

ASoCS: An Architecture Concept for Self-Optimizing Parallel and Distributed Computer Systems

Rainer Buchty, Jie Tao, Wolfgang Karl
Universität Karlsruhe (TH)
Institut für Technische Informatik
Zirkel 2, PO Box 6980
76128 Karlsruhe, Germany
{buchty|tao|karl}@ira.uka.de

Georg Acher, Jürgen Jeitner, C. Trinitis
Technische Universität München
Institut für Informatik, LRR
Boltzmannstr. 3
85748 Garching bei München, Germany
{acher|jeitner|trinitic}@in.tum.de

Abstract

Application optimization usually means optimization for a given, static computer architecture. In such systems, the programmer optimizes towards given objective functions with respect to the existing system. This scenario changes dramatically when dealing with reconfigurable systems. Here, the underlying system architecture may change and be changed during runtime. Such changes might be both, measures of self-healing or a direct result of self-optimization. In any case, it is not possible to optimize in a classic sense anymore since the objective functions might change with each reconfiguration. Optimization therefore becomes an effort of finding a best compromise between – sometimes contradictory – objective functions and the current system configuration.

In this paper we outline an architecture- and application-independent framework for self-organizing parallel computer systems. Main focus of this framework is to provide a novel, uniform approach to modeling Self-X systems. Core of this framework is a layer model with flexible, decentralized monitoring and adaptive components on each system layer. Monitoring components feed an adaptive planning stage, which evaluates this data against given objective functions and instruments the adaptive components to perform required system updates.

The suitability and applicability of this framework is demonstrated by successful adaptation of two example scenarios: data locality optimization serves as a software-only application example, whereas a dedicated Software-defined Radio architecture is presented as an example for an architecture based on the framework sketched in this paper.

1 Introduction and Motivation

As a matter of constantly rising system complexity, modern systems are increasingly hampered by system deployment failures, hardware and software issues, and human errors.

Growing system complexity combined with increasing usability and reliability requirements have to lead to systems with self-organizing characteristics and capabilities. Such systems require introspection to acquire system-wide state-information to tune for desired performance, energy consumption, reliability, security, or other metrics. In a typical self-organizing system, a monitor probe will acquire state, and then a steering component will adapt the system – either dynamically at runtime, or statically as in profile directed feedback techniques for future executions.

Role model for such systems is the human body's vegetative nervous system which is able to flexibly and autonomously react to external influences. Such flexibility and autonomy is required to cope with increasing system complexity. To achieve these abilities, a computer system must develop certain basic properties, commonly noted as Self-X capabilities and typically defined as *Self-Awareness*, *Self-Configuration*, *Self-Optimization*, *Self-Healing*, and *Self-Protection*.

A system built upon these properties requires a closed control or feedback loop calling for flexible and powerful monitoring resources on all system layers. It is vital to avoid restriction to some few monitoring points as present in current systems: by extending monitoring to all system layers, it is possible to exploit emergence effects. This emergence will contribute to improved self-awareness.

Based on this data, a more suitable (i.e. optimized) system configuration can be determined. Reconfiguration requires presence of adaptive components on all system layers to supply necessary reconfiguration capabilities. Amount of reconfiguration is determined by analysis of monitoring data and evaluation against given rules or problem specifications, so-called objective functions. Such functions describe the desired, optimal system behavior for a given objective. Although each objective function is well defined, no equally well-defined corresponding optimal system state exists because of the often contradictory nature of objective functions. Optimization therefore means finding a best-suiting compromise to fulfill the objective functions' requirements. Hence, a weighing and evaluation process is required. This process is based on application type, field of use, and additional system conditions. Further complexity is added from the fact that objective functions are not necessarily one-dimensional but can also be multi-dimensional, aggregate functions.

Figure 1 shows a closed control loop based on the scenario described above: an existing system is enhanced by monitoring and adaptive components as required. The monitoring infrastructure supplies status data to the automatic planning stage. Within this stage, monitoring data is evaluated against objective functions, and required system reconfiguration is initiated. Reconfiguration takes place by instrumenting adaptive components.

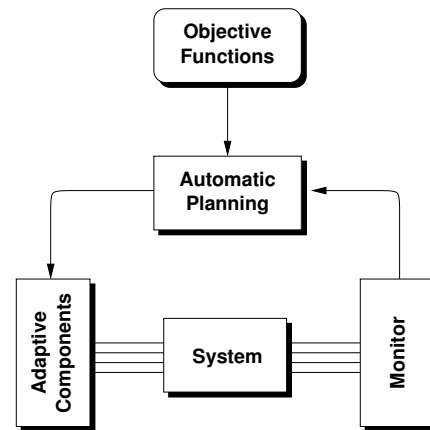


Figure 1: Closed-control system

In this paper we will present a novel architecture framework, ASoCS, targeting development of self-optimizing computer systems and applications for such systems. This framework is most versatile and neither bound to a specific target system architecture nor application scenario.

We will first describe existing approaches in Section 2 and then present our architecture in detail in Section 3. Section 4 will then show a prototypic implementation, where we used our architecture concept to enhance existing work within the field of NUMA data locality with self-x concepts. Section 5 briefly describes an architecture concept for Software-defined Radio applications based upon the concepts of the ASoCS framework. The paper closes with a conclusion in Section 6.

2 Related Work

The proposed architecture combines efforts from various research areas. Topics involved are monitoring, data assembly and interpretation, and system reconfiguration.

Monitoring techniques are used to gather data from various system layers. This data is then processed, interpreted, and evaluated against given objective functions. Following, a reconfiguration cycle takes place which will optimize the system with respect to objective functions. This optimization process ideally runs automatically without human interaction, i.e. the system turns into what is typically called an *autonomous system*.

Certain aspects of the above problems have been targeted in current and previous research activities and industrial initiatives. Most prominent is IBM's *Autonomic Computing* initiative targeting automated system administration without need for human interaction [13]. Similar projects exist from other companies [20, 3] and in academia [6, 7]. On hardware level current research interest focuses on feedback loops to adjust computing power and energy efficiency [5, 2].

Data for such feedback loops is provided by monitoring. On lowest hardware level, processor architectures offer so-called event counter registers offered by many processor architectures. These registers provide neces-

sary information for application profiling. Number and use of these registers are dependent on the individual architecture. Counter-based methods typically suffer from four basic limitations explained in detail in [23] and are complemented by additional monitoring techniques such as software-based profiling [8], or embedding monitoring routines on driver level [16, 12]. On higher level, monitoring devices are usually decoupled from post-processing software by APIs as described in [21], [1], or [17, 18, 19].

From the previous subsections we see that existing implementations and infrastructures are inherently limited. Implementations and applications have a rather limited scope resulting from the addressed scenario, and existing monitoring infrastructure in hardware is fixed and hardly configurable. Furthermore, no standardized universal Monitoring API exists, but are again bound to their specific field of use. An approach into this direction was taken within the APART project [10] and the associated EP-Cache project [9]. These, however, only target monitoring and analysis in parallel and distributed systems.

With respect to self-organizing systems, still no uniform, application-independent, and standardized monitoring API exists including reporting existing monitoring resources, permitting access to these resources, and – regarding self-organization – enabling reconfiguration.

3 The ASoCS Architecture Concept

In this section we outline our novel architecture concept for self-organizing or organic systems. This concept is based on a closed control loop system as sketched in Section 1 of this paper.

Key to self-optimization is self-awareness. A self-x architecture must first-most be able to detect and evaluate events on all system layers. However, a uniform monitoring infrastructure is not sufficient: to be able to deal with changing objective functions and therefore changing best-effort optimization scenarios, amount and type of monitoring and interpretation of this monitoring data also changes. Furthermore, in a changing system evaluating monitoring data by one distinct instance is not sensible. Especially with respect to parallel systems such analysis has to happen in-place, i.e. decentralized, where possible. However, data from all monitoring instances has to be correlated to enable system-wide best-effort optimization: using techniques like performance prediction and pattern matching adds the necessary amount of decentralized “intelligence” to fulfill required in-place optimizations.

We put therefore special focus on a novel monitoring infrastructure which is not only embedded into system layers, but furthermore contains necessary intelligence to enable decentralized analysis of data and enable correlation of all monitoring data. Hence, we introduce the concept of monitoring capsules (MC) and monitoring modules (MM), which we will explain in the following paragraph. Monitoring capsules provide appropriate interfacing required to dock or plug monitoring modules into the respective system layer. MCs are present on all system layers and can be considered as the low-level interface in which necessary monitoring functionality can be plugged. This functionality is provided by the monitoring modules which consist of the sensory part (data pickup) and defined pre-processing capabilities.

Splitting the monitoring resources into capsule (interface) and module (functionality) enables exchange of monitoring modules as required, so that the monitoring infrastructure itself can be reconfigured depending on the actual requirements. Monitoring modules will be stored in a repository from where they can be retrieved and docked into the appropriate monitoring capsule.

A dedicated API will serve as an abstraction layer and enable access to monitoring capsules and adaptive components by other system services, especially the adaptive planning (AP). AP is a virtual system service responsible for interpretation and evaluating monitoring data against given objective functions. It is virtual in a sense that it is no monolithic instance but pervades all nodes of a system and all layers within a single node. Based on the AP’s evaluation required monitoring modules are loaded into the respective capsules.

Data collected by the monitoring infrastructure will be buffered in a local performance repository. To explicitly address parallel and distributed systems, this data can be merged with local data from additional system nodes resulting in a global performance repository enabling scalability of the entire performance repository system. A dedicated query interface provides access to all local performance repositories and the monitoring infrastructure.

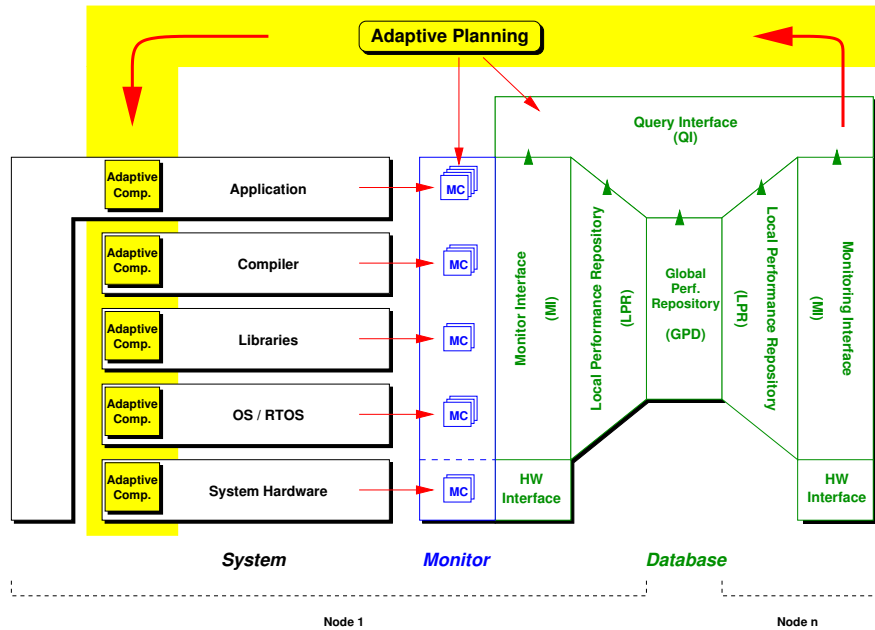


Figure 2: Layer concept for adaptive/autonomic computing systems

This query interface is used by AP which evaluates collected data against objective functions in a decentralized manner to determine appropriate system reconfiguration. This evaluation is based on certain metrics quantizing system parameters and objective functions; when adapting this architecture to distinct application, it is necessary to investigate and evaluate existing metrics for use in adaptive planning and to eventually develop novel metrics aiding in the evaluation process where needed.

The overall architecture concept is depicted in Figure 2. It enhances and refines the control loop scheme pictured in Figure 1. A system is split into a 5-tier hierarchical scheme: the bottom layer is formed by the system hardware, with the (Real-time) Operating System (OS) including hardware drivers on top. The OS is assisted by libraries which in turn are required by the compiler to finally create the desired application. Depending on its type, an application might influence only some or all levels. On each hierarchy level monitoring capsules exist. These capsules can be loaded with monitoring modules stored in a module repository. Data collected and preprocessed by monitoring modules is stored in a local performance repository.

Figure 3 shows a graphical representation (adaptation disc) of this feedback loop. This disc model provides a general template for structured and interoperable design of self-organizing or organic systems by independent groups: every single adaptive process of a self-organizing or organic system can be represented as a disc with segments for each component involved in the adaptive process. Each instantiation of a disc represents an optimization towards a particular objective function. A disc's sector comprises functionality or components to be configured to reach the respective aim, i.e. fulfill the objective function's requirements in a best-effort manner. As an illustration of use, Figure 3 shows discs for the two selected application scenarios presented in the following sections of this paper.

A single system can employ numerous discs representing various, different objective functions or different incarnations of a single objective function. One could illustrate such a collection of disc as a Wurlitzer machine: depending on the given base scenario (application and initial system configuration) a set of discs is addressed and "loaded". By doing so, the monitoring capsules will be loaded with required monitoring modules including the proper initialization of decentralized in-place analysis and evaluation.

During runtime, discs may be changed or replaced based on actual circumstances such as observed performance bottlenecks, user-defined system priorities, faults, or external influences. In such a case the system

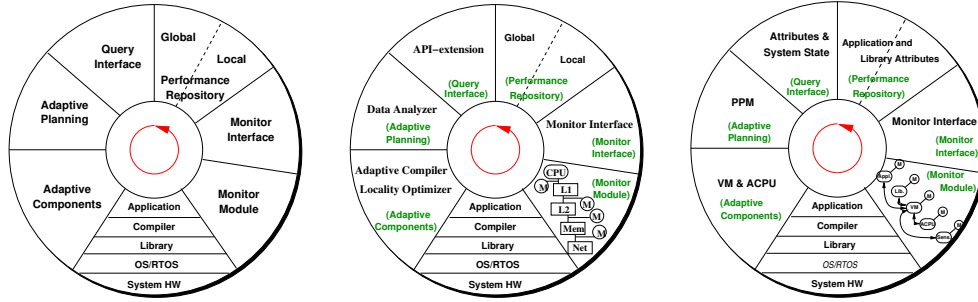


Figure 3: Generic disc (left), applied to Data Locality Optimization in NUMA systems (middle), and a dedicated SdR architecture (right)

selects the appropriate disc(s), loads required modules or performs required configuration described by each disc segment to initiate the adaptive mechanism described by the particular disc.

Contrary to projects and initiatives listed in Section 2, our framework is application- and system-independent, and addresses all system layers rather than specific system parameters; it is furthermore applicable to single nodes as well as parallel and distributed systems:

Decentralized and flexible **data acquisition methods** formed by Monitoring Capsules and Monitoring Modules, i.e. interface and functionality. **Adaptive components** provide the required amount of reconfigurability. Both, monitoring subsystem and adaptive components are present on every system layer.

To provide a uniform interface to this vast amount of – also changing – components, **interface, API, and management services** need to be well defined. By doing so, a hierarchical management service can be created which eases gathering of status information as well as initiating system reconfiguration.

By employing **Performance Repositories** and a dedicated **Query Interface (QI)** ASoCS scales with the size of a given system. The local performance repositories buffer and aggregate performance data of single system nodes and assist in generating a complete and emergent view of the entire system.

Adaptive planning receives status information from the monitoring infrastructure and evaluates this against one or more given objective functions. Based on this appropriate reconfiguration is initiated. AP will be responsible for coordinating self-awareness and self-configuration resulting in self-optimization, self-protection, and self-healing.

4 Prototypical Implementations

In order to evaluate the ASoCS concept, we targeted two different application scenarios. First, we have built a prototypical architecture and developed several components for a feedback loop with respect to data locality optimization on NUMA architectures. The reason for choosing NUMA locality as the initial objective function lies in the fact that we have been doing research work in the area of shared memory programming on top of NUMA architectures [22]. The second implementation is an architecture design for Software-defined Radio applications which we chose as an example for streaming-media architectures.

4.1 Data Locality on NUMA Systems

A typical feature of NUMA (Non-Uniform Memory Access) machines is their specific memory organization. On a NUMA machine, the main memory is distributed over the system, but globally organized into a shared virtual memory accessible from all processor nodes. Due to the different property of local and remote memory accesses, however, references targeting a remote memory can take up to two orders of magnitude longer than local accesses. As a consequence, unoptimized applications often suffer from poor data locality and the resulting high memory access latencies. For tackling this locality issue we use ASoCS to build a feedback loop.

4.1.1 Structure of the Feedback Loop

Figure 4 depicts how we map this problem onto the general framework of ASoCS. First of all, such data locality optimization needs a set of system parameters such as memory access distribution and remote access characteristics.

We assume Monitoring Capsules integrated in the NUMA network interface capable of observing all remote memory transaction. On top of these MCs, we use low-level APIs for preprocessing the original monitoring information. From there, data is aggregated and combined into the Global Repository. This work is done by the OMIS/OCM monitoring interface. Besides, OMIS/OCM also provides a Query Interface that delivers the memory locality information to higher levels. Finally, an Adaptive Planning and an Adaptive Component are needed for self-tuning. The former is implemented with a Data Analyzer that automatically analyzes the runtime interconnection traffic and detects access hot spots, while the latter is achieved with a Migration Component that transparently modifies the data layout via moving data to its dominating node.

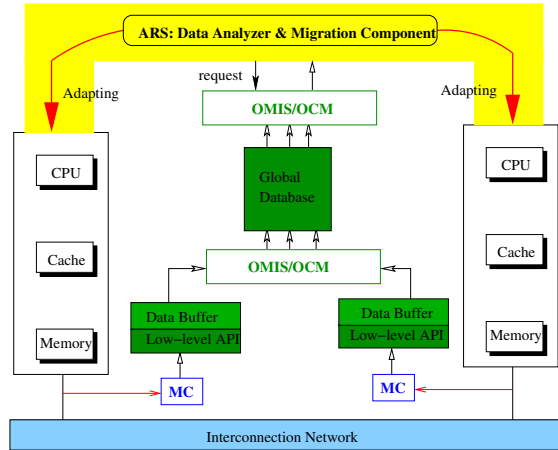


Figure 4: Memory Locality Adapting on NUMA Systems.

Monitoring Capsule: The design of this component is based on the SMiLE hardware monitor [15] which has been developed for observing the interconnection traffic on the SMiLE (Shared Memory in a LAN Environment) cluster connected via the Scalable Coherent Interface (SCI), a low-latency, high-bandwidth interconnection technology. For acquiring comprehensive data about the inter-node communication, we have designed two analysis modules for monitoring the memory transaction: static and dynamic. The former allows to explicitly program the hardware for event triggering and action processing on special memory regions of interest, while the latter is based on histogram-driven monitoring, in which all memory transactions through the local interconnect bus are monitored in order to provide fine-grain monitoring statistics across the complete application's working set.

A hardware Monitoring Capsule is also designed to hold these analysis modules, which can be dynamically loaded into the capsule at the runtime. The capsule implements two interfaces: a link interface and a PCI interface. The former is used to snoop a local bus, which connects a single node to the actual interconnection fabric, and extract information from the transactions delivered over this bus, while the latter, an interface between the PCI bus and the monitor, offers direct access to the host node enabling the users or system software to configure the hardware monitor and read the gathered monitoring data.

Low-level API and the Data Buffer: In order to avoid delivering fully detailed monitoring data which is not essential for further use, we have implemented a library of functions for data processing. This PAPI-like [4] standard API contains a set of routines capable of generating statistical information in the form of e.g. memory access histograms that show the number of accesses from all processor nodes to the whole working set at different granularity. These histograms can be used to detect communication bottlenecks where required data is mainly acquired from a remote processor node. For locally storing this monitoring data a Data Buffer is maintained on each processor. The buffer is organized as a histogram chain in order to enable fast searching of the needed information.

Data Aggregation and Querying Interface: We use the OMIS/OCM [24] monitoring system to combine monitoring data from all nodes. OMIS (On-line Monitoring Interface Specification) is a specification of an interface between a programmable on-line monitoring system for distributed computing and the tools that reside on top of it. It offers two interfaces: one for the interaction with different tools and the other for the interaction with the program and runtime system layers. OCM is an OMIS Compliant Monitoring system adhering to OMIS. It has been implemented for a series of loosely coupled environments including clusters and NoWs and

has initially been designed for message passing tools. It is structured into a core and several extensions. For this work we extended OCM for providing services with respect to the access of data delivered by the Monitoring Capsules. Further, we have extended OCM with a high-level Query Interface that provides the memory locality information to higher levels.

Adaptive Planning and Component: The remainder components include a Data Analyzer (Adaptive Planning) and a Migration Component (Adaptive Component). The former is used to analyze the monitoring data and determine whether to move a data page to another processor node. Based on the Querying Interface, the Data Analyzer is capable of accessing the memory access histograms created by the low-level API. It then compares the number of accesses to a data page from all processor nodes. If the accesses performed by a remote node exceed a predefined threshold, it is decided to move this page to the remote node. This decision is then delivered to the Migration Component, which uses system calls to move the data from the original location to the node that more requires it. This kind of adapting is performed periodically either at specified time or by synchronization points, and is held during the whole execution of the applications.

A critical issue with this approach is the migration algorithm used to make the migration decision. Commonly used page migration mechanisms are based on competitive algorithms, which migrate a page if the difference between the number of local references and the number of remote references concerning one node exceeds a predefined threshold. A similar one, called U-Mig, is also proposed within this work. As the local accesses can not be acquired by the monitors, the migration decision is based on the references performed by all remote nodes on the page under consideration. If the difference between the number of the remote accesses from the dominant node and the average remote accesses performed on the page exceeds a threshold, it is decided to move the page to the dominant remote node.

Using this algorithm, however, a correct decision can be made only after a large amount of references have been issued, resulting in late migrations and thereby a loss of performance. We implement therefore several novel migration algorithms, which base their analysis on memory references to multiple shared pages, in a way that the accesses to a set of pages are combined using a weighted distribution.

Overall, we have implemented a closed feedback loop for adapting the data distribution on NUMA systems. At the same time, we also established the basis framework of the ASoCS architecture. Most components within this framework can be applied to build other feedback loops with slight extension. For example, we are currently working on an adaptation disk for improving the cache performance. Monitoring data is acquired from cache monitors; the established repositories and Query Interface are directly applied; the existing Adaptive Planning component is slightly extended; and a new Adaptive Component is under development.

The prototypical implementation of the proposed architecture has been benchmarked with several OpenMP applications from the NAS parallel benchmark suite [14] and the SPLASH-2 benchmark suite [25]. It was an interesting observation to see that no constant threshold is optimal with respect to various applications; however, for the adaptive threshold all applications showed a good, either same or just slightly worse than the optimal threshold factor, performance. To give room for the second implementation, detailed results have been omitted but can be supplied upon request.

4.2 Software-defined Radio Architecture

In this Section, we describe an architecture suited for Software-defined Radio applications based upon the ASoCS principles.

The proposed architecture consists of three logical building blocks: on lowest level, an Adaptive CPU (ACPU) provides the actual reconfigurable computing hardware and necessary monitoring infrastructure. The actual hardware configuration is abstracted and assisted by a Virtual Machine (VM). and the overall system is controlled and configured by a dedicated management unit called Power-Performance-Management (PPM).

All these units are interconnected to form the closed control loop, the resulting architecture is shown in Figure 5.

4.2.1 Virtual CPU and Binary Compatibility

ACPU and VM together form the Virtual CPU (VCP) as seen by the programmer. The ACPU itself is a quasi-RISC architecture resembling a subset of common RISC architectures [11] enhanced by dedicated reconfigurable functional slots. These can be loaded with arbitrary application-supporting functions. The VM is used to ensure a configuration-independent binary format, therefore enabling binary compatibility among different ACPU configurations, and also keeps track of current resource usage and reports this information to the PPM.

An application is therefore compiled into a configuration-independent P-code representation and processed by the VM: upon processing a supporting instruction call, the VM decides whether this call should be executed natively, i.e. using a dedicated hardware instruction loaded into the ACPU, or call a software library function which emulates that instruction's behavior using either the static instruction set or, depending on the ACPU's configuration, less monolithic dedicated instructions.

This decision is made based on so-called attributes, which are part of the application description as well as the soft- and hardware system libraries. For an application these attributes define application requirements such as required minimum throughput, variable resolution, etc. and also list mandatory supporting instructions.

Similar information is provided in the system libraries which provide special operations to be loaded into the reconfigurable hardware as well as software representations of these operations based on the fixed standard instruction set.

Thus, the compilation stage not only involves compiling the source program into P-code to be processed on the VM, but also an initial configuration of the adaptive CPU, and optimization parameters for the PPM are created. These are based on comparing and weighing of all attributes. In addition, information about program phases is extracted to ease detection of initialization versus main computation loop. This avoids unnecessary optimization cycles.

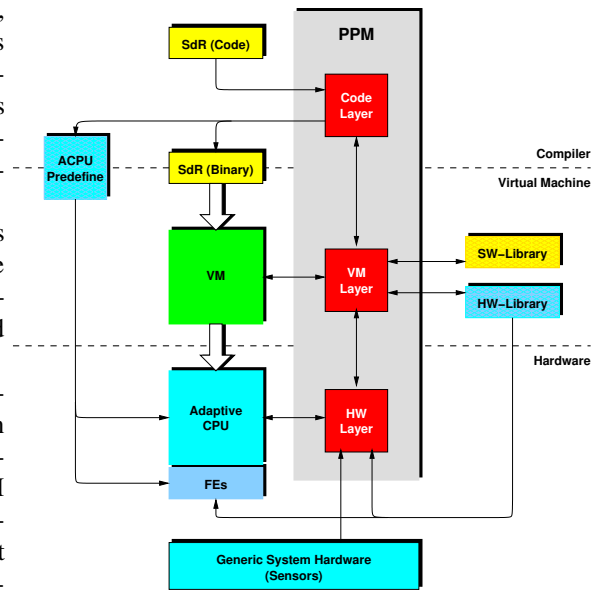


Figure 5: SdR Architecture

4.2.2 Performance-Power-Management

PPM forms the overall control unit of the proposed architecture. During compilation stage, the PPM assists the compiler in initial distribution of the application into generic program and dedicated function calls. With the help of PPM, also initial ACPU configuration is performed. These steps are based on the information contained in the attributed application. Further attributes are used to instruct PPM about the objective functions, i.e. what optimizations to perform. The PPM will also derive certain min/max parameters from the attributed application which are later used to evaluate the system configuration and decide whether reconfiguration should take place or not.

During program execution, the PPM serves as a control loop. It takes parameters reported by VM, ACPU, and external sensors to evaluate the current system configuration and determine possibly required system reconfigurations.

In case of reconfiguration, parts of the application are either moved from software to hardware execution, i.e. replacing function calls by dedicated instructions, or vice versa, or switching between similar representations of an algorithm fulfilling the changed requirements, such as e.g. enhanced precision. In parallel to hardware reconfiguration, the VM is instructed about the changes to properly translate P-code into appropriate instructions and function calls.

We are actively working on the implementation of this architecture, thus no measurements could be performed so far. As of now, the ACPU is being built in hardware and necessary code generation software (GNU binutils, GNU C compiler) are ported.

5 Conclusion

In this paper we described a novel architecture concept for self-optimizing parallel and distributed computer systems, and demonstrated applicability of this concept through a prototypical implementation.

The need for such an architecture concept was discussed in the first section, where it was shown that future systems must employ so-called Self-X capabilities to overcome current limitations resulting from increased system complexity. A basic sketch of such Self-X or autonomic systems and expected problems were outlined. Related work was presented, and it was shown that existing concepts and solutions currently address only certain aspects of autonomic computing or are inherently limited to dedicated applications and application scenarios.

Following, we presented the ASoCS concept in detail. The concept is application independent and scalable, thus able to address single-node as well as parallel and distributed systems. The applicability of this concept was demonstrated by a prototypical implementation targeting two independent fields, Data Locality optimization on NUMA systems, and power vs. performance optimization for SdR applications. For the first implementation, detailed information was given how the locality optimizer was modeled after the ASoCS concept, and early simulation results based various benchmark applications were shown.

References

- [1] 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.
- [2] R. Balasubramonian, D. Albonesi, A. Buyuktosunoglu, and S. Dwarkadas. Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. In *Proceedings of the International Symposium on Microarchitecture*, Dec 2000.
- [3] Jamie Beckett. Scaling IT for the Planet: Creating the worldwide computing utility. <http://www.hpl.hp.com/news/2001/oct-dec/planetary.html>.
- [4] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci. A portable programming interface for performance evaluation on modern processors. *The International Journal of High Performance Computing Applications*, 14(3):189–204, Fall 2000.
- [5] E. Chi, M. Salem, I. Bahar, and R. Weiss. Combining software and hardware monitoring for improved power and performance tuning. In *Boston Area Architecture Workshop (BARC) 2003*, Jun 2003.
- [6] Doug Burger et al. Scaling to the End of Silicon with EDGE Architectures. In *IEEE Computer*, pages 44–55, Jul 2004.
- [7] Philip K. McKinley et al. Composing Adaptive Software. In *IEEE Computer*, pages 55–65, Jul 2004.
- [8] J. Fenlason and R. Stallman. GNU gprof: The GNU Profiler. 1997.
- [9] Michael Gerndt and Wolfgang Karl. EP-Cache. February 2005. <http://www.bode.cs.tum.edu/gerndt/home/Research/EP-Cache/EPcache.htm>.
- [10] APART IST Working Group. Automatic Performance Analysis: Real Tools. 2004. <http://www.kfa-juelich.de/zam/RD/coop/apart/>.

- [11] Christian Heßmann. Design and Implementation of a Virtual Machine for efficient Translation and Execution in RISC Environments. 12 2004. Diploma Thesis.
- [12] 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.
- [13] Paul Horn. IBM's Perspective on the State of Information Technology.
<http://www.research.ibm.com/autonomic/manifesto>.
- [14] H. Jin, M. Frumkin, and J. Yan. The OpenMP Implementation of NAS Parallel Benchmarks and Its Performance. Technical Report NAS-99-011, NASA Ames Research Center, October 1999.
- [15] W. Karl, M. Leberecht, and M. Oberhuber. SCI Monitoring Hardware and Software: Supporting Performance Evaluation and Debugging. In *SCI Scalable Coherent Interface Architecture and Software for High-Performance Compute Clusters*, volume 1734 of *Lecture Notes in Computer Science*, chapter 24, pages 417–432. Springer-Verlag, Berlin, 1999.
- [16] 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.
- [17] 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.
- [18] 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.
- [19] Thomas Ludwig, Roland Wismüller, and Arndt Bode. Interoperable Tools based on OMIS. In *Proc. 2nd SIGMETRICS Symposium on Parallel and Distributed Tools SPDT'98*, page 155, Welches, OR, USA, August 1998. ACM Press. <http://www.acm.org/pubs/citations/proceedings/metrics/-281035/p155-ludwig/>.
- [20] Scott McNealy, Greg Papadopoulos, and Jonathan Schwartz. N1: Revolutionary IT Architecture for Business. <http://www.sun.com/software/solutions/n1>.
- [21] P.J. Mucci, S. Browne, C. Deane, and G. Ho. PAPI: A portable interface to hardware performance counters. In *Proceedings of the Department of Defense HPCMP User Group Conference*, Jun 1999.
- [22] M. Schulz, J. Tao, C. Trinitis, and W. Karl. SMiLE: An Integrated, Multi-paradigm Software Infrastructure for SCI-based Clusters. In *Proceedings of the IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid)*, pages 247–254, Berlin, Germany, May 2002.
- [23] B. Sprunt. The basics of performance-monitoring hardware. In *IEEE Micro*, pages 64–71, Jul/Aug 2002.
- [24] R. Wismüller. Interoperability Support in the Distributed Monitoring System OCM. In *Proceedings 3rd International Conference on Parallel Processing and Applied Mathematics - PPAM'99*, pages 77–91, Kazimierz Dolny, Poland, September 1999.
- [25] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: characterization and methodological considerations. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 24–36, June 1995.