Using Arbitrary Memory Regions for SCI Communication

Friedrich Seifert*, Joachim Worringen†, Wolfgang Rehm*

Abstract—**The gap between the memory bandwidth and the bandwidth of modern high–performance interconnects is getting smaller. This increases the relative performance penalty for each copy operation on a data buffer to be transferred between two user processes on different nodes of a cluster. The optimal solution is direct data transfer between the source and destination buffers, without any in–between copy operations, so–called zero–copy transfer. Support for this zero–copy transfer mode is inherent in the design of the SCI interconnect as its communication is based on transparent remote read and write accesses. However, with the available SCI driver software, the user needs to allocate a designated SCI shared memory segment which then can be exported to or which represents remote memory imported from other nodes. In this paper, we present a solution which allows to use arbitrary regions of a process' address space for SCI communication by mapping it into the global SCI address space. Such a technique allows advanced and efficient usage models of SCI like true zero–copying in communication libraries.**

Keywords—**SCI, interconnect, Linux, memory–locking, zero–copy**

I. Introduction

Current high–speed interconnects (e.g. SCI [1], Myrinet [2], cLAN [3], Servernet [4]) for the PCI I/O bus of commodity clusters deliver more than 2 GBit/s bandwidth on the link–level. However, this does not mean that this bandwidth is generally available for message exchange between user–level processes running on distinct nodes in such a cluster. The reason for this is that for different, often compatibility–motivated reasons, communication protocols are used which perform additional copy operations on the data to be transmitted, reducing the effective bandwidth.

A. The Need for Thin Communication Protocols

Protocols which are optimized to avoid any temporary copies of data to exploit the link–level bandwidth to the highest degree possible are commonly called thin communication protocols. A variety of such protocols has been defined and implemented (e.g. U–Net [5], VIA [6]; SHRIMP [7], and VMMC [8], [9]) on different interconnects, making more or less use of the on–board hardware of the PCI–to–interconnect adapters. The SCI protocol inherently defines the thinnest communication protocol available, namely transparent access of the CPU to remote memory locations. Next to this CPU driven communication, current PCI–SCI adapters also feature a DMA engine to allow data transfer with minimized CPU load. However, this does not allow data transfers from and to any user–allocated memory area, but only from memory areas allocated by the SCI driver. Although it is defined by the SISCI API [10] to register user memory with the driver, this feature has not been supported so far. For this reason DMA transfers can only be realized by intermediate copies to driver allocated SCI segments (called *regular segments* in the following) which has been examined in [11]. Three copy operations are required there to transport data between two user–allocated segments on distinct nodes. In this paper we present a solution that allows to export and import arbitrary user memory under Linux making it available for DMA operations, thus, enabling zero–copy protocols in higher level communication libraries such as MPI. This has been achieved by extending both the IRM and SISCI drivers of the Dolphin PCI–SCI adapter. We also show the ideas of making user memory available for remote PIO access.

*Lehrstuhl für Rechnerarchitektur, TU Chemnitz Straße der Nationen 162, D-09107 Chemnitz, Germany e-mail: {friedrich.seifert,rehm}@informatik.tu-chemnitz.de, WWW: http://www.tu-chemnitz.de/informatik/RA .

†Lehrstuhl für Betriebssysteme, RWTH Aachen Kopernikusstr. 16, D-52056 Aachen, Germany. e-mail: contact@lfbs.rwth-aachen.de, WWW: http://www.lfbs.rwth-aachen.de .

B. Necessity of Memory Locking

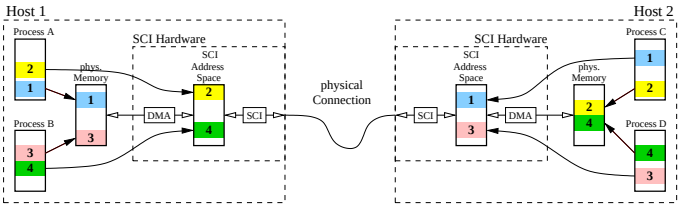

Fig. 1. Address mapping and data path for inter-node communication via the I/O bus

Inter–node communication in commodity clusters is performed via network adapters located in the I/O bus of the nodes (typically a PCI bus). Figure 1 (from [12], slightly modified) illustrates such a setup for two nodes including the data path between two user buffers for inter–node communication and the mapping of virtual to physical addresses for communicating processes.

The network adapter located in the I/O bus does not know anything about virtual addresses and thus needs to write to and read from physical addresses. This requires that the mapping to physical addresses for a user–specified range of virtual addresses (representing a send or receive buffer) does not change until the communication is complete. The pages must not be swapped out, which is achieved by locking the memory. This implies that for a system with no swapping or paging enabled, memory locking is not required.

C. Organization of the Paper

The next section presents an efficient mechanism for the Linux operating system to lock arbitrary ranges of virtual process memory. Section III explains how the kernel driver of the Dolphin PCI–SCI adapter has been extended to handle user–allocated memory. The resulting performance effects are described in section IV. Section V summarizes the achievements,

compares them with related work and concludes for work to be done in the future.

II. Memory Locking with Linux

Zero–copying means that the network hardware transfers data directly from or to user memory specified by virtual addresses. However, nearly all NICs, among them the PCI–SCI adapter, operate on physical instead of virtual addresses. This imposes two requirements:

- the virtual addresses must be translated into physical addresses and passed to the NIC
- the physical addresses must not change during the I/O operation, i.e. the virtual pages must be locked.

The standard way for locking memory on UNIX is the `mlock()` system call, but it has got some strong limitations. First, only super user processes are allowed to call it, and secondly, `mlock()` calls do not stack, that means one `munlock()` operation releases the given memory range even if has been locked several times. The permission problem can be solved inside the driver. The second limitation might be acceptable if only dedicated buffers are used that are locked during MPI startup and unlocked on MPI shutdown, for instance. However, since buffer addresses passed to MPI can not be predicted in any way, a network driver that supports zero–copy communication protocols must be able to lock and unlock any address range reliably at any time.

In order to solve the problem for Linux the so called Locked Memory Manager (LMM) has been developed at Chemnitz University of Technology in the course of a VIA implementation [13], [14]. The LMM is a separate kernel module providing a small interface to lock and unlock user space memory and to retrieve the physical addresses. It is based on a recent kernel mechanism called *kiobufs*. It was originally developed to enable raw disk I/O. A kiobuf describes the set of physical pages belonging to a part of a process' user address space, where all pages are locked. While it provides a safe way to get the physical addresses, its major limitation is that any physical page can be held at most one kiobuf at a time. Therefore an additional mechanism has been developed to allow for multiple lockings. We give only a brief explanation here. A detailed description can be found in [13], [14].

The LMM's central data structure is the Locked Memory Area (LMA). An LMA describes a continuous range of virtual addresses with the same lock count. There is one kiobuf per LMA, that describes the corresponding physical pages. LMAs are kept in per–process lists. The lock count specifies how many lock operations have been performed on that area. When a new area is to be locked, the LMM scans the current process' list of LMAs and changes them appropriately.

Figure 2 shows an example situation. Depending on the type of intersection of the new area and the existing LMAs, different actions have to be taken. These are: create a new LMA, increment the lock count of an existing LMA or split an LMA and increment the lock count of one part. To unlock an area the LMM steps through the LMA list and decrements the lock counts of the LMAs in question. Since LMAs with the same lock count are never melted together, an unlock operation will always span whole LMAs which eases the implementation of this operation.

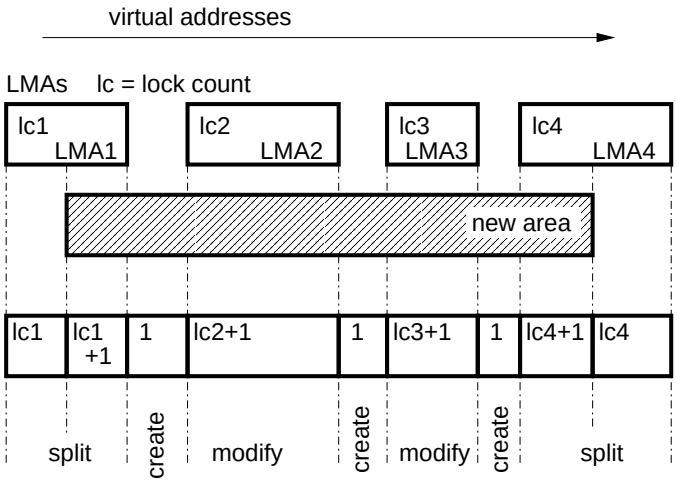

Fig. 2. Locking a new area

Once an area has been locked, a user of the LMM, which is another driver, can retrieve the physical addresses of all pages belonging to this area. Another LMM function returns the physical addresses grouped into consecutive blocks to allow for convenient building of scatter/gather lists.

III. SCI Segment Registration

If the SCI driver is told to establish a local SCI segment, it allocates the required memory area as one continuous chunk of physical memory which is locked to these addresses. Under Linux, this is done via the bigphysarea mechanism [15] or via a similar mechanism which was introduced for this purpose in the 2.4 release of the kernel. However, the latter method can fail in case of high memory fragmentation so that using bigphysarea is still the safest way.

A. User–allocated memory in SCI address space

Using user–allocated memory for SCI communication represents a new challenge since such memory does not fulfill the requirements of continuity and residency. While the second condition can be met afterwards by means of the locking mechanism introduced above, the non–continuity problem can not be removed. Hence, the SCI driver must be changed in order to be able to handle user memory. The major problem lies in the mappings between the different address spaces. Regular SCI segments span a continuous range of physical memory and are consequently continuous in the SCI space, too. Hence, only the start address and the length are necessary to locate a segment in those address spaces. Virtually continuous user memory is scattered across the physical memory due to the demand paging mechanism. One could assume that there are blocks of consecutive pages within a virtual area, but experiments have shown that user memory consists of nearly no physically continuous pages. More than 99 percent of the pages are non–continuous, even for memory areas as large as 4096 pages. The consequence is that optimizations for continuous blocks will not improve performance if not making it worse.

Further, the amount of information that has to be transferred during an import operation is proportional to the size of the seg-

ment as the importing node needs the addresses of all individual pages. Due to the non–continuous physical addresses DMA operations must be made page wise which will reduce the maximally achievable bandwidth as will be seen below.

The usage of user–allocated memory can be divided into three categories with ascending complexity:

1. local user segments as DMA source
2. remote user segments as DMA target
3. remote user segments for PIO access

In the following we will describe what these steps require and how it has been resp. can be achieved.

B. Local user segments as DMA source

Any DMA transfer with regular SCI segments involves two intermediate memory copies. First, the data must be copied from the user buffer to a local SCI segment on the sender side. Then a DMA transfer is initiated from the local segment to the remote segment. Since that remote segment is a local SCI segment on the receiver's side a final memory copy to the user buffer is needed. The first copy operation could be avoided if the DMA engine could fetch the data directly from the user buffer. Fortunately, the SISCI API knows about a special function for this purpose, although it had not been implemented in the SCI driver so far. It is possible to create so–called empty local segments, i.e. the driver does not allocate memory for it. After that, a user buffer can be registered with this segment by means of `SCIRegisterSegmentMemory()`.

For this first stage, empty segments must also be private, that means they cannot be exported to other nodes. The difference to regular local segments is that the physical addresses are non–continuous. Consequently, the physical addresses must be stored with the empty local segment. This is done during the `SCIPrepareSegment()` SISCI call, where the memory range to be registered is locked and the page addresses are retrieved by means of the LMM. Furthermore, the initiation of a DMA transfer (`SCIEnqueueDMATransfer()`) had to be changed. While a single DMA control block per transfer request is sufficient in conjunction with regular segments, user segments require a separate control block for each physical page. As figure 3 shows, the performance degradation caused by the scattered DMA transfer is less than 20 percent and thus quite acceptable. However, this is only true for properly aligned user buffers, i.e. aligned at 128 byte boundaries at least. For other alignments the performance drops to values as low as 134 MB/s. Originally we thought a solution to this problem would be to copy the first bytes to the next 128 byte boundary by PIO mode and then start the DMA transfer. For example assume, the local PCI address starts eight bytes beyond a page boundary, the remote SCI address is page–aligned. To get to an 128 byte aligned PCI address the first 120 bytes are transferred by PIO mode. The PCI address is well–aligned now, but the SCI address is not anymore. When we implemented this we realized that there is no difference in bandwidth whether PIO optimization is performed or not. This led us to the conclusion that the DMA engine of the PCI–SCI adapters used is sensitive to PCI address alignment as well as to SCI address alignment. Consequently, we removed the PIO optimization again, which reduced the time to connect to a remote segment slightly, as we will see in the next section.

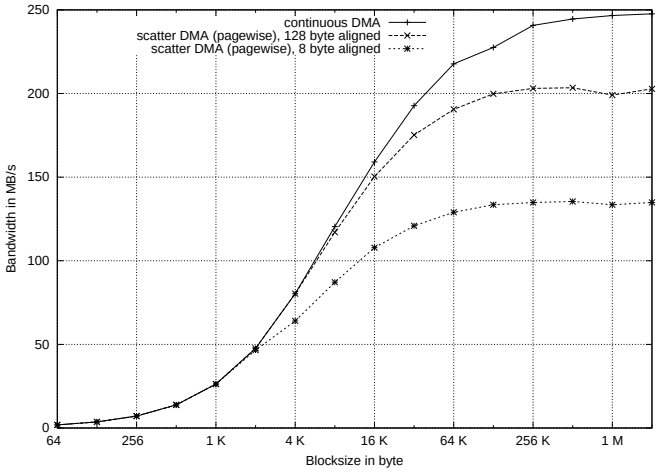

Fig. 3. Continuous DMA vs. scattered DMA (blocksize 4KB)

The application has to ensure that the DMA queue is large enough. Due to page alignment issues it should comprise $segmentsize/pagesize + 1$ entries.

C. Remote user segments as DMA target

The next step is to make user segments available to remote nodes so that they can be used as DMA target. This removes the need for the second memory copy on the receiver side. The only restriction is that such remote segments cannot be used for PIO mode. That is subject of section III-D.

The major challenge here consists in transferring the page addresses from the exporter to the importer. As opposed to regular, continuous SCI segments, which are identified by a single address, user segments are described by a list of addresses, the length of which depends on the segment size. We have realize this by using the IRM driver's internal mail box mechanism. The connection to a regular segment is established by the importer sending a connection request and the exporter responding by a single acknowledgement message (called `ACK` here) containing the segment's start address and its size. We have introduced two new types of acknowledges for user segments: `ACK1` (1st acknowledge) and `ACKI` (i-th acknowledge). In case of a user segment the response to a connect request consists of one `ACK1` and n `ACKI` messages, where $n \geq 0$. `ACK1` contains the segment size, the number of pages, the so–called *page offset* and up to four page addresses. Since user segments can start at any (minimally aligned) address, but exporting and importing is based on whole pages one must know where the segments actually starts within its first page. This is the meaning of page offset. For the same reason the number of pages is needed in addition to the segment size. As IRM mail box messages are rather short (64 byte in total) page addresses are stored in a compressed format. For a system page size of 4 KB only 20 are significant, for larger pages even less. Thus, we can put eight addresses into a 20 byte buffer (five double words). An `ACKI` message contains up to eight page addresses.

Upon reception of `ACK1` the importer can calculate the number of expected `ACKI` messages. When all messages have arrived and the page addresses have been copied to the data struc-

ture describing the remote segment, the connect operation is finished in the same way as for regular segments.

At this point the importer has got all information needed to make DMA transfers into the remote segment. Not even ATT (address translation table) mappings are necessary since the remote SCI addresses of DMA control blocks are built solely from the physical addresses of the remote memory pages and the remote node ID.

Things were slightly different when we implemented the PIO optimization for DMA operations explained above. Then, we allocated and set up an ATT entry to point to the first page of the segment and mapped the corresponding PCI memory into kernel address space.

Combining the first two steps (III-B and III-C) it is possible to perform DMA transfers directly from and to user segments. Performance numbers are given in section IV.

Up to two DMA control blocks are necessary for a single page, since a new block must be started at every local and remote page boundary. Hence, the DMA queue must comprise $(segmentsize/pagesize + 1) * 2$ entries.

D. Remote user segments for PIO access

In order to access a remote segment by PIO mode, the ATT and the process page tables must be setup appropriately. The process page table translates virtual addresses into physical addresses belonging to the PCI–SCI adapter, which translates them into SCI addresses using the ATT.

The simplest way is to use one ATT entry per host page. However, this would limit the amount of total importable memory to 16 MB (assuming 4 KB pages) due to the ATT size of 4096 entries in the current PCI–SCI adapters. The only solution is to increase the SCI page size, i.e. the ATT granularity. This leads to the following, more complex strategy: The importer determines the SCI pages the physical pages of the remote segment reside within. Then it checks if these pages are already covered by an ATT entry. If not, new ones must be set up. When all physical pages are mapped by means of the ATT, their addresses as well as the corresponding ATT indexes are stored with the imported segment. Now the process page tables must be set up. Note, that due to the larger SCI pages more memory than explicitly exported is visible to the outside. However, safe operation is ensured by the CPU's memory management. The page table entries are generated from the ATT index, representing one SCI page–sized part of the PCI–SCI adapter's PCI memory window, and the offset of the physical page to be mapped within that SCI page. This requires a number of changes in the IRM driver's memory map function. Finally, the page offset is added to the virtual start address of the imported segment so that it really points to the first byte of the remote user buffer.

After that, remote user segments can be accessed both by PIO and by DMA just as regular, driver–allocated segments. Although we have a fairly clear understanding of what must be done to achieve this step, the implementation is subject of near future work.

IV. Performance Evaluation

All test described in this section have been conducted using Dolphin's PCI–64/66 PCI–SCI adapters (64 Bit/ 66 MHz) on SuperMicro 370DLE mainboards with ServerWorks ServerSetIII LE chip set, two Pentium III 800 MHz, FSB clock 133MHz and 512MB ECC-RAM 133MHz. Except the first one all test were run on uniprocessor kernels since we discovered lockups during DMA transfers with SMP kernel. This issue still waits for its solution, but we suspect an interrupt handler problem.

First of all we evaluated the achievable bandwidth for DMA from areas which do not consist of physically continuous memory. We did this by splitting a DMA transfer from a regular segment into pieces of 4 KB. The results were already shown in section III-B.

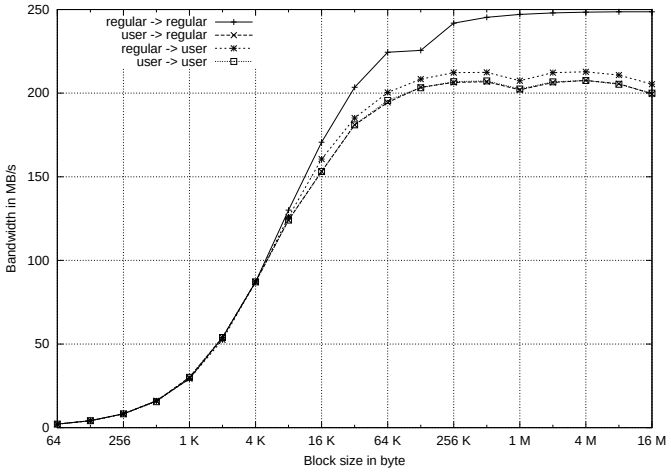

Fig. 4. DMA bandwidth for all user/regular segment combinations

Then we implemented the user segment registration and exporting/importing and measured DMA bandwidth between regular and user segments in all combinations. All segments were page–aligned. Figure 4 shows the results. The graph titles refer to the types of segments, "regular → regular" stands for a transfer from a regular local segment to a regular remote segment. The absolutely maximum bandwidth of 249 MB/s can only be achieved on regular segments since the DMA engine must be set up and started only once for the whole block. "regular → user" yields the highest bandwidth for user segments with 212 MB/s. Transfers from a user to a regular segment and between user segments achieve only 5 MB/s less. Hence, the DMA engine seems to operate slightly better on linearly ascending local PCI addresses. However, we have got no explanation for the small performance slumps at 1 and 16 MB yet. Note, the small advantage over the values from figure 3 is caused by the uniprocessor kernel used here.

We have made some more detailed analysis of transfers between user segments using several local and remote alignments. The results are shown in figure 5. For equal alignments on both sides the performance is the same as for page aligned segments shown in figure 4. The other graphs are entitled in the form "*local alignment → remote alignment*". If the alignments differ by half a page, the peak bandwidth is as low as 180 MB/s. The explanation is that a new DMA control block is needed whenever a new page with its specific address starts, either locally or remote. Hence, for 2 KB offsets the DMA transfer is scattered

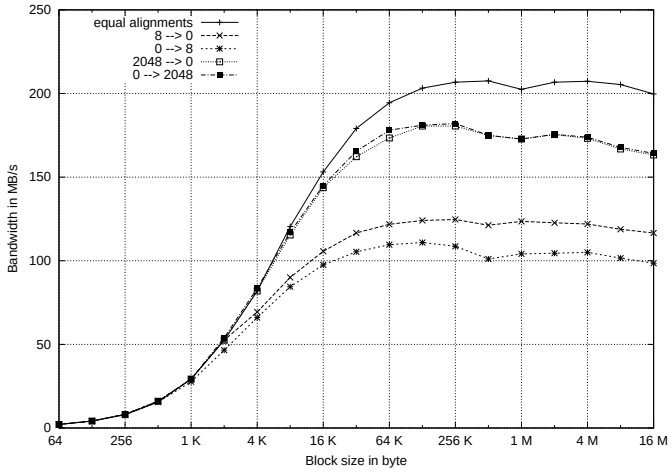

Fig. 5. DMA bandwidth different local and remote alignments

into chunks of 2 KB and the full DMA bandwidth cannot be exploited.

The situation is even worse for the minimal alignment difference of eight byte. The performance drops to 100 MB/s, which is less than half of the maximum. The reason is that the DMA transfer now consists of eight byte and 4088 byte chunks in turn, where the extremely short blocks obviously limit the bandwidth. What is interesting is the gap between "8 $\rightarrow$ 0" and "0 $\rightarrow$ 8". The number of DMA chunks and their sizes are equal in both cases. What differs is the alignment of these chunks. While the tiny blocks are page–aligned locally for "8 $\rightarrow$ 0" they always start at offset 4088 in the other case. One explanation could be that unaligned local PCI addresses additionally reduce the performance already limited by the small size. This means that any higher level software should try to use only well–aligned buffers. Since that is practically impossible for buffers passed by an application the PCI–SCI hardware should be optimized to perform well on arbitrary alignments.

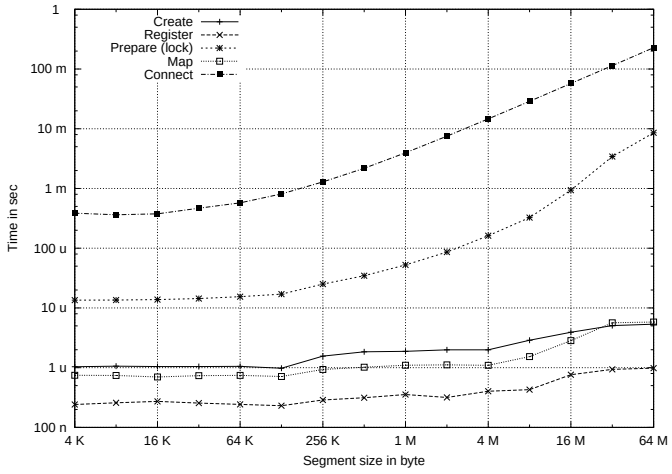

Fig. 6. Timings for SISCI calls to register a user-allocated segment

True zero copy protocols require some kind of on–the–fly registration of the communication buffers. Hence, the performance of these operations is also of high importance. We measured the time spent in the necessary SISCI calls `SCICreateSegment()`, `SCIRegisterSegmentMemory()`, `SCIPrepareSegment()` and `SCIMapSegment()`, which are needed to prepare the local buffer, and `SCIConnectSegment()`.

Figure 6 shows the results. The most expensive operation on the local segment is `SCIPrepareSegment()` since this is when the memory locking is performed. The total time to make a single page ready for DMA transfer is about $15.5\mu s$, while the throughput reaches its peak for 4 to 8 MB blocks at 25 GB/s.

Segment size	time in μs	# `ACKI`
4 KB	389	0
8 KB	389	0
16 KB	399	0
32 KB	484	1
64 KB	600	2
128 KB	813	4
256 KB	1280	8

TABLE I
CONNECT TIMES AND NUMBER OF `ACKI`S

The costs to establish a connection to a remote segment is one order of magnitude higher, starting from about 390 μs for up to 16 KB segments to 57 ms for 16 MB, see also table I. These high times are caused by the acknowledgement messages introduced in section III-C. As explained there, the `ACK1` message can carry up to four page addresses, i.e. a well–aligned 16 KB segment. That is why the connect times a nearly constant up to that size. Beyond that one or more `ACKI`s are sent. Linear regression yields a constant overhead of about 380 μs and about 110 μs for every additional `ACKI` message. The task of future work is to reduce the expense of `ACKI` messages. Currently a remote interrupt is triggered for each message delivered, which is then processed by the interrupt handler. A possible optimization could be to process several messages during one interrupt.

Finally, we measured the net bandwidth of a complete DMA transfer as seen from the initiator including all necessary preparation steps: local segment creation, registration of the user segment, preparation and mapping and connection establishment to the remote segment.

Figure 7 compares the bandwidths of pure DMA to transfers including only local preparation (designated "register+transfer") resp. local preparation and connection establishment ("register+connect+transfer"). While the local preparation adds a penalty of about 60 MB/s for medium size and 20 MB/s for large blocks, the connection setup makes the performance drop to 110 MB/s. The small message bandwidth is worse than FastEthernet. Although an optimization of the connect operation could improve the net performance, it will still stay far below the peak. Consequently, user–level data transfers — either by PIO or by DMA — are only sensible if expensive setup operations are not performed for every new transfer. This requires some kind of "caching" mechanism, that keeps registered and connected segments once they have been set up and does not release

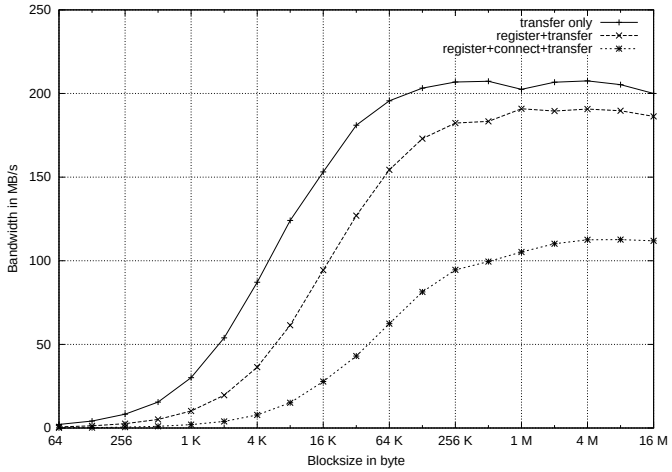

Fig. 7. Effective bandwidth for DMA from user-allocated segments

them except in case of resource shortage. Only the first transfer will show relatively bad performance, subsequent operations can fully exploit the zero–copy advantages.

V. Summary, Outlook and Related Work

We have presented an extension of the current SCI driver to support exporting and importing of arbitrary user memory in Linux which offers new and more effective applications of SCI based communication. The necessary memory locking is realized by the Locked Memory Manager. With this solution, user segments can be used as DMA source and target allowing for zero–copy protocols. The required scattered DMA transfer causes a performance penalty of about 20 percent on the maximum bandwidth on properly aligned buffers, but avoids two memory copy operations for transfers between two user–allocated segments on different nodes. Since the current hardware exposed severe performance degradation on unaligned buffers, higher software layers should try to use only buffers aligned to at least 128 byte. Besides, the hardware should be revised to perform better in such cases.

Utilization of this technique in communication libraries like MPI is required to evaluate the effective performance gain as seen by an application. Furthermore, PIO to remote user–allocated segments needs to be implemented as described in section III-D and the connect operation for user segments should be optimzed.

Another approach to utilize user memory for SCI communication was taken in [16] with VIA over SCI. However, that was implemented using PIO transfers instead of DMA. In [17] and [18] an advanced PCI–SCI bridge and system software design is introduced which allows much easier handling of user memory due to more sophisticated address translation mechanisms. The main difference to the Dolphin hardware consists in an additional address translation for inbound data, called upstream translation. This removes the need to transfer all page addresses to the importing node. Furthermore, the VIA support allows to start DMA operations directly from user level without a kernel call.

References

[1] IEEE: *The Scalable Coherent Interface (SCI), 1992. Standard 1596.*

[2] Nanette J. Boden, Danny Cohen, Robert E. Felderman, Alan E. Kulawik, Charles L. Seitz, Jakov N. Seizovic, Wen-King Su: *Myrinet: A Gigabit-per-Second Local Area Network.* IEEE Micro vol 15(1), pp. 29-36, February 1995.

[3] Emulex cLAN1000 High Performance Host Bus Adaptor.
URL: http://wwwip.emulex.com/ip/products/clan1000.html

[4] R. Horst and D. Garcia: *Servernet SAN I/O Architecture*, Hot-interconnects V, Stanford, 1997.

[5] Th. v. Eicken, A. Basu V. Buch, W. Vogels: *U-Net: A User-Level Network Interface for Parallel and Distributed Computing.* In Proc. 15th ACM Symposium on Operating System Principles. Copper Mountain, Colorado, December 3-6, 1995.

[6] Compaq, Intel and Microsoft Corporations: *The Virtual Interface Specification. Version 1.0.* Dec 16, 1997.

[7] Cezary Dubnicki, Angelos Bilas, Yuqun Chen, Stefanos N. Damianakis, Kai Li: *Shrimp Project Update: Myrinet Communication.* IEEE Micro vol. 28 (1), pp. 50-52, January/February 1998.

[8] Mattias A. Blumrich, Cezary Dubnicki, Edward W. Felten, Kai Li, Malena R. Mesarina: *Virtual-Memory-Mapped Network Interfaces.* IEEE Micro vol. 15 (1), pp. 21-25, February 1995.

[9] Cezary Dubnicki, Angelos Bilas, Kai Li, James Philbin: *Design and Implementation of Virtual Memory-Mapped Communication on Myrinet.* In Proceedings of the IEEE 11th International Parallel Processing Symposium. April 1997.

[10] *ESPRIT Project 23174 — Software Infrastructure for SCI (SISCI)*, 1998.

[11] Joachim Worringen: *SCI-MPICH - The Second Generation.* Proc. SCI Europe 2000 (conference stream of EuroPar 2000), pp. 10-20, Munich, Germany, August 2000.
URL: http://www.lfbs.rwth-aachen.de/users/joachim/publications

[12] Mario Trams, Wolfgang Rehm: *A new generic and reconfigurable PCI-SCI bridge.* In Proc. SCI Europe '99, helt in conjunction with EuroPar '99, pp. 3-11, Toulouse, France, September 1999
URL: http://www-user.tu-chemnitz.de/~mtr/publications.html

[13] Friedrich Seifert, Wolfgang Rehm: *Proposing a Mechanism for Reliably Locking VIA Communication Memory in Linux.* In Proc. 1st IEEE International Conference on Cluster Computing CLUSTER2000, Nov 28 - Dec 1, 2000, Chemnitz, Germany.
URL: http://www.tu-chemnitz.de/informatik/RA/papers/p97/papers.html

[14] Friedrich Seifert, Wolfgang Rehm: *Reliably Locking System V Shared Memory for User Level Communication in Linux.* To appear in Proc. IEEE International Conference on Cluster Computing CLUSTER2001, Oct. 8-11, 2001, Newport Beach, California, USA.
URL: http://www.tu-chemnitz.de/informatik/RA/papers/p97/papers.html.

[15] URL: http://www.uni-paderborn.de/cs/heiss/linux/bigphysarea.html

[16] Karim Ghouas, Knut Omang, Hakon Bugge: *VIA over SCI - Consequences of a Zero Copy Implementation, and Comparison with VIA over Myrinet.* In Proceedings of Communication Architectures for Clusters, CAC'01, San Fransisco, Apr 2001.

[17] M. Trams, W. Rehm, D. Balkanski, S. Simeonov: *Memory Management in a combined VIA/SCI hardware.* In Proceedings of PC-NOW 2000, Internalional Workshop on Personal Computer based Networks of Workstations, Cancun, Mexico, May 1-5 2000.

[18] Friedrich Seifert: *Design and Implementation of System Software for Transparent Mode Communication over SCI.* Study Thesis, Dept. of Computer Science, University of Technology Chemnitz, 1999.
URL: http://www.tu-chemnitz.de/ sfri/publications.html