulation environment [16] where it will be used as the information-gathering entity required for monitoring and correlating memory accesses to steer autonomous defragmentation, locality optimization, and brokering. This environment basically consists of a number of CPUs and memory banks connected through a shared interconnection network.

As a first step towards full self-management, we therefore extended the existing simulation infrastructure to provide associative counter arrays in each component to be able to detect incoming and outgoing accesses for each component, hence, monitoring individual commands to record typical request/response behavior taking place within the SaM protocol.

The setup was verified using a basic functionality test. For this test, we monitored randomly generated memory accesses, proving the functionality of our monitoring approach. This monitor was then subsequently used for further development of the

SaM memory allocation protocol, where we successfully verified and measured the behavior of the protocol additions and alterations.

With this setup, we were easily able to introduce and verify a new, more efficient allocation mechanism replacing the formerly used strategy by simply triggering to the individual access messengers (i.e. event IDs of individual accesses); while certainly overkill for just protocol optimization, the software prototype is, however, the initial and vital step to explore self-management, leading to fully autonomous self-optimization as required within the SaM context.

We also developed a **hardware prototype** to prove that our approach is suitable for high-performance communication technologies required for on-chip multicore connectivity as e.g. employed within DodOrg, and give a first estimation of the associated hardware costs. We therefore chose HyperTransport [9] as a state-of-the-art interconnection system, using an existing interface core [22], which was extended by an associative counter array to monitor memory accesses from CPU to memory. The array consists of 64 individual 8-bit counters with a tag size of 40 bits.

Targeting a Xilinx Virtex4FX100, this monitor accounts 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; compared to the original, unaltered core the monitor-equipped core shows an increase of 36% in logic and 33% in memory.

6 Conclusion

In this paper we presented a novel approach to sustained real-time monitoring. The approach not only introduces increased flexibility, but also addresses the specific topic of dynamically changing and self-adapting systems.

The approach employs associative counter arrays, introducing the flexibility required for such systems as counters are no longer bound a single event or predefined small group of events. Doing so requires the introduction of unique event IDs so that events are inherently self-defining. Applying a simple match mask enables scaling of the monitoring resolution so that counters may react to individual events or configurable groups of events, such as e.g. events coming from a distinct source or memory accesses with configurable address granularity.

This concept is biologically inspired and based on the hormone concept where solely the designated receiver decides whether to receive and how to react to hormones. No centralized monitoring instance exists, instead a distributed systems of producers (cells emitting hormones) and receivers (cells receiving hormones) exist. Hence, the entire system is inherently scalable and fail-safe.

To evaluate the concept, we extended a simulation infrastructure where we applied this style of monitoring. It required very little programming work and proved very successful in the process of expanding, optimizing, and verifying a communication protocol for de-central memory management and access.

We furthermore developed a hardware prototype to demonstrate the general suitability for high-performance real-time systems, and evaluated the hardware requirements such as chip area and ease of integration into existing hardware infrastructures.

By the outcome of this work we are convinced that the proposed method does provide the necessary flexibility and ease of use as required for dynamical and adaptive systems. We were able to show the required real-time capabilities, and the hardware requirements proved to be modest so that they can be easily integrated in existing interface structures such as the used HTX core.

We will therefore further pursue this work and not only refine the unique event identifier scheme and optimize our existing prototypes, but also be soon able to test our concept against real-world examples, running dedicated benchmark suites. Doing so will generate more universal results with respect to hardware requirements and communication overhead.

7 Acknowledgements

References

1. Uwe Brinkschulte Alexander von Renteln. Reliablity of an Artificial Hormone System with Self-X Properties. In *Parallel and Distributed Computing and Systems*, Cambridge, Massachusetts, USA, November 19-21 2007.
2. AMD. AMD Athlon processor, x86 Code Optimization Guide. 2002.
3. J.M. Anderson, L.M. Berc, J. Dean, S. Ghemawat, M.R. Henzinger, S.-T.A. Leung, R.L. Sites, M.T. Vandevoorde, C.A. Waldspurger, and W.E. Weihl. Continuous profiling: Where have all the cycles gone? In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, Oct 1997.
4. 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 (ARCS'06)*, volume P81 of *GI-Edition Lcture Notes in Informatics (LNI)*, pages 230–245, March 2006.
5. 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.
6. Compaq Computer. Alpha 21264 Microprocessor Hardware Reference Manual.
7. J. Fenlason and R. Stallman. GNU gprof: The GNU Profiler. 1997.
8. Robert Hockauf, Wolfgang Karl, Markus Leberecht, Michael Oberhuber, and Michael Wagner. Exploiting Spatial and Temporal Locality of Accesses: A New Hardware-Based Monitoring Approach for DSM Systems. In David Pritchard and Jeff Reeve, editors, *Euro-Par'98 Parallel*

Processing, 4th International Euro-Par Conference, Southampton, UK, September 1-4, 1998 Proceedings, volume 1470 of *Lecture Notes in Computer Science*, Berlin, September 1998. Springer Verlag.

9. HyperTransport Consortium. HyperTransport: Low latency Chip-to-Chip and beyond Interconnect. `http://www.hypertransport.org`.

10. IBM. PowerPC 740/PowerPC 750 RISC Microprocessor User's Manual. 1999.

11. Intel. Intel Itanium Architecture Software Developer's Manual. 2000.

12. Intel. Intel Architecture Software Developer's Manual Volume 3: System programming Guide. 2002.

13. C. Liao, M. Martonosi, and D.W. Clark. Performance monitoring in a myrinet-connected shrimp cluster. In *Proceedings of the International Conference on Measurement and Modeling of Computer Systems (Sigmetrics'98)*, Aug 1998.

14. T. Ludwig, R. Wismüller, V. Sunderam, and A. Bode. *OMIS – On-line Monitoring Interface Specification (Version 2.0)*. Shaker Verlag, Aachen, Germany, 1997. ISBN 3-8265-3035-7.

15. Thomas Ludwig and Roland Wismüller. OMIS 2.0 — A Universal Interface for Monitoring Systems. In M. Bubak, J. Dongarra, and J. Wasniewski, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 1332 of *Lecture Notes in Computer Science*, pages 267–276, November 1997.

16. Oliver Mattes. Developing a decentral memory management for distributed systems (self-aware memory). Master's thesis, Universität Karlsruhe (TH), 2007.

17. Sun Microsystems. Ultra-SPARC IIi User's Manual. 1997.

18. Motorola. MPC7450 RISC Microprocessor Familiy User's Manual. 2001.

19. P.J. Mucci, S. Browne, C. Deane, and G. Ho. PAPI: A portable interface to hardware performance counters. In *Proceedings of the Department of Depense HPCMP User Group Conference*, Jun 1999.

20. B. Sprunt. Pentium 4 performance-monitoring features. In *IEEE Micro*, pages 72–82, Jul/Aug 2002.

21. B. Sprunt. The basics of performance-monitoring hardware. In *IEEE Micro*, pages 64–71, Jul/Aug 2002.

22. Ulrich Brüning et al. HTX Board Universal Reference Design. `http://www.hypertransport.org/products/productdetail.cfm?RecordID=75`.