Teilprojekt B6

2.1 Teilprojekt B6

Anwendungsoptimierte Kommunikation auf Clusterarchitekturen

2.1.1 Antragsteller

Prof. Dr. Ing. Wolfgang Rehm	Dr. Matthias Pester
27.09.1950	29.04.1954
Professur Rechnerarchitektur und Mikroprogrammierung	Professur Numerische Algebra
Fakultät für Informatik	Fakultät für Mathematik
Technische Universität Chemnitz	Technische Universität Chemnitz
09107 Chemnitz	09107 Chemnitz
Tel.: (0371) 531-1420	(0371) 531-2656
Fax: (0371) 531-1806	(0371) 531-2657
rehm@informatik.tu-chemnitz.de	pester@mathematik.tu-chemnitz.de

2.1.2 Projektbearbeiter

Dipl.-Inf.(FH) M.Meyer
Dipl.-Inf. S.Schindler
Dipl.-Inf.(FH) C.Dinkelmann
Dipl.-Inf. M.Trams
MSc. D.Balkanski
Dipl.-Inf. S.Seifert
Technische Universität Chemnitz, Fakultät für Informatik

2.2 Ausgangsfragestellung/Einleitung

Ziel des Teilprojektes war eine Optimierung der Kommunikation für SFB–relevante Applikationen innerhalb eines Clusters. Dabei sollte diese Problemstellung auf verschiedenen Ebenen bearbeitet werden — Von algorithmischer Sicht über die Kommunikationsbibliothek bis hin zur Kommunikationshardware. Entsprechend wurde Teilprojekt B6 in vier einzelne Teilaufgaben aufgeteilt:

A: Effiziente Kommunikationsalgorithmen in parallelen FEM–Systemen

B: MPI–Infrastrukturen zur effizienten Anpassung an heterogene Kommunikationssysteme

C: VIA–konformer PCI–SCI Adapter für optimiertes Message Passing

D: Entwicklung einer objektorientierten Schnittstelle zu MPI

In der Realisierung wurde der Schwerpunkt auf Teilaufgaben B: *MPI–Infrastrukturen zur effizienten Anpassung an heterogene Kommunikationssysteme* sowie C: *VIA–konformer PCI–SCI Adapter für optimiertes Message Passing* gelegt.

Einerseits hat sich gezeigt, dass diese beiden Themen die beiden bewilligten halben Personalstellen vollkommen in Anspruch nehmen und andererseits sind durch Weggang geplanter ergänzender Bearbeiter (aus der Grundausstattung) für die Teilthemen A: *Effiziente Kommunikationsalgorithmen in parallelen FEM–Systemen* und D: *Entwicklung einer objektorientierten Schnittstelle zu MPI* die Grundlagen für eine effiziente Bearbeitung dieser geschwächt worden.

Im folgenden detaillierten Bericht wird daher nur auf die Teilaufgaben B und C eingegangen.

2.3 Forschungsaufgaben/Methoden

2.3.1 Teilaufgabe B: MPI–Infrastrukturen zur effizienten Anpassung an heterogene Kommunikationssysteme

Es war absehbar, dass sich MPI [MPI95] in der letzten Zeit gegenüber PVM und anderen als führende Message–Passing Bibliothek durchgesetzen würde [GLS94]. Es waren sowohl freie (z.B. MPICH vom Argonne National Lab/ Mississippi State University [GLS96] oder LAM vom Ohio Supercomputer Center [LAM96]) als auch kommerzielle Implementierungen (z.B. ScaMPI [Scali] von der norwegischen Firma Scali speziell für SCI–Technologie) erhältlich.

Die bereits erwähnte MPICH–Implementierung ermöglicht auf Grund der internen Schnittstelle ADI–2 eine effiziente und schnelle Realisierung eines angepassten MPI–Systems. LAM bietet dagegen interessante Ansätze zu dynamischen Prozessen in MPI. Kommerzielle Angebote (z.B. Scali) zeichnen sich oft durch sehr hohe Leistungen, allerdings meist nur für ein Kommunikationsmedium, aus.

Als Weiterentwicklung des MPI–Standards stand zu Beginn des Antragszeitraumes der damals neue MPI–2 Standard [MPI97] zur Verfügung, welcher neben der bisherigen Message–Passing Funktionalität auch neue Elemente, wie z.B. eine dynamische Prozeßverwaltung oder Remote Memory Access, vorsah.

Am Lehrstuhl Rechnerarchitektur der TU Chemnitz wurde sich schon seit einigen Jahren vor der Antragstellung mit der Entwicklung von Kommunikationsbibliotheken auf der Basis des MPI–Standards beschäftigt. So wurde mit der Entwicklung eines Shared Memory Devices in [LAN96] erstmals eine effiziente Abbildung von messagebasierter Kommunikation der API–Schnittstelle auf ein physisch unterliegendes Shared–Memory–System erreicht.

Mit SCI–MPI [WGR97] wurden erstmals verschiedene Kommunikationsmedien innerhalb eines angepassten MPI–Systems kombiniert. Hier wurden auch erste Untersuchungen zu globalen Operationen innerhalb heterogener Systeme durchgeführt. Das dabei entwickelte System stellt allerdings nur eine Teilimplementierung des MPI–Standards dar. Die bei der Entwicklung gewonnen Erkenntnisse sollen im Antragszeitraum auf Grundlage der MPICH–Implementierung in ein neues, vollständiges MPI–System umgesetzt werden. Dazu wurden mit [BEY97], [RAD97] und [SCH97] Vorarbeiten im Bereich der Deviceentwicklung geleistet. In [SCH97] wurden erste Ansätze zur Realisierung einer Verknüpfung verschiedener Kommunikationsmedien mittels eines sogenannten Multidevices vorgestellt.

Ziel der Arbeiten im Teilthema B war die Entwicklung einer leistungsfähigen Kommunikationsbibliothek für offene, skalierbare, heterogene Systeme.

In einem solchen System wurde vor allem ein Mechanismus zur Verbindung verschiedener Kommunikationsmedien benötigt. Dieser sollte durch ein sog. Multidevice auf der Basis von MPICH realisiert werden, welches die interne ADI-2 Schnittstelle von MPICH benutzt. MPICH wurde gerade wegen dieser Schnittstelle als Grundlage zur Realisierung des Gesamtsystems ausgewählt. Sie ermöglicht durch ihren im Vergleich zum MPI-Standard minimierten Funktionsumfang eine schnelle Realisierung eines eigenen MPI-Systems. So muss allgemeine, hardwareunabhängige Funktionalität, wie z.B. virtuelle Topologien oder Prozessgruppen, nicht mehr selbst implementiert werden, sondern es kann hierbei auf eine vorgegebene Realisierung zurückgegriffen werden. MPICH ermöglicht außerdem eine Portabilität auf verschiedene Architekturen.

Zum konkreten Zugriff auf die Hardware benutzt das Multidevice seinerseits sog. Subdevices (z.B. Multi–Threaded oder SCI Device), welche die zugrundeliegende Hardware optimal nutzen können. Bei der Entwicklung des VIA/SCI–Subdevices erfolgt dabei eine enge Zusammenarbeit mit dem Teilthema C.

In der Umsetzung der Aufgabenstellung haben sich relativ schnell unvorhergesehene Probleme bzgl. der ADI-2 Schnittstelle ergeben. Zwar wurde zunächst ein Multidevice basierend auf dieser Schnittstelle erfolgreich entwickelt, jedoch traten mit neueren MPICH–Releases diverse Inkompatibilitäten an dieser Schnittstelle auf. Auch sind konzeptionelle Probleme bei MPICH aufgetreten, welche eine effiziente Realisierung eines VIA/SCI–Subdevices für MPICH verhinderte.

Aus diesen Gründen wurde beschlossen, mit der Konzeptionierung einer eigenen MPI–Bibliothek zu beginnen und diese umzusetzen. Ein Hauptkriterium stand bei der Entwicklung auch darin, in erster Linie SFB–relevante MPI–Funktionalität zu implementieren. So konnte in relativ kurzer Zeit ein, wenn auch nicht optimaler, Prototyp geschaffen werden, auf dem eine im Rahmen von Teilprojekt D2 entwickelte Applikation (MISTRAL — numerische Simulation von dispersen Mehrphasenströmungen) zum Laufen gebracht werden konnte.

Ein weiterer Schwerpunkt der MPI–Entwicklung lag in der Fortführung des Multidevice–Konzeptes. So wurde das Design der Kommunikationsbibliothek so gestaltet, dass ein Multidevice in diesem Sinne nicht mehr benötigt wird. Vielmehr wird eine solche Funktionalität schon vom Kern selbst realisiert.

2.3.2 Teilaufgabe C: VIA–konformer PCI–SCI Adapter für optimiertes Message Passing

Während die physisch mögliche Kommunikationgeschwindigkeit stetig anwuchs, hat sich dieses Wachstum nicht mehr in gleicher Weise auf die Applikationen ausgewirkt. Dies betraf insbesondere die Latenzzeiten. Am Beispiel des TCP/IP Protokollstacks wurde dies in [VIA1] besonders eindrucksvoll aufgezeigt. Während der Anteil der Latenzzeit, der durch die eigentliche physische Übertragung der Daten anfällt, auch schon bei relativ langsamen Technologien wie Fast Ethernet sehr klein geworden war, hat sich der Anteil des Protokollstacks nur unwesentlich verringert.

Die wichtigste Schlussfolgerung daraus war, dass ein noch schnelleres Medium damit für kurze Nachrichten keine signifikante Verbesserungen bringt.

Als Lösung dieses Problem haben sich mehrere Ansätze herauskristallisiert. Zum einen muss der Protokollstack verkleinert und besser an die Anforderungen der Applikation angepasst werden. Zum anderen muss versucht werden, das Betriebssystem weitestgehend zu umgehen (jeder Einsprung in den Kernel kostet Zeit). Ferner muss jedes Umkopieren von Daten vermieden werden (*zero–copy–protocol* wird angestrebt).

Prinzipiell wurde mit SCI eine Umgehung des Betriebssystems durch die Realisierung von Distributed Shared Memory (DSM) ermöglicht. Damit werden jedoch lediglich Shared Memory Applikationen besser unterstützt. Message Passing Operationen, wie sie von unseren mathematischen und physikalischen Anwendungen (Projekte in A, C und D) verwendet werden, müssen hier auf Shared Memory abgebildet werden. Das kostet mindestens zwei zusätzliche Kopieroperationen beim Versenden von Daten. Performance–Untersuchungen am PCI Bus [PCI97], [HPVM] haben damals auch gezeigt, dass die Datenübertragungsrate im Shared Memory Modus (d.h. programmed IO) selbst bei modernen PCI Chipsätzen nicht an die maximale Bandbreite des PCI Bus herankommt. Die volle Leistung war nur dadurch zu erreichen, indem die Kommunikationshardware direkt auf die Daten im Hauptspeicher zugreift (DMA). Obwohl sich dieses Problem im Laufe des Antragszeitraumes etwas verbessert hat, besteht bei programmed IO nach wie vor das Problem, dass die CPU bei größeren Detentransfers zu stark in Anspruch genommen wird.

Es hat sich jedoch herausgestellt [WGR97], dass die Verwendung des Shared Memory beim Versenden von sehr kurzen Nachrichten, die zum Beispiel zur Synchronisation verwendet werden, aufgrund der sehr kleinen Latenzzeit bessere Ergebnisse als ein DMA–Modus liefert. Aus diesem Grund sollten einer Kommunikationsbibliothek beide Übertragungsmodi zur Verfügung gestellt werden.

Im Hinblick auf DMA Transfers unter Umgehung des Betriebssystems hat sich der Begriff *Protected User Level DMA* etabliert. Im internationalen Umfeld gab es zur Antragstellung eine Reihe von Projekten in dieser Richtung. Dazu zählen unter anderem das SHRIMP Projekt [SHRIMP1, SHRIMP2], das FLASH Projekt [FLASH] sowie das U–NET [UNET]. Weitere Arbeiten wurden in [HPCA] aufgezeigt. Weiterhin wurde eine neue Spezifikation, die sogenannte Virtual Interface Architecture (VIA) [VIA2] entwickelt, welche eine einheitliche Schnittstelle für Protected User Level DMA bzw. Message Passing Hardware/Systemsoftware allgemein definiert.

Damalige PCI–SCI Implementationen von Dolphin [Dolphin], der TU–München [SMiLE] sowie des CERN [CERN] boten noch keine Realisierungen derartiger architektonischer Merkmale auch für SCI. Die PCI–SCI Bridge des CERN war ursprünglich für reinen Datentransfer entworfen worden, bei dem Aspekte wie Speicherschutz keine Rolle spielen. Im Falle des Projektes an der TU–München stand eine allgemeine Untersuchung von SCI–Traffic bei Shared Memory Applikationen im Vordergrund. Dolphins Implementierung hat lediglich einen Shared Memory Modus angeboten.

Vorbereitend auf Teilthema C waren zur Antragstellung eine Diplom– sowie eine Studienarbeit in Bearbeitung. Die Diplomarbeit beschäftigte sich mit der Konzeption einer PCI–SCI Adapterkarte. Hauptschwerpunkte dieser Diplomarbeit waren eine bessere Unterstützung von SMP–Systemen im Shared Memory Modus (Trennung von Transaktionen verschiedener Prozesse um Transaction Ordering besser auflösen zu können), sowie eine grundlegende Einbettung von Protected User Level DMA. Die Studienarbeit befaßte sich mit der Aufgabe, die Hardware in das Betriebssystem Linux einzubinden. Dazu war einerseits ein Treiber zur Verwaltung der Hardware und zum Exportieren und Importieren von Speicherbereichen zu entwickeln, und andererseits das Memory Management von Linux um die Unterstützung für Shared Memory über SCI zu erweitern.

Innerhalb der Teilaufgabe C sollte sich mit dem Entwurf eines Hardwaremodells beschäftigt werden, welches Message Passing Applikationen schon auf der Hardwareebene möglichst gut unterstützt. Dies umfasste die Entwicklung einer entsprechenden PCI–SCI Adapterkarte sowie deren Einbindung in das Betriebssystem. Bezüglich der Hardware sollten dabei die Schwerpunkte auf einer Unterscheidung in Prozesse auch in der Kommunikationshardware sowie die damit verbundene Möglichkeit für Protected User Level DMA liegen. Ein wichtiger Schritt in diese Richtung ist die Einführung einer zweiten Address Translation Table, um auch Adressen von Upstream Transactions (sprich Zugriffe der Kommunikationshardware auf den Hauptspeicher) übersetzen zu können. Bisherige PCI–SCI Implementationen beinhalteten lediglich eine Downstream Address Translation Table.

Für den konkreten Aufbau der Hardware war dabei im Rahmen eines unabhängigen Projektes eine Zusammenarbeit mit der Chemnitzer Firma TBZ–PARIV GmbH geplant und ist auch zustande gekommen.

Mit Blick auf die Systemsoftware müssen diverse Erweiterungen des Betriebssystems (Linux) vorgenommen werden. Dies betrifft sowohl Treiberentwicklungen, als auch Änderungen am Linux–Kernel selbst.

Die Änderungen am Kernel selbst betrafen hauptsächlich das Memory Management System. Speziell musste die Verwaltung der Page–Tabellen modifiziert werden.

Während des Antragszeitraumes konnten wesentliche Ziele erreicht werden. So wurde ein PCI–SCI Adapter entwickelt, welcher prinzipiell in der Lage ist die Anforderungen eines zero–copy Protokolls für Message–Passing Bibliotheken zu erfüllen. Obwohl diese Hardware derzeit die geplante Funktionalität noch nicht in vollem Umfang zur Verfügung stellt, hat sich gezeigt, dass sie im Hinblick auf die reinen Leistungsparameter doch recht gut mit denen kommerzieller Entwicklungen (im besonderen von Dolphin) mithalten kann.

Parallel zur Hardware wurde auch ein entsprechender Linux–Treiber sowie eine Vielzahl von Dienstprogrammen entwickelt, welche zum Betrieb, zum Testen und zur Weiterentwicklung erforderlich sind.

Eine weitere wichtige Entwicklung war die eines sog. Emulators der Hardware. Mit Hilfe dieses Emulators war es beispielsweise möglich, die in Teilaufgabe B erwähnte Anwendung aus SFB Teilprojekt D2 (MISTRAL) zusammen mit der MPI–Eigenentwicklung sowie einem rudimentären VIA/SCI Device auf dem Forschungscluster zu testen, um damit die Funktionsfähigkeit der MPI–Implementation zu überprüfen.

Literaturverzeichnis

[BEY97] Uwe Beyer. *Optimierung einer multithreaded MPI–Implementierung*. Diplomarbeit. TU Chemnitz, Fakultät für Informatik, Professur Rechnerarchitektur, 1997.

[CERN] *Application of the Scalable Coherent Interface to Data Acquisition at LHC*. `http://www.cern.ch/RD24`

[Dolphin] Dolphin Interconnect Solutions `http://www.dolphinics.com`

[FLASH] J. Heinlein, K. Characholoo, S. Dresser, and A. Gupta. *Integration of Message Passing and Shared Memory in the Stanford FLASH Multiprocessor*. In Proc. of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems, pages 38–50, 1994.

[GLS94] Gropp, Lusk, Skjellum. *Using MPI* . MIT Press, 1994.

[GLS96] Gropp, Lusk, Skjellum. *A High–Performance, Portable Implementation of the MPI Message Passing Standard*. `http://www.mcs.anl.gov/mpi/mpicharticle/paper.html`

[HPCA] E. P. Markatos and M. G. H. Katevenis. *User–Level DMA without System Kernel Modification*. Proceedings to The Third International Symposium on High–Performance Computer Architecture. p. 322ff.

[HPVM] *SYSTEM BANDWIDTH TESTS AND FM PERFORMANCE TESTS RESULTS*. `http://www-csag.cs.uiuc.edu/projects/comm/hcl.html`

[LAM96] *MPI Primer / Developing with LAM*. `http://www.osc.edu/lam.html`

[LAN96] Oliver Langer. *An Implementation of MPI – The Shared Memory Device*. Computer Architecture Technical Report RA–TR–96–08. TU Chemnitz, 1996.

[MPI95] *A Message–Passing Interface Standard*. `http://www.mcs.anl.gov/Projects/mpi/index.html`, Juni 1995.

[MPI97] Message Passing Interface Forum. *MPI–2: Extensions to the Message–Passing Interface*. `http://www.mpi-forum.org/docs/docs.html`

[PCI97] M. Trams, F. Seifert, W. Rehm. *PCI Performance Messungen mit dem HP PCI Exercizer/Analyzer*. Computer Architecture Technical Report RA-TR-97-05. TU Chemnitz, 1997. `http://www.tu-chemnitz.de/informatik/RA/papers/p97/papers.html`

[RAD97] Thomas Radke. *More Message Passing Performance with the Multithreaded MPICH Device*. Computer Architecture Technical Report RA–TR–97–04. TU Chemnitz, 1997.

[Scali] *ScaMPI*. `http://www.scali.com/html/scampi.html`

[SCH97] Sven Schindler. *Weiterentwicklung der MPICH–Implementation für SCI–Cluster*. Studienarbeit. TU Chemnitz, Fakultät für Informatik, Professur Rechnerarchitektur, 1997.

[SHRIMP1] *The Scalable High-performance Really Inexpensive Multi-Processor (SHRIMP) Project.*
`http://www.cs.princeton.edu/shrimp`

[SHRIMP2] M.A.Blumrich, C.Dubnicki, E.W.Felten and Kai Li. *Prodected, User–Level DMA for the SHRIMP Network Interface.* Dept. of Computer Science, Princeton University, 1996.
`http://www.cs.princeton.edu/shrimp/html/papers_stack_18.html`

[SMiLE] *Shared Memory in a LAN-like Environment.* Projekt am Lehrstuhl für Rechnertechnik und Rechnerorganisation der TU-München.
`http://wwwbode.informatik.tu-muenchen.de/Par/arch/smile`

[UNET] M. Welsh, A. Basu, T.v. Eicken. *Incorporating Memory Management into User–Level Network Interfaces.* Dept. of Computer Science, Cornell University, 1997.
`http://www2.cs.cornell.edu/U-Net`

[VIA1] Intel Corporation. *Intel and the VI initiative.*
`http://www.intel.com/procs/SERVERS/isv/vi/vi2/index.htm`

[VIA2] Virtual Interface Architecture Homepage.
`http://www.viarch.org`

[WGR97] Jörg Werner, Lothar Grabowsky, Thomas Radke. *SCI–MPI An optimized implementation of a MPI Subset for SCI connected SMP–systems.* Computer Architecture Technical Report RA–TR–97– 03. TU Chemnitz, 1997.

2.4 Ergebnisse

2.4.1 Teilaufgabe B: MPI–Infrastrukturen zur effizienten Anpassung an heterogene Kommunikationssysteme

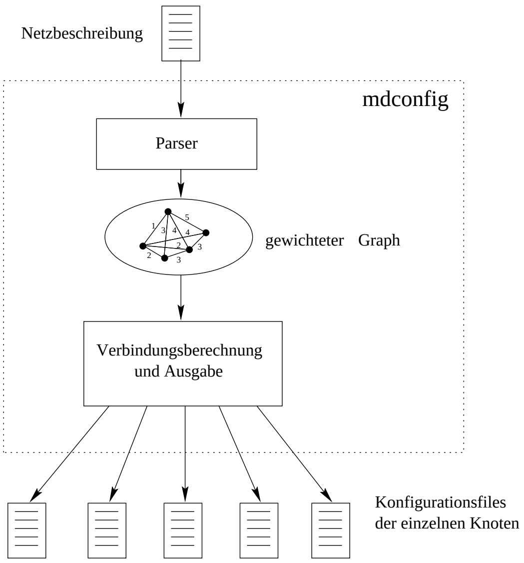
2.4.1.1 Das MPICH Multidevice

Zunächst wurde planmäßig mit der Entwicklung des Multidevices begonnen. Es wurde ein Multidevice für MPICH geschaffen, welches als Subdevices zunächst ein sog. Multi-Threaded Device (für Kommunikation innerhalb eines SMP–Systems) und ein SCI Device (für Kommunikation innerhalb eines Clusters via handelsüblicher SCI Hardware) enthielt. Entsprechende Arbeiten sind in [SCH99D] und [SR99] dokumentiert und veröffentlicht.

Neben dem eigentlichen Multidevice wurde auch ein entsprechendes Konfigurationstool namens mdconfig entwickelt, welches zum Einstellen des Multidevices verwendet wird. Abb. 2.1 stellt die prinziplelle Arbeitsweise von mdconfig dar.

Mit Hilfe von mdconfig wird demnach die zugrunde liegende Architektur (Kommunikationsmöglichkeiten und Eigenschaften) ausgewertet. Die ermittelten Ergebnisse

Abbildung 2.1: Arbeitsweise von mdconfig



werden später vom Multidevice selbst benutzt, um eine Entscheidung über das im konkreten Fall zu benutzende Subdevice zu fällen.

Bei der Gegenüberstellung des Multidevices mit einem "natürlichen" ADI-2 Devices konnte gezeigt werden, dass das Multidevice selbst trotz der zusätzlichen Schicht nicht zu größeren Leistungsverlusten führt. Tabelle 2.1 zeigt dies am Beispiel des Multi–Threaded Devices (Auszug aus [SR99]).

Tabelle 2.1: Leistungsvergleich von Multidevice und normalem ADI–2 Device (Multi–Threaded)

Message Größe in Byte	Multidevice Transfer Zeit in ns	ADI–2 Device Transfer Zeit in ns
1	51.8	50.8
16	52.0	51.3
64	52.2	51.5
256	52.7	51.8
1024	54.6	55.7
32768	208	199
524289	6184	6159

Obwohl ursprünglich für weitere Entwicklungen und Optimierungen der MPI Kommunikationsbibliothek MPICH als Basisimplementation dienen sollte (insbesondere in Hinsicht auf die in Teilaufgabe C zu entwickelnde VIA/SCI Hardware), hat sich dies aus verschiedenen Gründen als unpraktikabel erwießen. Zum einen wurden Entwicklungen auf Basis der ADI-2 Schnittstelle durch ständige Weiterentwicklung von MPICH selbst erschwert (Kompatibilitätsprobleme), und zum anderen haben sich prinzipielle Design–Probleme ergeben, welche die Integration wichtiger innovativer Merkmale (insbesondere Zero-Copy) äußerst schwierig gestalteten.

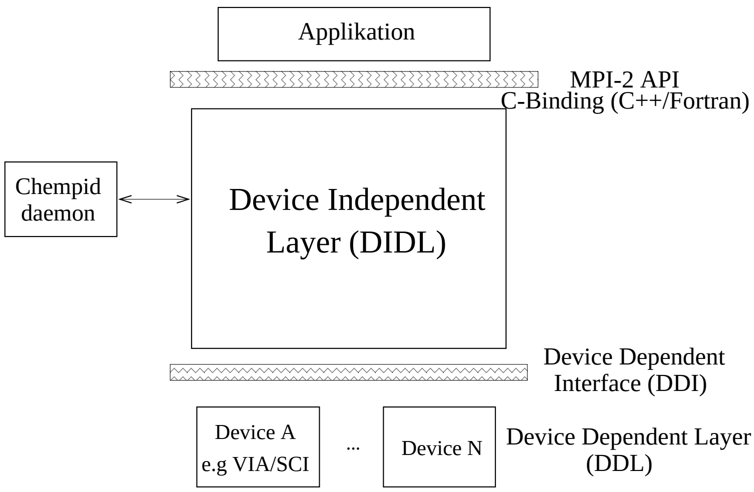
Aus diesen Gründen wurde beschlossen, mit der Entwicklung einer eigenen MPI–Implementation zu beginnen (CHEMPI — CHEmnitz MPI).

2.4.1.2 CHEMPI

Ziel von CHEMPI war und ist die Entwicklung eines Subsets der MPI–2 Funktionalität, welcher ausreichend für die in anderen Teilprojekten innerhalb des Sonderforschungsbereiches entwickelten MPI–Applikationen ist. Auch wurde von vorn herein großer Wert auf ein möglichst effizientes Zusammenspiel zwischen der MPI–Bibliothek und der in Teilaufgabe C entwickelten Kommunikationshardware gelegt. Damit sollten die prinzipiellen Hürden umgangen werden, die sich bei der Benutzung von MPICH für diesen Zweck ergeben hatten.

Der prinzipielle Aufbau von CHEMPI wurde dabei ähnlich zu MPICH gestaltet. D.h. die grundlegende allgemeine MPI Funktionalität wird in einer komplexeren Schicht (*Device Independent Layer*) realisiert, während die Anbindung an konkrete Hardware im sog. *Device Dependent Layer* umgesetzt wird. Abbildung 2.2 veranschaulicht diesen Zusammenhang.

Abbildung 2.2: Struktureller Aufbau von CHEMPI



Damit wird auch bei CHEMPI die prinzipielle Möglichkeit gegeben, verschiedene Hardwarearchitekturen als Kommunikationsmedium einzusetzen. Dabei sind die Erfahrungen, die mit dem Multidevice für MPICH gemacht worden [SCH99D, SR99] in das Design von CHEMPI eingeflossen. D.h. CHEMPI unterstützt ein solches Multidevice schon von Haus aus, bzw. es ist nicht erforderlich ein solches Device für CHEMPI zu entwickeln, da die entsprechende Funktionalität schon im *Device Independent Layer* integriert ist.

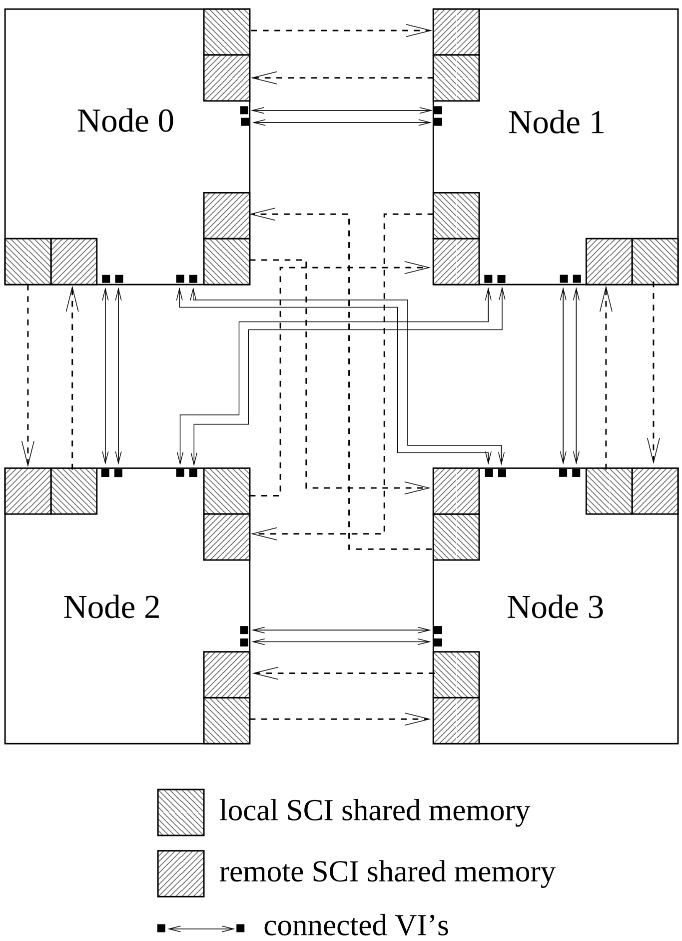
Obwohl mehrere Devices im Sinne eines Multidevices unterstützt werden, liegt das Hauptaugenmerk aber auf der Kombination von Scalable Coherent Interface und Virtual Interface Architecture (Teilaufgabe C). Die Möglichkeit für verschiedene Kommunikationsmedien ist aber nach wie vor von großer Bedeutung für das Gesamtkonzept. Zum einen lassen sich dadurch zu Test– und Entwicklungszwecken einfachere Devices (z.B. TCP/IP) anbinden, zum anderen kann andere Hardware angebunden werden, um die Güte der in Teilaufgabe C realisierten Kommunikationshardware zu bestimmen.

2.4.1.3 Das VIA/SCI Device

Wie oben erwähnt, ist das VIA/SCI Device von großer Bedeutung für das Gesamtkonzept von CHEMPI, da nur durch dieses Device aufgrund seiner Architektur bestmögliche Leistungen erreicht werden können. Nähere Hardware– und Betriebssystemspezifische Eigenschaften der dem VIA/SCI Device zugrundeliegenden Architektur werden im Rahmen von Teilaufgabe C beschrieben.

Abbildung 2.3 zeigt die Art und Weise in der VIA/SCI Devices verschiedener Knoten logisch miteinander verbunden sind.

Abbildung 2.3: Verbindungsrelationen des CHEMPI VIA/SCI Devices



Das besondere hierbei ist die Verbindung über Message–Passing ähnliche Kanäle (verbundene Virtual Interfaces — VIs) und über Distributed Shared Memory. Ziel dieser Kombination ist eine gute Leistungsausbeute für kleine und auch größere Messages.

So sollen kleine Messages bevorzugt via Distributed Shared Memory (insbesondere durch Schreiboperationen in entfernten Speicher) übertragen werden, während größere Messages über den Protected User Level DMA Mechanismus der Virtual Interface Architecture versendet werden.

2.4.1.4 Potential eines VIA/SCI Devices

Um das Leistungspotential eines VIA/SCI Devices abschätzen zu können, wurde eine Testimplementierung auf Basis kommerzieller SCI (Dolphin) und VIA (GigaNet) Hardware vorgenommen. Bei kleinen Messages wurden diese dabei über den Distributed Shared Memory von SCI übertragen, und bei größeren Messages wurde auf die Remote DMA der VIA Hardware zurückgegriffen.

Abbildungen 2.4 und 2.5 zeigen den Vergleich dieser Testimplementierung mit ScaMPI, einer kommerziellen MPI–Implementation für SCI von Scali, und mit MPI/Pro, einer kommerziellen MPI–Implementation für GigaNet's VIA Hardware von MPI Software Technology.

Unabhängig davon wurden SCI und VIA (GigaNet cLAN) MPI Implementationen bezüglich ihrer Leistunsfähigkeit betrachtet [SBR00]. Abbildung 2.6 zeigt das Verhältnis der mittels eines einfachen Benchmarkes (NetPipe) aufgenommenen MPI–Kommunikationsleistung von SCI (ScaMPI) und VIA (GigaNet cLAN mit MPI/Pro).

2.4.1.5 VIA/SCI Device mit Zero–Copy Möglichkeit

Um richtig zero–copy auf Basis einer kombinierten VIA/SCI Hardware betreiben zu können, sind spezielle Vorkehrungen nötig. Obwohl die Hardware dabei wesentliche Mechanismen zur Verfügung stellt (z.B. jede beliebige Speicherstelle eines Anwendungsprozesses kann der Hardware zur Verfügung gestellt und/oder von einem entfernten Prozess importiert werden [TRBS00]) sind die Ressourcen doch beschränkt. D.h. exportierbarer und importierbarer Speicher sind begrenzt. Hinzu kommt noch, daß es sich bei solchen sog. Memory–Registration Operations um relativ zeitaufwendige Operationen handelt [SR00].

Das bedeutet, dass derartige Registrierungen nicht jedesmal durchgeführt werden können wenn die MPI Applikation kommunizieren möchte, und unmittelbar danach wieder entfernt werden, um Hardwareressourcen freizugeben. Schließlich würde dies auch dem angestrebten Paradigma eines *User–Level Hardware Access* widersprechen.

Um dem zu begegnen wurde in [JOR00] ein Mechanismus entwickelt, welcher es ermöglicht diese Ressourcen dynamisch zu verwalten.

Vereinfacht gesagt bedeudet dies beispielsweise, dass wenn Speicher in der Hardware registriert werden soll (z.B. um ihn zu exportieren, sprich entfernten Prozessen zugänglich machen), dies zunächst durch entsprechend zeitaufwendige Systemrufe realisiert wird. Da die Wahrscheinlichkeit groß ist, dass dieser Speicherbereich in naher Zukunft wieder für den selben Zweck benutzt wird (es kann sich z.B. um ein Ergebnisfeld einer MPI–Applikation handeln), bleibt diese Registrierung für einen unbestimmten Zeitraum erhalten. Damit kommt eine weitere Nutzung dieses

Abbildung 2.4: Vergleich des VIA/SCI Testdevices mit ScaMPI und MPI/Pro (Bandbreite)

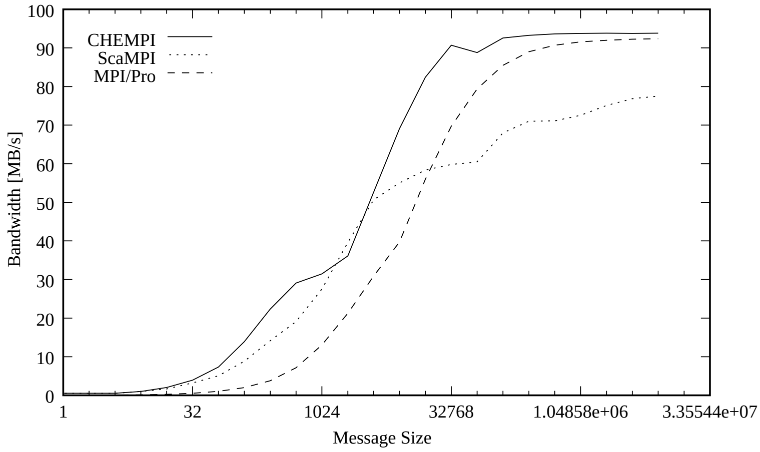
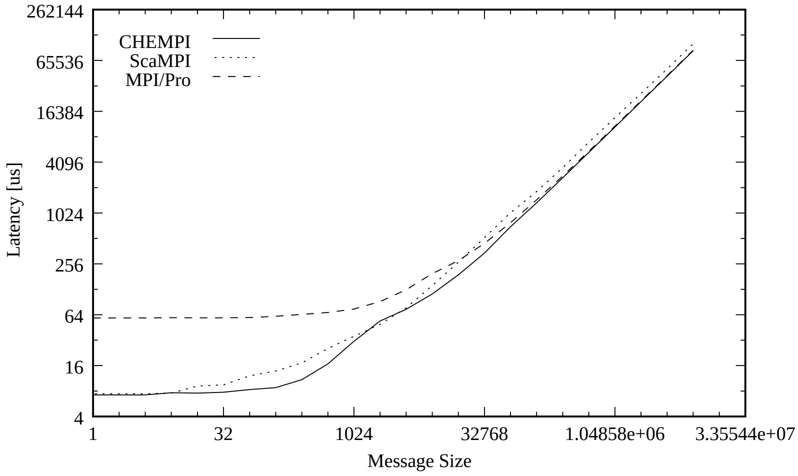


Abbildung 2.5: Vergleich des VIA/SCI Testdevices mit ScaMPI und MPI/Pro (Latenzzeit)



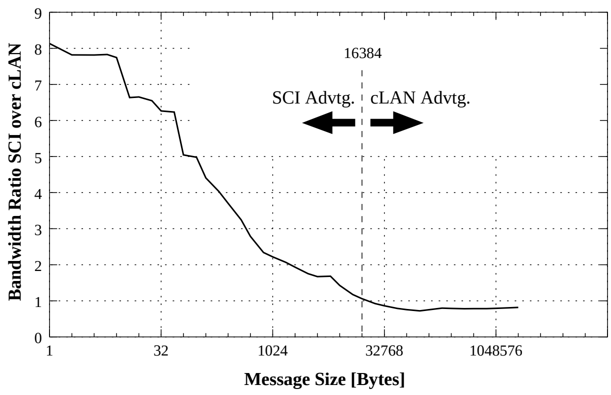
Bereiches ohne neuerliche Registrierung aus und die Hardwareressourcen können sofort ohne weitere Systemrufe genutzt werden.

Neigen sich dabei verfügbare Ressourcen dem Ende zu, so werden diese automatisch entzogen und neu verteilt.

Dieser Mechanismus setzt dabei unmittelbar auf der *ExtVIPL* (siehe Abschnitt 2.4.2.4 auf Seite 114) auf und fungiert als eine Art Wrapper dieser Library. Für die darüberliegende MPI bzw. CHEMPI Device Schicht ist diese Funktionalität weitestgehend transparent.

Bezüglich der Implementation des eigentlichen VIA/SCI Devices mit Zero–Copy Fähigkeit für CHEMPI ist momentan eine Diplomarbeit in Bearbeitung [JOR01].

Abbildung 2.6: Verhältnis von SCI Bandbreite (über ScaMPI) zu VIA cLAN Bandbreite (über MPI/Pro)

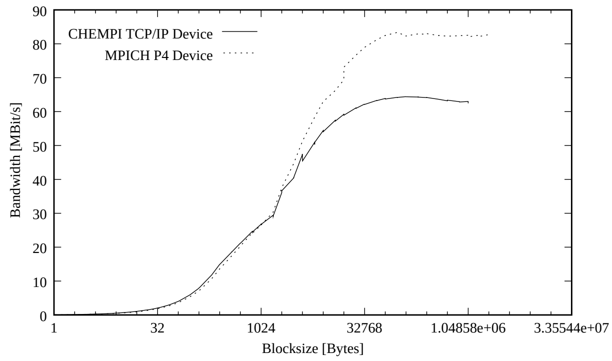


2.4.1.6 TCP/IP Device

Um CHEMPI auch mit Standard–Hardware (z.B. einfache FastEthernet Karten) betreiben zu können, wurde zunächst ein TCP/IP Device auf Basis des p4–Devices von MPICH entwickelt [SCHM00].

Ein Leistungsvergleich zum p4–Device von MPICH hat dabei zumindest im Bereich der kleineren Messages nicht schlecht abgeschnitten, wie aus Abbildung 2.7 ersichtlich wird.

Abbildung 2.7: Vergleich des CHEMPI TCP/IP Devices mit dem MPICH p4 Device



Bei größeren Messages liegt die Leistung des CHEMPI TCP/IP Devices lediglich bei etwa 75% des p4–Devices. Hier besteht jedoch noch die Möglichkeit zu Verbesserungen.

Hierzu wird voraussichtlich in Kürze eine Diplomarbeit bearbeitet werden, in der komplett vom p4–Device Abstand genommen werden soll.

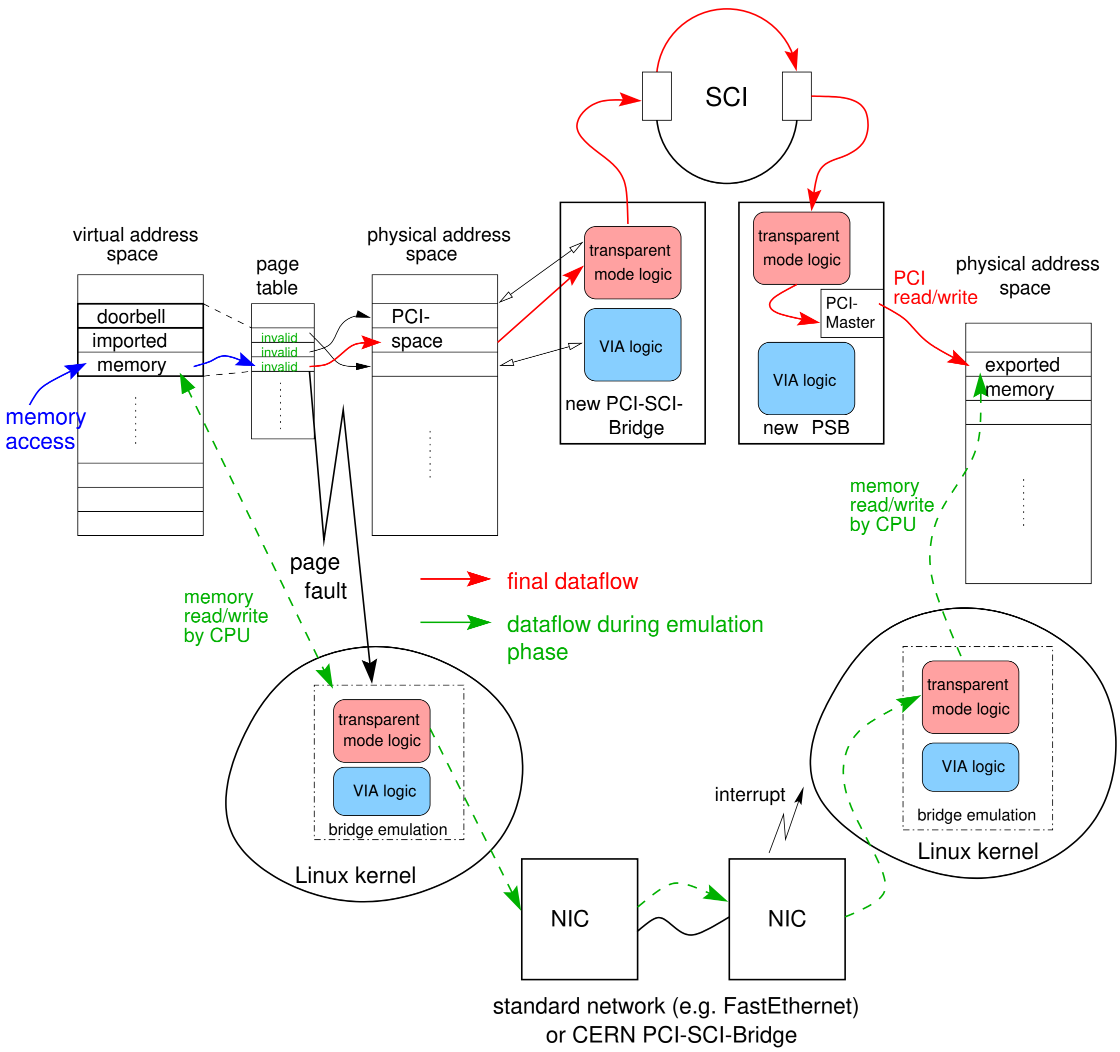
2.4.2 Teilaufgabe C: VIA–konformer PCI–SCI Adapter für optimiertes Message Passing

Wie vorgesehen wurde mit der Konzeption [TRA98] einer Hardware begonnen, welche die Vorzüge eines Distributed Shared Memory von SCI mit Protected User Level DMA der Virtual Interface Architecture verbindet. Danach wurde mit der konkreten Umsetzung des Konzeptes begonnen.

2.4.2.1 Hardware Emulation

Parallel zur Realisierung der Hardware wurde der Hardware–Emulator entwickelt [SEI99]. Abbildung 2.8 zeigt die prinzipielle Arbeitsweise des Emulators.

Abbildung 2.8: Prinzip der Hardware–Emulation

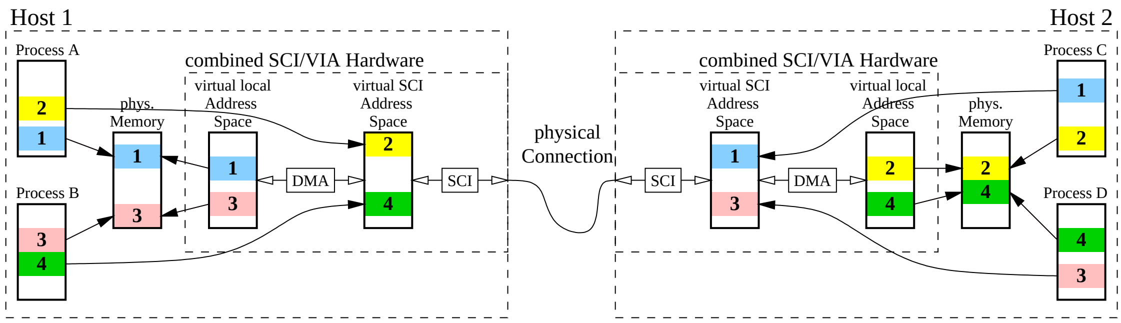


Der Haupteingriffspunkt des Emulators in das System ist dabei das Paging bzw. die Memory Management Unit des Prozessors. Da die Hardware selbst ausschließlich per Memory–Mapped IO gesteuert wird, kann diese daher vollständig emuliert werden. Die über den Emulator erreichbare Leistung ist zwar extrem gering (hier wurden 120kB/s bei Schreiboperationen auf entfernten Speicher gemessen [SEI99D]), jedoch reicht dies für prinzipielle Tests und Entwicklungen an höheren Softwareschichten (z.B. MPI Bibliothek) aus.

2.4.2.2 Logische Hardwarearchitektur

Das Prinzip der Kommunikationshardware beruht auf einer für Message Passing ausgelegten Kombination von SCI und VIA [TRS99, TRBS00, RTBS00]. Dabei wurde im wesentlichen die in der VIA vorgesehene virtuelle Sicht auf den lokalen Speicher in eine herkömmliche SCI Architektur integriert. Abbildung 2.9 zeigt diesen Sachverhalt anhand eines einfachen Beispiels mit 4 Prozessen auf 2 Knoten. Die

Abbildung 2.9: Prinzipbeispiel der Verschmelzung von VIA mit SCI



einzelnen Prozesse können dabei Speicher von anderen Prozessen im Sinne eines Distributed Shared Memory von SCI importieren und direkt darauf arbeiten, oder sie können Gebrauch von der DMA Engine machen, um größere Datenblöcke zu verschieben. DMA Transfers können direkt von den Anwendungsprozessen angestoßen werden (ohne Betriebssystemaufrufe), da die Hardware eventuelle Schutzverletzungen nicht zulässt (Protected User Level DMA).

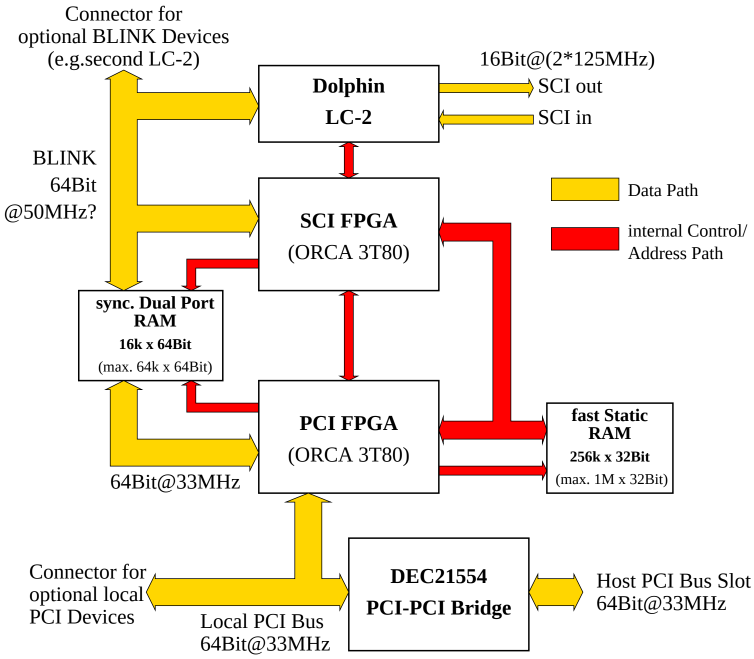
2.4.2.3 PCI–SCI Hardware

Es wurde erfolgreich eine auf FPGAs (Field Programmable Gate Arrays, programmierbare Schaltkreise) und einem kommerziell erhältlichen SCI Link Controller der Firma Dolphin entwickelt und aufgebaut [TR99, TSR00C]. Abbildung 2.10 zeigt die physische Architektur dieser Karte. Aufgrund der Programmierbarkeit der Hardware wurde diese Karte Generic PCI–SCI Bridge, oder kurz GPSB genannt.

Stand der Entwicklung dieser Hardware ist dabei die Möglichkeit Speicher zu exportieren als auch zu importieren, sowie auf entfernten Speicher zu schreiben. Ein einfacher Paket Transfer Modus, welcher zum Datenaustausch der Systemsoftware vorgesehen ist, wurde ebenfalls implementiert.

Erste Messungen der Latenzzeit von Schreiboperationen auf entfernten Speicher ergaben einen Wert von ca. 4μs. Obwohl diese Latenzzeit um einiges höher ist als

Abbildung 2.10: Blockschaltbild der PCI–SCI Bridge



bei kommerzieller Hardware von Dolphin (ca. 2μs), ist dies doch besser als erwartet. Schließlich handelt es sich hierbei mehr um ein Proof–of–Concept mit relativ langsamen FPGAs im Gegensatz zu der ASIC–Lösung in Dolphins Hardware.

2.4.2.4 Die erweiterte Virtual Interface Provider Library

Einhergehend mit der architekturellen Verschmelzung von SCI und VIA wurde die ursprüngliche Virtual Interface Provider Library (VIPL) erweitert, so dass es mit der sog. *ExtVIPL* nicht nur möglich ist bestimmte Hardwarefunktionen im Sinne der konventionellen VIA anzusprechen, sondern ebenfalls lokalen Speicher zu exportieren (entfernten Knoten/Prozessen zu Verfügung stellen) und entfernten Speicher zu importieren.

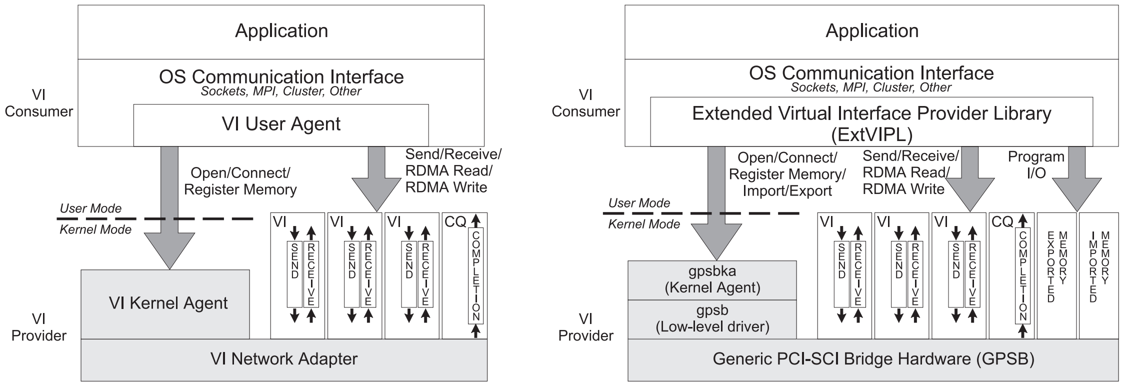
Abbildung 2.11 stellt diesen Sachverhalt an einer Gegenüberstellung des ursprünglichen VIA–Schichtenmodells mit dem erweiterten Modell dar. Dieses erweiterte Modell spiegelt dabei deutlich die zugrundeliegende Hardwarearchitektur wider.

2.4.3 Hardware–Treiber und Dienstprogramme

Obwohl dies nicht unmittelbar zu konzeptionell neuen Ergebnissen führt, ist mit der Entwicklung neuer Hardware die Entwicklung entsprechender Treibersoftware etc. notwendig.

Derzeit unterstützt der Treiber die neueste Linux–Kernel Version (2.4.x) und ist sowohl auf Systemen mit Intel–Architektur (AMD eingeschlossen) als auch auf Alpha–Systemen lauffähig. Eine Unterstützung der Alpha–Architektur war dabei nicht trivial, da es hier doch erhebliche architekturelle Unterschiede gibt.

Abbildung 2.11: Vergleich der herkömmlichen VI Architektur (links) mit der erweiterten (rechts)



Die Unterstützung der Alpha–Architektur war jedoch sehr wichtig, da diese Systeme derzeit mit zu den führenden im Hinblick auf Rechen– und Kommunikationsleistung zählen. Aus diesem Grund wurde auch im Jahr 2000 im Rahmen des Sonderforschungsbereiches ein kleinerer Alpha–Cluster angeschafft, welcher zum einem als Rechencluster dem SFB zur Verfügung steht, und zum anderen als Testplattform für die hier entwickelte Hardware genutzt wird.

Abgesehen von der üblichen Treiberfunktionalität selbst (Initialisierung, Interruptbearbeitung, Bereitstellung von Systemrufen etc.) integriert dieser Treiber einen einfachen Messaging–Mechanismus, welcher wiederum von der Hardware selbst als einfachster Transfer–Modus unterstützt wird (der sog. Manuelle Pakettransfer Modus). Dieser Mechanismus soll als eine Art Verbindungsaufbaumedium dienen, den sich die SCI/VIA–Systemsoftware zu Nutze macht, um z.B. globale Speichermappings aufzubauen. Entsprechende Arbeiten dazu laufen zur Zeit.

Neben dem Treiber selbst wurde eine Vielzahl von Konfigurations– und Testprogrammen entwickelt. Um nur einige zu nennen:

- Programmierung des Flash–ROMs zur FPGA–Initialisierung
- Programmierung diverser Konfigurations–EEPROMs
- Test der Hardwarekomponenten
- Abfrage von Statusregistern
- ...

Literaturverzeichnis

Begutachete Literatur

[RSTB00] Wolfgang Rehm, Stanislav Simeonov, Mario Trams und Daniel Balkanski: *Message Passing on PCI-SCI Interfaces*, Proceedings of Bulgarian Academy of Sciences, vol.8, 2000, Publisher: Academic Publishing “Acad. M. Drinow”

[RTBS00] Wolfgang Rehm, Mario Trams, Daniel Balkanski und Stanislav Simeonov: *A New Architectural Concept for Highly Efficient Message-Passing on PCI–SCI Network Interfaces*, 14th International Conference "System for Automation of Engineering and Research"SAER'2000, 18.–20. September 2000, St. Konstantin (Varna), Bulgarien

[SBR00] Friedrich Seifert, Daniel Balkanski und Wolfgang Rehm: *Comparing MPI Performance of SCI and VIA*, Proceedings of the SCI-Europe 2000, 29.–30. August 2000, München, im Rahmen der Euro-Par 2000.

[SR00] Friedrich Seifert und Wolfgang Rehm: *Proposing a Mechanism for Reliably Locking VIA Communication Memory in Linux* Proceedings of the CLUSTER2000 — IEEE International Conference on Cluster Computing, 28.Nov.–1.Dez. 2000, Chemnitz, ISBN 0-7695-0896-0, Seiten 225–232.

[SR99] Sven Schindler und Wolfgang Rehm: *Multiple Devices Under MPICH* , PASA'99 5. Workshop Parallele Systeme und Algorithmen, 7. Oktober 1999, Jena, im Rahmen der ARCS'99 — 15. GI/ITG-Fachtagung Architektur von Rechensystemen.

[SRD00] Sven Schindler, Wolfgang Rehm und Carsten Dinkelmann: *An optimized MPI library for VIA/SCI cards* , Proceedings of the Asia-Pacific International Symposium on Cluster Computing (APSCC'2000) im Rahmen der Fourth International Conference/Exhibition on High Performance Computing in Asia-Pacific Region (HPCAsia2000), 14.–17. Mai 2000, Beijing/China, Volume II, Seiten 895–903.

[TRS99] Mario Trams, Wolfgang Rehm und Friedrich Seifert: *An Advanced PCI-SCI Bridge With VIA Support*, 2.Workshop Cluster Computing, 25./26. März 1999, Universität Karlsruhe. Chemnitzer Informatik-Berichte CSR-99-02 (157 S.). ISSN 0947-5125

[TR99] Mario Trams und Wolfgang Rehm: *A New Generic and Reconfigurable PCI-SCI Bridge*, Proceedings of the SCI-Europe'99, 2.–3. September 1999, Toulouse/Frankreich, im Rahmen der Euro-PAR'99.

[TRBS00] Mario Trams, Wolfgang Rehm, Daniel Balkanski und Stanislav Simeonov: *Memory Management in a combined VIA/SCI Hardware*, Proceedings of the PC-NOW 2000, International Workshop on Personal Computer based Networks of Workstations im Rahmen des International Parallel and Distributed Processing Symposium (IPDPS 2000), 1.–5. Mai 2000, Cancun/Mexico. Springer LNCS Series, ISSN 0302-9743, ISBN 3-540-67442-X, Seiten 4–15.

[TSR00C] Mario Trams, Ralph Schlosser und Wolfgang Rehm: *Design Choices and First Results of Our VIA-Capable PCI-SCI Bridge*, Proceedings of the CLUSTER2000 — IEEE International Conference on Cluster Computing, 28.Nov.–1.Dez. 2000, Chemnitz, ISBN 0-7695-0896-0, Seiten 349–350.

Sonstige Arbeiten (Diplomarbeiten, Technical Reports etc.)

[JOR00] Lars Jordan *Entwicklung eines effizienten Speichermanagementes für das CHEMPI VIA/SCI Device*, Studienarbeit, TU Chemnitz, Fakultät für Informatik, Oktober 2000

[JOR01] Lars Jordan *Entwicklung eines VIA/SCI Devices fr CHEMPI*, Diplomarbeit (in Bearbeitung), TU Chemnitz, Fakultät für Informatik

[SCH99D] Sven Schindler: *Entwurf und Implementierung eines ADI–2 Multidevices*, Diplomarbeit, TU Chemnitz, Fakultät für Informatik, März 1999

[SCHM00] Ralf Schmidt *Entwicklung eines TCP/IP Devices für CHEMPI*, Studienarbeit, TU Chemnitz, Fakultät für Informatik, Oktober 2000

[SEI99] Friedrich Seifert: *Design and Implementation of System Software for Transparent Mode Communication over SCI*, Studienarbeit, TU Chemnitz, Fakultät für Informatik, Februar 1999

[SEI99D] Friedrich Seifert: *Development of system software to integrate the Virtual Interface Architecture (VIA) into the Linux operating system kernel for optimized message passing*, Diplomarbeit, TU Chemnitz, Fakultät für Informatik, Oktober 1999

[TRA98] Mario Trams: *Design of a system-friendly PCI-SCI bridge with an optimized user-interface*, Diplomarbeit, TU Chemnitz, Fakultät für Informatik, September 1998

Projektseiten im Internet

[CHEMPI] Projektseite von CHEMPI:
`http://www.tu-chemnitz.de/informatik/RA/projects/chempi-html`

[HARD] Projektseite der Hardwareentwicklung:
`http://www.tu-chemnitz.de/~mtr/VIA_SCI`

[SYSSOFT] Projektseite der VIA/SCI Systemsoftware:
`http://www.tu-chemnitz.de/~sfri/ra/via_linux`

2.5 Offene Fragen/Ausblick

Bezüglich der MPI Implementation CHEMPI gilt es, insbesondere das VIA/SCI Device konsequent zu vervollständigen, um das Potential der Hardware voll ausschöpfen zu können. Desweiteren muss die Funktionalität von CHEMPI weiter vervollständigt werden, wobei hier der Schwerpunkt auf der wirklich benötigten MPI bzw. MPI–2 Funktionalität liegt.

Die im Rahmen von Teilaufgabe C erarbeiteten Hard– und Systemsoftwarelösungen müssen noch weiter vervollständigt werden. Bei der Hardware betrifft dies insbesondere die DMA Engine, welche zur Entlastung des Prozessors und zur Steigerung der Bandbreite benötigt wird.

Ferner müssen folgende Dinge näher untersucht werden:

- Inwiefern lässt sich CHEMPI bzw. die Message Passing Bibliothek allgemein noch besser durch die Hardware unterstützen?
- Wie lassen sich Message Passing Applikationen durch die Systemsoft– und Hardware im Hinblick auf Ressourcenausnutzung besser unterstützen?

- Wie kann man den (gegenüber der Hardware so großen) Software–Overhead im Kommunikationsinterface minimal halten, um effiziente Kommunikation, insbesondere eff. globale Operationen erzeugen zu können?
- Wie ist die obige Fragestellung in heterogenen Clustersystemen zu beantworten?