# Exploiting Spatial and Temporal Locality of Accesses: A New Hardware-Based Monitoring Approach for DSM Systems

Robert Hockauf, Wolfgang Karl, Markus Leberecht, Michael Oberhuber, and Michael Wagner

Lehrstuhl fr Rechnertechnik und Rechnerorganisation (LRR-TUM)
Institut fr Informatik der Technischen Universitt Mnchen
Arcisstr. 21, D-80290 Mnchen, Germany
{karlw, leberech, oberhube}@informatik.tu-muenchen.de

**Abstract.** The performance of a parallel system with NUMA characteristics depends on the efficient use of local memory accesses. Programming and tool environments for such DSM systems should enable and exploit data locality.
In this paper we present an event-driven hybrid monitoring concept for the SMiLE SCI-based PC cluster. The central part of the hardware monitor consists of a content-addressable counter array managing a small working set of the most recently referenced memory regions. We show that this approach allows to provide detailed run-time information which can be exploited by performance evaluation and debugging tools.

## 1 Introduction

The SMiLE[1] project at LRR-TUM investigates high-performance cluster computing based on Scalable Coherent Interface SCI as interconnection technology [5]. Within this project, an SCI-based PC cluster with distributed shared memory (DSM) is being built. In order to be able to set up such a PC cluster we have developed a PCI-SCI adapter targeted to plug into the PCI bus of a PC. Pentium-II PCs equipped with these PCI-SCI adapters are connected to a cluster of computing nodes with NUMA characteristics (non-uniform memory access).

The SMiLE PC cluster as well as other SCI-based parallel systems available and accessable at LRR-TUM serve as platforms for the software developments within the SMiLE project, studying how to map parallel programming models efficiently onto the SCI hardware.

The performance of such a parallel system with DSM depends on the efficient use of local memory accesses through a parallel program. Although remote memory accesses via hardware-supported DSM deliver high communication performance, they are still an order of magnitude more expensive than local ones. Therefore, programming systems and tools for such parallel systems with NUMA

[1] SMiLE: Shared Memory in a LAN-like Environment

characteristics should enable and exploit data locality. However, these tools require detailed information about the dynamic behaviour of the running system.

Monitoring the dynamic behaviour of a compute cluster with hardware-supported DSM like the SMiLE PC cluster is very exacting because communication might occur implicitely on any read or write. This fact implies that monitoring must be very fine-grained too, making it almost impossible to avoid significiant probe overhead with software instrumentation.

In this paper we present a powerful and flexible event-driven hybrid monitoring system for the SMiLE SCI-based PC cluster with DSM. Our PCI-SCI adapter architecture allows the attachment of a hardware monitor which is able to deliver detailed information about the run-time and communication behaviour to tools for performance evaluation and debugging. In order to be able to record all interesting measurable entities with only limited hardware resources, the monitor exploits the spatial and temporal locality of data and instruction accesses in a similar way as cache memories in high-performance computer systems do.

Simulations of our monitoring approach helps us to make some initial design decisions. Additionally, we give an overview of the use of the hardware monitor in a hybrid performance evaluation system.

## 2 The SMiLE PC Cluster Architecture

The Scalable Coherent Interface SCI (IEEE Std 1596-1992) has been chosen as network fabric for the SMiLE PC cluster.

The SCI standard [6] specifies the hardware interconnect and protocols allowing to connect up to 64 K SCI nodes (processors, workstations, PCs, bus bridges, switches) in a high-speed network. A 64-Bit global address space across SCI nodes is defined as well as a set of read-, write-, and synchronization transactions enabling SCI-based systems to provide hardware-supported DSM with low-latency remote memory accesses. In addition to communication via DSM, SCI also facilitates fast message-passing. SCI nodes are interconnected via point-to-point links in ring-like arrangements or are attached to switches. The logical layer of the SCI specification defines packet-switched communication protocols. An SCI split transaction requires a request packet to be sent from one SCI node to another node with a response packet in reply to it. This enables every SCI node to overlap several transactions and allows for latencies of accesses to remote memory to be hidden.

The lack of commercially available SCI interface hardware for PCs during the initiation period of the SMiLE project led to the development of our custom PCI-SCI hardware. Its primary goal is to serve as the basis of the SMiLE cluster by bridging the PCs I/O bus with the SCI virtual bus.

Fig. 1 shows a high-level block diagram of the PCI-SCI adapter, which is described in detail in [1]. The PCI-SCI adapter is divided into three logical parts: the PCI unit, the Dual-Ported RAM (DPR), and the SCI unit.

The PCI unit interfaces to the PCI local bus. A 64 MByte address window on the PCI bus allows to intercept processor-memory transactions which are then translated into SCI transactions. For the interface to the PCI bus, the PCI9060 PCI bus master chip form PLX Technology is used. Also, packets arriving from the SCI network, have to be transformed into PCI operations.

The packets to be sent via SCI are buffered within the Dual-Ported RAM. It contains frames for outgoing packets allowing one outstanding read transaction, 16 outstanding write transactions, 16 outstanding messaging transactions using a special DMA engine for long data transfers between nodes, and one read-modify-write transaction which can be used for synchronization. Additionally, 64 incoming packets can be buffered.

The SCI unit interfaces to the SCI network and performs the SCI protocol processing for packets in both directions. This interface is implemented with the LinkController LC-1 from Dolphin Interconnect Solutions, which realizes the physical layer and part of the logical layer of the SCI specification. The SCI unit is connected to the DPR via the B-Link, the non-SCI link side of the LC-1. Control information is passed between the PCI unit and the SCI unit via a handshake bus. Two additional FPGAs, responsible for both the PCI-related and B-Link-related processing complete the design.

## 3 The SMiLE Hardware Monitor Extension Card

### 3.1 Overview

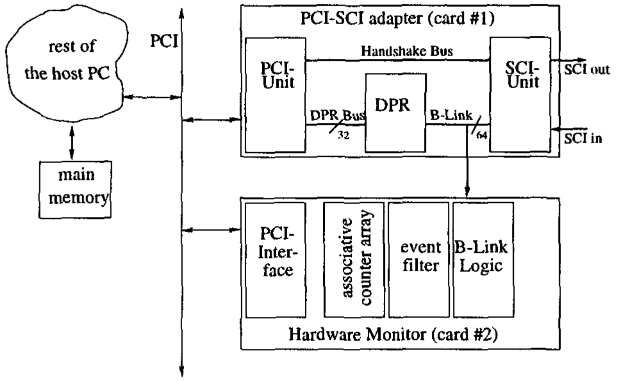

**Fig. 1.** The SMiLE PCI-SCI adapter and the monitor card installed in a PC

The fine-grain nature of SCI's remote memory transactions and their level of integration in the hardware makes it hard to trace and measure them in a manner necessary to do performance analysis and debugging.

As mentioned in section 2, the B-Link carries all outgoing and incoming packets to the physical SCI interface, Dolphin's LinkController LC-1. It is therefore a central sequentialization spot on which all remote memory traffic can be sensed. The SMiLE SCI hardware monitor thus is an additional PCI card attached to the original SCI adapter as shown in Fig. 1. Data that can be gathered from the B-Link includes the transaction command, the target and source node IDs, the node-internal address offset, and the packet data.

However, the design is naturally limited to remote memory traffic: a conventional plug-in PCI card does not enable us to monitor internal memory accesses. Nonetheless, remote memory accesses remain the most severe as mainly read accesses across the SCI network still account for more than one order of magnitude higher latencies than local accesses.

## 3.2 Working Principle

When memory access latency becomes an increasing problem during execution of machine instruction streams, and speeding up the whole main memory of a computer system appears not to be economically feasible, performance enhancements are usually achieved through the use of cache memories.

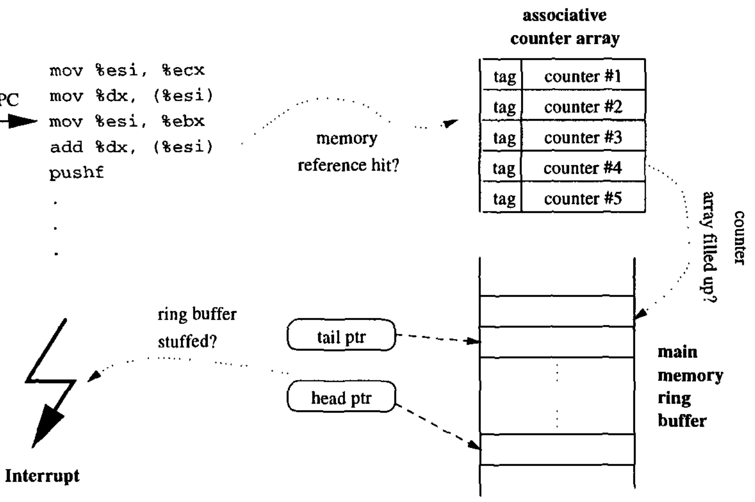

**Fig. 2.** The hardware monitor detects remote accesses. In order to allocate new counters, old counter value/tag pairs are flushed to the main memory ring buffer. Should it overflow, an interrupt is generated

As data accesses and instruction fetches in typical programs exhibit a great amount of temporal and spatial locality, cache memories only attempt to store a small *working set* of memory pages (cache lines). A prerequisite for this, however, is the use of a content-addressable memory as the covered address space usually dramatically exceeds the cache's capacity.

The same property holds for remote memory references in SCI-supported distributed shared memory execution of parallel programs: It is highly likely that a successive memory reference will access the same data item as its predecessor or some data close by. A hardware monitor the duty of which is to count memory accesses to specified memory regions can profit from this. The most recently addressed memory references and their associated counter can be stored in a register array with limited size. The detection of address proximity allows to combine counter values, as the accesses to neighboring data items may actually represent a single larger data object. E.g. it might be more interesting to count the accesses to an array rather than the references to its elements. Only recently unused memory references have to be flushed into external buffers, by this reducing the amount of data that has to be transported.

Drawn from these principles, the SMiLE hardware monitor uses the concept described in Fig. 2:

1. SCI remote memory accesses are sensed via the B-Link.
2. If the memory reference matches a tag in a counter array, the associated counter is incremented. If no reference is found, a new counter-tag pair is allocated and initialized to 1.
3. If no more space is available in the counter array, a counter-tag pair is flushed to a larger ring buffer in main memory. This buffer is supposed to be emptied by system software in a more or less cyclic fashion.
4. If a buffer overflow occurs nevertheless, a signal is sent to the software process utilizing the monitor in order to force the retrieval of the ring buffer's data.

Apart from a partition of counters working according to this principle (termed *dynamic* counters), the monitor also contains a set of counters that are statically preprogrammable (thus *static* counters) and are not covered in this work.

This addition to the monitor design was chosen in order to allow the user to include pre-known memory areas that are supposed to be monitored anyway, much like in conventional histogram monitors [10].

## 3.3 The Dynamic Coverage LRU Method

As one wants to reduce the amount of flushing and re-allocating counter-tag pairs, it makes sense to integrate the strategy of counter coverage adaption into the cache replacement mechanism. Under the prerequisite that counter tags can name not only single addresses but also continuous areas of memory for which the counter is to represent the accesses, a simple least-recently-used replacement algorithm can be adapted to this special task. The maximum address range, however, has to be predefined by the user. The Dynamic Coverage LRU method is shown in Fig. 3.

## 3.4 The Hardware Design

The resource saving principle of the monitor will hopefully allow us to construct the additional plug-in adapter with a small number of high-density FPGAs. The

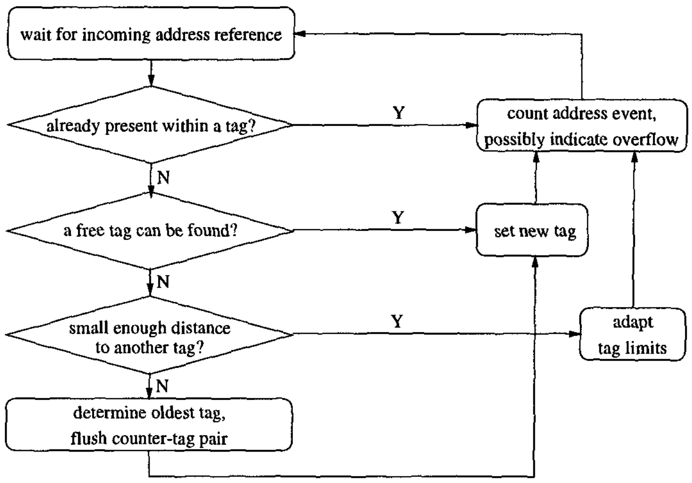
wait for incoming address reference
already present within a tag?
Y
N
count address event, possibly indicate overflow
a free tag can be found?
Y
N
set new tag
small enough distance to another tag?
Y
N
adapt tag limits
determine oldest tag, flush counter-tag pair

**Fig. 3.** The Dynamic Coverage LRU mechanism

same commercial PCI master circuit already used with the SCI bridge will serve again as its PCI interface.

The ring buffer will reside in the PCs main memory while its pointer registers will be located on the monitor hardware. Access to the monitor and its registers will be memory mapped in order to speed up accessing the device from user space. As SCI does not restrict the use of read and write transactions to main memory accesses, synchronization and configuration of a whole monitoring system can also be performed via SCI: remote monitor cards can be mapped into global shared memory in the same way that application memory is made accessible to other nodes.

## 4 Experimental Evaluation and Design Decisions

For initial performance assessments, a software simulator of the proposed hardware monitor served as the basis of a number of experiments. Traces were gathered by executing a number of the SPLASH-2 benchmark programs with the multiprocessor memory system simulator Limes [9]. For this, a shared memory system was assumed with the following properties:

**local read/write access latency** of 1 cycles, assuming they can be seen as normally cached on a usual system,

**remote write latency** of 20 cycles, corresponding to approx. 100 ns on a 200 MHz system (accounting for host bridge latencies when accessing the PCI bus),

**remote read latency** of 1000 cycles, corresponding to approx. 5 $\mu$s on a 200 MHz system [7], and

**remote lock latency** of 1000 cycles, as the processor gets stalled when waiting for the result of the remote atomic read-modify-write operation.

The global shared memory in the SPLASH-2 applications was distributed over the nodes in pages of 4 Kbytes in a round-robin fashion. Due to this the physical memory on a single node no longer represented a contiguous but rather interleaved area of the global shared memory. The simulated DSM system consisted of four nodes of x86 processors clocked at 200MHz.

Figures 4 and 5 represent the memory access histograms of the FFT and the radix sort programs in the SPLASH-2 suite generated by this configuration.

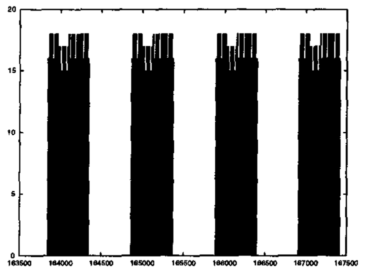

**Fig. 4.** Left: Memory access histogram for page 40 on node #0 of a 16K points FFT program in the SPLASH-2 suite run on a simulated 4-node SCI configuration

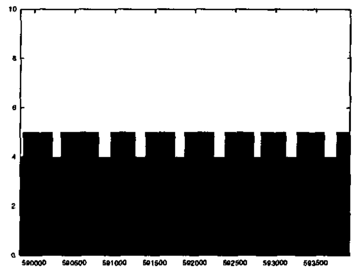

**Fig. 5.** Right: Memory access histogram for page 144 on node #0 of a 64K keys RADIX program in the SPLASH-2 suite run on a simulated 4-node SCI configuration

Both traces were run through the simulator while the number of dynamic counters was varied. This served to help us find the optimal hardware size of a physical implementation: constraints are an as lean as possible hardware realization which should still ensure low loads for buses and high-level tools.

Figures 6 and 7 display the results of these experiments.

The results follow intuition: the larger the number of counters is, the less flushes to the ring buffer occur. The same holds when the area that a counter can cover is increased: more and more adjacent acesses will be counted in a single register. Naturally, there is an optimum size for the maximum coverable range depending on the program under test: for the FFT increasing this maximal range to more than 16 bytes yields less counter range extensions. The reason for this lies in the FFT's mostly neighboring accesses to complex numbers, reflecting exactly a 16 byte granularity (two `double` values in C).

The RADIX results represent a different case: while for the overall work of the monitor the same holds as for the FFT (a larger number of counters and a bigger covered area account for less flushes), the number of counter range extensions only decreases for increased areas when a larger number of counters is working. This can, however, attributed to the linear access pattern in RADIX:

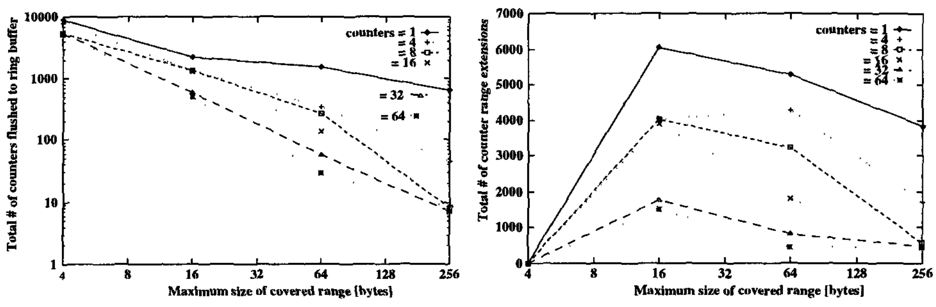

**Fig. 6.** Number of flushes to the ring buffer and counter coverage extensions for the FFT

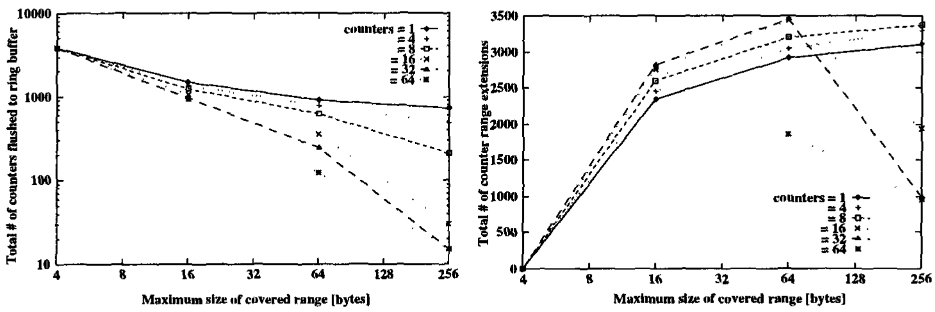

**Fig. 7.** Number of flushes to the ring buffer and counter coverage extensions for RADIX

successive reads and writes can always be put into the same counter until the maximum range has been reached. As soon as the counters are able to cover the whole 4K page (e. g. 256 byte range × 16 counters = 4096 bytes), the number of extensions decreases drastically.

A number of 16 or 32 counters appears to be a good compromise in a practical implementation of this monitor. Already in this size the number of accesses to main memory can be reduced significantly, even with relatively small ranges covered by the counters.

As the current monitor design always writes a packet of 16 bytes to the external ring buffer, this means that for 16 counters the amount of flushed data sums up to 84 KBytes for the FFT and 60 KBytes for RADIX. Given a simulated runtime for the FFT of 0.127 sec and 1.142 sec for RADIX, this yields a top average data rate of 661 KBytes/s. Roughly the same numbers hold for 32 counters. When the the Dynamic Coverage LRU is switched on with a maximum range of larger than 1 word, this rate instantly drops to 280 KBytes/s.

With typical PC I/O bandwidths of around 80 MB/s (PCI's nominal bandwidth of 133 MByte/s can only be reached with infinite bursts), monitoring represents a mere 0.8 % of the system's bus load and shouldn't influence program execution too much.

Switching off all monitor optimizations (range of 4 byte, only one dynamic counter) converts our hardware into a trace-writing monitor, creating trace data at a maximum rate of 1.06 MB/s.

An external ring buffer has to be sized according to these figures and the ability of a possible on-line monitoring software to make use of the sampled data. Derived from rules of thumb, ring buffers in the 1 MByte range should suffice to allow a reasonably relaxed access to flushed data without disturbing a running application.

It has to be noted that we are currently restricted to page-sized areas that can be covered by the dynamic counters of the monitor as PCI addresses are physical addresses. This means that consecutive virtual addresses may be located on different pages.

One solution to this problem would be to duplicate parts of the processor's MMU mechanisms tables on the monitoring hardware in order to be able to retranslate PCI into virtual addresses. The obvious issue of hardware complexity has so far excluded this possibility.

## 5 Related Work

The parallel programming in environments with distributed memory is supported by a number of tools so that there are strong efforts in a standardization of the monitoring interface [3].

Tool environments for DSM-oriented systems are less wide-spread. Mainly for software-based DSM systems performance debugging tools have been developed, which analyse traces for data locality [4] [2].

Over the last years, monitoring support has become increasingly available on research as well on commercial machines. For the CC-NUMA FLASH multiprocessor system the hardware-implemented cache coherence mechanism is complemented by components for the monitoring of fine-grained performance data (number and duration of misses, invalidations etc.) [11]. Modern CPU chips incorporate hardware counters which collect information about data accesses, cache misses, TLB misses etc. For some multiprocessor systems these information is exploited by performance analysis tools [12]

Martonosi et.al. propose a multi-dimensional histogram performance monitor for the SHRIMP multiprocessor [10].

## 6 Conclusion

In the paper we presented the rationale and the architecture of the distributed hardware monitor for the SMiLE DSM PC cluster. In order to be able to use the information provided by these hardware monitors a performance evaluation system needs a software infrastructure consisting of a local monitor library, a communication and managing layer providing a global monitor abstraction.

This items will be covered in an OMIS [8] compliant implementation of a prototypical DSM monitoring infrastructure.

## References

1. G. Acher, H. Hellwagner, W. Karl, and M. Leberecht. A PCI-SCI Bridge for Building a PC-Cluster with Distributed Shared Memory. In *Proceedings The Sixth International Workshop on SCI-based High-Performance Low-Cost Computing*, pages 1–8, Santa Clara, CA, Sept. 1996. SCIzzL.
2. D. Badouel, T. Priol, and L. Renambot. SVMview: A Performance Tuning Tool for DSM-Based Parallel Computers. In L. Boug, P. Fraigniaud, A. Mignotte, and Y. Robert, editors, *EuroPar'96 - Parallel Processing*, number 1123 in LNCS, pages 98–105, Lyon, France, Aug. 1996. Springer Verlag.
3. A. Bode. Run-Time Oriented Design Tools: A Contribution to the Standardization of the Development Environments for Parallel and Distributed Programs. In F. Hofeld, E. Maehle, and E. W. Mayr, editors, *Proceedings of the 4th Workshop PASA'96 Parallel Systems & Algorithms*, pages 1–12. World Scientific, 1997.
4. M. Gerndt. Performance Analysis Environment dor SVM-Fortran Programs. Technical Report IB-9417, Research Centre Jlich (KFA), Central Institute for Applied Mathematics, Jlich, Germany, 1994.
5. H. Hellwagner, W. Karl, and M. Leberecht. Enabling a PC Cluster for High-Performance Computing. *SPEEDUP Journal*, 11(1), June 1997.
6. IEEE Standard for the Scalable Coherent Interface (SCI). IEEE Std 1596-1992, 1993. IEEE 345 East 47th Street, New York, NY 10017-2394, USA.
7. M. Leberecht. A Concept for a Multithreaded Scheduling Environment. In F. Hofeld, E. Maehle, and E. W. Mayr, editors, *Proceedings of the 4th Workshop on PASA'96 Parallel Systems & Algorithms*, pages 161–175. World Scientific, 1996.
8. T. Ludwig, R. Wismüller, V. Sunderam, and A. Bode. OMIS — On-line Monitoring Interface Specification (Version 2.0). TUM-I9733, SFB-Bericht Nr. 342/22/97 A, Technische Universität München, Munich, Germany, July 1997.
9. D. Magdic. *Limes: An Execution-driven Multiprocessor Simulation Tool for the i486+-based PCs.* School of Electrical Engineering, Department of Computer Engineering, University of Belgrade, POB 816 11000 Belgrade, Serbia, Yugoslavia, 1997.
10. M. Martonosi, D. W. Clark, and M. Mesarina. The SHRIMP Performance Monitor: Design and Applications. In *Proceeding 1996 SIGMETRICS Symnposium on Parallel and Distributed Tools (SPDT'96)*, pages 61–69, Philadelphia, PA, USA, May 1996. ACM.
11. M. Martonosi, D. Ofelt, and M. Heinrich. Integrating Performance Monitoring and Communication in Parallel Computers. In *SIGMETRICS'96 1996 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, 1996.
12. M. Zagha, B. Larson, S. Turner, and M. Itzkowitz. Performance Analysis Using the MIPS R10000 Performance Counters. In *Supercomputing SC'96*, 1996.