

Data Monitoring in High-Performance Clusters for Computing Applications

G. Torralba, V. González, E. Sanchis, J. Tao, M. Schulz, and W. Karl

Abstract—The shared memory in a LAN-like environment (SMiLE) project at Lehrstuhl für Rechnertechnik und Rechnerorganisation, Technical University of Munich (LRR-TUM) investigates in high-performance cluster computing using system area networks. In the context of this project, a hardware monitor is being developed to observe the system area network (SAN) traffic. This hardware monitor is, therefore, capable of delivering detailed information about the run-time communication behavior of applications running on SMiLE clusters. The central part of this monitor consists of a content-addressable counter array managing a small working set of the most recently referenced memory regions.

Index Terms—Cluster, scalable coherent interface, monitoring.

I. INTRODUCTION

THE shared memory in a LAN-like environment (SMiLE) project [1] at the Lehrstuhl für Rechnertechnik und Rechnerorganisation, Technical University of Munich (LRR-TUM), investigates in high-performance cluster computing. It utilizes the scalable coherent interface (SCI) as the interconnection technology. This network fabric, which is standardized in the IEEE standard [2], uses a state-of-the-art split transaction protocol and currently offers link speeds of up to 667 MB/s with the latest LinkController3™ chip (LC3) [3]. The basic topology for scalable coherent interface (SCI) networks consists of small ringlets (as shown in Fig. 1) to avoid the potential large latency caused by passing a packet through many intermediate nodes. These basic structures can be hierarchically organized to form systems containing up to 64-K nodes.

One of the fundamental features of SCI is the capability to access directly remote physical memory without software intervention resulting in a hardware distributed shared memory (DSM) abstraction. This can be used to implement efficiently both message-passing and shared-memory programming models. The latter is implemented by merging the hardware capabilities for DSM of SCI [hardware distributed shared memory (HW-DSM)] with software DSM techniques. This creates a hybrid DSM system that allows to exploit directly the communication benefits of the underlying network while still

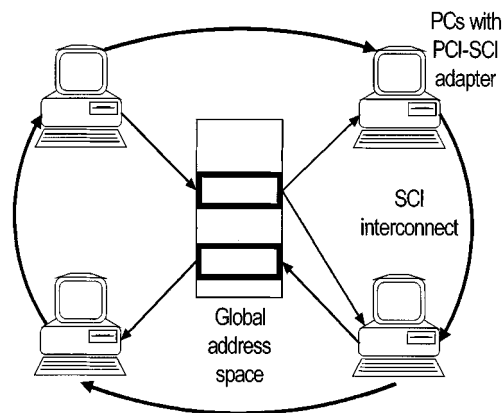


Fig. 1. SCI-based personal computer (PC) clusters with nonuniform memory access (NUMA) characteristics.

providing a true-shared memory abstraction to the programmer [4].

It is often the case that applications initially perform much worse than expected and require extensive fine tuning. While the performance of message-passing codes can be easily assessed and optimized using standard tools, the same task is much more difficult for shared memory codes in such a hybrid-DSM environment. The main source of inefficiencies is often excessive remote-memory access across the network. Although via hardware-supported DSM these accesses deliver an extremely high communication performance, they are still an order of magnitude more expensive than local ones. To improve the efficiency of parallel applications a tool set consisting of both a run-time adaptive system with a locality optimizer and comprehensive visualization is needed. The goal of such a tool set is to improve the locality of memory reference and to minimize remote memory accesses.

Unfortunately, the growing complexity of hardware makes this kind of performance evaluation and characterization more difficult, especially in a DSM environment on top of a NUMA architecture like a PC-cluster with HW-DSM support. The challenge is to design a powerful performance monitor able to deliver detailed information about the run-time communication behavior and to help the run-time adaptive system in making correct decisions concerning partitioning and redistribution of data and threads.

A hardware monitor alone, however, cannot be the final stage. A comprehensive on-line monitoring infrastructure must be developed along with it [5] to allow the analysis and visualization of the observed behavior and the on-line optimization of applications, either automatically through run-time intervention or by providing the programmer with precise performance hints.

Manuscript received June 15, 2001; revised January 14, 2002. This work was supported in part by ESPRIT Project EP 22582 Working Group on the Scalable Coherent Interface SCIWG.

G. Torralba, V. González, and E. Sanchis are with the Department of Electronic Engineering, University of Valencia, Burjassot, Valencia 46100, Spain (e-mail: gloria.torralba@uv.es; vicente.gonzalez@uv.es; enrique.sanchis@uv.es).

J. Tao, M. Schulz, and W. Karl are with the Department of Computer Science, Technical University of Munich, LRR, D-80290 Munich, Germany (e-mail: tao@in.tum.de; schulzm@in.tum.de; karlw@in.tum.de).

Publisher Item Identifier S 0018-9499(02)03917-5.

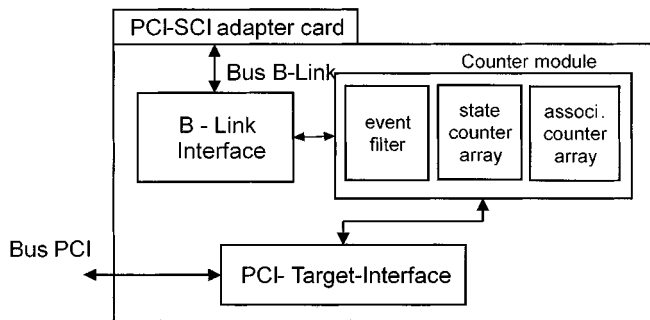


Fig. 2. Architecture of the SMiLE hardware monitor.

The main goal of all tools implemented within this framework is to improve the locality of memory accesses through a tight cooperation of monitoring hardware and tools [6].

The remainder of the paper is organized as follows. Section II describes the general design principles of the hardware monitor, followed by the description of the interface to the network, the so-called B-Link interface, in Section III. The counter module and the host interface are explained in Sections IV and V, respectively, while the visualization tool is introduced in Section VI. Section VII presents the current overall status and introduces some continuing work planned for the future. The paper is then rounded up by the discussion of related work in Section VIII and some concluding remarks in Section IX.

II. SMiLE HARDWARE MONITOR

The SMiLE hardware monitor [7], as shown in Fig. 2, consists of three modules: a B-Link interface, a counter module, and a peripheral component interconnect (PCI) interface in cooperation with the commercially available PCI bridge PCI 9060 [8]. The B-Link interface is responsible for acquiring the network traffic and it is the connection to the SCI network adapter, in this case a PCI-SCI adapter card developed at TUM. The data is then handed on to the counter module, which keeps track of data related to the parameters defined by the user, which can then be transferred to the host through the PCI interface.

III. B-LINK INTERFACE

The B-Link is a 64-bit-wide synchronous bus and a logical part of the PCI-SCI adapter [9]. It serves as the carrier of all the packets going to and coming from the SCI link. It has been defined by Dolphin interconnect solutions (ICS) [10] for their own SCI adapter cards and it is the host-side interface of the Link Controller1 link chip (LC-1) [11], which is responsible for transmitting any SCI packet to the respective node. Therefore, the B-Link interface is the central point of the hardware monitor on which all remote memory accesses can be monitored. The B-Link packet format is shown in Fig. 3. The monitored information includes transaction commands (read, write, lock, or unlock), source and destination identifiers, physical memory addresses and a description of the packet type (incoming, outgoing, response, or request).

The information needed by the counter module can be found in the first and second words. The actual payload data and the

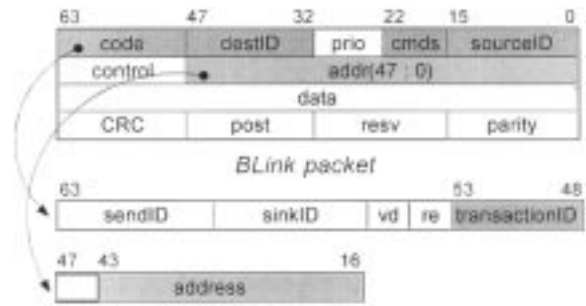


Fig. 3. The B-Link packet at the PCI-SCI adapter card [9].

next fields are not relevant for monitoring and are, therefore, ignored.

The packet type information is extracted from the first 64 bits of the B-Link packet, more specifically from the following fields.

- Code: only needed transaction identifiers (transactionID).
- Command: for decoding response, request, write, read, or lock packet (cmds).
- Destination identifier: target node for packet (destID).
- Source identifier: source node for packet (sourceID).

The current monitor is only intended for clusters with up to 32 nodes, which reduces the bit range of the last two fields to 5 bits.

The address information is extracted from the B-Link packet in the second 64 bits. The output generated is the SCI-address, which is the page-number or physical location of the data higher address (16 bits) and the page-offset or physical location of the data lower address (10 bits).

The storage of the transaction identifiers and addresses is implemented in the B-Link interface using a 64×27 bit SRAM. This memory is basically used as a translation table to look up and retrieve addresses (page-number and page-offset) for each transactionID if the packet type is “response.” If the packet type is “request,” the transactionID and the address are stored into the table. Additionally, a column has been added to indicate actions that trigger responses (read or lock).

IV. COUNTER MODULE

The primary part of the hardware monitor is the counter module also called flexible range logic (FRL). It receives the monitored event information from the B-Link interface block. Its three components—the event filter, the static counter array, and the associative counter array, depicted in Fig. 4—allow the programmer to use the FRL in two different working modes: static and dynamic. The static mode, for which the event filter and the static counter array [12] are responsible, allows users to set up. It process actions on freely definable SCI regions. This mode can be used to monitor data structures or parts of arrays and it will likely be applied in combination with language-specific profiling extensions. The dynamic working mode [13] is a histogram-driven monitoring in which the counting is not controlled by events, as in the static mode, but rather all packets through the B-Link are monitored. This provides a fine-grained monitoring statistics across the complete working set of the application. The dynamic mode is, therefore, suitable to obtain

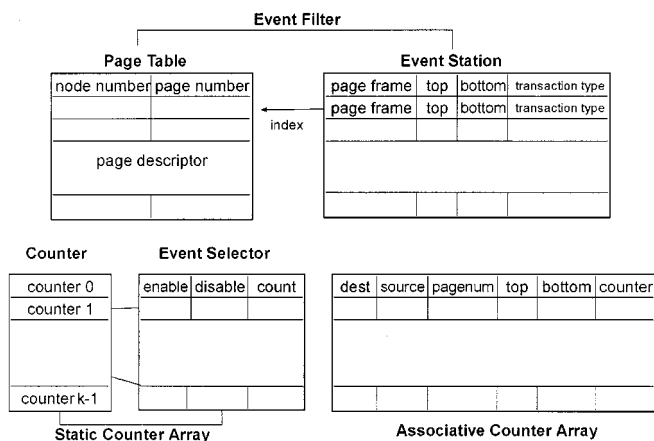


Fig. 4. The hardware structure of the counter module.

an overview of the application’s behavior without the need of annotating the code or focusing the monitor on specific areas of interest.

This hardware monitor has been simulated using a software simulator [6]–[14]. Several experiments were performed using various benchmarks, mostly from the Stanford parallel applications for shared memory (SPLASH-II) suite [15], in order to find the right tradeoff between counting resources and hardware complexity. These experiments have shown that the optimal number of counters that keep the necessary hardware resources at a workable level while providing enough monitoring details is between 16 and 32.

A. The Static Part of the FRL

The main purpose of the static part [12] within FRL is to register and count well-defined events. These events either are *global events* (transactions within the SCI network) or *timer events*. The global events are driven by the monitoring software or a clock timer and they can be used to implement some kind of interaction between the user and the monitoring hardware. The static part uses two tables to define memory locations within the global memory: *the page table* to specify the pages and the *static-event table* to define the SCI events to be monitored.

B. The Dynamic Part of the FRL

The data collected within the dynamic mode [13] of the FRL allow the creation of memory access histograms. The user is able to use the gathered information to gain an overview of the application run-time behavior without any specific knowledge about the application structure. This allows for easy detection of bottlenecks and communication hot spots and it is a very useful first step in the fine-tuning process. It can be followed by a more precise evaluation of problems using the static mode described above. Therefore, the modes complement each other and provide the user with a comprehensive tool set for application tuning.

The SCI transactions will be referred to as events in the following description of the dynamic part. An event consists of a 5-bit source identifier (*sourceID*), a 5-bit destination identifier (*destID*), a 26-bit SCI-address, and a transaction type. In a first

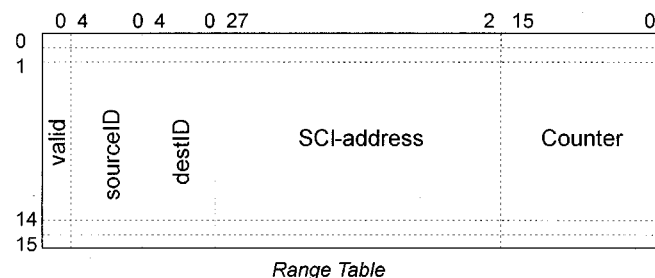


Fig. 5. Range table structure.

step, all incoming events are filtered based on their transaction types.

The core of the dynamic part is the *range table structure*, shown in Fig. 5. The fields *sourceID*, *destID* and *SCI-address* hold the corresponding data of an event, *valid* indicates that a range table entry is filled with a valid event attribute and the remaining 16 bit are used as the actual *counter* to keep track of the frequency of similar events. During the utilization of the monitor in this dynamic mode, every occurring relevant event is saved in this range table. If an event appears for the first time, a free-range table entry is filled with the event attributes, the *valid* bit of the entry is set and the counter is initialized with “1.” If a similar event is observed later, the corresponding counter is incremented. The fields *sourceID* and *destID* of two similar events have to be equal.

The dynamic part uses only 16 entries to keep the necessary hardware resources at an acceptable level. This resource limitation is compensated with a simplified less recently used (LRU) algorithm, which is used to swap range table entries to a buffer in the RAM of the host machine.

V. IMPLEMENTATION OF THE HARDWARE MONITOR

The PCI-target interface connects the SMiLE hardware monitor with the input/output (I/O) bus of the PC and is used to both configure the monitor from the host side and transfer the acquired data back to the host.

In this section, the PCI-target-interface will be described together with the evaluation and prototyping process that was undertaken to implement it.

A. PCI-Target Interface

The PCI-target interface is the connection between a host, which is used to run the monitor tool set, and the FRL. It is used to set user parameters that influence the monitoring process as described in the previous section, as well as to give the host software access to the acquired data in the counter registers. It includes the components *Swap_out*, *Write/Read*, *Pointer administration*, and further components for the initialization and control of the static and dynamic FRL.

The administration takes place at the *PCI_target* interface level within the component *Swap_out*. It is also responsible for the management of the counter value pointing into the host ring buffer for both static and dynamic FRL. It can be activated by the following two different events:

- 1) if the user demands via the PCI interface: all 16 counter values for static or dynamic events and their identifier are stored in the ring buffer;
- 2) in the case of a counter overflow of the static or dynamic FRL: the counter value or the index (counter number) together with the overflow identifier are stored in the ring buffer.

If an overflow of the ring buffer is detected during any of the above operations, the PCI-target interface raises an interrupt and, thereby, it triggers the complete save of all data by the host system software.

For the administration of the write/read operation three pointers are used: a read and a write pointer, indicating the current state of the ring buffer and an auxiliary pointer, which always contains the storage address increased by one of the last written memory location. When this auxiliary pointer reaches the value of the reading pointer, the ring buffer is full and the host is notified as described above.

In addition, the following parameters are necessary for the initialization and control of the monitor and can also be accessed by the host software through the PCI-target interface: the own *node identifier* to distinguish between inbound and outbound packets, the *maxim range area* to determine the granularity of segments to be monitored by the dynamic FRL, and the *transaction type* (read, write, lock, etc.) filtered within the dynamic FRL.

B. Hardware Monitor Prototype

The evaluation of the PCI-target interface logic and the implementation of a hardware monitor prototype have been done in a Hardware Object Technology™ System (HOT II) board, Virtual Corporation [16], which is a PCI based reconfigurable computing system containing both hardware and software components. One of the main features of this system is that the hardware components can be reconfigured dynamically by an executable program (so-called run time reconfiguration).

The evaluation setup used for the SMiLE hardware monitor can be seen in Fig. 6. The hardware is attached to a PCI card and a mezzanine daughter board.

The FPGA (XC4062XLA) contains the PCI core for the PCI-bus, the HOT II interface for a comfortable use of the hardware resources and the PCI-target interface required to initialize and control the FRL. The daughter board holds the B-Link interface and either the static FRL or the dynamic FRL logic.

In the case on the prototype of the hardware monitor, this can be used dynamically to load the dynamic or the static part of the FRL, as never are both required at the same time. This saves space on the prototype and reduces the complexity of the verification process.

VI. VISUALIZATION TOOL

As the low-level address information offered by the SMiLE hardware monitor is not understandable for users, a visualization tool is essential for the animation of the memory behavior of programs. The visualized data presentation in the form of tables, curves, and graphs enable a programmer to understand the programs more effectively.

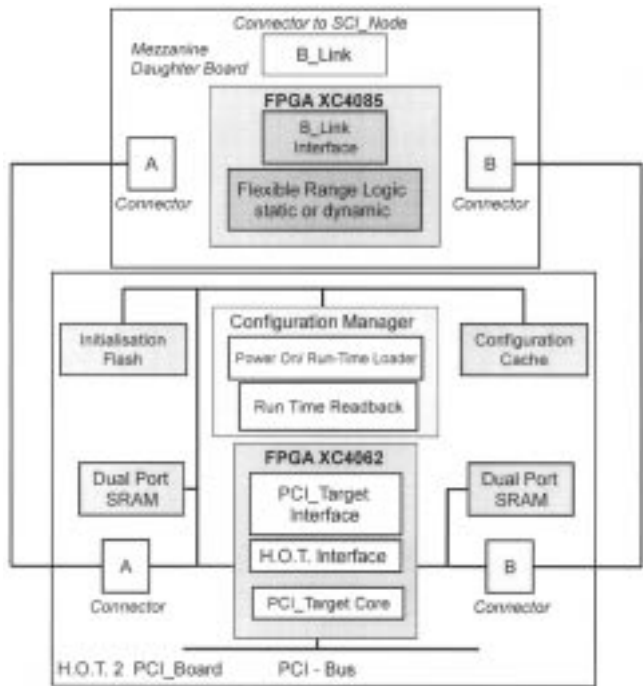


Fig. 6. Hardware monitor implemented at the HOT II Board.

The visualization tool has been implemented using Java in combination with the swing library to ensure maximum portability for the visualization front-end across platforms and it takes full advantage of the advanced graphic capabilities of swing, which reduces the implementation efforts. Access to raw data on the memory behavior of monitored programs is offered by the application programming interface functions (C-API) [17]. The visualization tool and the C-API functions are connected through the Java native interface (JNI). Corresponding C-API functions are called when the visualizer shows the various aspects of the memory behavior of monitored applications.

The current version of the visualization tool provides a number of displays of the monitored data. It shows the run-time data transfer among processors, the statistical results and the final static information about different pages, memory areas, as well as given data structures and arrays. Figs. 7 and 8 illustrate the results obtained with the visualization tool of a sample code, the RADIX program from SPLASH2-Benchmark suite [15], with a working set size of 256 KB.

The “run-time show” window (Fig. 8), reflects the run-time data transfer. The four components represent four memories located on the different nodes and the arrows between them represent the flow of data among nodes. Rectangles between components represent calculators, which record the number of data transfers between two nodes. Another window shown in Fig. 7, the “access histogram” window, exhibits the memory operations to all pages. The information offered by this window includes numbers of pages, numbers of nodes on which the pages are located, source nodes of accesses, and the numbers and ratios of accesses. The C-API functions *Monitor_get_packets* and *Monitor_histogram_info* provide the necessary data for these two display windows separately.

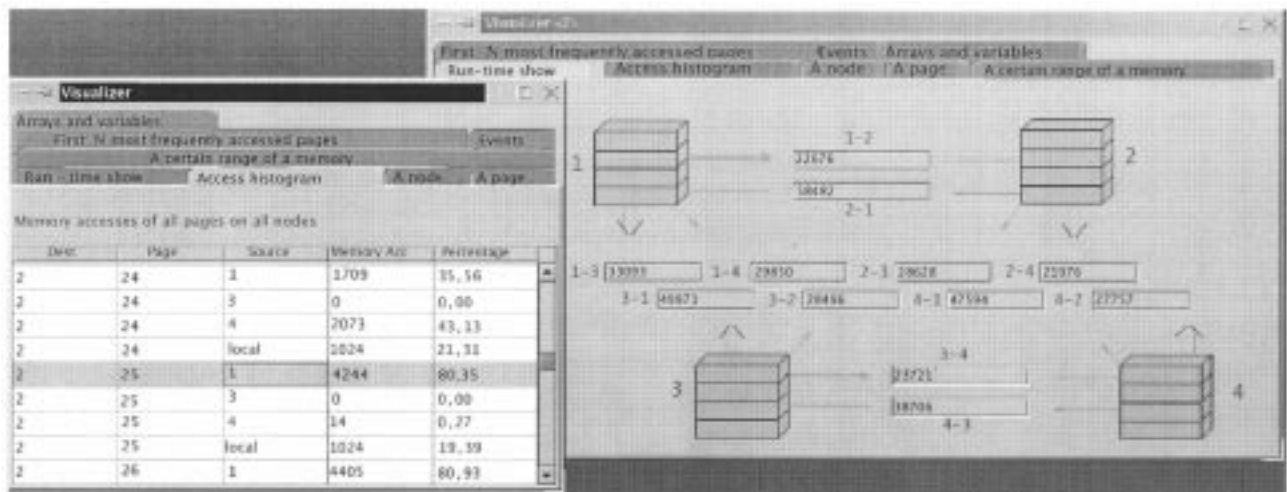


Fig. 7. Run-time show and access histogram windows of the visualization tool.

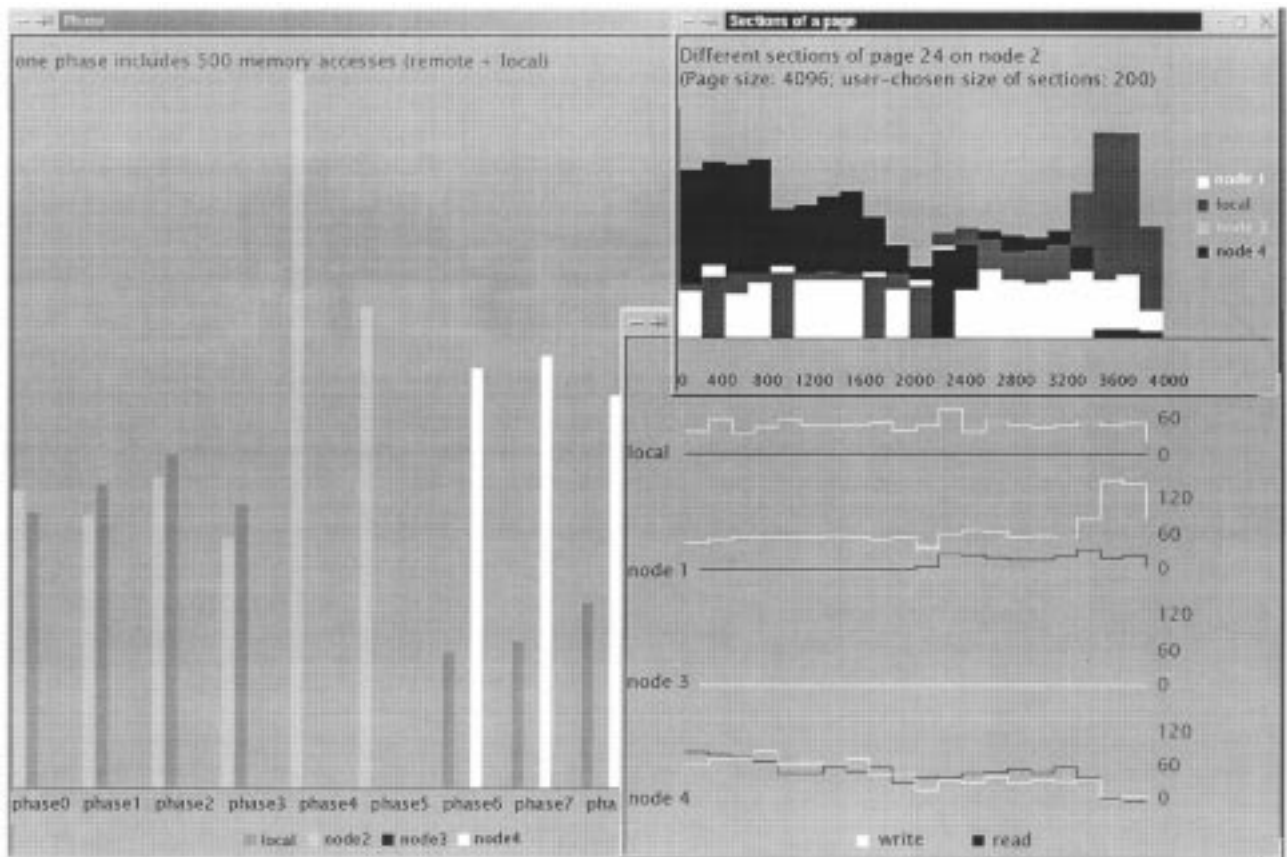


Fig. 8. Memory access cases to a single page.

Fig. 8 illustrates the memory access to a single page. It contains three subwindows. The “section” subwindow exhibits the behavior of different sections of a page and the “read/write” curves offer the contrast between the numbers of read and write operations to different sections, while the “phase” diagram provides the statistics of memory accesses to a page. These subwindows offer detailed information on a page enabling a better understanding of a single page. The C-API functions

Monitor_sections_info, *Monitor_sections_read*, *Monitor_sections_write* and *Monitor_phase_info* are responsible for offering the original data.

The visualized information described above can be used to detect “access” hot spots and bottlenecks caused by inappropriate distributions to modify the memory layout of the application with the goal of minimizing remote memory accesses and to further optimize memory locality.

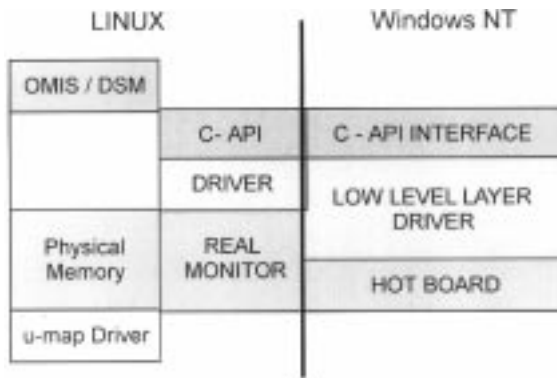


Fig. 9. Overview of the SMiLE hardware and software infrastructure.

VII. FUTURE WORK

Related to the software, the visualization tool is working with a simulator and it is ready to work in a real system when the interface tool will be debugged. The interface tool [17] has been defined as a commodity set of functions that the visualization tool needs in order to provide statistical information about the run time on each SCI node attached to one monitor. The driver tool [17] has been written and the main functions to control and manage the monitor, the read and write pointers and the ring buffer have been tested.

Related to the hardware, the dynamic mode of the hardware monitor prototype has been tested in a Windows NT (WNT) environment. This is due to the fact that the dynamic part of the FRL is implemented at the evaluation board (HOT II, described in Section V) and this system only allows developing designs under WNT. The static part of the FRL was not described in VHDL language. After the static part will be translated to VHDL, both the static and dynamic parts will be integrated into a single field programmable gate array (FPGA). The evaluation board used until now will not be able to allocate both designs due to the limited size, so the final prototype will be implemented in a bigger board.

The most convenient operating system of the SMiLE cluster is Linux, even though it has been working on WNT. The final step is then to translate the whole hardware monitor design to this environment. This means programming the new driver and interface tools and translating the hardware design into a more powerful device, the so-called *real monitor* (shown in Fig. 9).

VIII. RELATED WORK

Over the last years, basic monitoring support has become increasingly available on research as well as on commercial machines. For the cache coherent nonuniform memory access multiprocessor and flexible architecture for shared memory (CC-NUMA FLASH) multiprocessor systems, the hardware cache coherence mechanism is complemented by components for the monitoring of fine-grained performance data (number and duration of misses, invalidation etc.). Also some modern CPU chips incorporate hardware counters that collect information about data accesses, cache misses, etc. For some multiprocessor systems, this acquired data can be

exploited by performance analysis tools [18]. The information gained from those approaches, however, is not at that level of detail and granularity as with a direct monitoring of the NUMA interconnection fabric, as is possible with the SMiLE monitoring approach.

More closely related to the SMiLE monitoring system is the monitoring approach proposed by Martonisi *et al.* [19] for the scalable high-performance really inexpensive multiprocessor (SHRIMP) system. It is based on Myrinet [20] and it is implemented as an additional Myrinet Control Program run by the LANai processor [21]. However, as the traffic over Myrinet is generally based on larger packets and intended for message passing systems, also here, the granularity of information is less than in the SMiLE approach.

Another trace instrument for SCI-based systems has been designed at the Trinity College, Dublin, Ireland [22]. In contrast to the SMiLE monitor, it allows full tracing of the complete network traffic and stores the data in a large memory space on the monitoring hardware itself. While this increases the accuracy of the monitoring, it requires an offline evaluation due to the large amount of data acquired and it is, therefore, not suited to support adaptive run-time systems or on-line visualization tools.

IX. CONCLUSION

Precise and low-level performance monitoring is essential for the development of efficient parallel applications. This is especially true for distributed shared memory systems that rely on hardware support for communication in a hybrid fashion. In those systems, it is impossible to observe any communication in software without incurring a substantial overhead, as all communication is performed implicitly at in advance unknown locations. Only additional hardware support in the form of a hardware network monitor is capable of performing this task.

The design and the initial implementation work of such a monitor for the SMiLE system has been presented and discussed in detail in this work. Attached to an SCI adapter card through the B-Link, the main data path between the PCI bus and the SCI link for all common SCI cards, this monitor is capable of observing the complete traffic from and to a node. With its counter module or FRL, which is capable of both the static monitoring of given memory segments as well as the general observation of *a priori* unknown access hotspots, it is capable of providing the user with a complete assessment of the application's behavior and potential performance bottlenecks.

In order to reach an efficient implementation of this hardware design in the forms of FPGAs, several complexity reducing concepts have been applied. Most importantly, the counter registers, responsible for a great percentage of the total complexity have been limited to only a small width. To compensate for this, the monitor includes within its own PCI host interface several mechanisms that enable the swapping of full counters into main memory. This allows, in combination with an appropriate driver infrastructure on the main host node, to use transparently the monitor beyond the limited count width without increased hardware complexity or limited functionality.

This comprehensive hardware approach alone, however, is only the first, but necessary step toward efficient and precise

performance monitoring. In addition, an online monitoring software infrastructure is required to allow an evaluation, visualization, and dynamic application optimization based on the acquired data. Together with previous research [6]–[23] within SMiLE, this will result in a complete DSM programming tool set intended to help the programmer during the whole application development, debugging, and optimization process.

REFERENCES

- [1] W. Karl, M. Leberecht, and M. Schulz, "Supporting Shared Memory and Message Passing on Clusters of PC's With a SMiLE," . Berlin, Germany: Springer-Verlag, 1999, vol. 1602, Lecture Notes in Computer Science.
- [2] "Scalable coherent interface," IEEE SCI, IEEE Std. 1596, 1992.
- [3] "LinkController3™ Specification D666 LC3," Dolphin interconnect solutions AS, Oslo, Norway, 2000.
- [4] W. Karl and M. Schulz, "Hybrid-DSM: An efficient alternative to pure software DSM systems on NUMA architectures," in *Proc. 2nd Int. Workshop Software DSM Int. Conf. Supercomputing*, Santa Fe, NM, May 2000.
- [5] W. Karl, M. Schulz, and J. Trinitis, "Multilayer online monitoring for hybrid-DSM systems on top of PC clusters with a SMiLE," . Berlin, Germany: Springer-Verlag, 2000, vol. 1786, Lecture Notes in Computer Science.
- [6] W. Karl, M. Schulz, and J. Tao, "Using the SMiLE Monitoring Infrastructure to Detect and Lower the Inefficiency of Parallel Applications," . Berlin, Germany: Springer-Verlag, 2000, vol. 1823, Lecture Notes in Computer Science.
- [7] R. Hockauf, J. Jeitner, W. Karl, R. Lindhof, M. Schulz, V. González, E. Sanchis, and G. Torralba, "Design and implementation aspects for the SMiLE hardware monitor," in *Proc. SCI Eur. 2000 Conf. Stream of Euro-PAR 2000*, Munich, Germany, Aug. 2000, pp. 47–55.
- [8] "PCI9060 PCI master bus interface chip for adapters and embedded systems," PLX Technol., Mountain View, CA, 1995.
- [9] H. Hellwagner and A. Reinefeld, *SCI: Scalable Coherent Interface Architecture and Software for High-Performance Compute Clusters*. Berlin, Germany: Springer Verlag, 1999, pp. 89–101.
- [10] —, *A Backside Link (B-Link) for Scalable Coherent Interface (SCI) Nodes*. Oslo, Norway: Dolphin Interconnect Solutions AS, 1996.
- [11] —, *Link Controller LC-1 Specification*. Oslo, Norway: Dolphin Interconnect Solutions AS, 1995.
- [12] R. Hockauf, "A hardware monitor for the observation of communication events in a SCI based PC cluster, implemented in programmable logic," Tech. Univ. Munich, Germany, Dept. Comput. Sci., 1999.
- [13] J. Jeitner, "Dynamic run time analysis in a SCI based PC cluster," Tech. Univ. Munich, Germany, Dept. Comput. Sci., 2000.
- [14] W. Karl, M. Leberecht, and M. Schulz, "Optimizing data locality for SCI-based PC-clusters with the SMiLE monitoring approach," in *Proc. PACT'99*, Newport Beach, CA, Oct. 1999.
- [15] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *Proc. 22nd Int. Symp. Computer Architecture*, Santa Margherita Liguin, Italy, June 1995, pp. 24–36.
- [16] [Online]. Available: <http://www.vcc.com>
- [17] G. Torralba, "Connecting the SMiLE hardware monitor and the visualizer: Driver and interface tools," Dept. Electron. Eng., Univ. Valencia, Spain, 2001.
- [18] M. Zagha, B. Larson, S. Turner, and M. Itzkowitz, "Performance analysis using the MIPS R10000 performance counters," in *Proc. Supercomputing (SC)'96*, Pittsburgh, PA, 1996.
- [19] M. Martonosi, D. Ofelt, and M. Heinrich, "Integrating performance monitoring and communication in parallel computers," in *Proc. SIGMETRICS'96 ACM Sigmetrics Conf. Measurement and Modeling of Computers Systems*, Philadelphia, PA, 1996.
- [20] N. Boden, D. Cohen, R. Felderman, J. Seizovic, A. Kulawik, C. Seitz, and W.-K. Su, "Myrinet: A Gigabit-per-second local area network," *IEEE Micro*, vol. 15, pp. 29–36, Feb. 1995.
- [21] C. Liao, M. Martonosi, and D. W. Clark, "Performance monitoring in a Myrinet connected SHRIMP cluster," in *Proc. SIGMETRICS Symp. Parallel and Distributed Tools*, Madison, WI, 1998.
- [22] M. Manzke and B. Coghlan, "Non-intrusive deep tracing of SCI interconnect traffic," in *Proc. SCI Europe '99*, Toulouse, France, 1999.
- [23] J. Tao, W. Karl, and M. Schulz, "Memory access behavior analysis of NUMA-based shared memory programs," in *Proc. 2001 Int. Conf. Computational Science*, San Francisco, CA, May 2001.