VIA2SISCI – A new library that provides the VIA semantics for SCI connected clusters

Torsten Mehlan, Wolfgang Rehm

{tome,rehm}@cs.tu-chemnitz.de

Chemnitz University of Technology
Faculty of Computer Science
Straße der Nationen 62, 09111 Chemnitz, Germany [*]

Abstract:
Normally the SISCI interface provides a Distributed Shared Memory (DSM) [PTM97] abstraction using the Scalable Coherent Interface (SCI). This paper describes and discusses the design and the concepts behind a library called VIA2SISCI that we have developed. The library maps the semantics of the Virtual Interface Architecture (VIA) to SISCI–semantics and establishes a middleware between SISCI and high–level communication facilities. We focus on several important concepts of VIA that had to be mapped to the SISCI services. The presented concepts may be interesting and useful beyond the scope of this paper.

1 Introduction

High-speed networks were introduced to achieve effective communication within cluster systems. One of them is the Scalable Coherent Interface (SCI) [Dg00]. SCI was used a often within cluster computers to reduce communication overhead. As of today a single SCI link can transmit up to 320 MByte per second Also network latency is much lower than expected from conventional network links. The latency period for transmitting a 4 byte message is considered to be about 1.4 μs. Compared with conventional ethernet communication over TCP/IP these values are really impressive. The small delay of network transmissions is considered to be the highlight of SCI technology. Most other network technologies don't achieve better latency results. Even the upcoming InfiniBand technology has a latency larger than 4 μs Depending on the behaviour of the applications the use of SCI networks can lead to a significant speedup.

The SISCI programming interface serves as the native SCI interface. The specification of the SISCI programming interface [Gf99] was released in March 1999. This interface

[*]The research is part of the project 7455/1180 and funded within the framework for technology promotion by means of the European Fund for Regional Development (EFRE) 2000-2006 as well as by means of the SMWA Saxony ministry.

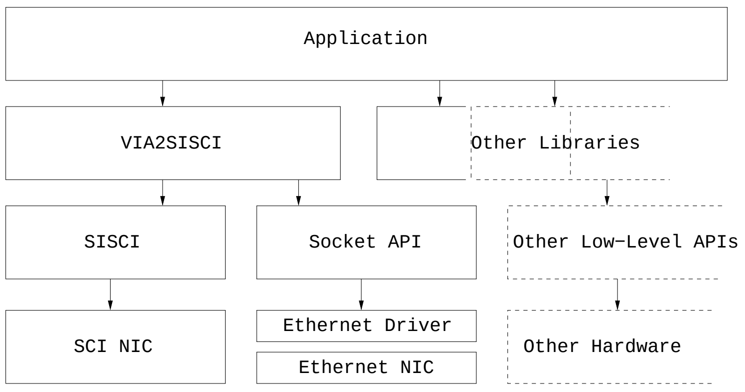

Figure 1: Arrangement of VIA2SISCI

provides access to the Distributed Shared Memory abstraction provided by the underlying SCI hardware. We developed a middleware that enables message passing on top of shared memory clusters running SCI. Other work was performed to enable MPI over SCI. For instance [WB99] and [Hl99] worked on this topic. The specification of the VI Provider Library was selected to serve as target interface. Therefore we selected the name VIA2SISCI to refer to our implementation. Figure 1 shows the arrangement of the VI Provider Library, SISCI and VIA2SISCI.

In the following the organization of the paper is given: Section 2 gives an overview about SISCI and the VI Provider Library. In section 3 are shown all the concepts necessary to map the semantics of VIA to the SISCI services. The resulting design of the current VIA2SISCI version is explained in section 4. Section 5 presents several performance results. Finally section 6 contains a conclusion and an outlook on future developments.

2 Fundamentals

We start with an introduction of the properties and capabilities of VIA and SCI. The following discussion introduces basic concepts of both network technologies. Of course this paper doesn't include a complete reproduction of the specification of VIA or SISCI. We just give as much details as necessary to understand the challenge of creating VIA2SISCI. Please refer to [Gf99] and [IMC98] to find an extensive discussion of VIA and SISCI.

Even if the VI Provider Library and SISCI pursue different programming paradigms the capabilities of the underlying hardware are similar. Both network adapters are capable to directly access the memory areas of the virtual address space of the process. This becomes necessary since the actual data transfer takes place without any additional copy of the payload.

The following statements are related to the user interface of VIA and SISCI. For a complete specification refer to [Gf99], [Ds03] and [IMC98]. The user process has to request SCI enabled memory from the library. Once the memory is published in this way the user processes of other nodes are free to establish a connection. After this work is done the memory can be used as like as local memory would be used. The VI Provider Library follows a completely different paradigm than SISCI. The Virtual Interface (VI) serves as the most important resource type visible to the user. Once the VI is created it has to be connected to exactly one remote VI. Each data transfer takes place between two connected VIs. The actual data transfer is started by posting a descriptor. The sender submits a gather list within the descriptor. The receiver may provide a matching scatter list within a corresponding receive descriptor. Once the sender has posted the descriptor the network hardware starts the transfer behind normal process execution.

For test purposes we used the research cluster of our chair. The configuration of the machines is presented in the following:

- 2 x Intel Xeon CPUs running at 2.4 GHz core frequency, 4x100MHz FSB, 512K L2 Cache
- 1 x Super Micro P4DMS-6GM mainboard (Intel E7500 chipset)
- 4 x 512 MB CL2.5 Registered ECC PC2100 DDR SDRAM Module
- 1 x ATI Rage XL integrated PCI Grafic Controller, 8MB RAM
- 1 x Intel 82544GC Gigabit Ethernet on-board NIC
- 1 x Intel 82557 10/100M Fast Ethernet on-board NIC
- 1 x Dolphin's PSB66 SCI-Adapter D331

3 Mapping VIA to SISCI

In this section we discuss several possibilities how to express VIA semantics in terms of SISCI functions. The following functional groups are covered by this section: The memory management is discussed first. Next the way of processing data transfers is brought into focus. Finally the concept of event signaling is presented.

3.1 Memory Management

At first we address the memory management. Both communication facilities rely on special treating of communication memory. Any interference into the memory management of the kernel is slow. Both communication libraries are designed to avoid frequent invocation of the memory management functions. The challenge is to provide the semantics of memory registration as specified by VIA [IMC98] while preserving a good performance.

The problem appears after consideration of the semantics of VIA memory registration. The *user* has to allocate memory and passes the appropriate pointer to the VIA subsystem. Instead the SISCI subsystem is not as flexible as expected. Even if communication memory has to be prepared as well, the semantics are slightly different. Our SCI cards would require additional address translation capabilities to handle arbitrary memory locations. Thus the only possibility to receive communication memory requires the authorization of SISCI functions to select an appropriate address. The user is not allowed to *register* allocated memory. To sum it up the SISCI interface doesn't allow for memory registration in terms of VIA.

We found three solutions that are discussed now. The first one consists of the creation of a small kernel module. This module would be responsible for modifying the page table of the kernel. Thus the pages of the user allocated memory can be remapped in order to reside in the memory area used by the SCI hardware. This solution would be the only one offering full transparency to user applications without affecting performance. Regardless of this major advantage we decided to discard the solution for several reasons. The use of a kernel module wouldn't fit into the design of VIA2SISCI. Furthermore manipulating the memory management of the GNU/Linux kernel is considered to be error–prone.

The second solution provides VIA compliant behaviour without playing with the memory management. Unfortunately this comes for the price of a performance loss. Let us explain the way we achieve VIA compliant memory registration. The user allocated memory area is passed to the appropriate function. VIA2SISCI just saves the pointer and size information and creates a shadow segment. This shadow segment is a memory area requested from the SISCI subsystem. The memory information structure holds both the pointer to the original memory and the pointer to the SCI segment. Unfortunately we have to live with two additional copy operations. Other applications may take advantage from this solution. Some high–level libraries like to frequently register and deregister memory areas. This process may be improved by maintaining a buffer of valid SCI segments. This may avoid the kernel transition when memory have to be registered.

The good news is that the third solution eliminates the performance drawbacks. Unfortunately we loose the property of being compliant with the VIA specification [IMC98]. To get rid of the data copies we introduce two functions. These functions serve as alternative to the memory registration and deregistration of the VI Provider Library. Instead of taking a pointer to some memory the new registration function returns a pointer to the appropriate SCI segment. Unfortunately the user has to be aware of this deviation from the interface specification in order to benefit from the full SCI performance.

The VIA2SISCI distribution will provide both the VIA compliant solution as well as the non–compliant solution. This becomes possible since different function names are used for each solution. VIA2SISCI manages all memory areas in a suitable way. Depending on the memory properties the data will be transfered either with copies or without copies.

3.2 Initiating a Data Transfer

Within VIA each data transfer is specified by a descriptor. The four different descriptor types provide slightly different semantics of data transmission. There are send descriptors and receive descriptors implementing the pure message passing paradigm. Alternatively two types of RDMA descriptors allow for bypassing the pure message passing semantics. Data transfers are always associated with at least one descriptor.

As already mentioned the SISCI paradigm follows the Distributed Shared Memory approach. Even if the VIA framework defines RDMA operations there are significant differences between RDMA in terms of VIA and DSM in terms of SISCI. Once the connection is established data transfers start implicitly. Since the receiver does not participate actively in one–sided communication the receiving process doesn't take influence on the incoming data transfer.

In the following we present our solutions implementing descriptor processing. A descriptor may contain a gather list of data sources. Correspondingly the receiver may provide a scatter list. If present the scatter list has to provide sufficient space to store all data.

The SISCI interface doesn't know anything about descriptors. Thus the information of both send descriptor and receive descriptor has to be available at the sending node. But the receive descriptor is posted at the receiving node. To make the necessary information available at one node we have to move one descriptor. There are two different approaches to the problem. The first solution becomes possible since the VIA specification requires descriptors to reside at registered memory. We assume that registered memory always is located inside an SCI segment. Thus the descriptor is directly accessible from any remote node. Unfortunately the SCI read–operation is not effective. Furthermore the descriptor doesn't have to reside within an SCI segment. Section 3.1 explains why.

The second solution works different. The receiver has to write a copy of the receive descriptor into a dedicated buffer. This buffer space is local to the sender. This solution is more advantageous than the first one. The transmission of receive descriptors is done by SCI write operations. In addition several receive descriptors can be transmitted at once. Buffering of descriptors helps to reduce the average latency of data transmission.

The concept of early transmission of necessary buffer information may be useful to other applications. All kinds of rendezvous protocols perform the transmission of management data prior to payload data transmission. If a sufficient amount of management information becomes available it may be transfered immediately. This may lead to smaller latencies for payload data.

Now we know the data source and the data destination since both descriptors are present. What do we need further? Well, the address specification is somewhat different between VIA and SISCI. VIA requires that each contiguous data block is given by virtual address and length. Nevertheless SISCI uses a different semantic. The appropriate functions expect an SCI segment to be specified. The actual data location is given by an offset into the SCI segment and the length. Therefore the offset has to be calculated. The calculation of remote offsets requires some information about remote memory to be present at the sender. During SCI connection setup the appropriate information is stored at the local node. Af-

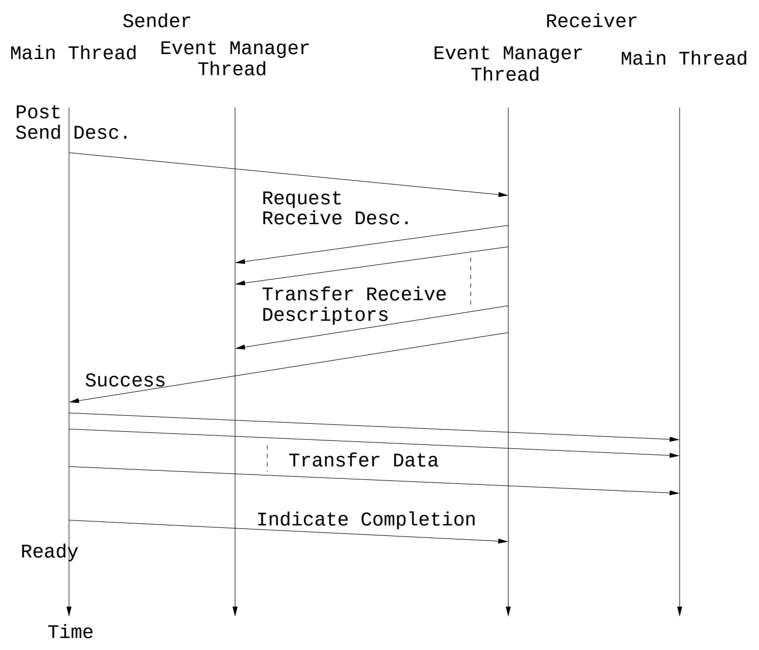

Figure 2: Time schedule of network transactions of a single data transfer

ter the connection setup the necessary address information will be available without any network transmissions.

3.3 Event Signaling

The last concept to discuss is the event management. This section describes the way how VIA2SISCI indicates events across nodes. The current event management is the result of a difficult decision. We had to weigh between issues of portability, performance and complexity.

The VI Provider Library contains a sophisticated event scheme. The data transmission goes behind the program execution per default. Thus the transmission completion occurs asynchronously. The management of asynchronous data transmissions requires the implementation of cross–node event signalling. Consider the case of a regular send–receive message. The receiver waits for the completion of the corresponding receive descriptor. The sender has to notify the receiver after the data transfer completes. Now we have to answer the question how to receive remote events at any other node and at any time.

The SISCI interface provides the Remote Interrupt mechanism to signal events across nodes. Remote Interrupts are not associated with any kind of remote memory access or data transmission. During the tests it became clear that the implementation of remote

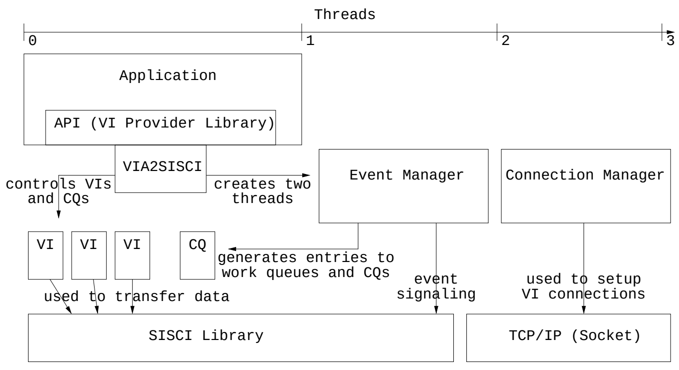

Figure 3: Overview of the design of VIA2SISCI

interrupts is not reliable. The SCI subsystem of the test bed did not offer the correct behaviour according to the specification.

We developed an alternative way to signal events like transfer completion. The event mechanism of VIA2SISCI founds on shared memory segments. During startup VIA2SISCI creates a thread that executes the event manager. This thread is responsible for catching and distributing events. To accomplish this task a small SCI segment is used. The event manager continuously polls the appropriate memory segment. A private area within the memory segment is assigned to each local VI. In case of an event the connected VI has to set a flag indicating the event type. The event manager will call the appropriate event handler as soon as the flag is detected. Unfortunately the event manager causes high CPU load. On the other hand fast event detection becomes possible.We expect that many users will insist on an interrupt based solution. This will be a subject to future work.

4 Design of VIA2SISCI

This section deals with the actual design of the VIA2SISCI middleware. The design decisions were driven by the considerations presented in section 3. We give an overview about the way how things actually work in the current VIA2SISCI version.

The most general approach to the VIA2SISCI design is given by figure 3. When the application initializes the library two threads are created. Among the main thread executing the user application we can find an event manager thread and a connection manager thread. The event manager catches events that are triggered by remote nodes. There are four different event types. To keep control all along the event management has to be executed by a thread. The connection manager supports connection setup and disconnection of VIs. Due to the weak performance demands of this task we are able to use regular TCP/IP connections. This becomes possible since connection setup is considered to be slow.

4.1 VI and Work Queues

Now we want to have a look at the way how the VI works. The VI serves as the access point to the most important services of the VIA framework. Basically the VI is the interface to connection management, data transfer and completion notifications. Thus each VI manages a private state. Once the VI is created it enters the idle state. State transitions are driven by the connection manager or due to errors. During the connection process the connection manager changes the state of the appropriate VI. Correspondingly a disconnection request will result in a state transition.

The actual data transfer takes place as soon as a descriptor enters the send queue of a connected VI. To post a descriptor to the send queue the application has to call `VipPostSend()` Behind the appropriate call the descriptor is enqueued and the processing starts. At first we have to check if a receive descriptor is needed. If this test succeeds we have to obtain the next receive descriptor of the remote VI. If there are still some receive descriptors this task will be skipped. Next the gather list of the descriptor is processed. Each contiguous piece of data is transfered using an SCI function. Which function is used depends on the settings of the VIA2SISCI configuration file.

The user has full control about the kind of SCI transfer. There are three options. The SISCI interface provides memory copy, block transfer and DMA transfer. After all data is transmitted we have to care about some administrative tasks. The new status of the receive descriptor has to be transfered to the remote node. Finally an event is sent to the remote node. Now the control can be returned to the caller.

Processing the descriptor happens synchronously. In contrast to the VI specification the data transfer is not performed in parallel with the program execution. Since normal code doesn't make assumptions about timing the discrepancy shouldn't be that significant.

4.2 Address Calculation

Source and destination of data transfers is specified in different ways in VIA and SISCI. The application has to provide a virtual address and a length within VIA descriptors. The Distributed Shared Memory paradigm of SISCI behaves different. The SCI hardware is not aware of the page table. Therefore SISCI is unable to handle virtual addresses of processes running remotely. Instead the user has to specify SCI segments and offset values.

The VIA2SISCI distribution has to manage the distinct address representations. At first we have to care about the SCI handles. We need appropriate handles to refer to SCI segments. Locally the memory manager maintains a hash table that holds information about local SCI memory. The handles of local memory and the start addresses are available from this table. Remote SCI memory segments have to be connected before they can be used. A buffer holds all handles of remote memory segments and address information. Once the handles and start addresses of SCI segments are available we have to calculate the byte offset. Normally this is done by subtracting the original address from the start address of the segment. Actually this task becomes complicated since we support two different memory

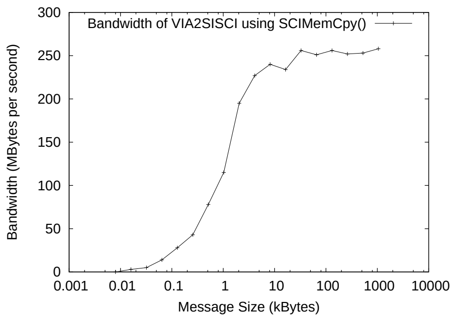

Figure 4: Bandwidth using SCI memory copy

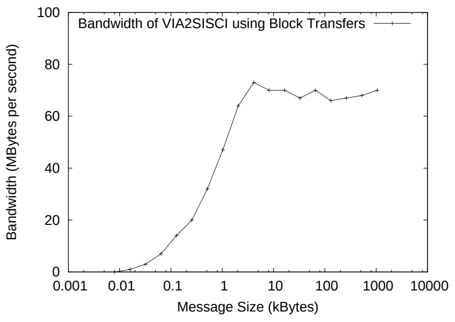

Figure 5: Bandwidth using block transfers

management schemes. Nevertheless the data derived from this calculation is suitable to start an SCI transfer.

5 Performance Results

This section covers the results of the performance measurements applied to VIA2SISCI. We wrote a small test application to measure the round trip time of different message types. The application fits into a client–server scheme. Server and client start to exchange messages to determine the round trip time. The layout of the hardware of our test bed was already introduced by section 2.

Figure 4 shows the bandwidth of a VIA2SISCI transmission while using the SCI memory copy function. The significant delay of the first transmission is caused by management tasks of VIA2SISCI. Consider the necessity of the transmission of receive descriptors to the sender. This takes place when the application posts the first descriptor. The benchmark posts several descriptors at one time. Thus our optimization becomes possible.

An alternative way of data transmission is provided by SISCI block transfers. The user can force VIA2SISCI to use the SISCI block transfer instead of memory copy. Figure 5 contains the corresponding results. Obviously the performance of block transfers keeps behind the performance of memory copy.

Figure 6 presents the performance while using DMA transfers. The VIA2SISCI configuration file allows for specifying DMA transfer to be used in each case. The resulting performance values don't show any unusual effects. For large messages the DMA transfer achieves the same performance like memory copy.

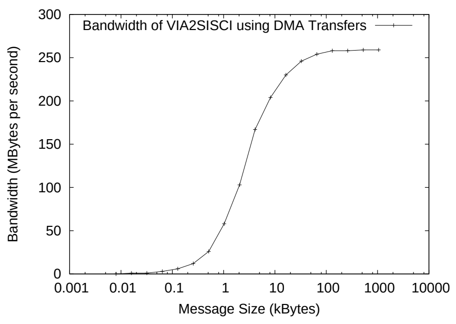

Figure 6: Bandwidth of VIA2SISCI using DMA transfers

6 Conclusion and Outlook

This paper dealt with two different programming paradigms that are popular in parallel computing. The shared memory paradigm was represented by the SISCI interface. In contrast the VI Provider Library was designed with respect to the message passing paradigm. Starting from both interfaces we discussed several alternatives to implement the VI Provider Library in terms of SISCI.

We are glad to present a running version of the VIA2SISCI middleware. Despite all problems we achieved an implementation of VIA2SISCI running with a common subset of SISCI functions.

References

[Dg00] David B. Gustavson, SCI Overview
http://www.scizzl.com/Perspectives.html, 2000

[PTM97] Jelica Protic, Milo Tomaevic, Veljko Milutinovic,
Distributed Shared Memory Concepts and Systems, ISBN 0-8186-7737-6, August 1997

[IMC98] Intel, Microsoft, Compaq, VI Architecture Specification,
http://www.viarch.org, 1998

[WB99] Joachim Worringen, Thomas Bemmerl, MPICH for SCI-connected Clusters,
Proc. SCI Europe '99, Toulouse, France, September 1999

[Hl99] L.P. Huse, K. Omang, H. Bugge, H. Ry, A.T. Haugsdal, E. Rustad,
ScaMPI - Design and Implementation, Springer-Verlag Lecture Notes in computer science 11734, 1999

[Gf99] F. Giacomini, T. Amundsen, A. Bogaerts, R. Hauser, B. D. Johnsen, R. Nordstrøm, Low–level SCI software functional specification,
http://www.dolphinics.com/pdf/documentation/SISCI_API-2_1_1.pdf, March 1999

[Ds03] Dolphin Interconnect Solutions, Source Code of the SISCI Developers Kit,
http://www.dolphinics.com/support/os/source.html, 2003