Entwurf und Implementierung einer Windows-spezifischen, TCP/IP-basierten Geräteschnittstelle für MP-MPICH

Silke Schuch, Carsten Clauß, Thomas Arens, Stefan Lankes, Thomas Bemmerl
Lehrstuhl für Betriebssysteme,
RWTH Aachen University,
Kopernikusstr. 16, 52056 Aachen, Germany
E-Mail: {silke, carsten, arens, lankes, bemmerl}@lfbs.rwth-aachen.de

Abstract

Im Bereich der Clusterverbundsysteme führen Windows-basierte Systeme mit wenigen Ausnahmen [17] ein Nischendasein. Allerdings wird in der Zukunft jeder Bürokraft einer mittelständischen Firma durch die Entwicklung von Dual Core CPUs eine zweite Ausführungseinheit zur Verfügung stehen, die durch die Verwendung normaler Büroanwendungen kaum ausgelastet wird. Diese brachliegende Rechenleistung kann u.a. dazu verwendet werden, rechenintensive Programme innerhalb eines Firmennetzes parallel auszuführen. Allerdings muss sich die Systemsoftware für Clusterverbundsysteme den Gegebenheiten einer Büroumgebung anpassen und die effiziente Unterstützung eines TCP/IP-basierten Netzes innerhalb einer Windows-dominierten Umwelt gewährleisten.

Der Lehrstuhl für Betriebssysteme entwickelte innerhalb des Projekts MP-MPICH[1] *eine optimierte MPI-Bibliothek für Windows. Hierbei wurden sowohl das Scalable Coherent Interface (SCI) als auch TCP/IP-basierte Netze effizient integriert, wobei die Integration des Scalable Coherent Interface u.a. in [21, 25] beschrieben wurde. Bei der Integration der TCP/IP-basierten Netze wurde nicht die BSD-konforme sondern die Windows-spezifische Schnittstelle verwendet, die unter dem Namen Windows Sockets 2 bekannt ist. Zwar basiert diese Schnittstelle auch auf den Berkeley Sockets, allerdings wurde diese u.a. mit überlappenden Sendeoperationen erweitert. In diesem Artikel wird die Integration der Windows Sockets innerhalb einer MPI-Bibliothek präsentiert und die Unterschiede zu einer traditionellen, BSD-konformen Lösung erläutert. Anhand aussagekräftiger Benchmarks wird die Leistungsfähigkeit dieser Implementierung belegt und somit ihre Einsatzfähigkeit untermauert.*

[1] http://www.lfbs.rwth-aachen.de/content/mp-mpich

Workshop Kommunikation in Clusterrechnern und Clusterverbundsystemen (KiCC 2005), Chemnitz, Germany, November 2005

1. Einleitung

MP-MPICH [1] ist das Projekt für die Implementation der MPI-Bibliothek des Lehrstuhls für Betriebssystem der RWTH-Aachen. MP-MPICH basiert auf MPICH 1.2.0 [6] und erfüllt den MPI-1 Standard [14], zusätzlich werden auch einige Funktionen des MPI-2 Standards [15] unterstützt.

Unter dem Oberbegriff MP-MPICH werden die Projekte SCI-MPICH [24], NT-MPICH und Meta-MPICH [12, 19] zusammengefasst.

SCI-MPICH ist der Teil, der für die Unterstützung des Scalable Coherent Interface (SCI) [7] als Cluster-Interconnect zuständig ist. Hierbei werden für die Betriebssysteme Linux, Solaris und Windows SCI-Schnittstellen zur Verfügung gestellt.

Die generelle Unterstützung von Windows-Rechnern als Clusterknoten erfolgt im Projekt NT-MPICH. Die im folgenden besprochenen Geräteschnittstellen ch_wsock2 und ch_usock sind daher Teile von NT-MPICH. Zusätzlich werden auch Tools für den Start entfernter Prozesse angeboten, da dies bei Windows nicht direkt durch das Betriebssystem unterstützt wird.

Das Projekt MetaMPICH beschäftigt sich mit der Kopplung von Clustern. Dies kann zum einen durch dedizierte Router-Prozess geschehen. In diesem Fall erfolgt die Kopplung für die Rechenprozesse auf den Knoten transparent über ein WAN wie ATM[2] [4] oder Gigabit-Ethernet. Zum anderen besteht die Möglichkeit, für Inter- und Intra-Cluster-Kommunikation unterschiedliche Kommunikationsmedien auszuwählen. So kann die Intra-Cluster-Kommunikation über schnelle Cluster-Interconnects erfolgen, z.B. SCI. Die Inter-Cluster-Kommunikation erfolgt über ein auf allen Knoten verfügbares sekundäres Netzwerk z.B. Ethernet. Eine Kombination von direkter und Router-basierter Kommunikation ist möglich.

Obwohl viele schnelle Cluster-Interconnects wie SCI, Myrinet [23] und Infiniband [8, 9] existieren, ist die Unterstützung von Ethernet als primärem TCP/IP Interconnect sinnvoll, wie die Anwendungsbeispiele im folgenden Abschnitt zeigen.

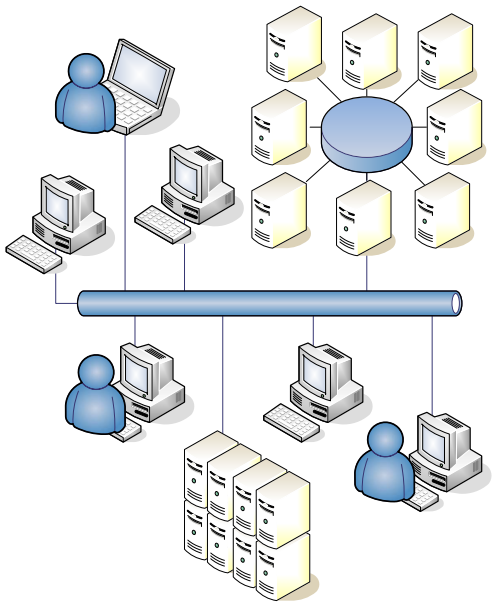

Abbildung 1. Beispiel Office Grid

1.1 Anwendungsgebiete von Ethernet

Zusätzlich zu dedizierten Clustern können auch reguläre
Arbeitsstationen als Rechenknoten in einer parallelen Umgebung eingesetzt werden. Eine solche Umgebung wird von uns als *OfficeGrid* bezeichnet, siehe dazu auch Abbildung 1.

Diese Arbeitsplatzrechner, die beispielsweise in Sekretariaten stehen, haben oft Microsoft Windows als Betriebssystem. Um diese Rechner als bisher ungenutzte Ressource für parallele Applikationen zur Verfügung zu stellen, ist es daher notwendig, ein MPI-Bibliothek speziell für Windows einzusetzen. Arbeitsstationen in einem Firmen- oder Universitätsnetzwerk sind in der Regel

[2] Asynchronous Transfer Mode

mit Ethernet vernetzt, eine TCP/IP-Geräteschnittstelle kann hier so eingesetzt werden, dass diese Rechner an einer MPI-Applikation mitrechnen können. Des Weiteren ist Gigabit-Ethernet als kostengünstige Alternative auch in dedizierten Clustersystemen einsetzbar.

Im Rahmen von MetaMPICH kann Ethernet zur Kopplung von Clustern eingesetzt werden. Für alle genannten Anwendungsfälle kann die selbe TCP/IP-Geräteschnittstelle, das so genannte TCP/IP-Device, im Rahmen der MPI-Applikationen Verwendung finden. Durch die Entwicklung von *Dual-Core* [2] und *Multi-Core* CPUs [20] werden in naher Zukunft mehrere Ausführungseinheiten innerhalb eines Rechners zur Verfügung stehen. Es kann also sinnvoll sein, auch mehrere MPI-Prozesse auf einem Rechner zu starten. Die Prozesse, die lokal auf einem Knoten liegen, benutzen zur Kommunikation dabei das selbe Device, wie Prozesse, die auf entfernten Knoten liegen. Um die Kommunikation zwischen Prozessen, die auf dem selben Knoten liegen, zu beschleunigen, wird diese nicht über TCP/IP durchgeführt, sondern über gemeinsame Speicherbereiche. Details zur Initialisierung und Implementation der TCP/IP-Geräteschnittstelle für Windows, d.h. des Devices *ch_wsock2*, finden sich im folgenden Abschnitt.

2. Implementationsdetails

Wie in Abbildung 2 gezeigt, setzen MPI-Applikationen auf die standardisierte MPI-API auf. NT-MPICH hat innerhalb der MPI-Implementation das Schichtenmodell von MPICH übernommen. MPIP ist dabei lediglich eine Instrumentierungsschnittstelle, während MPIR für die Transformation von komplexen MPI-Funktionen zu einfachen Punkt-zu-Punkt-Kommunikationen zuständig ist. Zum Beispiel werden in diesem Teil die kollektiven Operationen realisiert. Der MPIR-Teil setzt wiederum auf der ADI-2 Schnittstelle auf, die unabhängig vom verwendeten Interconnect realisiert wurde. Der MPID-Teil realisiert die geräteabhängige Funktionalität, wobei bei der Portierung auf eine neue Plattform hauptsächlich das so genannte *Device* angepasst werden muss, dass die direkte Schnittstelle zum Interconnect darstellt.

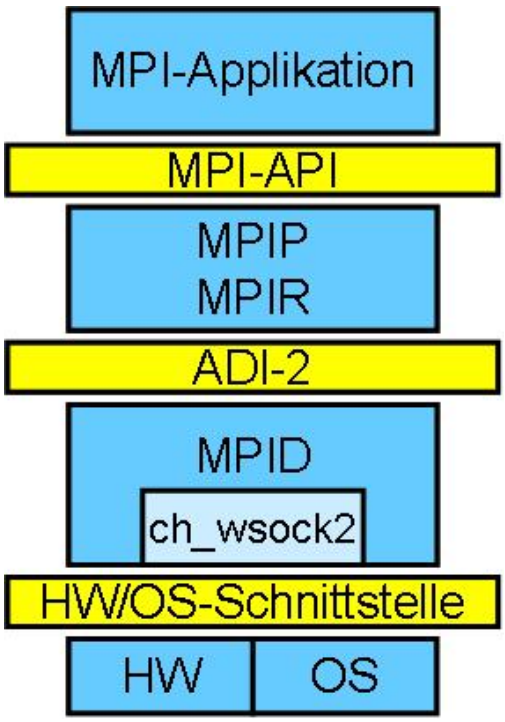

Abbildung 2. Struktureller Aufbau von NT-MPICH

Wie bereits erwähnt, unterstützt das Device *ch_wsock2* sowohl die Kommunikation über TCP/IP als auch über gemeinsame Speichersegmente. Die Entscheidung, welcher Kommunikationsweg zwischen zwei Prozessen gewählt wird, wird in der Initialisierungsphase getroffen, die im Abschnitt 2.1 genauer erläutert wird.

Das Device ch_wsock2 basiert, wie der Name schon sagt, bei der TCP/IP-Kommunikation auf die Windows-Socket-2-API [13]. Diese Sockets bieten gegebenüber den Berkley-Sockets eine erweiterte Funktionalität an, auf die im Abschnitt 2.2 genauer eingegangen wird. Wie die Send- und Recv-Funktionen diese Besonderheiten der Windows-Sockets ausnutzen, wird in den entsprechenden Abschnitten 2.3 und 2.4 beschrieben.

2.1 Initialisierung

Das Device ch_wsock2 basiert auf einer Master-/Client-Architektur. Über die Startparameter der Applikation erhält der Masterprozess Informationen über die Gesamtzahl der MPI-Prozesse, während die Clients Informationen über den Rechner erhalten, auf dem der Masterprozess läuft. Durch diese Information können die Clients auch herausfinden, ob sie auf dem gleichen Rechner laufen, wie der Masterprozess. Die Initialisierung der Kommunikation erfolgt generell über Sockets. Der Master öffnet für jeden Client einen Socket auf einem festen Port und wartet, bis sich die vorgegebene Anzahl Clients verbunden hat. Nachdem sich die Clients mit den Master verbunden haben, senden sie jeweils eine Initialisierungsnachricht an ihm. Diese Nachricht enthält Informationen über die verfügbaren IP-Adressen des Clients sowie die Angabe ob sich der Client auf dem selben Rechner befindet, wie der Master. Der Master sendet an alle Clients Nachrichten mit Informationen über die Clients mit kleinerem Rang, die Clients mit höherem Rang verbinden sich daraufhin jeweils mit allen Clients kleineren Ranges. Hier werden wieder Nachrichten mit Informationen zu IP-Adressen und Rechnerzugehörigkeit versandt. Nachdem alle Prozesse mit allen anderen verbunden sind, wird für Prozesse, die auf dem gleichen Rechner liegen, die Kommunikation mit Hilfe von gemeinsamen Speichersegmenten initialisiert. Verantwortlich hierfür ist jeweils der Prozess mit dem kleineren Rang. Für diese Prozesse werden die nicht mehr benötigten Socket-Verbindungen getrennt. Alle übrigen benutzen die etablierten Verbindungen, um über diese zukünftig MPI-Nachrichten zu versenden.

2.2 Windows-Sockets

Mit Windows-Sockets-2 bietet Microsoft die Möglichkeit, Nachrichten asynchron zu senden und zu empfangen. Ein Socket wird dabei als nicht-blockierend angelegt und bei einer Sende- oder Empfangs-Operation wird zusätzlich eine so genannte Overlapped-Struktur angegeben. Der Rückgabewert des Aufrufs gibt an, ob die Operation bereits zu Ende geführt wurde, oder sich noch in Ausführung befindet (pending). Anhand dieser Struktur hat der Programmierer die Möglichkeit zu überprüfen, in welchem Status sich die entsprechende Operation befindet. Dies kann beispielsweise durch die Funktion *WSAGetOverlappedResult* realisiert werden, die die Werte `OK`, `Pending` oder `Error` für *Operation beendet*, *Operation in Bearbeitung* oder *Operation wurde mit einem Fehler beendet* zurück liefert. Innerhalb der Overlapped-Struktur kann ein Event-Handler angegeben werden, der manuell abgeprüft werden kann. Alternativ zur Angabe eines Event-Handlers besteht die Möglichkeit beim Aufruf einer Overlapped-Operation einen Zeiger auf eine Funktion anzugeben, die nach der Komplettierung der Operation aufgerufen werden soll.

Außerdem können Events auch direkt an einen Socket gebunden werden. So muss beispielsweise nicht aktiv auf den Eingang einer Nachricht gewartet werden, sondern der Prozess kann blockieren und wird durch ein Event aufgeweckt, sobald eine Nachricht bereitsteht. Solche Konstrukte sind mit traditionellen BSD-Sockets nicht zu realisieren.

Trotz der Erweiterung um Overlapped-Kommunikation sind die Windows-Sockets kompatibel zu den BSD-Sockets. Jede Applikation, die BSD-Sockets verwendet, kann problemlos unter Windows compiliert werden. Zudem können Applikationen, die auf BSD-Sockets basieren, mit denen kommunizieren, die die Windows-Erweiterungen verwenden.

2.3 Die Send-Funktionalität

Da die Funktion MPI_Isend vom Standard als asynchrone Funktion definiert wird, kann die entsprechende Funktion in der Device Implementation durch einen nichtblockierenden Sendevorgang realisiert werden. Innerhalb von ch_wsock2 wird hier mit Hilfe der Overlapped-Funktionen gesendet, das heißt die Socket-Send Funktion puffert intern die Daten und blockiert nicht bis zum Abschluss des Sendevorgangs. Falls der Sendevorgang jedoch unverzüglich abgeschlossen werden konnte, beispielsweise beim Versand kleiner Nachrichten, wird dies mit Hilfe eines Flag in der entsprechenden MPI_Request-Struktur vermerkt. Ansonsten wird die Overlapped-Struktur der Sendeoperation in der MPI_Request-Struktur gespeichert.

Der Aufruf von MPI_Test überprüft, ob der Sendevorgang erfolgreich abgeschlossen wurde. Diese Funktion fragt zunächst das entsprechende Flag in MPI_Request ab. Ist dieses nicht gesetzt, überprüft sie mit Hilfe der gespeicherte Overlapped-Struktur, ob der Sendevorgang abgeschlossen wurde.

Die Funktion MPI_Send ist eine blockierende Funktion. Dennoch wird mit Hilfe der Overlapped Funktionen gesendet, also mittels eines nicht blockierenden Socket-Send innerhalb der Device Implementation. Allerdings wird auf höherer Ebene innerhalb des blockierenden MPI_Send-Aufrufs auf die Beendigung der Operation gewartet. Dabei wird die Windows-Funktion *WaitForMultipleObjects* verwendet, die es ermöglicht, auf mehrere Events zu warten. Es wird zum einen auf die Beendigung der angestoßenen Socket-Sendeoperation und zum anderen auf Events an den Sockets gewartet, die für die eingehende Nachrichten zuständig sind. Somit ist während des Sendens auch das gleichzeitige Empfangen von Nachrichten möglich.

2.4 Die Recv-Funktionalität

Die MPI_Irecv Funktion wurde analog zur MPI_Isend Funktion implementiert. Innerhalb des Device ch_wsock2 wird mit Hilfe der Overlapped-Funktionen ein asynchroner Socket-Receive Aufruf ausgeführt. Der sofortige Abschluss der Receive-Operation wird wiederum durch ein Flag in MPI_Request dargestellt. Zusätzlich wird innerhalb des Device für jeden Receive Aufruf eine sogenannte *ReceiveRequest* Struktur gespeichert. Innerhalb dieser Struktur wird durch ein Flag angezeigt, ob eine Nachricht vollständig empfangen wurde. Über Informationen in MPI_Request kann die zugehörige interne *ReceiveRequest* Struktur identifiziert werden.

Die Funktion MPI_Wait überprüft, ob eine Nachricht vollständig angekommen ist. Sie fragt das Device solange ab, bis dies der Fall ist (polling). Dabei wird nicht nur auf eine spezielle Nachricht gewartet, sondern es werden auch alle anderen Sockets auf eingehende Nachrichten überprüft. Falls eine Nachricht, für die ein MPI_Irecv Aufruf vorliegt, empfangen wurde, wird in der zugehörigen Struktur *ReceiveRequest* das Flag *is_complete* gesetzt. Sobald dieses Flag für die Nachricht, auf die gewartet wird, gesetzt ist, kehrt die Funktion MPI_Wait zurück.

Das blockierende MPI_Recv entspricht einem MPI_Irecv mit anschließendem MPI_Wait.

3. Benchmarks

MPI wurde für Aufgaben entwickelt, die große Mengen an Rechenleistung benötigen. Eine MPI-Implementierung muss sowohl die zur Verfügung stehende Rechenleistung wie auch die Band-

breite der Cluster-Interconnects so effizient wie möglich nutzen, da die Geschwindigkeit einer auf MPI basierenden Anwendung maßgeblich durch die Leistungsfähigkeit der MPI-Implementierung selbst bestimmt wird. Zusätzlich gewinnt die Performance einer MPI-Implementierung dadurch an Bedeutung, dass sie üblicherweise auf einer großen Anzahl von Rechnern eingesetzt wird und die durch nicht genutzte Rechenleistung entstehenden Mehrkosten mit der Anzahl der Rechner multipliziert werden.
Leistungsmessungen (Benchmarks) sind also unverzichtbar und für die Entwicklung und Nutzung einer MPI-Implementierung gleichermaßen von Bedeutung. Für die Bestimmung der Performance von NT-MPICH wurden die im Nachfolgenden kurz beschriebenen Pallas-Benchmarks [18] eingesetzt. Da die HPC Abteilung der Firma Pallas im September 2003 an die Firma Intel verkauft wurde, ist diese Version des Benchmark-Paketes inzwischen auch unter dem Namen *Intel MPI Benchmarks* [10] bekannt geworden.

3.1 Single-Transfer-Benchmarks

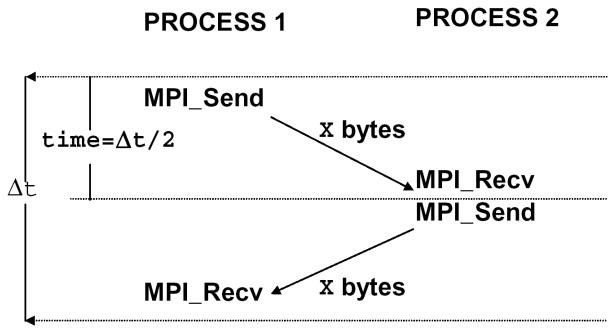

Abbildung 3. PingPong (Quelle: Intel ® MPI Benchmarks, Users Guide and Methodology Description)

Bei diesen Benchmarks wird eine einzelne Nachricht zwischen zwei Prozessen ausgetauscht. Im Unterschied zu den Collective-Transfer-Benchmarks werden keine Berechnungen ausgeführt; es wird also lediglich die Geschwindigkeit des Nachrichtenaustauschs und somit des jeweiligen Devices sowie der zugrunde liegenden Hardware gemessen. Die Kommunikation dieser Benchmarks beruht dabei auf den einfachen MPI-Funktionen `MPI_(I)Send`, `MPI_Recv` und `MPI_Wait`.

- PingPong – Misst die Zeit, die die Übertragung einer Nachricht dauert (siehe Abb. 3). Dies ist der klassische Test, um Durchsatz und Latenz bei Versand einzelner Nachrichten zwischen zwei Prozessen zu messen.

- PingPing – Zeit für die Übertragung einer Nachricht, während deren Versendung eine zweite Nachricht empfangen wird (siehe Abb. 4). Für diesen Test wird der asynchrone Nachrichtenversand mittels MPI-Isend eingesetzt.

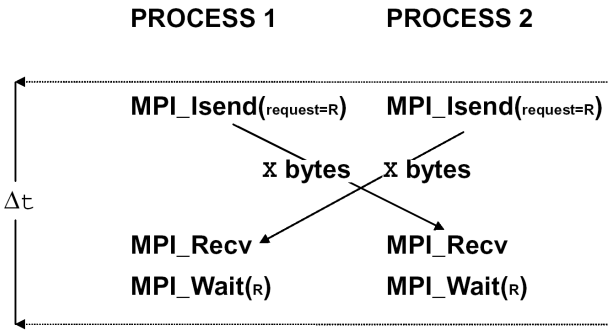

Abbildung 4. PingPing (Quelle: Intel ® MPI Benchmarks, Users Guide and Methodology Description)

3.1.1 Parallel-Transfer-Benchmarks

Bei dieser Art von Tests wird das Verhalten des Systems bei einfacher Kommunikation zwischen mehr als nur zwei beteiligten Knoten gemessen.

Diese Tests heißen:

- `Exchange` — jeder Prozess sendet eine Nachricht jeweils an seinen "linke" und "rechte" Nachbarn und empfängt entsprechend von diesen auch Nachrichten.
- `Sendrecv` — bei diesem Test wird mit der Funktion `MPI_Sendrecv` Nachrichten entlang einer periodischen Kette aller aktiven Prozesse gesendet.

3.1.2 Collective-Transfer-Benchmarks

Mit ihnen lassen sie die Eigenschaften von den globalen, bzw. kollektiven MPI-Funktionen genauer untersuchen. Dabei können folgende Tests durchgeführt werden, die entsprechend der von ihnen verwendeten Funktionen benannt sind:

- `Reduce`, `Reduce_scatter`
- `Allreduce`, `Allgather(v)`, `Alltoall`
- `Bcast`, `Barrier`

Wie in Abbildung 2 zu sehen, ist das Device ein Teil der MPID-Schicht. Die kollektiven Operationen sind in der MPIR-Schicht implementiert, ein Benchmark dieser Operationen hängt daher auch von der Effizienz deren Implementierung ab.

3.2 Benchmark-Ergebnisse

Die Benchmarks wurden auf dem Xeon-Cluster (CLOX) des Lehrstuhls für Betriebssysteme gestartet. Es handelt sich dabei um acht Doppelprozessor Intel Xeon 2.4 GHz Maschinen mit 512

kByte Cache und 1 GByte Hauptspeicher. Die Rechner sind sowohl über einen SCI-2D-Torus als auch über Fast- und Gigabit-Ethernet verbunden.

3.2.1 Vergleich MPI-Implementationen

Im Folgenden wird NT-MPICH mit folgenden drei anderen MPI-Implementationen für Windows verglichen: MPICH.NT [11], WMPI-II [3] und MPI/Pro [22].

- MPICH.NT ist wie NT-MPICH eine freiverfügbare, auf MPICH basierende MPI-Bibliothek für Windows. Man kann sich die Frage stellen, warum es überhaupt zwei Portierungen von MPICH auf Windowsplattformen gibt. Die Antwort ist, dass obwohl NT-MPICH deutlich vor MPICH.NT entwickelt wurde, das ursprüngliche MPICH von ANL ab der Version 1.2 auch Windows als Plattform zu unterstützen begann, wobei dieser Teil der Bibliothek dann als MPICH.NT bekannt wurde. Im Gegensatz zu NT-MPICH unterstützt MPICH.NT nicht die MPI-Erweiterung MPE im vollen Umfang und da MPICH.NT nicht mehr weiterentwickelt wird (ANL verweist hier auf die Windows-Portierung von MPICH2), wird sich das auch nicht mehr ändern. Als Kommunikationsmedien wird von MPICH.NT nur TCP und Shared-Memory unterstützt, wobei hier die erreichbaren Leistungen deutlich hinter NT-MPICH liegen, wie unsere Benchmark-Ergebnisse zeigen werden.

- MPI/Pro ist eine kommerzielle MPI-Bibliothek der Firma *Verari Systems*, die den vollen MPI-2-Standard sowohl für Linux-, Windows-, als auch Mac-OS-Systeme unterstützt. Als Kommunikationsmedien können mit MPI/Pro neben jeglichen TCP-basierten Netzen und Shared-Memory sowohl Myrinet, als auch InfiniBand zum Einsatz kommen. Weitere erwähnenswerte Eigenschaften von MPI/Pro sind die IMPI[3]-Konformität, sowie die garantierte Thread-Sicherheit, so dass mit MPI/Pro auch hybride Parallelisierungsstrategien z.B. mit OpenMP möglich sind.

- WMPI-II ist ebenfalls eine kommerzielle MPI-Implementierung, welche von der Firma *Critical Software* speziell für Windowsplattformen entwickelt wurde, und ebenfalls den vollen MPI-2-Standard unterstützt und völlige Thread-Sicherheit garantiert. WMPI-II stützt sich dabei wie NT-MPICH auf die speziellen Windows-API-Funktionalitäten, wie z.B. die vollständig asynchronen Socket-Kommunikationsfunktionen, wobei aber Kommunikation nur über TCP und Shared-Memory angeboten wird. Trotz der Orientierung auf Windows-Systemen ist von Critical-Software inzwischen auch eine Portierung von WMPI-II auf Linux-Systeme erhältlich.

Die im Folgenden mit den verschiedenen MPI-Implementationen durchgeführten Single-Transfer-Benchmarks messen jeweils nur die Ethernet-Werte, eine Messung der Shared-Memory Kommunikation wird nicht durchgeführt. Beim PingPong Test in Abbildung 5 sind die Latenzen aller Implementationen vergleichbar. Die Bandbreiten von WMPI-II und MPI/Pro sind etwas höher als von NT-MPICH. Das nicht-kommerzielle MPICH.NT zeigt eine deutlich niedrigerer Bandbreite für große Nachrichten. Im Vergleich dazu schneidet NT-MPICH beim PingPing Test in Abbildung 6 besser ab. Die Latenzen sind wieder ähnlich, die Bandbreite von NT-MPICH ist hier höher, als beim kommerziellen WMPI-II. Der PingPing Test untersucht das Verhalten bei gestörter

[3]Interoperable Message Passing Interface [5]

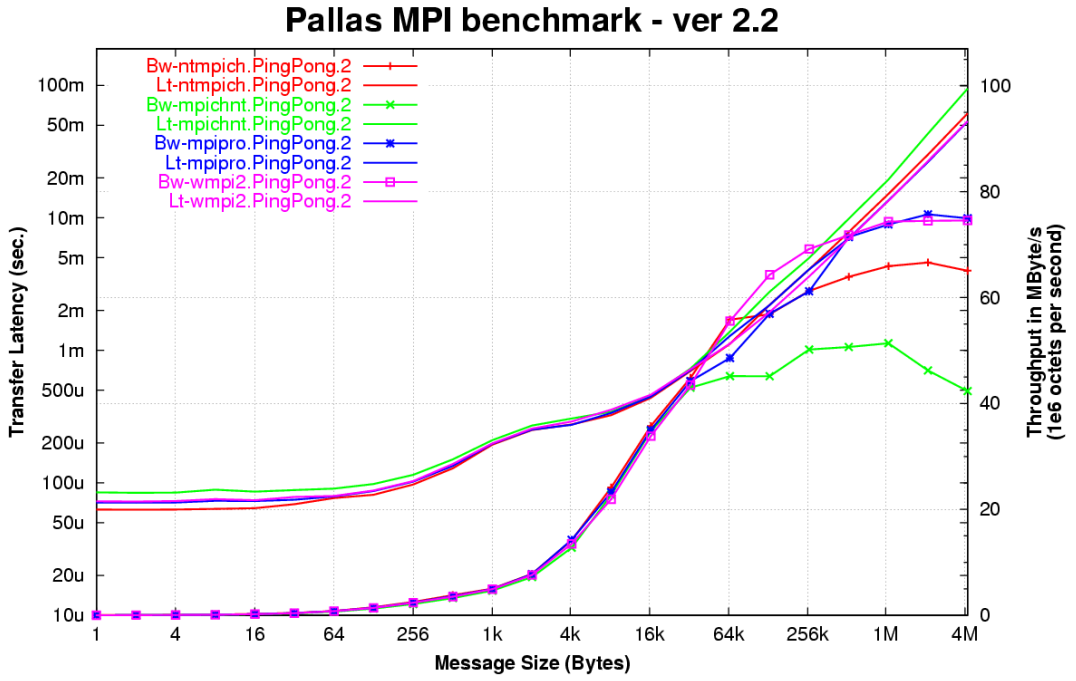

Abbildung 5. PingPong mit NT-MPICH, MPICH.NT 1.2.5, MPI/Pro 1.6.4.1, WMPI II 2.4.0

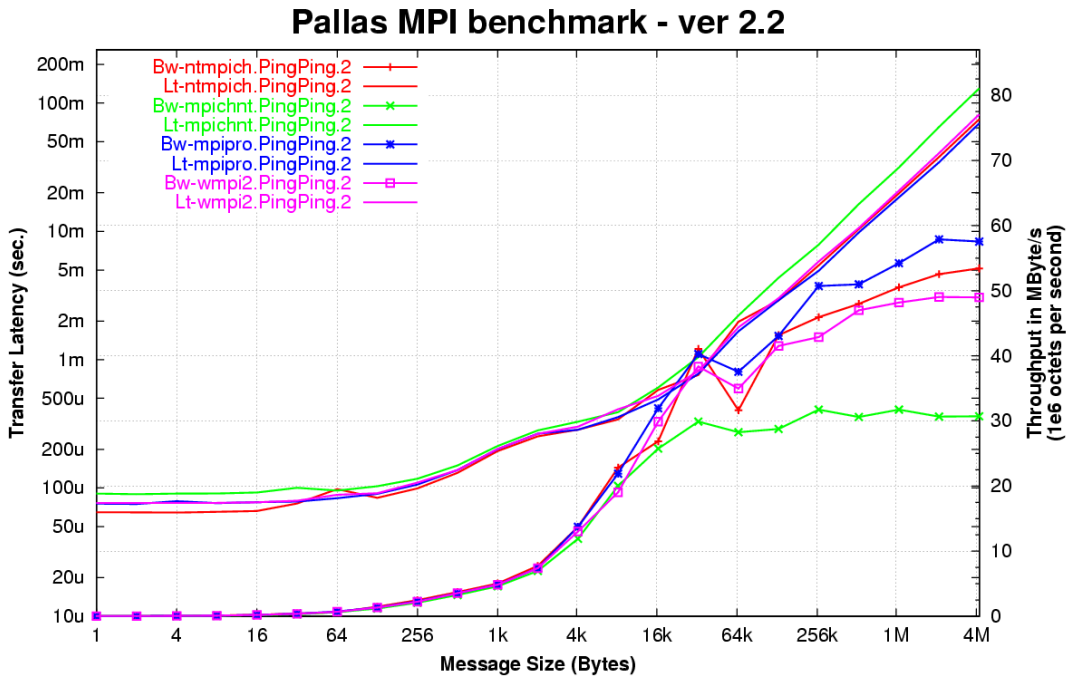

Abbildung 6. PingPing mit NT-MPICH, MPICH.NT 1.2.5, MPI/Pro 1.6.4.1, WMPI II 2.4.0

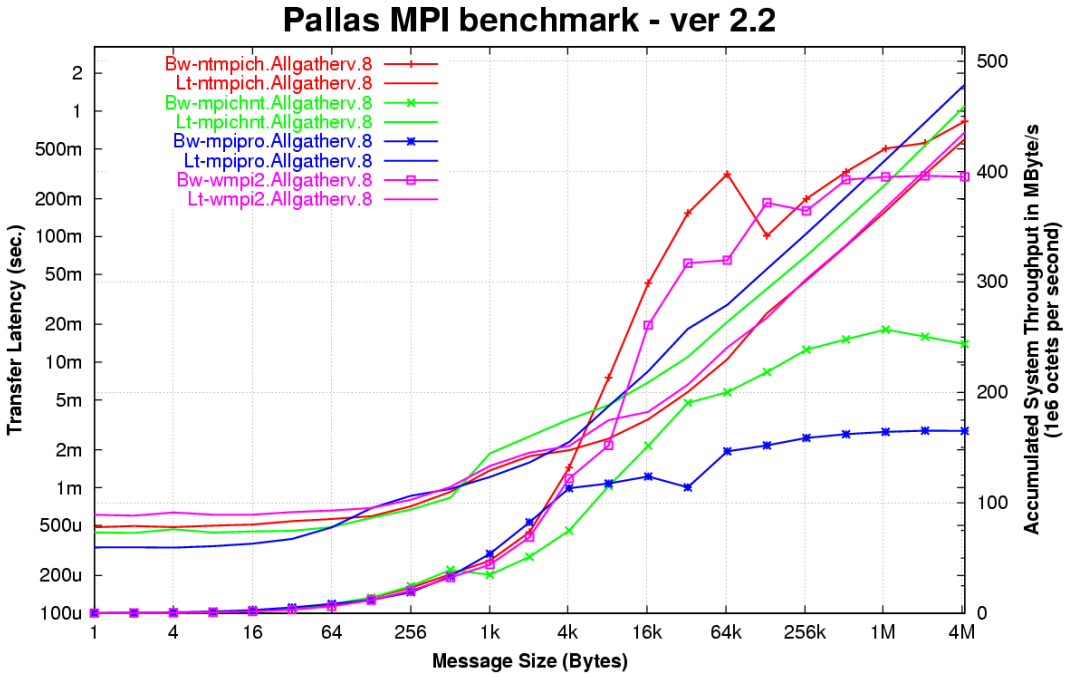

Abbildung 7. Allgatherv mit NT-MPICH, MPICH.NT 1.2.5, MPI/Pro 1.6.4.1, WMPI II 2.4.0

Kommunikation, das heißt, die Prozesse senden gleichzeitig asynchron und empfangen sofort die Nachricht des Anderen. Hier zeigt die ch_wsock2 Implementation ihre Stärke, die Bandbreiteneinbußen gegenüber PingPong sind bei NT-MPICH geringer, als bei WMPI-II und MPICH.NT, und gleich dem kommerziellen MPI/Pro.

Um den Vergleich der Implementationen bei mehr als zwei Prozessen zu haben, wurde die kollektive Operation *Allgatherv* ausgewählt. Die gemessenen Ergebnisse sind in Abbildung 7 gezeigt. Hier wird allerdings nicht nur die Effizienz des Device bewertet, sondern auch die Güte der Implementation der jeweiligen kollektiven Operation.

3.2.2 Effizienz der ch_wsock2 Implementation

In Abbildung 8 ist der Vergleich zwischen reiner Ethernet-Kommunikation und mit Shared-Memory kombinierter Kommunikation aufgetragen. Um diesen Fall zu messen, können keine Single-Transfer-Benchmarks eingesetzt werden, da diese Punkt-zu-Punkt Verbindungen messen und daher die gleichzeitige Verwendung von zwei Devices nicht unterstützen. Aus diesem Grund wurde für diesen Vergleich der kollektive Allgatherv-Benchmark ausgewählt. Im Fall „8on8“ wurden 8 Prozesse auf 8 Clusterknoten gestartet, d.h. es existiert ein Prozess pro Knoten und die Kommunikation erfolgt ausschließlich über Ethernet. Bei „8on4“ liegen 8 Prozesse auf 4 Knoten, d.h. pro Knoten gibt es zwei Prozesse, welche mittels Shared-Memory kommunizieren. Wie in der Abbildung 8 zu sehen ist, ist die akkumulierte Bandbreite bei zwei Prozessen pro Rechner fast so hoch, wie bei einem Prozess pro Rechner.

Der Hauptvorteil der Verwendung von Overlapped-Sockets liegt bei ch_wsock darin, dass die blockierenden MPI-Operationen intern asynchron realisiert sind. Um solche Besonderheiten zu

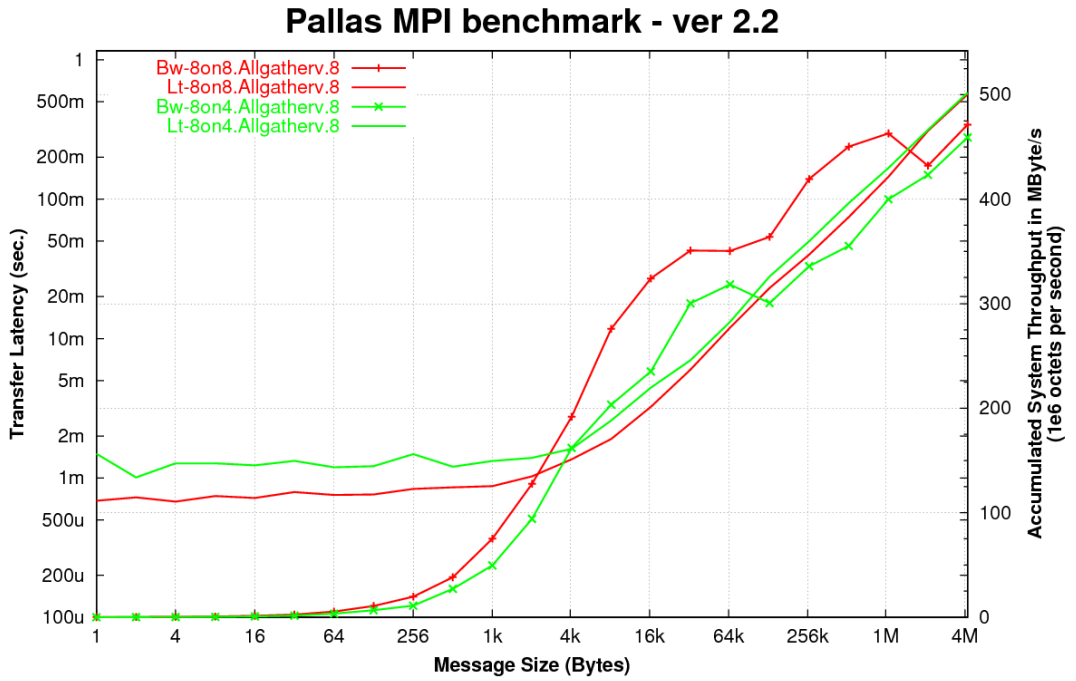

Abbildung 8. Vergleich reiner Ethernet-Kommunikation mit der Kombination Ethernet / Shared Memory

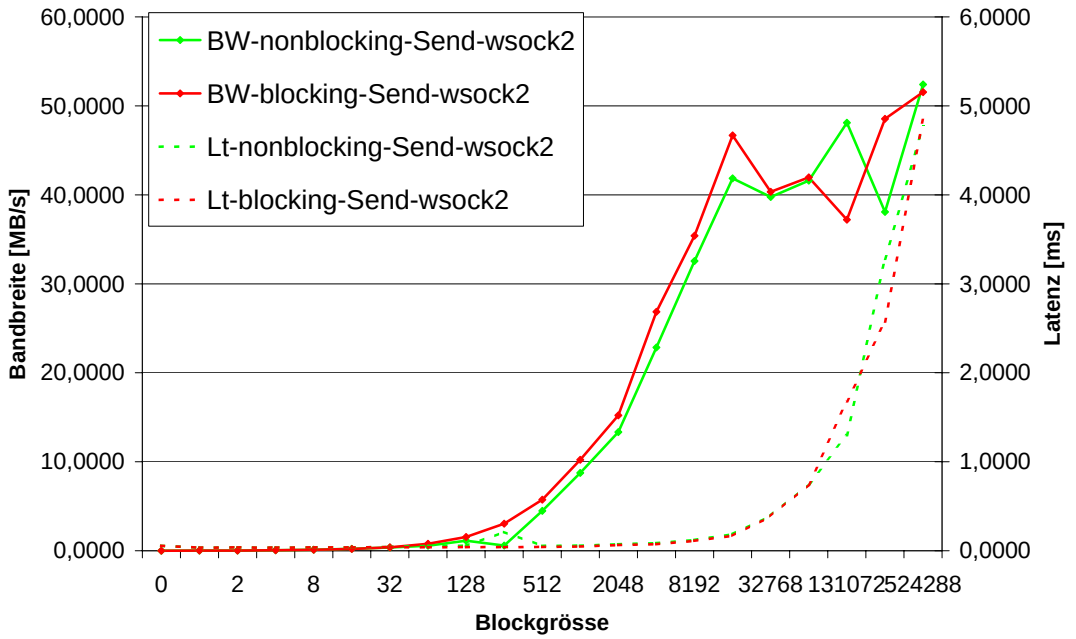

Abbildung 9. Vergleich blockierendes und nicht blockierendes PingPing mit ch_wsock2

visualisieren sind die Pallas-Benchmarks PingPong und PingPing nicht geeignet. PingPing verwendet MPI_Isend und bei PingPong erfolgen Versand und Empfang der Nachrichten in optimaler Reihenfolge. Daher haben wir den PingPing Benchmark einzeln implementiert, einmal nach Vorgabe: MPI_Isend, MPI_Recv, MPI_Wait. Um den direkten Vergleich mit den synchronen Operationen zu haben, wurde das asynchrone MPI_Isend durch MPI_Send ersetzt. Beide Partner führen beim synchronen PingPong erst gleichzeitig MPI_Send aus und dann gleichzeitig MPI_Recv. Für diesen Test wurde das Rendevouz-Protokoll für große Nachrichten abgeschaltet und generell das Eager-Protokoll eingesetzt, damit es nicht zu Verklemmungen kommt. Der Vergleich zwischen synchronem PingPong mit MPI_Send und asynchronem PingPong mit MPI_Isend ist in Abbildung 9 gezeigt. Wie hier zu sehen ist, sind die Bandbreiten bei beiden Tests vergleichbar. Die explizite Verwendung einer asynchronen Send-Operation bietet hier also keine Vorteile gegenüber der Verwendung eines synchronen Send, welches intern asynchron realisiert ist.

4. Linux Interoperabilität

Da am Lehrstuhl für Betriebssysteme bisher kein eigenes Ethernet-Device für Linux existierte, kam die Frage auf, ob ch_wsock2 nicht auf Linux zu portieren wäre. Durch die am Lehrstuhl entwickelte Bibliothek NT2Unix [16] standen bereits viele Portierungen von Windows-API Aufrufen auf Linux zur Verfügung. Einige benötigte Spezialfälle wie Overlapped I/O waren in der ursprünglichen NT2Unix nicht vorhanden. Für das Projekt ch_usock, also das Device ch_wsock portiert auf Unix Sockets, wurden die benötigten API-Funktionen ergänzt. Um die Portierung zu vereinfachen unterstützt das Device ch_usock keine Shared-Memory Kommunikation auf SMP-Maschinen. Bei der Portierung wurden zusätzlich einige im Device ch_wsock2 gemachten Optimierungen weggelassen, somit waren diese Devices nicht mehr identisch. Da am Lehrstuhl für Betriebssysteme in der Regel für verschiedenen Plattformen gleichzeitig entwickelt wird, so sind die Quelltexte für das SCI-Device unter Windows und Linux/Solaris in großen Teilen identisch, war die Existenz zweier unterschiedlicher Devices für Linux und Windows nicht optimal. Von daher wurde der ch_usock Quellcode wieder auf Windows zurückportiert, was mit wenig Aufwand möglich war. Innerhalb des Devices ch_usock wurden weiterhin Windows Aufrufe verwendet, da die Abbildung der Windows-API Aufrufe auf Linux System Calls in der Bibliothek NT2Unix vorgenommen werden. Somit waren die selben Quelltexte sowohl unter Linux als unter Windows die Grundlage für das Ethernet Device ch_usock. Durch die Kompatibilität zwischen Windows Sockets und Berkeley Sockets kann dieses Device auch in *Mixed-Konfigurationen* eingesetzt werden. Das heißt, die MPI-Applikation wird einmal unter Linux compiliert und einmal unter Windows. Als MPI-Unterstützung wird unter jedem Betriebssystem jeweils die passende MPI-Bibliothek mit dem Device ch_usock eingesetzt. Die Prozesse unter Windows und Linux können

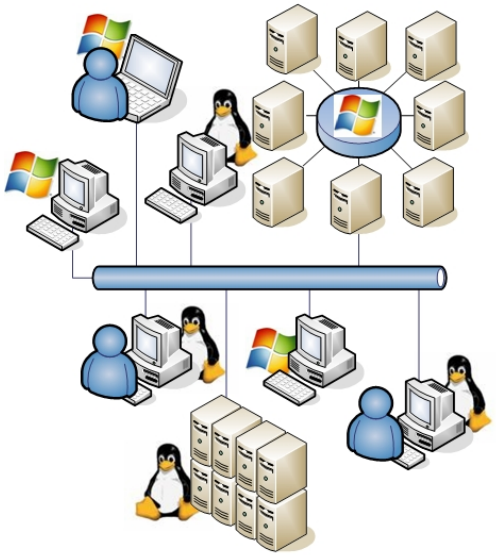

Abbildung 10. Heterogenität im Office Grid

nun miteinander kommunizieren und bilden eine gemischte MPI-Applikation. Diese so genannte Mixed-Konfiguration beinhaltet also kommunizierende MPI-Prozesse auf Linux und Windows gleichzeitig. Damit ist MP-MPICH auch in heterogenen *Office Grids* (siehe Abbildung 10) einsetzbar, bei denen Arbeitsplatzrechner und/oder Clusterknoten nicht nur mit Windows, sondern auch mit Linux verfügbar sind.

4.1 Vergleich ch_usock / ch_wsock2

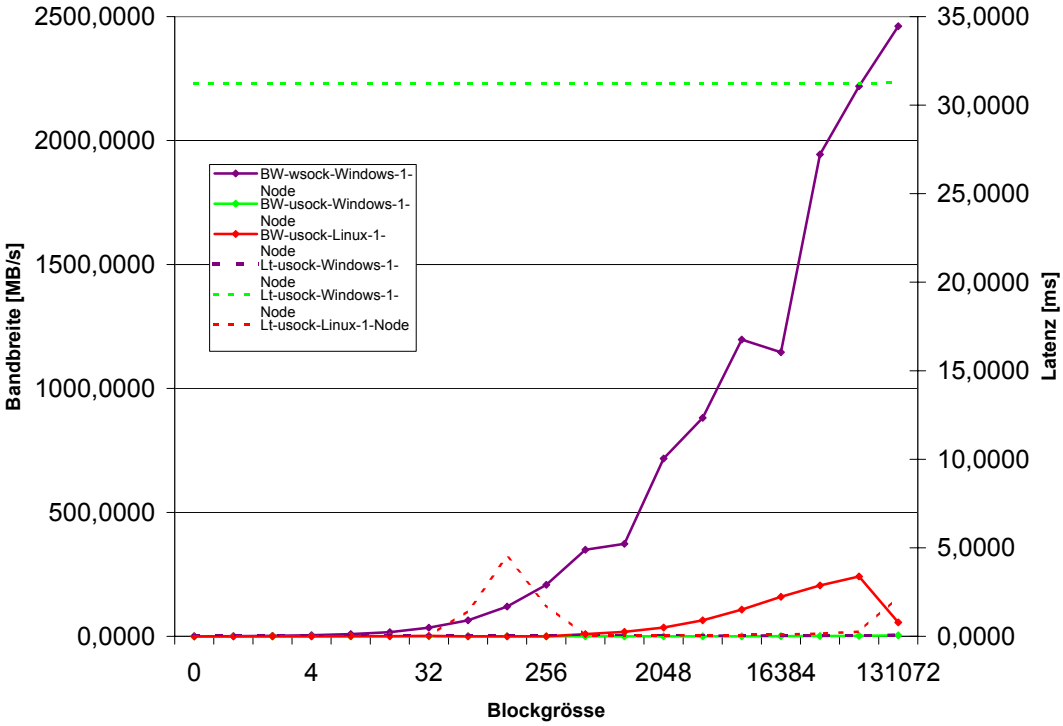

Abbildung 11. Vergleich lokale Kommunikation (PingPong)

In Abbildung 11 ist der Vergleich der Bandbreiten und Latenzen bei lokaler Kommunikation aufgezeigt. Dazu wurden je zwei PingPong-Prozesse auf einem Clusterrechner gestartet, unter Linux mit dem Device ch_usock und unter Windows zum einen mit ch_usock und zum anderen mit ch_wsock2. Wie deutlich zu sehen ist, ist die Bandbreite bei ch_wsock2 unter Windows mit Shared-Memory Kommunikation sehr hoch. Mit 2,5 GByte pro Sekunde bei grossen Paketen und minimalen Latenzen ist die Shared-Memory Unterstützung von ch_wsock2 sehr effizient implementiert.

Insbesondere im direkten Vergleich mit ch_usock unter Windows ist zu sehen, dass die Unterstützung dieser Kommunikationsform unter Windows erforderlich ist. Da ch_usock nur TCP/IP benutzt wird hier zur Kommunikation zwischen Prozessen auf einem Rechner die IP-Adresse 127.0.0.1 benutzt und somit das so genannte *Loopback-Device* des Rechners eingesetzt. Dieses Device lieferte im PingPong Benchmark eine Bandbreite von maximal 4 MByte pro Sekunde und

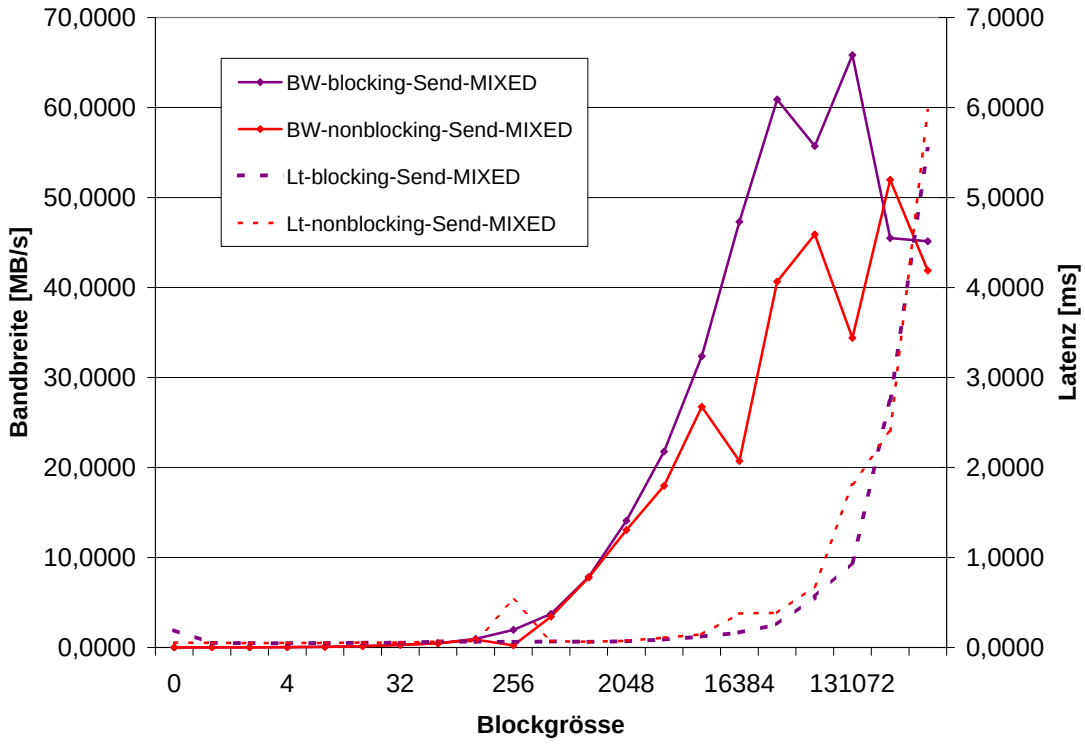

Abbildung 12. PingPing 1 Linux und 1 Windows Rechner

Latenzen von über 30 ms und ist damit in modernen Systemen zur effizienten Prozesskommunikation faktisch nicht einsetzbar.

Im Vergleich dazu ist das Loopback-Device unter Linux wesentlich effektiver implementiert als unter Windows. Der PingPong Test auf einem Linux-Rechner mit ch_usock ergibt eine Bandbreite von bis zu 250 MByte pro Sekunde und Latenzen ähnlich denen unter Windows bei Einsatz von gemeinsamem Speicher.

Wie dieser Benchmark zeigt, wäre die Unterstützung von Shared-Memory Kommunikation zusätzlich zur Kommunikation über Ethernet ist in Linux Systemen sinnvoll, in Windows Systemen ist diese aber absolut notwendig.

Abbildung 12 illustriert den in Kapitel 3.2.2 beschriebenen Vergleich zwischen PingPing mit blockierendem MPI_Send und nicht blockierendem MPI_Isend. Der Benchmark lief dabei mit einem Prozess auf einem Linux Rechner und einem Prozess auf einem Windows Rechner. Es wurde hier nur das Eager-Protokoll verwendet. Wie zu sehen ist, führt die effiziente Implementation des blockierenden MPI_Send dazu, dass hier die Bandbreite höher ist, als bei Verwendung der asynchronen Send Funktion. Auf Windows-Seite wird dies durch die Verwendung der Overlapped-Option der Windows Sockets erreicht. Auf Linux Seite wurde diese Overlapped-Funktionalität in der Bibliothek NT2Unix effizient durch die Verwendung von Threads nachgebildet.

5. Zusammenfassung und Ausblick

Mit ch_wsock2 wurde ein schnelles TCP/IP Device für Windows implementiert. Als besonderes Feature bietet ch_wsock2 zusätzlich zur Kommunikation über Ethernet die Möglichkeit, auf Multiprozessor oder Multicore Rechnern mittels gemeinsamen Speichers zu kommunizieren.

Das neue Device ch_usock bietet nicht nur die Möglichkeit schneller Ethernet-Kommunikation auf Linux, sondern auch in der gemischten Kombination zusammen mit Windows-Rechnern.

Bis jetzt implementiert ch_usock allerdings noch nicht die komplette MPI-API. Hier wird also zukünftig noch eine Erweiterung der Schnittstelle erforderlich sein.

Die Kombination von Ethernet und Shared Memory Kommunikation zusammen in einem Device bringt jedoch auch Nachteile mit sich. So können beispielsweise kollektive Operationen nicht im Hinblick auf eine heterogenen Kommunikationsstruktur optimiert werden. Beispielsweise wäre es bei der Kopplung zweier SMP-Maschinen vorteilhaft, wenn bei der Bildung eines Minimalwertes über alle Prozesse erst die lokalen Minima auf den SMP Maschinen gebildet würden und erst danach über die langsame Ethernet-Verbindung. Für eine solche Optimierung müssten jedoch auf höherer Ebene Informationen über die Verbindungen zwischen den Prozessen zur Verfügung stehen. Eine generelle Multi-Device-Struktur würde dies ermöglichen. Das bedeutet, dass innerhalb der MPID Schicht zwei getrennte Devices für Ethernet und gemeinsamen Speicher existieren und dort Informationen über das verwendetet Device zwischen zwei Prozessen gespeichert werden. Für diesen Ansatz ist ch_usock die ideale Grundlage für ein generelles Ethernet Device. Zusätzlich zu ch_usock würde der Applikation ein komplettes Shared-Memory Device zur Verfügung stehen.

Literatur

[1] Chair for Operating Systems, RWTH-Aachen, University. *MP-MPICH – User Documentation & Technical Notes*.

[2] M. T. Chapman. The Benefits of Dual-Core Processors in High-Performance Computing. White Paper, June 2005.

[3] criticalsoftware. WMPI. http://www.criticalsoftware.com/solutions/products/wmpi/index.html.

[4] M. de Prycker. *Asynchonous Transfer Mode*. Prentice Hall Verlag GmbH, 1996.

[5] W. George, J. Hagedorn, and J. Devaney. Status report on the development of the interoperable mpi protocol. In A. Skjellum, P. V. Bangalore, and Y. S. Dandass, editors, *Proceedings Third MPI Developer's and User's Conference*. MPI Software Technology Press, march 1999.

[6] W. D. Gropp and E. Lusk. *User's Guide for* `mpich`*, a Portable Implementation of MPI*. Mathematics and Computer Science Division, Argonne National Laboratory, 1996. ANL-96/6.

[7] IEEE. *ANSI/IEEE Std. 1596-1992, Scalable Coherent Interface (SCI)*, 1992.

[8] InfiniBandSM Trade Association. *InfiniBandTM Architecture Specification*, October 2004. Volume 1 - General Specifications.

[9] InfiniBandSM Trade Association. *InfiniBandTM Architecture Specification*, October 2004. Volume 2 - Physical Specifications.

[10] Intel GmbH. Intel ® MPI Benchmarks - Users Guide and Methodolgy Description. Technical report, Intel GmbH, Hermülheimer Str. 10, D-50321 Brühl, Germany, Nov. 2004.

[11] A. N. Laboratory. MPICH.NT. http://www-unix.mcs.anl.gov/mpi/mpich/mpich-nt/.

[12] M. Pöppe and J. Worringen. *Meta-MPICH, user documentation & technical notes*. Lehrstuhl für Betriebssysteme, RWTH Aachen, 2002.

[13] Microsoft. *Windows Sockets 2 Application Programming Interface, An Interface for Transparent Network Programming Under Microsoft Windows*. Manual.

[14] MPI Forum. MPI: A message-passing interface standard. *International Journal of Supercomputing Applications*, 1994.

[15] M. P. I. F. MPIF. MPI-2: Extensions to the Message-Passing Interface. Technical Report, University of Tennessee, Knoxville, 1996.

[16] S. M. Paas, T. Bemmerl, and K. Scholtyssik. Win32 API Emulation on UNIX for Software DSM. In *Proceedings of the 2nd USENIX Windows NT Symposium*, pages 39–46, Seattle, Washington, USA, August 1998.

[17] S. M. Paas, M. Dormanns, T. Bemmerl, K. Scholtyssik, and S. Lankes. Computing on a cluster of pcs – project overview and early experiences. In *Proceedings of the 1st Workshop Cluster-Computing*, pages 217–229, TU Chemnitz-Zwickau, November 1997.

[18] Pallas GmbH. Pallas MPI Benchmarks - PMB, Part MPI-1. Technical report, Pallas GmbH, Hermülheimer Str. 10, D-50321 Brühl, Germany, 2000.

[19] M. Pöppe, S. Schuch, and T. Bemmerl. A Message Passing Interface Library for Inhomogeneous Coupled Clusters. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS 2003), Workshop on Communication Architecture for Clusters (CAC 2003)*, Nice, France, April 2003.

[20] R. Ramanathan. Intel Multi-core Processors: Leading the Next Digital Revolution. White Paper, 2005.

[21] F. Seifert, J. Worringen, and W. Rehm. Using arbitrary memory regions for sci communication. In *Proceeding of SCI Europe 2001*, pages 59–64, Dublin, Ireland, October 2001.

[22] verari. MPI/Pro. http://www.verari.com/mpi.asp.

[23] VITA Standards Organization. *ANSI Standard ANSI/VITA 26-1998, American National Standard for Myrinet-on-VME Protocol Specification*, 1998.

[24] J. Worringen. SCI-MPICH: The Second Generation. In *SCI Europe 2000*, pages 10–20, 2000.

[25] J. Worringen, F. Seifert, and T. Bemmerl. Efficient asynchronous message passing via sci with zero-copying. In *Proceeding of SCI Europe 2001*, pages 65–74, Dublin, Ireland, October 2001.