Sicherheit und Effizienz in einer Active-Message-Kommunikationsschicht

Michael Eberl, Hermann Hellwagner, Wolfgang Karl, Markus Leberecht

Lehr– und Forschungseinheit Informatik X
Rechnertechnik und -organisation / Parallelrechnerarchitektur (LRR-TUM)
Institut für Informatik der Technischen Universität München
D-80290 München, Germany
phone: +49-89-289-25357, fax: +49-89-289-28232
email: leberech@informatik.tu-muenchen.de

Zusammenfassung

Active Messages haben sich als effizientes Kommunikationsverfahren insbesondere auf Kommunikationstechnologien durchgesetzt, die einen direkten Zugriff des Benutzers ohne Intervention des Betriebssystems zulassen. Als Nachteil der leichtgewichtigen Kommunikation erwiesen sich jedoch die nicht ausreichenden Schutzmechanismen, vor allem bei der Verwendung mehrerer Prozesse, die sich gleichzeitig einer Active-Message-Bibliothek bedienen. Die Spezifikation 2.0 der Berkeley-Active-Messages unternimmt nun den Versuch, für das bekannte und schnelle Kommunikationsverfahren Schutzabstraktionen vorzusehen. Im Rahmen dieser Arbeit wird die Implementation eines solchen Active-Message-Layers der Version 2.0 auf einem Cluster von SCI-gekoppelten Arbeitsplatzrechnern beschrieben. Wir können zeigen, daß die zusätzlichen Schutzmechanismen nur wenig Einfluß auf die Leistung haben und somit der Vorteil der Active Messages, die leichtgewichtige, feingranulare Kommunikation, erhalten bleibt.

1 Einführung und Motivation

Die Kommunikation innerhalb von Parallelrechnern basiert auf dem Austausch von Nachrichten. Kommunikationsbibliotheken wie PVM oder der MPI-Standard tragen diesem Umstand Rechnung, indem sie Programmierschnittstellen zur Verfügung stellen, denen die vereinheitlichte Abstraktion eines Nachrichten verschickenden und Nachrichten empfangenden parallelen Rechners zugrunde liegt. Sie stellen somit eine der wenigen standardisierten Infrastrukturen im Bereich paralleler Programmierung dar.

Der Nachteil dieser Konzepte, die sich auch auf Netzwerken von Arbeitsplatzrechnern (Networks of Workstations, NOW) umsetzen lassen, besteht häufig in ungenügenden Leistungsdaten: Während lange Nachrichten in der Regel mit einem nahe an der physikalischen Bandbreite des Mediums bzw. der Protokolle liegenden Durchsatz verschickt werden können, benötigen

kurze Nachrichten nicht selten überlange Aufsetzzeiten, so daß die Latenz von Anwendung zu Anwendung im Zweifelsfall unökonomisch hoch wird.

Diesen Umstand näher betrachtend und die Gründe dafür suchend, entwickelten David Culler et al. [CKK+94] das Konzept der *Active Messages*. Üblicherweise wird bei asynchronen nachrichtenorientierten Kommunikationsmodellen der Empfang einer Nachricht durch eine Kopieroperation aus einem Puffer der Kommunikationsschicht in den Puffer der Anwendung realisiert, um später der Anwendung die Benutzung der Nachricht zu überlassen. Das Konzept der Active Message koppelt diese beiden Aktionen, um so den Aufwand der Kopieroperationen einsparen zu können.

Eine *Active Message* besteht daher außer aus der eigentlichen Nachricht noch aus einem Funktionszeiger, der auf den sogenannten *Active Message Handler* des Empfängers verweist. Dieser Handler, eine kurze, nichtblockierende Funktion der Anwendung dient zur Behandlung der angekommenen Nachricht. Der Empfang einer Active Message besteht lediglich aus dem Aufruf des passenden Message Handlers mit den Daten der Nachricht als Argument. Nach Terminierung der Handlerfunktion kann damit die Nachricht als empfangen und bearbeitet gelten, so daß der entsprechende Puffer weiteren Nachrichten zur Verfügung steht.

2 Das Active-Message-Konzept

Wie von Eicken, Culler et al. in [vECGS92] berichten, benötigen echt synchrone Datenübertragungsprotokolle stets ein Drei-Phasen-Schema, welches eine Überlappung von Kommunikation und Berechnung vom Prinzip her ausschließt und die Netzwerklatenz im Programmablauf sichtbar macht.

Asynchrone Datenübertragungsverfahren hingegen erlauben es der Datenquelle, einen Sendebefehl ohne Verzögerung abzusetzen, während hierzu jedoch sowohl beim Sender als auch beim Empfänger Pufferspeicher und damit zusammenhängende Kopieroperationen benötigt werden.

Durch diese Nachteile ergeben sich bei vielen Kommunikationsbibliotheken unnötig hohe Nachrichtenaufsetz-, Empfangs- und -sendezeiten. Durch Active Messages, die im Gegensatz zu den herkömmlichen Verfahren nur das nötige Minimum an Pufferspeicher und Kopieroperationen benötigen, läßt sich auf vielen Hardwarearchitekturen der Kommunikationsaufwand um nahezu eine Größenordnung reduzieren.

Als Nachteil dieses Verfahren kann jedoch der mangelnde Schutz einer möglichen Active-Message-Kommunikationsbibliothek gelten. Durch die Benutzung von Funktionszeigern zur Beschreibung von Handlerfunktionen sind gegen eine fälschliche Verwendung der Nachrichten keinerlei Schutzmaßnahmen vorgesehen. Hinzu kommt, daß das ursprüngliche Modell der Active Messages von einer SPMD-Ausführung der sie benutzenden Programme ausgeht. Mehrbenutzerverwendung des Verfahrens scheint auch wegen der engen Verzahnung mit den Hardwaremechanismen nicht angebracht.

2.1 User-Level-Kommunikation und SCI

Ein Unterschied existierender Active-Message-Implementationen, der diese in zwei Klassen einteilt, liegt im Zugriffsverfahren auf das Kommunikationsmedium. In den meisten Fällen muß auf die gemeinsame Ressource „Netzwerk“ bzw. „Netzwerkschnittstelle“ zur Wahrung von Schutzabstraktionen über den Betriebssystemkern zugegriffen werden. Dieser Wechsel in das Betriebssystem erfordert zusätzliche Zeit und verlangt erneut aufgrund der Organisationsstruktur vieler Betriebssysteme ein Umkopieren von Nachrichtendaten. Daher existieren unterschiedliche Ansätze, Netzwerkschnittstellen direkt in Form von Programmierbibliotheken und spezieller Hardware dem Programmierer zur Verfügung zu stellen. [vEBBV95] [BDFL96]

Netzwerkhardware, die unmittelbar vom Benutzer aus ansprechbar ist, bietet hier somit einen weiteren Vorteil und eine weitere Verringerung der Latenzzeiten. Ihr Nachteil liegt allerdings in den in der Regel nur unzureichend vorhandenen Schutzmechanismen und der nicht vorhandenen Ressourcenverwaltung für nebenläufige Prozesse mit jeweils eigenen Kommunikationsanforderungen.

Ausgehend von einer Maschinenorganisation, die entweder direkten Zugriff auf die Kommunikationshardware zuläßt oder verbietet, unterscheidet von Eicken somit zwei Formen seiner eigenen Active-Message-Version U-Net [vEBBV95], eine Zero-Copy- und eine True-Zero-Copy-Version. Erstere benötigt weiterhin eine einzelne Kopie auf dem Wege zwischen Applikation und Netzwerkinterface, während im True-Zero-Copy-Fall direkt aus der Anwendung in die Anwendung übertragen werden kann, hierfür allerdings Benutzerzugriffe auf die Netzwerkschnittstelle vonnöten sind.

Im Rahmen des SMiLE-Projektes (Shared Memory in a LAN-like Environment [KH95] [HKL97] [AHKL96]) am Lehrstuhl für Rechnertechnik und Rechnerorganisation der TU München wird ein Netzwerk von PCs aufgebaut, welches sich mit Hilfe einer selbstentworfenen PCI-SCI-Brückenkarte der modernen Kommunikationstechnologie des Scalable Coherent Interface (SCI [Soc93]) bedient, um damit einen Rechner mit verteiltem gemeinsamen Speicher zu verwirklichen. Da SCI die Dienste eines Busses (Lese-, Schreib- und atomare Transaktionen, ggf. auch cachekohärent) auf einer schnellen und skalierbaren Verbindungstechnologie zur Verfügung stellt (1 GBit/s bis 1 GByte auf unterschiedlichen Medien), sind Zugriffe auf entfernten Speicher aus einer Anwendung heraus ohne Softwareintervention möglich. Die daraus sich ergebenden geringen Latenzen zeigen deutlich das Potential des direkten Zugriffes auf die Netzwerkhardware.

2.2 Erweiterungen in der Active Message Specification 2.0

Active Messages in ihrer ursprünglichen Form waren nicht zur Benutzung durch den Anwendungsprogrammierer vorgesehen, sondern sollten Betriebssystemen, Kommunikationsbibliotheken und Compilern als einfacher gemeinsamer Mechanismus zur Verfügung stehen. Dennoch setzte sich dieses Kommunikationsschema auch für den Parallelprogrammierer durch. Dadurch reichten verschiedene Punkte der ursprünglichen Definition der Active Messages nicht mehr aus, so z. B. die Namens- und Schutzmodelle, die bislang von reinen SPMD-Anwendungen ausgegangen waren, Fehlermodelle, sowie die mangelnde Integration von Transportoperationen, Kommunikationsereignissen und vielfädigen Anwendungsprogrammen.

Diese Nachteile werden durch die Active-Message-Spezifikation Version 2.0 aufgehoben, indem die Abstraktion eines Kommunikations*endpoints* definiert wird: Ein *Endpoint* entspricht entfernt einem Socket oder Port, weist doch aber einige Unterschiede auf. Eine Anwendung kann mehrere Endpoints von der Active-Message-Ebene anfordern, die sie über *Endpoint Tags* ansprechen kann und die ihr allein zur Verfügung stehen. Über eine Tag-Überprüfung wird die Authentifizierung innerhalb einer Anwendung sichergestellt. Ein Endpoint enthält dabei folgende Komponenten:

- einen Sendepool, um Nachrichten ausgehend von diesem Endpoint zu senden,
- einen Empfangspool, um Nachrichten für diesen Endpoint zu empfangen,
- eine Handler-Tabelle, die Handler-Indizes auf die korrekten Funktionszeiger abbildet,
- ein Segment des virtuellen Speichers der Anwendung, in das Transfers langer Nachrichten direkt gespeichert werden können,
- eine Übersetzungstabelle zur Abbildung von Indizes auf globale Endpoint-Namen und Tags sowie
- ein Tag zur Überprüfung ankommender Nachrichten.

Ein Endpunktbündel *(Endpoint Bundle)* entsteht aus der Zusammenfassung mehrerer Endpoints eines Prozesses. Zusätzlich zu den Endpoints sind hier noch drei weitere Komponenten vorhanden:

- eine Synchronisationsvariable, die anzeigt, ob ein Endpoint ein Kommunikationsereignis generiert hat,
- eine Ereignismaske, deren Zustand angibt, bei welchen Ereignissen die Synchronisationsvariable gesetzt wird und
- ein Flag, welches den Zugriffsmodus (nebenläufig oder sequentiell) auf die Endpoints oder das Bundle angibt.

Über die zusätzlichen Angaben innerhalb des Bundles können sich mehrere Threads bei der Benutzung der Endpoints koordinieren. Endpoints können hierbei immer nur einem Bundle zugeordnet sein, die Spezifikation unterstützt jedoch die dynamische Migration von Endpoints zu anderen Bundles.

Da die Authentifizierung über Tags erfolgt, deren Vergabe systemglobal und unter Betriebssystemschutz ablaufen kann, ist jetzt ein ausreichender Schutz vor Mißbrauch oder fehlerhafter Ausführung im Active-Message-System gewährleistet.

3 Die Active-Message-Kommunikationsschicht für das SMiLE-System

Eine Active-Message-Kommunikationsschicht, die auf der zweiten Version der Active-Message-Spezifikation [Mai95] beruht, wurde am LRR-TUM in einer Prototypvariante für das endgültige SMiLE-System auf einem Cluster von SCI-gekoppelten Workstations des Typs UltraSparc-1 und UltraSparc-2 implementiert. Es handelt sich hierbei um die gleiche Hardware-Konfiguration, die auch für [ISSW96] eingesetzt wurde, dort jedoch mit einer Implementation der Active Messages in der ursprünglichen Ausführung ohne Schutzmechanismen und nebenläufige Benutzung. Vier Ringe von jeweils zwei SCI-Knoten wurden dabei über einen Vierfach-SCI-Switch miteinander gekoppelt.

Rechnersystem	HW-RT-Latenz	Rohbandbreite
SCI NOW	9,3 μs	26 MB/s
Meiko CS-2 (DMA)	19,0 μs	40 MB/s
Myrinet NOW	20 μs	35 MB/s
ATM NOW	65 μs	15 MB/s

Tabelle 1: Rohbandbreiten und -latenzen für verschiedene Kommunikationstechnologien

Rohbandbreiten einiger moderner Verbindungsnetze sind aus Tabelle 1 zu entnehmen. Es fällt auf, daß SCIs Technologie insbesondere kurze Latenzen bei vernünftig hohen Bandbreiten verspricht, die es gilt, im Rahmen einer Kommunikationsschicht dem Programmierer mit möglichst geringen Verlusten zur Verfügung zu stellen.

Durch den direkten Zugriff auf entfernten Speicher aus einer Applikation heraus eignet sich SCI hervorragend zur Implementation einer solchen Active-Message-Schicht: kleine Nachrichten können direkt aus den Prozessorregistern in das Netzwerk injiziert werden, eine aufwendige Pufferverwaltung entfällt.

Während die Zahlen für SCI hierbei aus eigenen Messungen stammen, sind die Ergebnisse für die Meiko CS-2 (Parallelrechner mit proprietärem Verbindungsnetzwerk) und für Myrinet aus [ISSW96] entnommen, während das ATM-Netwerk in [vEBBV95] untersucht wurde.

3.1 Aufbau des SCI-Active-Message-Systems

Das Active-Message-System besteht im wesentlichen aus zwei Teilen:

- Dem **Active-Message-Dämon**, der auf jedem physikalischen Knoten des Systems einmal angestartet wird und zur Initialisierung der gemeinsamen Speichersegmente dient, die anderen beteiligten Dämonprozessen sowie der Bibliothek zur Verfügung gestellt werden. Darüberhinaus dient die Menge von Dämonen eines AM-Systems zur zentralisierten und kontrollierten Tagvergabe. Zur Vereinfachung des Gesamtkonzeptes findet die Kommunikation zwischen Dämonprozessen auch über (feste) Endpoints statt, die beim Hochfahren des Systems angelegt werden.

- Der **Active-Message-Bibliothek**, die zu Anwendungsprogrammen hinzugebunden wird und alle Datentypen und Funktionen der Programmierschnittstelle enthält. Die Sende- und Empfangspools werden hierbei über einen gemeinsamen Speicher zwischen Dämonprozeß und Bibliothek verwirklicht.

Da SCI enfernte Speicheroperationen unterstützt, ist bei dieser Implementation von Active Messages eine bedeutende Optimierung möglich: Durch Abbilden entfernten Speichers in lokalen Speicher können der Sendepool von Knoten A und der Empfangspool von Knoten B dem gleichen physikalischen Speicherbereich entsprechen. Da entfernte Schreiboperationen durch die Netzwerkhardware gepuffert werden, entfernte Leseoperationen jedoch den Prozessor bis zur Ankunft blockieren, wurde daher der effizientere Weg gewählt und jeweils der Empfangsbereich (oder Lesebereich) lokal angelegt und auf dem entfernten Knoten als Sendebereich abgebildet. (Siehe Abb. 1.)

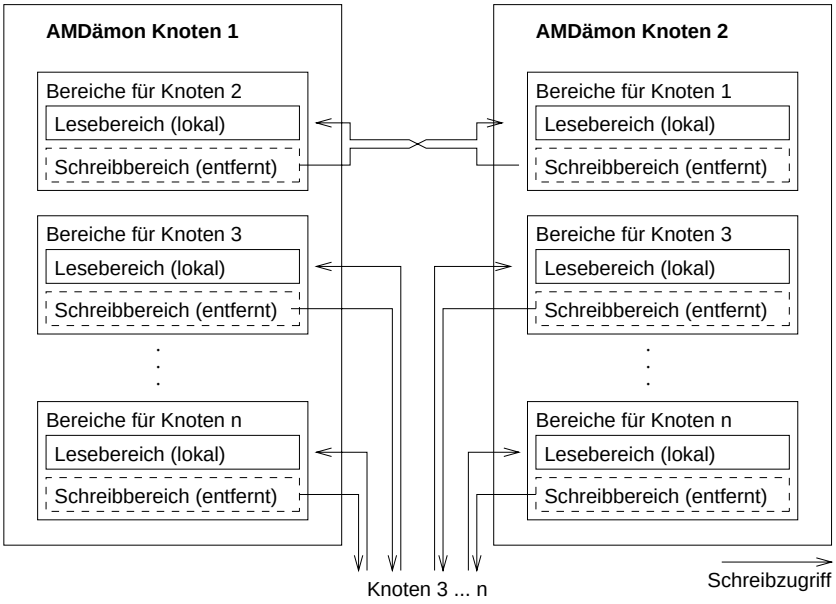

Abbildung 1: Anordnung der gemeinsamen Speicherbereiche zweier Active-Message-Dämonprozesse

Jeder Empfangsbereich ist dabei selber als Ringpuffer mit einer Standardgröße von 32 Nachrichten mittlerer Länge à 1024 Bytes aufgebaut, um ein nichtblockierendes Senden einer Reihe von Nachrichten zu ermöglichen. Längere Nachrichten werden direkt in das abgebildete virtuelle Speichersegment des Endpoints übertragen.

Da die derzeit auf dem Markt befindliche SCI-Hardware keinerlei Erzeugung entfernter Interrupts ermöglicht und deren Benutzung über UNIX-Signale in aller Regel auch zuviel Overhead mit sich brächte, arbeitet die AM-Schicht lediglich mit Polling der betreffenden Endpoints einer Anwendung, um den Empfang einer Nachricht zu bemerken. Der Nachteil der kurzzeitig möglichen Blockierung bei gefülltem Ringpuffer wird aufgehoben durch den Vorteil der ökonomisch genutzten Rechenzeit: Die Anwendung beschäftigt sich nur dann mit dem „Empfang“ einer Nachricht (dem Überprüfen der Puffer und dem Ausführen der Handlerfunktion), sobald sie sich selber dazu Zeit nimmt. Ist der Ringpuffer des betreffenden Endpoints leer, kehrt der Polling-Aufruf ohne Verzögerung zurück.

4 Messungen und Ergebnisse

Vergleichende Messungen der Ping-Pong-Umlaufzeiten einer kurzen Active Message unseres Systems mit anderen Active-Message-Implementationen (Tab. 2) zeigen deutlich, daß der zusätzliche Verwaltungsaufwand für die Sicherungsmechanismen vergleichsweise gering ist. Es zeigt sich sogar ein marginal besseres Verhalten als in der auch auf SCI auf der gleichen Hardware-Konfiguration ausgemessenen Active-Message-Version 1.1. Schauser et al. [ISSW96] verdeutlichen konsequenterweise auch, daß ihre Implementation noch klares Optimierungspotential aufweist.

Interessant in diesem Zusammenhang sind auch die scheinbar schwachen Leistungen der U-Net-Active-Message-Implementation. Obwohl dort eine direkt vom Benutzer aus zugreifbare Netzwerkhardware vorlag, die eine True-Zero-Copy-Implementation ermöglichte, so schränkten doch die ATM-Protokolle selbst die physikalisch mögliche Bandbreite und Latenz deutlich ein. Angesichts eines technologisch maximalen Durchsatzes von ca. 15 MB/s und einer Ping-Pong-Latenz von minimal ca. 65 μs erscheinen die Ergebnisse der U-Net-AM-Implementation, die eigene Schutzmechanismen abweichend von der AM-Spezifikation 2.0 enthält, im richtigen Licht.

Machine	Roundtrip-Latenz	Bandbreite mit AM-Store
SCI NOW mit AM 2.0 (mit Switch)	15 μs	21 MB/s
SCI NOW mit AM 1.1 (mit Switch)	16,5 μs	13 MB/s
Meiko CS-2	23 μs	32 MB/s
Myrinet NOW	28,9 μs	34 MB/s
U-Net (ATM NOW, True Zero Copy)	71 μs	14 MB/s

Tabelle 2: Roundtrip-Latenzen und Transferraten für verschiedene Active-Message-Implementationen

Wie aus Abb. 2 zu erkennen ist, zeigt die verwirklichte Active-Message-Schicht selbst bei 64 KByte großen Nachrichten nur leichte Sättigungserscheinungen, so daß die gemessenen 21 MB/s nicht notwendigerweise die obere Grenze des Durchsatzes repräsentieren müssen, dabei aber bereits 80 % der verfügbaren Leistung ausnutzen. Als Softwareoverhead, der in der Active-Message-Schicht pro Nachricht hinzukommt, sind ca. 6 μs pro Nachricht anzusetzen.

Ein Teil dieses Overheads läßt sich auf das Polling der Eingangspuffer zurückführen: Pro Verbindung an einem physikalischen Knoten benötigt die Bibliothek eine durchschnittliche Zeit von ca. 0,25 bis 0,3 μs zur Abfrage der angeschlossenen Endpoints. Dieser Wert bleibt konstant über einen weiten Bereich von angeschlossenen Teilnehmern, unabhängig davon, ob diese einer oder mehreren Anwendungen zuzuordnen sind. .

Ein Problem verdeutlicht jedoch Abb. 3: Eine Erhöhung der Teilnehmerzahl im Rahmen einer Anwendung erhöht auch gleichzeitig die Latenz eines Polling-Aufrufes, d. h. mit zunehmender Parallelität eines Programmes steigen die Kosten der Kommunikation. Dieses ungewünschte Verhalten, welches die Skalierbarkeit auf große Prozessorzahlen einschränkt, liegt in der Pufferverwaltung begründet, die für jede mögliche physikalische Verbindung einen Nachrichtenpuffer zu prüfen hat. Diese Anordnung ist jedoch die einzige, die mit der aktuellen SCI-Technik einsetzbar ist, da es die Speichertransaktionen von SCI erfordern, daß senderseitig

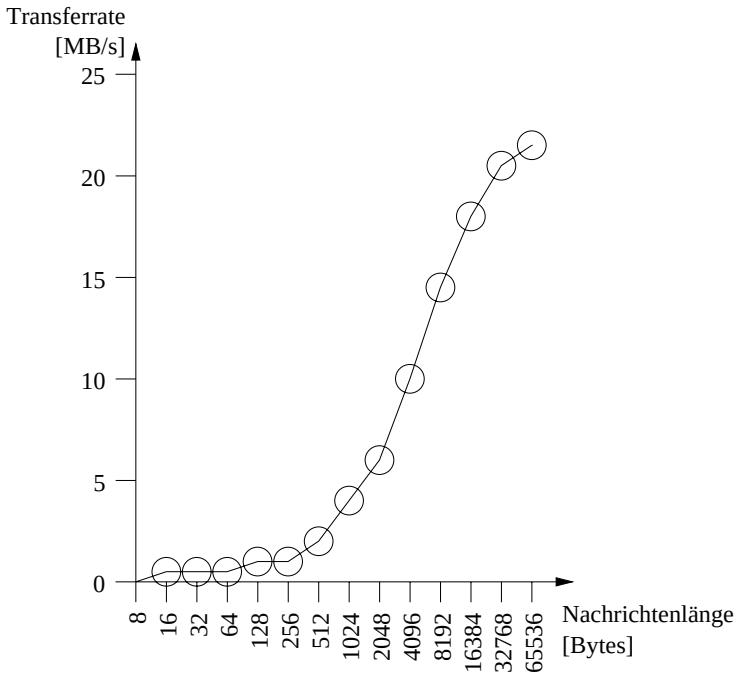

Abbildung 2: Datendurchsatz für Store-Active-Messages auf SCI

über die Empfangsadresse entschieden wird. Mögliche Lösungen für dieses Skalierungsproblem, wie z B. die Einführung einer speziellen *Enqueue*-Transaktion in SCI, wurden daher bereits in [ISSW96] vorgeschlagen.

5 Zusammenfassung und Ausblick

Es zeigt sich, daß eine moderat optimierte Active-Message-Kommunikationsbibliothek, die sich die Schutzverfahren der AM-Spezifikation 2.0 zu eigen macht, sehr wohl mit Implementationen der SPMD-Variante der Active Messages im Hinblick auf Latenz und Durchsatz mithalten kann. Inwieweit der festgestellte Overhead dann doch größer als eine aggressiv optimierte AM-Implementation der Version 1.1 auf einem Netzwerk von Arbeitsplatzrechnern unter SCI ist, bleibt abzuwarten.

SCI als Wahl für das Kommunikationsmedium stellt sich als vorteilhaft heraus. Die Möglichkeit des direkten Zugriffes auf entfernten Speicher eröffnet die Chance zu einer True-Zero-Copy-Implementation ohne Kopieraufwand, während die physikalischen Daten und die inhärente Zuverlässigkeit von SCI es einfach machen, trotz einer Softwareschicht nahe an dessen Rohbandbreite und -latenz zu gelangen.

Ein Problem jedoch stellt sich bei Benutzung einer Netzwerktechnologie wie SCI, die originär zum Aufbau von Systemen mit verteiltem gemeinsamen Speicher gedacht ist: Der Initiator einer Transaktion (Lesen oder Schreiben) bestimmt jeweils die Adresse des betreffenden Datums. Diese Eigenschaft macht es in Verbindung mit dem Puffermanagement schwierig, skalierbare Latenzen für den Empfang einer Active Message zu erhalten. Eine Umkehrung der Datenrichtung (Pull-Messaging) ist nicht angebracht aufgrund der Nichtüberlappbarkeit von entfernten

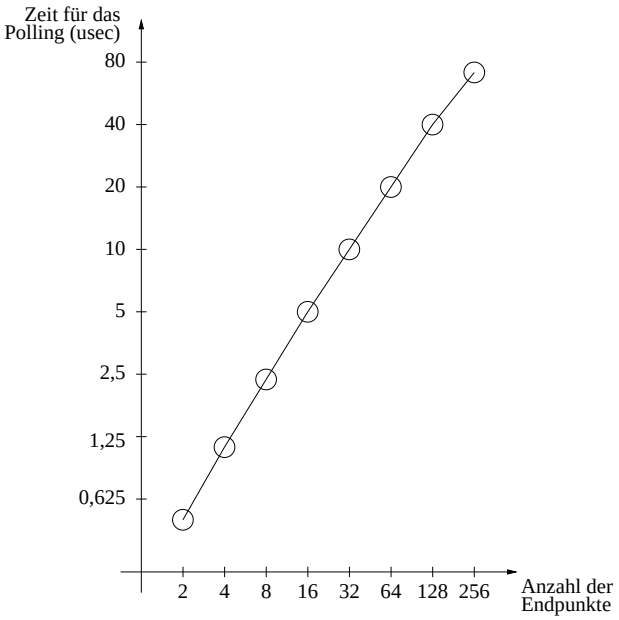

Abbildung 3: Zeit für das Polling von mehreren Endpoints

Leseoperationen im Gegensatz zur Möglichkeit der Pufferung entfernter Schreiboperationen.

Die beschriebene Active-Message-Bibliothek wird im Rahmen des SMiLE-Projektes als Softwareinfrastruktur genutzt, um andere Systeme darauf zu implementieren. Beispiele hierfür sind eine virtuelle Datenflußmaschine im Teilprojekt MuSE (Multithreaded Scheduling Environment [Leb96]) und andere, auf Active Messages aufgesetzte Programmiermodelle. Darüberhinaus steht sie als Kommunikations-API jedem Anwendungsprogrammierer auch im Mehrbenutzerbetrieb zur Verfügung.

Literatur

[AHKL96] G. Acher, H. Hellwagner, W. Karl, and M. Leberecht. A PCI-SCI Bridge for Building a PC Cluster with Distributed Shared Memory. In *Proceedings of the 6th International Workshop on SCI-Based High-Performance Low-Cost Computing*, Santa Clara, CA, September 1996.

[BDFL96] M. A. Blumrich, C. Dubnicki, E. W. Felten, and K. Li. Protected User-Level DMA for the SHRIMP Network Interface. In *Proceedings of the 2nd International Symposium on High-Performance Computer Architecture*, Februar 1996.

[CKK+94] D. Culler, K. Keaton, C. Krumbein, L. T. Liu, A. Mainwaring, R. Martin, S. Rodrigues, K. Wright, and C. Yoshikawa. *Generic Active Message Interface Specification Version 1.1*. University of California, Berkeley, Internet-available document edition, November 1994.

[Ebe96] Michael Eberl. Realisierung einer DSM-Implementierung von Active Messages am Beispiel eines SCI-gekoppelten Sun Workstation Clusters. Diplomarbeit, August 1996. TU München.

[Gus92] D. B. Gustavson. The scalable coherent interface and related standards projects. *IEEE Micro*, page 10ff, Februar 1992.

[HKL97] Hermann Hellwagner, Wolfgang Karl, and Markus Leberecht. Enabling a PC Cluster for High-Performance Computing. *SPEEDUP Journal*, 11(1), June 1997.

[ISSW96] Maximilian Ibel, Klaus E. Schauser, Chris J. Scheiman, and Manfred Weis. Implementing Active Messages and Split-C for SCI Clusters and Some Architectural Implications. In *Proceedings of the 6th International Workshop on SCI-Based High-Performance Low-Cost Computing*, Santa Clara, CA, September 1996.

[KH95] Wolfgang Karl and Hermann Hellwagner. SMiLE – Entwurf eines Multirechnersystems auf SCI-Basis. In *Mitteilungen der GI/PARS-Berichte*. GI/ITG-PARS, Dezember 1995.

[Leb96] Markus Leberecht. A Concept for a Multithreaded Scheduling Environment. In *Proceedings of the 4th PASA Workshop*, KFA Jülich, April 1996. GI/ITG.

[Mai95] Alan M. Mainwaring. Active Message Application Programming Interface and Communication Subsystem Organization. Technical report, Computer Science Division, University of California at Berkeley, 1995. draft.

[Soc93] IEEE Computer Society. *IEEE Standard for Scalable Coherent Interface (SCI)*. The Institute of Electrical and Electronics Engineers, Inc., 345 East 47th Street, New York, NY 10017, USA, August 1993.

[vEBBV95] Thorsten von Eicken, Anindya Basu, Vineet Buch, and Werner Vogels. U-Net: A User-Level Network Interface for Parallel and Distributed Computing. In *Proceedings of the 15th ACM Symposium on Operating System Principles*. ACM, Dezember 1995.

[vECGS92] Thorsten von Eicken, David E. Culler, Seth C. Goldstein, and Klaus E. Schauser. Active Messages: a Mechanism for Integrated Communication and Computation. In *Proceedings of the 19th International Symposium on Computer Architecture*. ACM Press, Mai 1992.