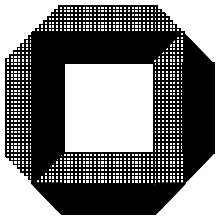
# RESH

## Rechnernetze als Supercomputer und Hochleistungsdatenbanken

## Zwischenbericht Oktober 1998

Joachim Blum, Matthias Gimbel,
Bernhard Haumacher, Holger Leuck,
Michael Philippsen, Thomas Warschko

Interner Bericht 23/98



**Universität Karlsruhe**

Fakultät für Informatik

# Zusammenfassung

Dieser Zwischenbericht stellt die Ergebnisse der Forschergruppe RESH im Zusammenhang dar, die in den ersten neun Monaten der Projektlaufzeit erarbeitet wurden. Dargestellt werden die Ergebnisse der einzelnen Teilprojekte sowie Ausblicke auf geplante weitere Arbeitseinheiten.

Am 6. Oktober 1998 wurden diese Zwischenergebnisse in Kurzvorträgen vorgestellt, um die Aktivitäten der einzelnen Teilprojekte besser auf einander abzustimmen. Ebenfalls im Bericht enthalten sind die Poster, mit denen die Forschergruppe RESH anläßlich des Tags der Informatik am 6. November 1998 ihre Arbeit der interessierten Öffentlichkeit vorstellte

# Inhaltsverzeichnis

# 1 Einführung

Ziel der Forschergruppe ist es, Technik und Anwendung von vernetzten Rechnern als Hochleistungsrechenanlagen („Cluster Computing“) voranzutreiben. Solche Rechnernetze bergen gewaltiges Potential:

- preisgünstige und skalierbare Superrechnerleistung
- riesigen Hauptspeicher
- preisgünstigen, ausfallsicheren und skalierbaren Hintergrundspeicher mit
- gewaltiger E/A-Bandbreite.

Durch die Vernetzung leistungsfähiger Einzelrechner wird es möglich, diese Leistung auch Anwendern zur Verfügung zu stellen, die zwar hohe Rechenleistung brauchen, für die aber ein Superrechner zu teuer ist, Superrechnerleistung nicht ständig benötigt wird oder das zum Betrieb notwendige Personal nicht zur Verfügung steht.

Die Haupthindernisse, die bislang einen Durchbruch vernetzter Plattformen verhinderten, sind die ungenügende Kommunikationsleistung sowie die erheblichen Schwierigkeiten bei der Programmierung von verteilten Anwendungen. Für diese Probleme sollen im Rahmen dieses Projekts Lösungen erarbeitet werden. Diese Lösungen sollen gleichzeitig eine Plattform für anspruchsvolle Anwendungen bieten.

Das Projekt weist also zwei Hauptstoßrichtungen auf: Erstens soll die Grundtechnologie von Rechnerkopplungen vorangetrieben werden. Dazu sind Arbeiten im Bereich schneller Netze, schlanker Kommunikationsprotokolle, Betriebssystemanpassung (bzw. -umgehung), Programmierumgebungen und Bibliotheken wie PVM, MPI oder Fast/Active Messages erforderlich.

Die zweite Hauptstoßrichtung sind Anwendungen, und zwar in den Bereichen Data-Mining, Bildfolgenauswertung und Sichtsteuerung von Robotern in Echtzeit. Diese Anwendungen sollen die Rechnerkopplungstechnik testen und validieren.

Die Gruppe gliedert sich in folgende Projektbereiche:

| | |
|---|---|
| **Projekt T1:** | Intelligente Netzwerke, Protokolle und Systeme |
| **Projekt T2:** | Programmierumgebungen und Übersetzer, Anwenderunterstützung |
| **Projekt L1:** | Paralleles Data-Mining |
| **Projekte N1 & N2:** | Bildfolgenauswertung als Anwendungsprobe zur quantitativen Untersuchung von Parallelisierungsansätzen |

Die enge Zusammenarbeit dieser Projektbereiche, insbesondere zwischen den Grundlagenbereichen T1 und T2 und den Anwendungen (L1, N1 und N2) sollen sich die einzelnen Teilprojekte gegenseitig befruchten: Zum einen ist eine direkte Unterstützung der Anwender gewährleistet, zum anderen erhalten auch die Entwickler der Basistechnologien eine schnelle und präzise Rückkoppelung und Validierung ihrer Arbeiten.

# 2 T1: Intelligente Netzwerke, Protokolle und Systeme

## 2.1 Ausgangssituation

Am Institut für Programmstrukturen und Datenorganisation läuft bereits eine Entwicklung zur Parallelverarbeitung in Workstation/PC-Clustern, genannt ParaStation. Die ParaStation-Rechnerkopplung löst die Aufgabe des Zusammenschaltens von handelsüblichen Einzelrechnern zu einem Parallelrechner, ohne die gewohnte Einzelrechnernutzung zu beeinträchtigen. ParaStation ist speziell an die Bedürfnisse der Parallelverarbeitung in Workstation-Clustern bzw. Workstation-Farmen zugeschnitten. Insbesondere bedeutet dies sehr kurze Kommunikationsverzögerungszeiten (Latenzzeiten), minimalen Protokolloverhead, keinen Betriebssystem-Overhead und einen hohen Durchsatz. Die Skalierbarkeit reicht dabei von der Kopplung zweier Arbeitsplatzstationen bis hin zu Farmen von Arbeitsplatzstationen mit mehreren hundert Prozessoren. Die Vorführung einer ParaStation-Anlage mit acht vernetzten Arbeitsplatzstationen hat bereits im November 1995 stattgefunden. ParaStation steht somit als funktionfähige Plattform mit Beginn der Forschergruppenaktivitäten allen Prokjektpartnern zur Verfügung.

Die Arbeiten an ParaStation brachten viele Erkenntnisse und Einsichten in die Problematik des „Cluster Computing“ zu Tage, warfen aber auch gleichzeitig neue Frage- und Problemstellungen auf, so daß auf diesem Gebiet auch weiterhin Forschungsbedarf besteht.

Mit einer weiteren Steigerung der Übertragungsgeschwindigkeit, einer Steigerung der räumlichen Ausdehnung sowie der Bereitstellung weiterer Plattformen, soll das ParaStation2-Projekt zum Teil eine recht praxisorientierte Zielsetzung verfolgen. Die eigentliche Herausforderung jedoch liegt im Entwurf und der Realisierung eines Kommunikationssubsystems, das die Leistungsfähigkeit des bisherigen Systems weiter steigert, um mit den steigenden Prozessorgeschwindigkeiten Schritt zu halten. Dazu zählen intelligente Hochleistungsnetzwerke, schlanke Hochgeschwindigkeitskommunikationsprotokolle sowie die effiziente Integration dieses Kommunikationssubsystems in bestehende oder zukünftige Betriebssysteme.

## 2.2 Ziele

- **Steigerung der Übertragungsgeschwindigkeit und räumlichen Ausdehnung:** Der Einsatz von Glasfaserverbindungen ermöglicht Übertragungsgeschwindigkeiten im Gbit/s-Bereich (Steigerung um Faktor 7) sowie eine räumlichen Ausdehnung von bis zu 1km zwischen je zwei Arbeitsplatzrechnern (Steigerung um Faktor 100). Damit wäre es möglich, z.B. alle Arbeitsplatzrechner innerhalb eines Gebäudes als ParaStation2-Parallelrechner zu nutzen. Der derzeit eingesetzte Netzwerkprozessor erlaubt bis zu 65.000 angeschlossene Stationen (Arbeitsplatzrechner), so daß in diesem Punkt praktisch keinerlei Beschränkung besteht. Im Zusammenhang mit der Erweiterung auf eine größere Anzahl von Einzelrechnern sind dann auch Fragen der Ausfallsicherheit zu betrachten.

- **Intelligente Hochleistungsnetzwerke:** Die ParaStation-Kommunikationshardware stellt bisher eine gesicherte Datenübertragung, automatische Wegwahl, Flußkontroll- und Synchronisationsmechanismen zur Verfügung. Offen sind Fragen nach einer echten Multiuser-/ Multitasking-Unterstützung, kollektiven Kommunikationsoperationen, sowie das Bereitstellen weiterer Protokollfunktionalitäten, wie etwa einer dynamischen und adaptiven Wegfindung oder einem dynamischen Lastausgleich innerhalb des Netzwerkes.

- **Bereitstellung neuer Plattformen:** Bei Anwendern, die einen Einstieg in die Parallelverarbeitung planen, besteht berechtigtes Interesse, dies auf Basis der bereits vorhandenen Infrastruktur (verfügbaren Arbeitsplatzrechnern) in Angriff zu nehmen. Durch die Verwendung einer standardisierten Schnittstelle der ParaStation-Kopplungskarte (PCI-Bus) ist die Möglichkeit der Umsetzung auf weitere Plattformen prinzipiell gegeben. Die dabei offenstehenden Möglichkeiten schließen auf der Hardwareseite Prozessoren der Firmen Intel, Digital, IBM und Sun ein. Im Bereich der Betriebssysteme sind zahlreiche UNIX-Varianten (OSF/1, Linux, Solaris, AIX) sowie Windows-NT denkbar. Die letztendlich verwirklichten Plattformen werden sich an den Bedürfnissen der beteiligten Projektpartner orientierten.

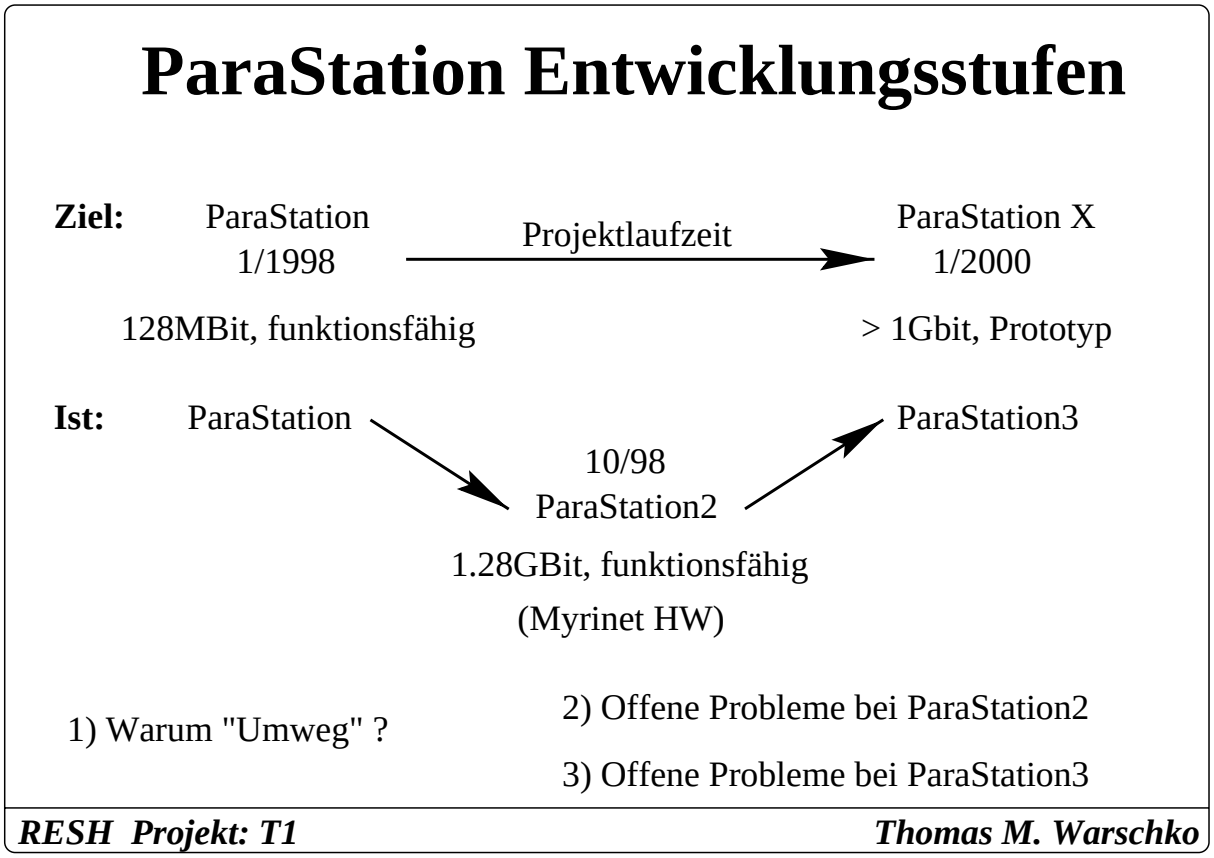
- **Schlanke Hochgeschwindigkeitskommunikationsprotokolle:** Kommunikationsprotokolle überbrücken die Kluft zwischen den Fähigkeiten der Kommunikationshardware und den Anforderungen

eines Betriebssystems sowie der gewünschten Funktionalität einer Kommunikationsschnittstelle. Die Protokolle innerhalb des ParaStation-Systems bilden da keine Ausnahme. Die Frage die sich vielmehr stellt ist, welche (sinnvolle) Funktionalität die Kommunikationshardware mitbringen sollte und welche Teile besser innerhalb eines Kommunikationsprotokolls realisiert werden sollten. Dies ist natürlich unter der Prämisse, einen wohldefinierten Ausgleich zwischen der Komplexität der Hardware und der Effizienz der Software zu finden.

- **Betriebssystemintegration:** Bisher basiert die ParaStation-Kommunikationsschnittstelle auf dem Prinzip der sogenannten User-Level Kommunikation, das einfach ausgedrückt das Betriebssystem aus dem Kommunikationspfad entfernt. Damit lassen sich zwar sehr leicht neue Protokolle implementieren und testen, aber die gewünschte Transparenz eines Kommunikationssubsystems, die ein Betriebssystem normalerweise bietet, geht verloren. Applikationen müssen auf eine andere, wenn auch äquivalente Kommunikationsschnittstelle aufsetzen und können nicht alleine aus dem Vorhandensein eines Hochleistungsnetzwerkes profitieren. Die Kommunikationsschnittstelle des Betriebssystems unterliegt derselben Beschränkung, denn solange das Betriebssystem das Hochleistungsnetzwerk nicht nutzen kann, können auch systeminterne Kommunikationsoperationen mit anderen Stationen nicht über das Hochleistungsnetzwerk abgewickelt werden. Die Problematik, die sich demnach stellt ist, wie ein Hochleistungsnetzwerk effizient innerhalb des Betriebssystem angekoppelt werden kann.

## 2.3 Aktueller Stand

### 2.3.1 ParaStation Hardware




Bei Projektstart im Januar 1998 stand ParaStation [36, 34] als Basisplattform bereits zur Verfügng und war voll funktionsfähig, so daß für die restlichen Projektpartner nicht das Problem einer geplanten, aber noch nicht existenten Hardware bestand. Parallel zur den anderen Projekten sollte die neue ParaStation Hardware entwickelt und am Ende der Projektlaufzeit die Funktionsweise an einem Prototyp demonstriert werden.

Abweichend von der ursprünglichen Planung wurde eine ParaStation-Zwischenstufe eingeschoben, die einen Teil der geplanten Weiterentwicklungen bereits realisiert (z.B. die gesteigerte Übertragungsleistung von 1.28 GBit/s). Die „Zwischenlösung“ ist in Betrieb und wird mittlerweile von allen Projektpartnern genutzt; das Fernziel (ParaStation3) wird ebenfalls weiter verfolgt.

Im folgenden wird auf die Gründe für diese Zwischenlösung eingegangen, sowie die offenen Fragen und Probleme bei der (Weiter-)entwicklung von ParaStation2 und ParaStation3 diskutiert. Dabei gilt folgende Konvention: ParaStation1 bezeichnet die ursprüngliche ParaStation Hardware aus dem Jahre 1995. Die Kombination aus der ParaStation-Software und der Myrinet Hardware der Firma Myricom [4] wird als ParaStation2 bezeichnet. Die Hardwareneuentwicklung läuft unter der Bezeichnung ParaStation3.

# Warum der Umweg ?

**Programmatische Gründe:**

- ParaStation etwas veraltet (zu langsam)
- 2 Jahre Wartezeit auf Prototypen ist eine lange Zeit

**Jetzt: ParaStation2 ist "State of the Art"**

**Wissenschaftliche Gründe:**

- die ParaStation HW ist etwas spartanisch, Myrinet stellt geradezu das andere Extrem da.
- Wissensgewinn (u.a. für ParaStation3):
  - Umgang mit DMA-Übertragung
  - Eigenintelligenz einer Netzwerkkarte



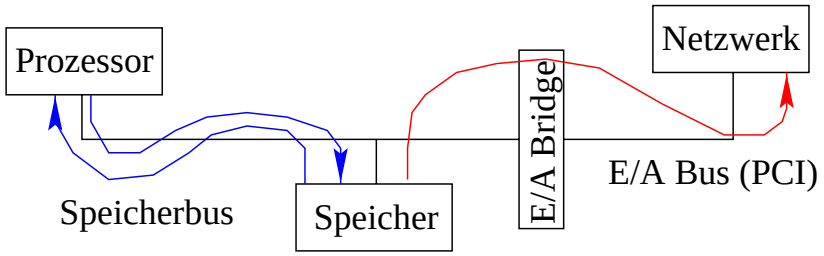
Im wesentlichen basiert die Entscheidung für die Entwicklung und den Einsatz einer Zwischenlösung auf zwei Argumentationsschienen:

1. **Programmatische Gründe:** Das Leistungsspektrum der ParaStation1 Hardware ($3\mu s$ Latenzzeit und 128 Mbit/s Durchsatz) wurde besonders beim Durchsatz von marktgängigen Lösungen wie FastEthernet (100 Mbit/s) oder ATM (155 Mbit/s) eingeholt, so daß ein gewisses Argumentationsproblem für den Einsatz von Spezialhardware entsteht. Lediglich die extrem kurzen Latenzzeiten von ParaStation sind nach wie vor unerreicht. Desweiteren sind zwei Jahre Wartezeit auf einen Prototypen äußerst lang – besonders in Hinblick auf neue LAN-Technologien wie Gbit/s ATM oder Gigabit-Ethernet, denn im Vergleich zu diesen Systemen wirkt die ParaStation1 Hardware geradezu steinzeitlich.
2. **Wissenschftliche Gründe:** Die Programmierschnittstelle von ParaStation1 ist äußerst spartanisch und läßt außer der implementierten Ansteuerungen keinen Spielraum für Alternativen. Die Myrinet-Adapter dagegen verwenden einen voll programmierbaren Netzwerkprozessor sowie verschiedene DMA-Einheiten, so daß hier vollkommene Freiheit bezüglich neuer Ideen besteht.

Mit der Adaption der ParaStation-Software auf die Myrinet-Hardware befindet sich ParaStation2 [37] jetzt wieder auf dem Stand der Kunst im Clustercomputing. Die erreichten Latenzzeiten ($12\mu s$) sind zwar etwas schlechter als die bei ParaStation, dafür ist ein Durchsatz von bis zu 1.28 Gbit/s konkurrenzlos (und Myricom hat für 1999 bereits 2.5 Gbit/s angekündigt).

Desweiteren fließen die Erfahrungen bei der Programmierung des Myrinet-Adapter als Wissensgewinn in die Entwicklung von ParaStation3 ein.

# ParaStation2: Die Kopier Problematik




1) Kopieren der Daten im Speicher  2) Transfer der Daten ins Netzwerk

Wozu Operation 1 ?  a) Übergang vom Benutzer- in Kernadreßraum
b) Transfer der Daten in einen DMA-Bereich

Varianten:  Einmal-Kopie: Operationen 1 und 2 nacheinander
Einmal-Kopie mit Fließband: Operationen 1 und 2 verschränkt
Ohne-Kopie: Operation 2

***RESH-T1: Ohne-Kopie 1/5*** ***Thomas M. Warschko***

Derzeit ist die Frage, ob ein Gbit/s Netzwerk in heutigen Systemen überhaupt mit dem notwendigen Datenstrom versorgen kann, noch offen. Erste Untersuchungen zeigen lediglich, daß unter üblichen Voraussetzung dies **nicht** möglich ist. Die Ursache dafür stellt die sogenannte Kopier-Problematik [38] dar.

Sollen Daten über ein Netzwerk verschickt werden, so werden diese zunächst vom Adreßraum des Benutzers in den Adreßraum des Betriebssystemkerns kopiert. Dies ist notwendig, da einerseits die Hardware nur vom Systemkern angesprchen werden kann und andererseits, da die Daten in einem DMA-fähigen Speicherbereich abgelegt sein müssen. Anschließend werden die Daten vom Systembereich in das Netzwerk übertragen und von dort aus weitergeleitet. Der Datentransfer zwischen Applikation und Netzwerk ist gewünscht; der Umweg über einen Puffer im Betriebssystem dagegen stellt einen systembedingten Overhead dar.

Ungeachtet von Systembeschränkungen lassen sich prinzipiell drei Datentransfervarianten identifizieren:

1. **Einmal-Kopie:** Die Transferoperationen 1 und 2 werden streng sequentiell ausgeführt.
2. **Einmal-Kopie mit Fließband:** Die Transferoperationen 1 und 2 zweier aufeinanderfolgender Kommunikationsperationen sind verschränkt.
3. **Ohne-Kopie:** Die Daten werden direkt aus der Applikation an das Netzwerk übergeben (sog. User-Level-Kommunikation).

Im folgenden wird das Leistungsverhalten der drei Varianten anhand unterschiedlicher Systeme diskutiert.

# Verhalten heutiger Systeme

| | Alpha, 600 MHz LX-Board | DualPentium 200 GA 586DX | Pentium-II, 400MHz BX-Board |
|---|---|---|---|
| Bmem (Peak) | 1600 MB/s | 528 MB/s | 800 MB/s |
| Bcpy (Real) | 125 MB/s | 42 MB/s | 125 MB/s |
| (%Bmem) | (8%) | (8%) | (16%) |
| DMA (in/out) (PCI = 133MB/s) | 67/120 MB/s | 40/40 MB/s | 124/124 MB/s |
| Einmal-Kopie | 44 MB/s | 21 MB/s | 64 MB/s |
| Einmal-Kopie mit Fließband | 44 MB/s (~350 Mbit/s) | 20 MB/s (~160 Mbit/s) | 62 MB/s (~500 Mbit/s) |
| Ohne-Kopie | 67 MB/s (~530 Mbit/s) | 40 MB/s (~320 Mbit/s) | 125 MB/s (1000 Mbit/s) |

***RESH-T1: Ohne-Kopie 2/5*** ***Thomas M. Warschko***

Der Parameter `Bmem(Peak)` gibt den Speicherdursatz laut Herstellerangaben für die untersuchten Systeme (DEC-Alpha, Pentium und Pentium II) an und läßt zunächst keinerlei Engpässe vermuten. Wird jedoch die reale Speicherbandbreite `Bcpy(Real)` gemessen, die idealerweise 50% des Maximaldurchsatzes betragen sollte, so ergibt sich mit 8% bzw. 16% des Idealwertes ein äußerst unbefriedigendes Bild. Ein ähnliches Verhalten zeigt auch die DMA-Transferleistung, die keineswegs den vom PCI-Bus spezifizierten Wert von 132 MByte/s erreicht.

Verwendet man diese Daten zur Berechung der Transferleistung der drei Datentransfervarianten, so ist das Ergebnis ebenfalls ernüchternd: Die klassischen Varianten Einmal-Kopie und Einmal-Kopie mit Fließband weisen inakzeptable Leistungswerte auf und können im besten Fall gerade einmal die Hälfte der Maximalleistung erreichen (und das bei Annahme einer idealen Portokollverarbeitung ohne aufwendige Protokolltürme). Lediglich die Ohne-Kopie Variante gibt Hoffnung, denn hier gelingt es wenigstens in einem Fall das Gbit/s Netzwerk mit dem notwendigen Datenstrom zu versorgen.

Damit stellt sich die Frage, warum Protokolle nicht alle die Ohne-Kopie Variante verwenden.

# Warum verwenden nicht alle Ohne-Kopie?

Allgemeine Antworten:

1) bis vor ca. 3 Jahren: GEHT NICHT (wg. Schutzmechanismen in UNIX)

2) Das ist unsauber (wg. USER-LEVEL-COMMUNICATION)

3) Sicherheitsbedenken (gemeinsame Speicherbereiche)

Speziellere Antworten:

1) Kein DMA-Betrieb möglich, PIO zu langsam

2) Kollidiert mit dem Konzept des virtuellen Speichers

3) TCP/IP-Turm müßte komplett neu geschrieben werden

4) Kein Betriebssystem bietet die notwendigen Voraussetzungen



Bis vor ca. 3 Jahren wurde die allgemeine Ansicht vertreten, daß Ohne-Kopie Protokolle nicht möglich sind, da die Schutzmechanismen des Betriebssystems (UNIX) dies verhindern. Genau dieselben Mechnismen die zum Speicherschutz eingesetzt werden, lassen sich jedoch nutzen, um die Voraussetzung für Ohne-Kopie Protokolle, nämlich das Einblenden der Hardware in den Adreßraum einer Applikation, zu schaffen. Heute ist das Verfahren unter der Bezeichnung „User-Level-Kommunikation“ bekannt, genießt aber immer noch den Beigeschmack eines „dirty hack“. Die Hauptargumente gegen dieses Verfahren sind meist Sicherheitsbedenken, die sich auf die geimeinsam benutzten Speicherbereiche beziehen.

Außer einer gewissen Abneigung gegen Ohne-Kopie-Protokolle gibt es allerdings auch ernstzunehmende Einwände:

1. Die Daten könnten nicht per DMA übertragen werden, da DMA-Bausteine mit physikalischen die Applikation jedoch mit virtuellen Adressen arbeitet (und die Applikation die physikalischen Adressen nicht in Erfahrung bringen kann). Außerdem sei die Programmgesteuerte Ein/Ausgabe (PIO) zu langsam.
2. Die Handhabung von physikalischem Speicher, wie ihn die DMA-Bausteine benötigen, kollidiert mit den Mechanismen eines virtuellen Speichers, speziell mit der Auslagerung von Speicherseiten.
3. Die Implementierung des TCP/IP-Turms ist auf eine Ohne-Kopie Strategie nicht ausgelegt und müßte neu geschrieben werden.
4. Derzeit bietet kein kommerzielles Betriebssystem (und auch die wenigsten Forschungsprototypen) die notwendigen Voraussetzung, um eine Ohne-Kopie Strategie transparent zu integrieren.

Die Einwände 3 und 4 gehen direkt an die Adresse von Herstellerfirmen, daß in diesem Bereich Handlungsbedarf existiert. Im folgenden wird auf die Einwände 1 und 2 näher eingegangen.

# Ohne-Kopie Sender:

Einblenden der Kommunikationshardware in den Adreßraum der Applikation

+ schneller Zugriff - Sicherheit

Einblenden eines (geschützten) Kommunikationsendpunktes in den Adreßraum einer Applikation

+ Sicherheit - Hardwareunterstützung

Kopieren der Daten direkt in die Kommunikationshardware (PIO)

+ kein DMA, genauso schnell, keine Adreßumsetzung

- kein DMA, hohe CPU Belastung

DMA-Transfer in die Kommunikationshardware

+ keine CPU-Belastung

- Adreßberechnung, keine Seitenüberschreitung, Datenausrichtung, Synchronisation



- Das Einblenden der Kommunikationshardware in den Adressraum einer oder mehrerer Applikationen stellt technisch kein Problem dar (gewöhnliche Speicherzuordnung der MMU) und ermöglicht einen direkten und damit schnellen Zugriff von der Applikation auf die Kommunikationshardware. Der Nachteil liegt in mangelnden Sicherheitsmechanismen, da sowohl die korrekte Interaktion mit der Hardware als auch die Koordination der Applikationen untereinander nicht von einer zentralen Instanz wie dem Betriebssystem gewährleistet wird.

  Würde die Kommunikationshardware etwas wie applikationsspezifische (geschützte) Kommunikationsendpunkte bereitstellen, die einzeln in den Adreßraum jeweils einer Applikation eingeblendet werden, wäre das oben beschriebene Sicherheitsproblem (trotz User-Level-Kommunikation) gelöst. Leider existiert derzeit keine Hardware, die Kommunikationsendpunkte anstatt Registersätze zur Verfügung stellen würde.

- Auf der Sendeseite ist das Kopieren von Daten aus dem Speicher in die Netzwerkhardware (PIO) in vielen Systemen genau so schnell wie die Verwendung von DMA-Operationen. Außerdem benötigt PIO keinerlei Umsetzung von virtuellen in physikalische Adressen und PIO benötigt keinen physikalisch zusammenhängenden Speicherbereich. Allerdings wird die CPU stark belastet, da sie den gesamten Datentransfer vornimmt.

- Die Vorteile der PIO sind die Nachteile der DMA (und umgekehrt). DMA entlastet die CPU benötigt aber eine Adreßumrechnung (virtuell in physikalisch), die Speicherseitengrenzen müssen beachtet werden (es kann nur physikalisch zusammenhängender Speicher übertragen werden), die Datenausrichtung (auf Maschinenwortbreite) muß gewährleistet sein, und DMA als asynchroner Prozeß (unabhängig von der CPU) bedarf speziellen Synchronisationsmechanismen. Die dazu notwendigen Vaerfahren sind jedoch alle aus der Konzeption von Betriebssystemen bekannt (und erprobt).

Die Sendeseite eines Ohne-Kopie-Protokolls ist vergleichsweise einfach – problematisch ist der Empfänger.

# Ohne-Kopie Empfänger:

Kopieren der Daten in den Hauptspeicher (PIO)

+ einfach - unmögliche Leistungsdaten (< 10MB/s)

Also: DMA ist unumgänglich ! (und damit fangen die Probleme an)

Pufferspeicher in der HW viel zu klein

DMA benötigt physikalische Adressen (Speicherseiten)

DMA benötigt einen zusammenhängenden Speicherbereich

Speicherseiten müssen vorher reserviert werden (RemotePageFault)

-> PrePost von erwarteten/möglichen Empfangsaufträgen

(vollständig neue Kommunikationssemantik)

Datenausrichtung (Alignment)



Das Kopieren von Daten zwischen Netzwerk und Hauptspeicher (PIO) ist zwar einfach (und auf Sendeseite auch schnell), beim Emfänger jedoch werden inakzeptable Leistungsdaten erreicht (kein „burst-read“ über den PCI-Bus). Demnach müssen die Daten per DMA in den Speicher übertragen werden.

- Der Pufferspeicher auf einer Netzwerkkarte ist in der Regel viel zu klein[1], als daß Daten dort längere Zeit zwischengespeichert werden könnten, so daß sich die Applikation erst bei Bedarf ihre Daten (per DMA) aus dem Netzwerkadapter abholt. Ergo müssen die Daten automatisch aus dem Adapter in den Hauptspeicher übertragen werden.
- DMA benötigt physikalische Adressen (Speicherseiten). Daraus folgt direkt, daß der vom Kommunikationssystem verwendete Pufferspeicher **nicht** als virtueller Speicher vom Betriebssystem verwaltet werden darf (dieses lagert nämlich Speicherseiten bei Bedarf aus und wenn eine DMA Operation vom Kommunikationsadapter eine ausgelagerte Speicherseite betreffen würde, hätten wir soetwas wie einen „remote page fault“). Außerdem benötigen DMA Operationen einen physikalisch zusammenhängenden Speicherbereich, was kontraproduktiv zur virtuellen Speicherverwaltung des Betriebssystem ist (das versucht nämlich gerade zusammenhängende Speicherbereiche zu vermeiden).
- Um die systemseitige Bereitstellung von Pufferspeicher zu vermeiden, könnte auch die Applikation den von ihr benötigten Zwischenpuffer anfordern und breitstellen (PrePost von Empfangspuffern). Dies stellt jedoch eine vollkommen neue Kommunikationssemantik dar (kaum eine Applikation kennt a priori ihren zur Laufzeit benötigten Empfangspufferspeicher; sie kennt nur Situationen in denen etwas empfangen werden kann oder muß).
- Die beliebige Ausrichtung von Daten in Bezug auf Speicherwortbeite (Datenalignment) wird von DMA-Bausteinen in der Regel nicht unterstützt und kann auch programmiertechnisch nicht umgangen/gelöst werden.

An der Lösung dieser Problematik durch entsprechende Kooperation zwischen Netzwerkhardware (Myrinet) und Betriebssystem wird derzeit im Rahmen des ParaStation2 Systems gearbeitet. Erst in ParaStation3 wird es Mechnismen geben, die einen Ohne-Kopie Sender aktiv unterstützen.

[1] Bei einem Gbit/s Netzwerk müssen im Extremfall pro Sekunde 125 MByte Daten gepuffert werden !!

# ParaStation3: Ideales HW-SW Interface

Ziel: flexible Hardware, großes Funktionsspektrum, geringe Kosten

**ParaStation3: Paketaufbau**


- ID: Sende- & Empfangsendpunkt sowie Typkennung

  = Basissupport für gesicherte Endpunkte

- Trennung Protokoll- & Benutzerdaten

  = Basissupport für Zero-Copy-Protokolle

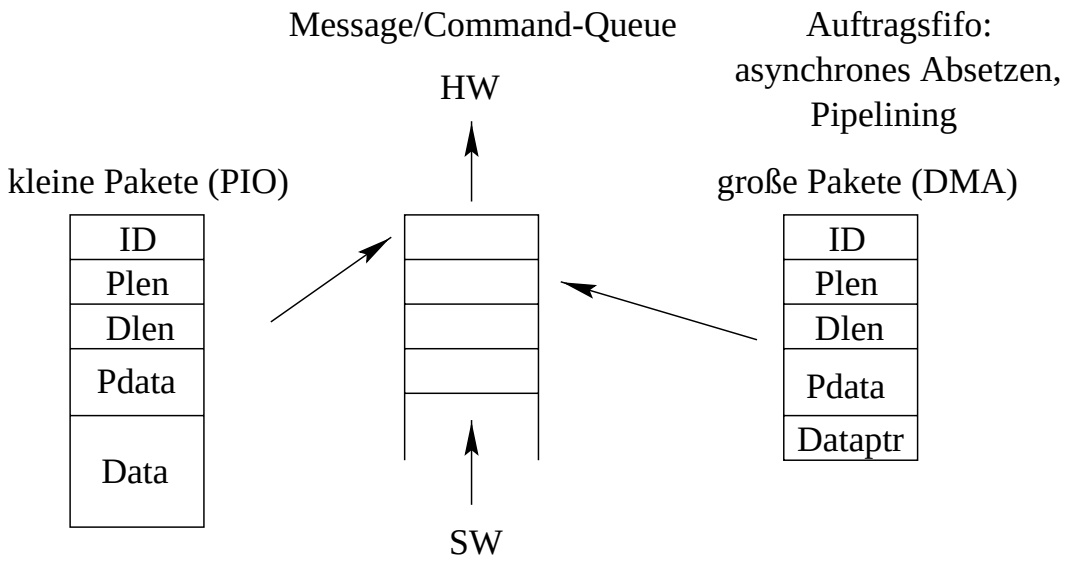


ParaStation3 soll Ohne-Kopie-Protokolle sowie einen sicheren Multiuserbetrieb aktiv unterstützen.

Zur Unterstützung von gesicherten Kommunikationsendpunkten trägt ein ParaStation3 Paket eine Kennung (`ID`), in der Sende- und Empfangsendpunkt sowie ein Pakettyp kodiert sind. Damit kann jedes Paket eindeutig einer Kommunikationsverbindung zugeordnet werden (Sende- und Empfangsendpunkt) und innerhalb dieser Kommunikationsverbindung sind logische Kanäle (mittels der Pakettypen) denkbar.

Desweiteren sind Protokoll- (`Pdata`) und Benutzerdaten (`Data`) streng getrennt, um die Implementation von Ohne-Kopie Protokollen zu erleichten. Beim Empfang einer Nachricht in den Adreßraum einer Applikation erwartet diese nämlich nur die Daten und nicht die Kontrollinformation in ihren Datenfeldern.

# ParaStation3: Sendeinstanz

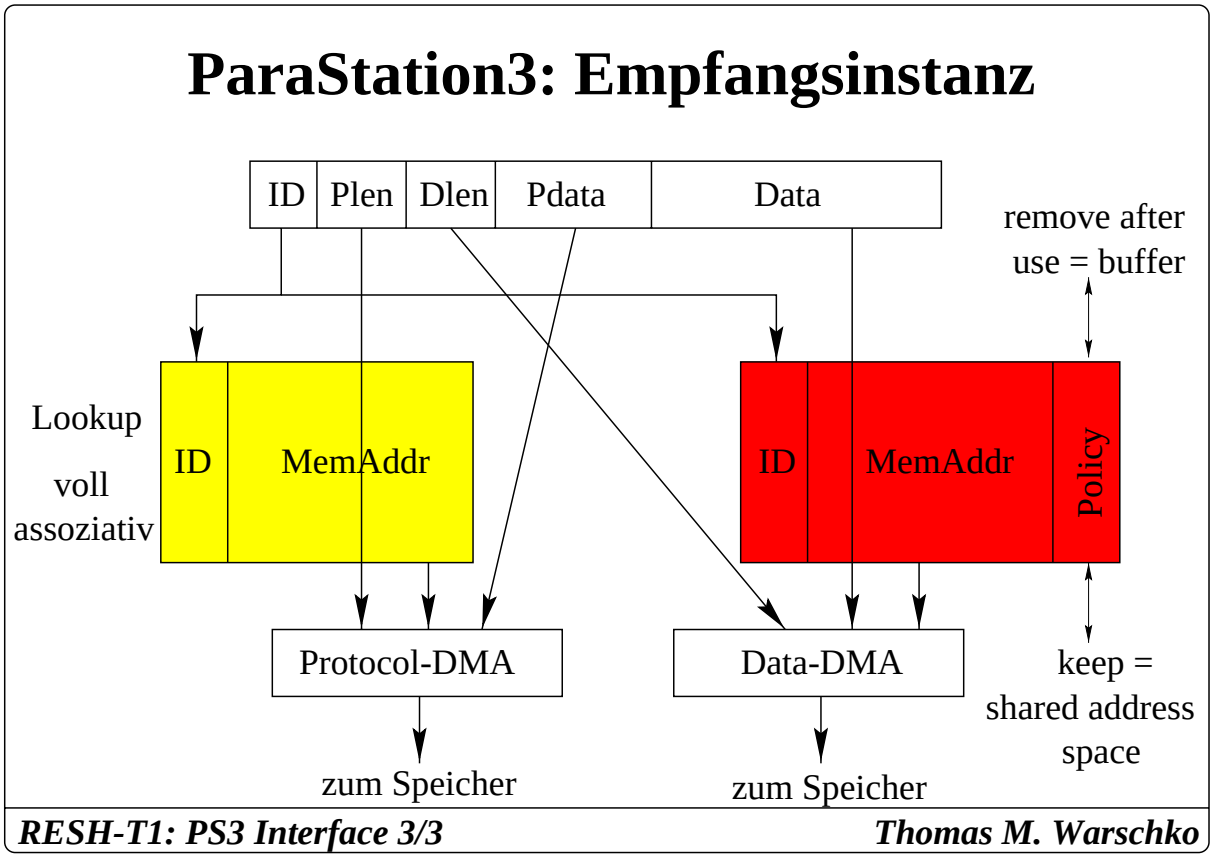



Mehrere M/CQs: Hardwareunterstützung für Multiprozeßumgebung



Die ParaStation3 Paketstruktur spiegelt sich direkt in der Auslegung der Sendeschnittstelle wider. Eine Applikation „sieht“ lediglich eine sog. *Message/Command-Queue*, in die sich mehrere Aufträge einreihen lassen. Ein einzelner Auftrag besteht aus einem Paketkopft gefolgt von einer (kleinen) Nachricht mit Benutzerdaten oder gefolgt von einer Speicheradresse, von der aus sich die Hardware die Daten per DMA laden kann.

Die Hardware selbst stellt mehrere dieser Message/Command-Queues zur Verfügung (im Idealfall eine pro Applikation), wodurch ein gesicherter Betrieb im User-Level gewährleistst werden kann. Einzig kritische Stelle sind die Speicherreferenzen zum Anstoßen der DMA. Über einen „Anmeldemechanismus“, der zweifelsohne stattfinden muß (nur das Betriebssystem kann die Adreßumrechnung für die DMA vornehmen), kann aber auch diese vermeindliche Sicherheitslücke geschlossen werden.




Der eigentliche Kern – sprich die Neuerungen – von ParaStation3 stecken in der Emfängerinstanz. Auch hier findet eine strenge Trennung von Protokoll- und Benutzerdaten statt.

Wird ein Paket empfangen, so wird mittels der PaketID in einem Assozoativspeicher die Adressen zum Ablegen der Protokoll- und Benutzerdaten ermittelt. Sind mehrere Einträge für eine ID vorhanden, so wird der erste Eintrag (Fifo Semantik) verwendet. Dieser Mechanismus ermöglicht die Realisierung von Ringpuffern, bei denen sowohl die Hardware also auch die Software immer den nächsten gültigen Puffer kennt. Sind die Adressen bekannt, dann treten die DMA-Einheiten in Aktion. Zuerst werden die Benutzerdaten an die ermittelte Adresse übertragen und danach die Protokolldaten (so kann die Applikation ihre Daten erst mit dem Erhalt der Protokollinformation sehen, also nachdem die Daten übertragen wurden).

Mittels einer kleinen Erweiterung des Zwischenspeichers für Datenadressen kann man sogar einen gemeinsamen Adreßraum realisieren. Wird ein Eintrag nach Gebrauch nämlich nicht aus dem Assozoativspeicher gelöscht (normaler Fall bei einem Ringpuffer), dann schreibt das System Datenpakete mit gleicher ID immer an dieselbe Speicheradresse. Da die Verwaltung der Protokolldaten davon unabhängig ist (hier wird weiterhin ein Ringpuffer verwendet), kann der logische Datenfluß problemlos mitverfolgt werden.

Einzig (sicherheits)kritischer Punkt beim Empfänger sind analog zum Sender die Speicherreferenzen für die DMA. Aber auch hier gilt dieselbe Argumentation, wie beim Sender. Da nur das Betriebssystem die Umrechnung von virtuellen auf physikalische Adressen vornehmen kann, können bei diesem Vorgang entsprechende Maßnahmen getroffen werden, die einen sicheren Betrieb gewährleisten.

Die ParaStation3 Hardware wird derzeit in Kooperation mit der Universität Mannheim entwickelt; die vorgeschlagene Strategie kann aber auch mittels der Myrinet Hardware emuliert werden. Dies wird in der Folgezeit geschehen.

## Inhalt

- OS Integration: Wo liegt sinnvollerweise die OS/Benutzer Schnittstelle?
- Programmierschnittstellen
- Lösungsansätze zu Problemen der User-Level Kommunikation
- ParaStation Single System View
- Myrinet Portierung
- Weitere Arbeiten

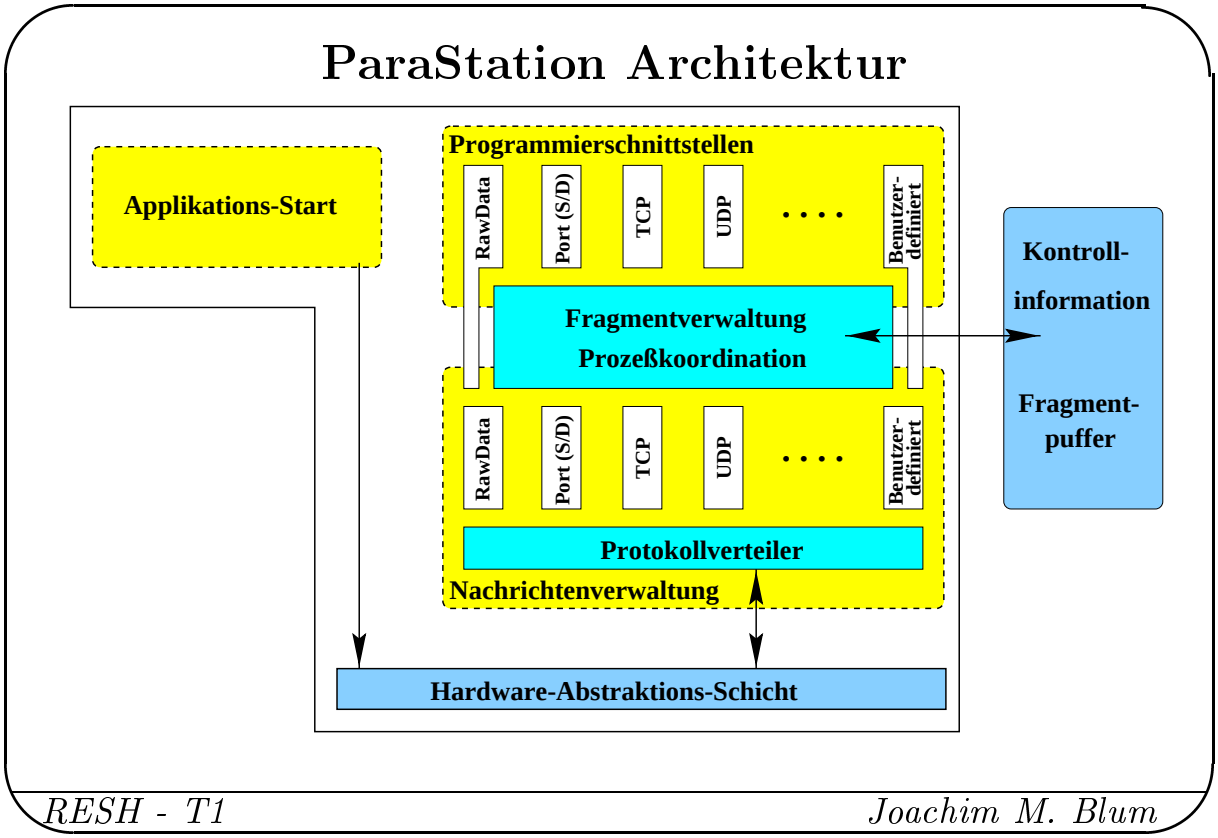
*RESH - T1* *Joachim M. Blum*

Bei Projektstart stand die Software in einem nutzbaren Zustand den Projektteilnehmern zur Verfügung. Durch die intensive Benutzung wurden allerdings ein paar wenige Schwachstellen erkannt. Diese Schwachstellen konnten zum Teil schon eliminiert werden.

Da bei ParaStation die Hardware offen im Benutzerbereich eingeblendet wird, kam es am Anfang des Jahres zu mehrere Abstürzen, die durch starke Belastung der Java Kommunikation ausgelöst worden sind. Als Lösung wurde daraufhin die Möglichkeit geschaffen, den Zugriff auf die Hardware wieder in den Kern zu integrieren. Dabei wurde festgestellt, daß die Latenz hierbei um ca. 5 $\mu$s auf ca. 15 $\mu$s bei der Socketkommunikation zunahm. Der befürchtete *Performancecrash* blieb somit aus. Dies bedeutet, daß man sich die erhöhte Sicherheit durch schlechtere Kommunikationsleistung erkaufen kann. Es wirft aber auch die Frage auf, ob *User-Level*-Kommunikation in dieser Form der Protokollschnittstellen überhaupt sinnvoll ist. Hierzu soll in Zukunft untersucht werden, wo die beste Applikation- Betriebssystem-Hardwareschnittstelle liegt.

Bezogen auf der ParaStation Software Architektur kann ich mir sehr gut vorstellen, daß nur noch die Schnittstelle zur Applikation im Benutzeradressbereich liegt. Allerdings soll die Implementierung so flexibel gehalten werden, daß auch jede andere Position möglich sein wird, damit man die Vor- und Nachteile jeweils evaluieren kann.

Mit dieser Integration der ParaStation Software in das Betriebssystem kommen wir dem Ziel einer binärcode transparenten Schnittstelle der ParaStation näher.




Auf dieser Abbildung kann man die Architektur der ParaStation Software erkennen. Dieses komplette Kommunikationssubsystem ist als Bibliothek implementiert, die man beim Bindevorgang statisch an die Prozesse bindet. Für die Javaschnittstelle wird auch eine dynamisch ladbare Bibliothek zur Verfügung gestellt.

Die Hardware- Abstraktionsschicht verbirgt die architektur- und betriebssystemabhängigen Teile der Bibliothek vor den Implementierungen der einzelnen Protokolle. Sie bietet eine sehr einfache Schnittstelle, die nur Basissende- und Empfangsoperationen zur Verfügung stellt.

Diese Hardware- Abstraktionschicht wurde in den letzten Monaten von der Bibliothek in den Treiber der ParaStation verschoben. Somit besteht nun die Möglichkeit, daß die Bibliothek nur über das Betriebssystem auf die Hardware zugreift. Bei Wahl dieser Zugriffsart ist ein sicherer Zugriff gewährleistet.

In Zukunft soll die Bibliothek- Betriebssystem- Schnittstelle an verschiedenen Stellen implementiert werden. Interessant sind v.a

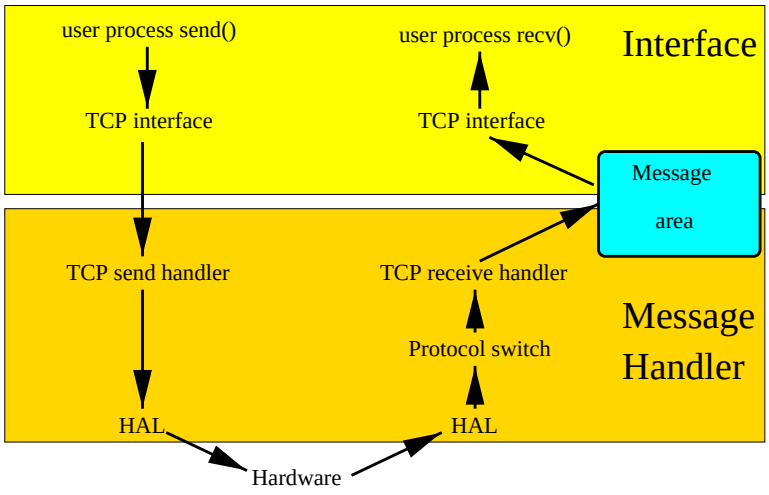
- Schnittstelle zwischen Nachrichtenverwaltung und dem Fragment-Puffer.
  Dies würde ermöglichen, daß ein prozeßspezifischer Fragment-Puffer verwendet werden könnte, was einen vollständigen Schutz zwischen den Prozessen ermöglichen würde. Insgesamt würde die Nachrichtenverwaltung die Fragment-Puffer aller Prozesse sehen, aber die Prozesse immer nur ihren eigenen.
- Schnittstelle zwischen Programmierschnittstelle und Applikation.
  Dies ist die Stelle, die in *normalen* Betriebssystemen verwendet wird. Der komplette Protokollstack liegt bei dieser Variante im Kern. Somit sind aber alle Aufrufe in die Bibliothek Aufrufe in den Kern, was zu erheblichen Leistungseinbußen führen kann. Die Applikation hätte keine Zugriff mehr auf den Fragment-Puffer. Dies würde jede Möglichkeit einer Korruption der Daten durch die Applikations elliminieren.
- Unscharfe Schnittstelle:
  Die Applikation behält Zugriff auf den Fragment-Puffer. Zudem werden manche Operationen im

Benutzerbereich und anderen Operationen im Kernbereich ausgeführt. Somit wird verhindert, daß man bei einfachen Informationsabfragen einen Adreßraumwechsel ausführen muß, was sich durch kürzere Antwortzeit bemerkbar macht. Jedoch hat man trotzdem die Gefahr, daß die Applikation Daten im Fragment-Puffer zerstört. Jedoch beschränkt sich diese Zerstörung nur auf eigene Daten.

Alle diese möglichen Alternativen sollen bewertet werden. Es ist aus heutiger Sicht allerdings noch nicht zu erkennen, ob sich die Verschiebung der Schnittstelle zur Applikation possitiv oder negativ auf das Laufzeitverhalten der Applikation auswirkt.

Ich erwarte allerdings, daß sich die bisher gemessene Verschlechterung der Latenzzeit von ca. 5 $\mu$s nicht viel verschlechtern wird, wenn man die Betriebssystemschnittstelle von der HAL zur API verschiebt. Diese Verschlechterung wäre für die meisten Anwendungen akzeptabel, da einhergehend ein deutlicher Schutz gewonnen wird. Für hochwertige Protokolle wie Java RMI werden die verlorenen Mikrosekunden keine entscheidene Rolle spielen. Jedoch muß auch in Zukunft die Möglichkeit eines Protokolls, das keine Bremsstrecken beinhaltet, in der ParaStation Architektur weiter erhalten bleiben.

## Beipiel: TCP Nachrichten Übertragung




Jede Applikation erhält
nur seine eigene Nachrichten
von der Schnittstelle

Message Handler
verteilt Nachrichten
für jede Applikation
Er läuft in:

- low level Software
- OS
- Kommunikations HW

*RESH - T1* *Joachim M. Blum*

Hier kann man erkennen, wie der Nachrichtentransport in der ParaStation Software implementiert ist. Die zwei wichtigsten Bestandteile sind hierbei die Programmierschnittstelle, die die semantische Lücke zwischen der von der Applikation benutzten API und den Datenstrukturen der ParaStationb Bibliothek schließt, und die Nachrichtenverwaltung (Message Handler), die die Dienste, die ein Protokoll laut Spezifikation zur Verfügung stellen muß, implementiert.

Hierbei kann der Nachrichtenverwaltung in der *low-level* Software, im Betriebssystem oder in der Kommunikationshardware ablaufen:

- **Low-Level Software**:
  Der Messagehandler läuft in der derzeitigen Implementierung in einer unteren Schicht der Software ab. Durch die Multiprozeßfähigkeit der ParaStation Software kann es allerdings passieren, daß ein Prozess Nachrichten für einen anderen Prozess empfängt. Wenn dies passiert, werden diese Nachrichten transparent für den Prozess in den Nachrichtenspeicher zwischengespeichert. Durch diese

Hilfestellung von anderen Prozessen benötigen diese Prozesse allerdings Schreibrechte auf Speicherplätze des Adressaten. Um dies zu ermöglichen haben alle Prozesse Zugriff auf einen *gemeinsamen Fragment-Puffer* (SHared Memory), in dem sie die nicht empfangenen Nachrichten abspeichern. Dies eröffnet allerdings eine Sicherheitslücke, die in [3] beschrieben ist.

- **Betriebssystem**:
  Das Betriebssystem ist eigentlich der übliche Platz um die ankommenden Nachrichten den entsprechenden Empfängern zuzuordnen. Durch die Einführung von User-Level Kommunikation kam das Betriebssystem allerdings in den Ruf zu langsam zu sein. Wie oben beschrieben, glaube ich nicht, daß dieser Ruf gerechtfertigt ist. Wenn man das Protokoll entsprechend implementiert, wird es auch performant sein.

- **Hardwareunterstützung**
  Mit der Verwendung der Myrinet Kommunikationskarte ist es nun möglich, diese Idee der Ausführung des Nachrichtendemultiplexens in der Hardware weiter zu verfolgen, da die Kommunikationshardware nun einen frei programmierbaren Prozessor besitzt, der wie in [3] beschrieben, die ankommenden Nachrichten an die applikationsspezifischen Nachrichtenpuffern weiterleiten kann. Somit würde, wie in [1] beschrieben, die Sicherheitslücke *gemeinsamer Fragment-Puffer (SHM)* bereinigt.

Sobald wir den Nachrichtendemultiplexer im Betriebssystem und in der Hardware implementiert haben, können wir die Designalternativen vergleichen und Schlüsse für das Software- Hardware Schnittstelle der neuen ParaStation3 Hardware ziehen.

Für alle diese Designalternativen sieht der Nachrichtentransport aber identisch aus. Hier werden nun die zeitliche Abfolge und die Funktionen der einzelnen Komponenten erläutert:

1. Der Sendeprozeß führt eine Sendeoperation aus.
2. In der Programmierschnittstelle wird der Aufruf in entsprechende Nachrichtenhandleraufrufe umgesetzt. Es kann hierbei vorkommen, daß ein Schnittstellenaufruf mehrere Handleraufrufe generiert. Jedes Protokoll implementiert hierbei seine protokollspezifischen Dienste (Adressumsetzung, Fragmentierung, usw.). Es kooperiert hierbei eng mit dem ReceiveHandler der Empfangsinstanz.
3. Der Handler übergibt die Nachrichten an die Hardware, die die Daten an den Empfänger weiterleitet.
4. Am Empfänger überpruüft ein Protokollverteiler, für welches Protokoll das entsprechende Nachrichtenfragment bestimmt ist und leitet es an die protokollspezifischen Empfangshandler weiter.
5. Der protokollspezifische Empfangshandler sucht die entsprechenden Zieladresse und speichert das Fragment in den Nachrichtenbereich ab. Hierbei entscheidet das Protokoll auf welche Art und Weise abgespeichert werden soll( Einfach-/Multistromorientiert oder Nachrichtenorientiert). Die Nachricht bleibt nun im Nachrichtenspeicher, bis der Empfängerprozeß eine entsprechende Nachricht empfangen möchte.
6. Der Empfängerprozeß ruft eine Empfangsroutine auf. Diese überprüft, ob eine entsprechende Nachricht schon im Nachrichtenspeicher vorliegt und übergibt diese gegebenenfalls. Wenn keine Nachricht vorliegt, wartet er bis eine Nachricht ankommt. Das Warten wird durch das ParaStation Coscheduling koordiniert.

## Coscheduling

Bei User-Level Kommunikation ohne Interrupts muß man nach neuen Nachrichten an der Hardware *pollen* .

**Problem:** CPU Zyklen gehen ungenutzt verloren. Andere Prozesse werden behindert

**Lösung:** ParaStation Coscheduling übergibt die CPU an andere Prozesse/Fäden, wenn es keine sinnvolle Arbeit gibt.

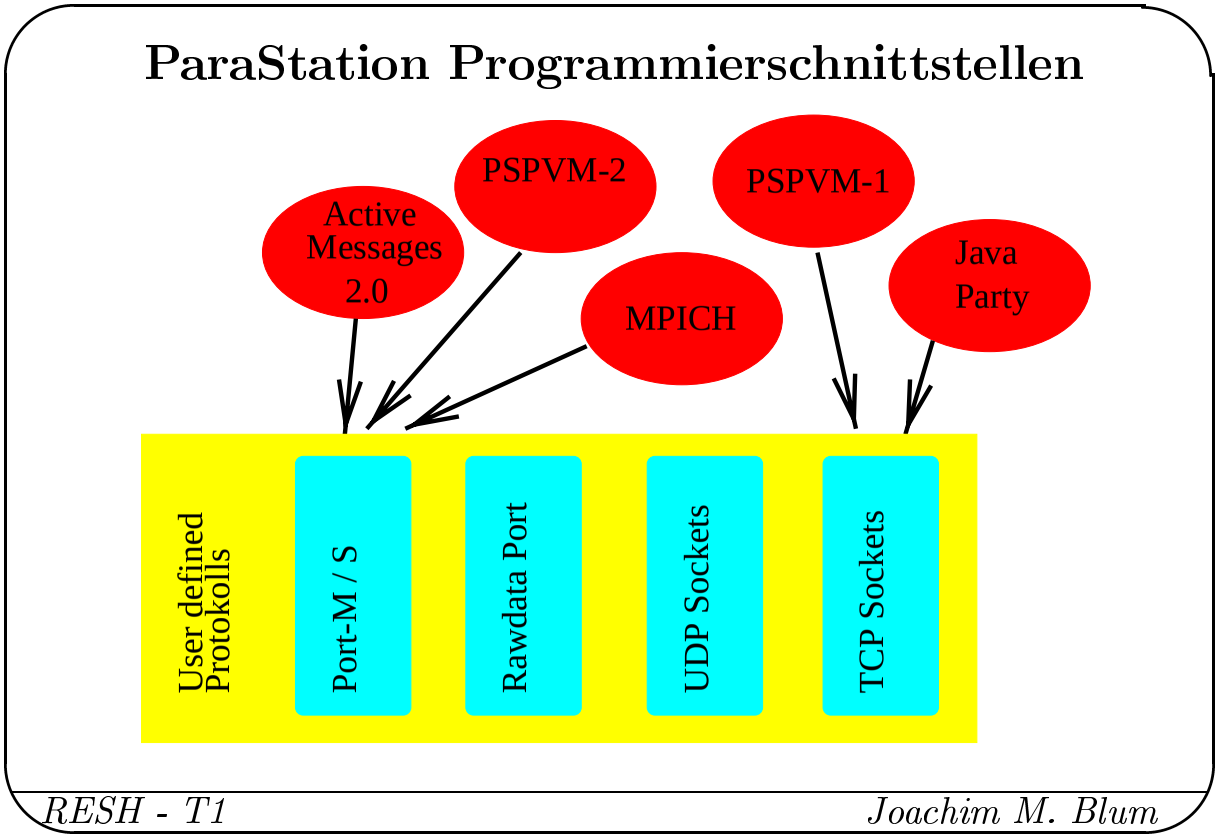
**Strategien:**

- sleep: Selbstaufgabe für eine bestimmte Zeit
- dynamic: übergabe an einen anderen ParaStation Task
- sched_yield: Suspendiert den eigenen Thread
- thread_switch: Posix `thread_switch()` Aufruf
- local_send: ruft Coschedule() nach jedem lokalen Senden auf.



Die ParaStation Software war das erste System, das bei *User-Level* Kommunikation mehrere Prozesse pro Knoten zuließ. Schon bald wurde erkannt, daß durch die fehlende Betriebssysteminteraktion Prozesse, die auf eine Nachricht warten, unnötig CPU Zyklen verschlingen. Dieser negative Effekt wurde daraufhin von uns in der ParaStation Software durch verschiedene Coscheduling Strategien gemildert. Obige Abbildung zeigt die verschiedenen Verfahren. Jedoch mußte bisher leider festgestellt werden, daß für multithreaded Applikationen, andere Strategieen als für singlethreaded Applikationen sinnvoll sind.

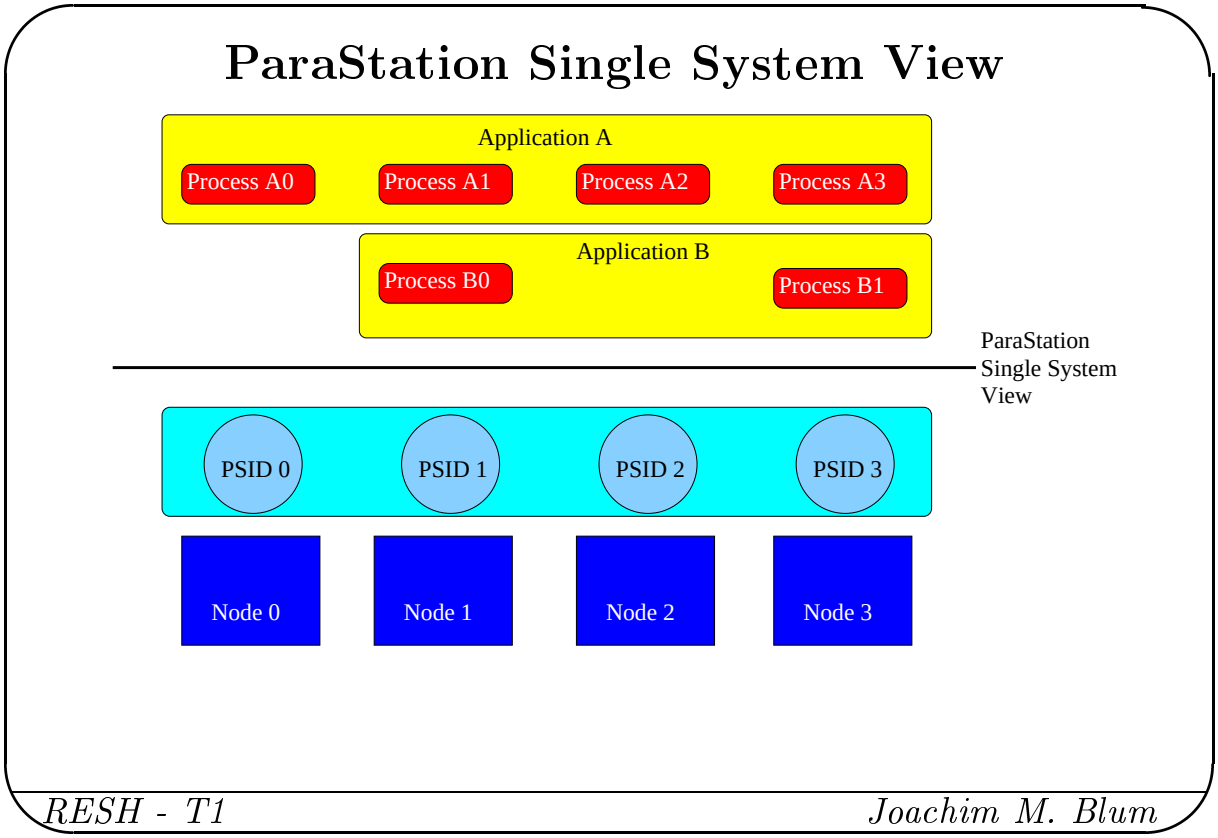
Insbesondere durch den Applikationsmix, der auf dem RESH-Cluster ausgeführt wird, müssen hier neue Verfahren entwickelt werden. Ein Problem stellt hierbei im Moment die Granularität des ParaStation Coscheduling dar. Obwohl die Einheit des Scheduling im Betriebssystem (Digital Unix) Threads sind, verwendet ParaStation nur Tasks. Hier müssen in Zukunft das ParaStation System Wissen über Threads erhalten. Durch diese Erweiterung erwarte ich eine Laufzeitverbesserung für einen Mix aus verschiedenen Applikationen.




Diese Abbildung zeigt einen kurzer Überblick über die Schnittstellenvielfalt der ParaStation Software und erläutert, auf welche interne Protokolle höherwertige Protokolle, wie PVM [24], MPI [10] und Java, aufsetzen. Es ist eindeutig ein Trend in Richtung ParaStation Ports [1] zu erkennen, da diese Schnittstelle eine deutlich größere Funktionalität bietet, als die standardisierten Unix-Sockets. Die Unix-Sockets bieten sich aber als schnelle Lösung immer an, da Portierung von existierenden Paketen sehr einfach ist.

Dank der Möglichkeiten der *Java Virtual Maschine* ist es uns gelungen, Java Applikationen vollkommen transparent über das ParaStation System kommunizieren zu lassen. Java nutzt hier die Möglichkeit, daß die ParaStation Sockets Unix-Sockets Aufrufe überladen. Da Java interpretiert wird, ist ein erneutes binden, wie bei C-Programmen, nicht mehr nötig.

Eine Weiterentwicklung wird die Ports-Schnittstelle in Java integrieren und somit die Möglichkeit schaffen, die volle Funktionalität des ParaStation Systems innerhalb von Java zu nutzen.




Ein weiteres Themengebiet für die ParaStation Software war die Erweiterung des ParaStation Single System Views. Die Dämonen verfügen nun über Lastinformationen der anderen Knoten und können somit bei der Erstellung neue Prozesse die Information verwenden, um den Zielknoten zu wählen. Außerdem wurde die Möglichkeit geschaffen das Cluster zu partitionieren. Hierzu kann ein Benutzer wählen, welche Knoten er für seine Applikationen verwenden möchte. Beim Erzeugen neuer Prozesse werden nur diese Knoten in Betracht gezogen.

# Leistungsdaten der ParaStation Protokolle auf Myrinet

| protocol-layer | Alpha 21164, 600 MHz ParaStation latency [$\mu s$] | ParaStation bandwidth [MB/s] | OS/Ethernet latency [$\mu s$] | OS/Ethernet bandwidth [MB/s] |
|---|---|---|---|---|
| HAL | 21.1 | 64.4 | | |
| rawdata | 22.9 | 62.0 | | |
| port-M | 27.3 | 53.7 | | |
| socket | 27.1 | 53.4 | 95 | 1.1 |
| PVM | 90.0 | 44.3 | 249 | 1.0 |
| PVM (port-M) | 30.2 | 45.5 | | |
| socket (self) | 2.5 | 930.2 | 190 | 54.0 |

Die Kommunikationsbibliothek ist bisher nur portiert und nicht optimiert auf Myrinet.

*RESH - T1* *Joachim M. Blum*

Neben den oben beschriebenen Erweiterungen der Software, wurde das ParaStation System auf eine neue Kommunikationshardware portiert. Dank der Hardware Abstraktionsschicht (HAL), die im Hardwareteilprojekt von T1 (siehe 2.3.1), umgestellt wurde, waren die Änderungen im kompletten Rest des ParaStation Systems nur minimal. Die Leistungszahlen in der obigen Abbildung verdeutlichen, daß es gelungen ist eine Portierung zu machen, jedoch die erhofften Leistungsdaten noch nicht erreicht werden können. In naher Zukunft wird die HAL Implementierung optimiert und gewisse Änderungen in der HAL-Schnittstelle vorgenommen. Mit diesen Änderungen erhoffen wir uns wieder eine TCP Latenz von unter 20 $\mu$s.

# NAS Parallel Benchmark Suite 2.0 auf ParaStation-1 Hardware

| NAS Parallel Benchmark auf ParaStation-1 and T3E | | | | |
|---|---|---|---|---|
| Test on no. of nodes Class A | 1 | 2 | 4 | 8 |
| BT ParaStation | n/a | n/a | 144.4 | n/a |
| BT Cray T3E-900 | n/a | n/a | 226.7 | n/a |
| CG ParaStation | 19.7 | 44.5 | 55.15 | 75.72 |
| CG Cray T3E-900 | | 46.5 | 86.0 | 241.4 |
| EP ParaStation | 4 | | | 31.96 |
| EP Cray T3E-900 | | 5.2 | 10.4 | 20.8 |
| IS ParaStation | 1.46 | 2.23 | 2.15 | 3.72 |
| IS Cray T3E-900 | | 6.6 | 12.9 | 22.1 |
| LU ParaStation | | | | 579.48 |
| LU Cray T3E-900 | | 134.4 | 270.4 | 531.1 |
| MG ParaStation | | | | 299.77 |
| MG Cray T3E-900 | | 171.5 | 313.9 | 720.8 |
| FT ParaStation | | | | 86.02 |
| FT Cray T3E-900 | | 85.3 | 169.5 | 330.4 |
| SP ParaStation | | n/a | 106.55 | |
| SP Cray T3E-900 | n/a | n/a | 172.4 | n/a |

*RESH - T1* *Joachim M. Blum*

Um einen besseren Vergleich des ParaStation Systems mit kommerziellen Parallelrechner zu bekommen, wurde der NAS Parallel Benchmark auf der ParaStation ausgeführt und mit dem schnellsten Parallelrechner (Cray T3E) verglichen. Wie die obige Abbildung zeigt, braucht das ParaStation System den Vergleich nicht zu scheuen. Bei Tests, die einen hohen Durchsatz verlangen, schneidet die alte ParaStation-1 Hardware erwartungsgemäß schlechter ab. Bei Tests, die latenzzeitkritisch sind, scheint das ParaStation System, das nur einen Bruchteil der T3E kostet, ebenbürtig zu sein.

## Weitere Arbeiten

- Neue Plattformen: Sun (Solaris), Alpha (Linux)
- Optimierung für die Myrinet Hardware: z.B. Zero/One-Copy
- OS Integration:
  - Wo muß die Benutzer/OS/HW Schnittstelle liegen
  - Gang scheduling
  - Erweiterung des Single System Views (z.B. paralleles Dateisystem)
  - Transparente Schnittstelle
- Unterstützung eines optimierten JavaPartys auf ParaStation



Als Abschluß sollen noch die geplanten Arbeiten für die nächsten Monaten vorgestellt werden. Da die neuesten Java Entwicklungen immer zuerst auf Sun-Solaris verfügbar sind, werden wir eine Portierung auf das System vornehmen. Hierbei erwarten wir jedoch wieder die Hauptarbeit bei der HAL- und Treiberentwicklung. Der komplette Rest wird wieder ohne großen Aufwand zur Verfügung gestellt werden können.

Wie schon in der Abbildung auf Seite 22 verdeutlicht, werden Optimierungen v.a. in der HAL und in der Kooperation mit dem Rest der Software wichtige Rollen bei der weiteren Entwicklungen der ParaStation sein.

Bei der OS Integration wird als erstes der Hauptaugenmerk auf einer idealen Platzierung der Applikations-Betriebssystemschnittstelle liegen. Mit der Integration der ParaStation Software in das Betriebssystem werden auch andere Punkte, wie *Gang Scheduling*, Erweiterung des *Single System Views* und transparente Schnittstellen vereinfacht.

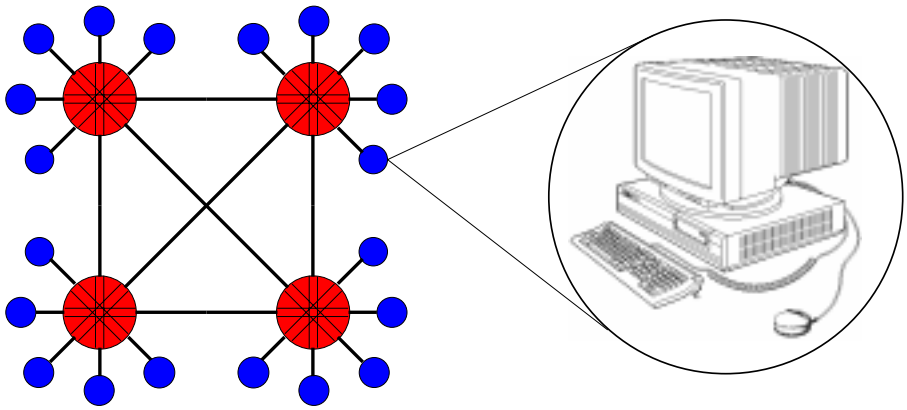
Weiterhin soll natürlich im RESH Projekt die Zusammenarbeit aller Teilprojekte weiter untestützt werden.

# Forschergruppe RESH

Rechnernetze als Superrechner und Hochleistungsdatenbanken

## ParaStation™ 2

### Effiziente Parallelverarbeitung auf der Basis gekoppelter Einzelplatzrechner



- Aufbau des Kommunikationsnetzes mittels PCI-Bus Einsteckkarten (1.28 Gbit/s Myrinet)
- Skalierbarkeit von 2 bis über 100 Stationen
- Leistung durch minimalen Protokoll- und Betriebssystem-Overhead, kurze Latenzzeiten und durch hohen Durchsatz
- Portabilität durch Einsatz standardisierter Programmier-umgebungen (PVM, MPI, Unix-Sockets, Java-RMI)

**Ansprechpartner / Kontakt:**

IPD Prof. Tichy
Universität Karlsruhe
Am Fasanengarten 5
D-76131 Karlsruhe
GERMANY

Tel: +49/721/608-4317
Fax: +49/721/608-7343
Email: parastation@ira.uka.de
URL: http://wwwipd.ira.uka.de/RESH
URL: http://wwwipd.ira.uka.de/ParaStation

# 3 T2: Programmierumgebungen und Übersetzer, Anwendungsunterstützung

## 3.1 Ausgangssituation

Es gibt auf bestimmten Parallelrechnern umfangreiche Anwendungsprogramme, die in hersteller- oder architekturspezifischen Programmierumgebungen erstellt worden sind. Diese sind derzeit nur schwerlich auf andere Parallelrechner und erst recht nicht auf Rechnerbündel zu portieren.

Um trotz der Architekturunterschiede ein gewisses Maß an Portabilität und damit Investitionssicherung zu erreichen, finden standardisierte Kommunikationsbibliotheken in neueren parallelen Anwendungsprogrammen Verwendung.

Diese Kommunikationsbibliotheken sind jedoch der kleinste gemeinsame Nenner und haben auf Parallelrechnern zwei wesentliche Nachteile: Erstens geht ein großer Teil der Rechnerleistung bei der Abbildung der Kommunikationsbibliotheken auf die zugrundeliegende Hardware als Reibungsverlust verloren. Zweitens sind aus Sicht des Software Engineerings Kommunikationsbibliotheken ein ungenügendes Programmiermodell, das leider zu oft zu fehlerträchtigen, schlecht wartbaren und unverständlichen Programmen führt.

Im Rahmen von T2 sollen beide Nachteile für die besonderen Randbedingungen von Rechnerbündeln behoben werden. Es soll einerseits eine effiziente und optimierte Abbildung gängiger Kommunikationsbibliotheken auf ParaStation erfolgen, um existierende parallele Anwendungsprogramme kosteneffizienter ablaufen lassen zu können. Andererseits soll eine Programmierumgebung erstellt werden, die es ermöglicht, neue parallele Anwendungsprogramme erheblich einfacher zu erstellen. Statt mit Detailproblemen der parallelen Hardware zu kämpfen, soll sich der Programmierer darauf konzentrieren können, seine Anwendungsprobleme zu bearbeiten.

Zusätzlich soll intensiv mit den Anwendern (Datenbanken und Bildauswertung, Projekte N1, N2 und L1) zusammengearbeitet werden. Diese enge Zusammenarbeit hat zwei Ziele:

- die optimale Unterstützung der Anwender bei der Parallelisierung ihrer Anwendungen sowie bei der Nutzung paralleler Basisbibliotheken bzw. paralleler Programmierumgebung und
- die direkte Rückwirkung auf die Basistechnologien. Durch die Erfahrungen, die aus der aktiven Mitarbeit in den Anwendungsprojekten gewonnen werden, kommt es zu ständigen Verbesserungen der Basistechnologien, die den Anwendern wieder zur Verfügung gestellt werden können.

## 3.2 Ziele

In **Teilprojekt A/Programmierumgebungen für Rechnerbündel** sollen bereits bekannte Kommunikationsbibliotheken daraufhin untersucht werden, welche Optimierungsmöglichkeiten sich zur Implementierung der Bibliothek aus den Eigenschaften eines Rechnerbündels im allgemeinen und aus den Eigenschaften der ParaStation im speziellen ergeben. Ferner wird untersucht, ob und in welchem Umfang weitergehende Verbesserungen der Kommunikationsleistung durch Verlagerung von Protokollfunktionen in Hardware prinzipiell möglich sind. Die frühe Verfügbarkeit und ständige Verbesserung bekannter Kommunikationsbibliotheken ist ein Beitrag, auf dem andere Arbeiten dieses Projekts (Projekte L1, N1, N2) aufbauen können.

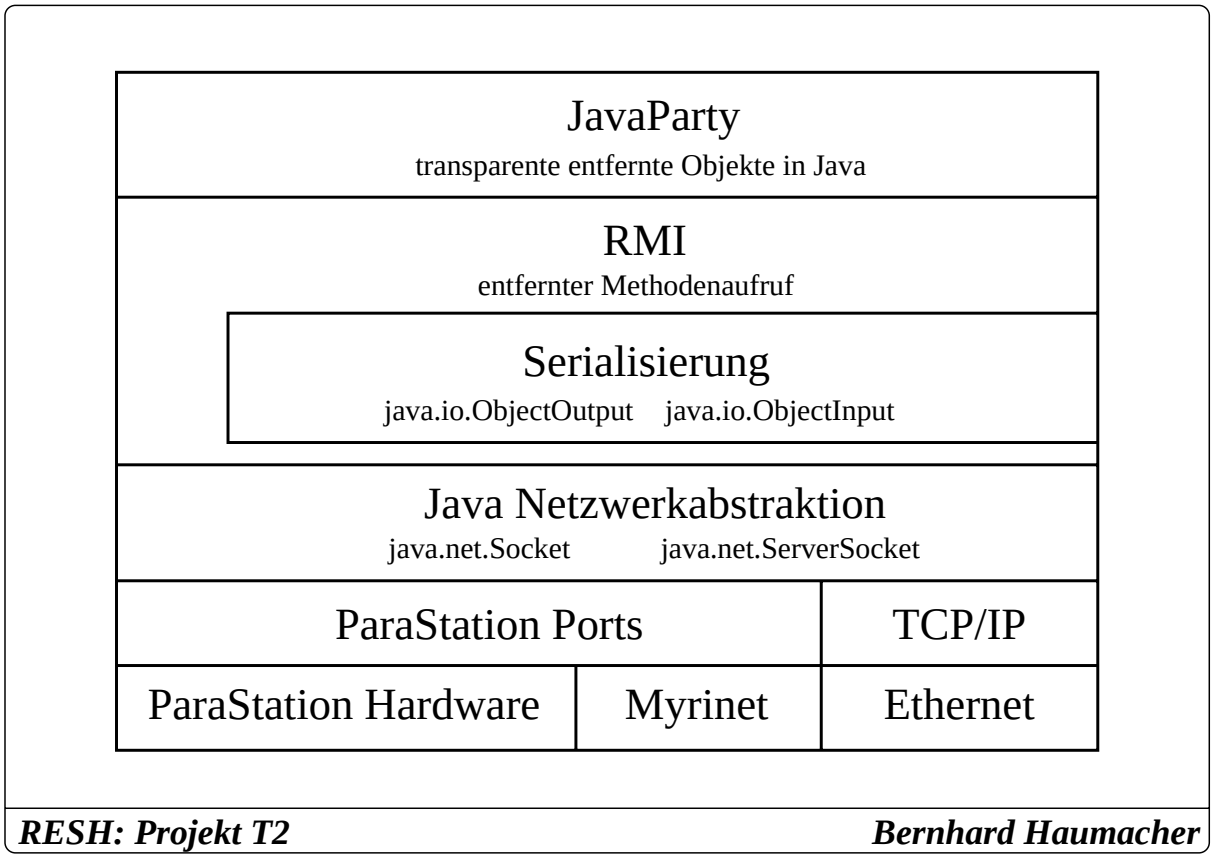
In **Teilprojekt B/Allzweck-Parallelprogrammierung von Rechnerbündeln** soll JavaParty, ein verteiltes Java für Netzwerke von Arbeitsplatzrechnern, die Allzweck-Parallelprogrammierung ermöglichen. Für JavaParty sollen prototypische Implementationen von Übersetzern und von Systemen zur Programmier- und Fehlersuchunterstützung erstellt werden, wobei die besonderen Anforderungen zu erforschen und zu berücksichtigen sind, die sich bei Netzwerken von Arbeitsplatzrechnern ergeben.

JavaParty wird zur erleichterten Allzweck-Parallelprogrammierung von Netzwerken von Arbeitsplatzrechnern führen und dabei Grundlagenfragen lösen, die aus dem Bereich der Programmiersprachen, des Übersetzerbaus und der Software-Technik stammen.

Zur Validierung der Ergebnisse unserer Untersuchungen planen wir die Durchführung praxisrelevanter und vergleichender Experimente sowohl mit den übertragenen Programmiermodellen als auch mit den prototypischen Werkzeug-Implementationen der neuen Techniken. Dies setzt eine enge Zusammenarbeit mit den Anwenderprojekten L1, N1 und N2 voraus.

In **Teilprojekt C/Anwenderunterstützung und Rückkopplung zur Basistechnologie** soll die Zusammenarbeit mit den Anwendern zu einer optimalen Unterstützung bei der Parallelisierung führen. Die Anwendungsprojekte sollen möglichst schnell und effizient auf ParaStation parallelisiert werden. Als Rückfluß aus der Zusammenarbeit erhoffen wir uns Erkenntnisse für die Weiterentwicklung der ParaStation-Hardware. Wenn die Spezifikationen der Hardware festliegen, werden optimale Schnittstellen und Protokolle für die verschiedenen Anwendungsklassen entworfen. Wir erwarten Funktionserweiterungen und neue Erkenntnisse für die Basistechnologien in Bereichen der verteilten Ein-/Ausgabe, der Konsistenzhaltung von verteilten Memory Mappings, der Synchronisation von verteilten Prozessen sowie bei Kommunikationsprotokollen und Schnittstellen, die spezifisch für bestimmte Anwendungsgruppen sind. Besonderes Augenmerk werden hierbei die Kooperationsbeziehungen der lose gekoppelten Betriebssysteme der einzelnen Rechner erhalten.

### 3.3 Aktueller Stand




Java bietet im Sprachumfang bereits umfangreiche Unterstützung für Programmierung mit mehreren Kontrollfäden (*multi-threading*). Echt parallel ausgeführt werden kann ein solches Programm allerdings nur, wenn eine Mehrprozessormaschine mit gemeinsamen Adressraum zur Verfügung steht und die Implementation der virtuellen Maschine die Verwendung mehrerer Prozessoren unterstützt. Für die Programmierung eines Rechnerbündels bietet die Sprache Java noch keine speziellen Hilfsmittel an. Um ein Java-Programm in einer verteilten Umgebung zur Ausführung zu bringen, stehen im Prinzip mehrere Wege offen.

- Umsetzung der notwendigen Netzwerkkommunikation in explizite Kommunikationsanweisungen mit Hilfe der Java Socket-Bibliothek.
- Verwendung der RMI-Bibliothek [31], die einen entfernten Methodenaufruf für Java zur Verfügung stellt.

Wie die Untersuchung [27] zeigt, erfordern beide Ansätze einen nicht vertretbaren Änderungsaufwand in einem Programm, das bereits für SMP Rechner parallelisiert wurden.

Als Alternative stellt JavaParty transparente entfernte Objekte zur Verfügung. Diese entfernten Objekte besitzen weitgehend Java-Objektsemantik [15] [27], Methoden dieser Objekte sind aber von allen Knoten der verteilten Umgebung aus aufrufbar. Entfernte Klassen werden lediglich mit einem zusätzlichen Schlüsselwort `remote` gekennzeichnet. Der JavaParty-Übersetzer übernimmt anschließend die Transformation in entsprechenden Java-Code plus RMI.

Die Programmierung von DMP Rechnern vereinfacht sich dadurch dramatisch. In einem Programm mit bereits mehrfädigem Kontrollfluß müssen lediglich diejenigen Klassen markiert werden, die von entfernten Knoten aus referenziert werden sollen. Da die JavaParty-Transformation Methodenaufrufe von entfernten Objekten auf RMI-Aufrufe abbildet, ist die Geschwindigkeit von RMI-Aufrufen ein besonders kritischer Faktor bei der Ausführung von JavaParty-Programmen.

## Wunsch und Wirklichkeit

### hohe Geschwindigkeit eines entfernten Methodenaufrufs

- ermöglicht feingranularere Parallelität
- macht Methodenaufruf als objektorientiertes Pendant zum Message-Passing attraktiv

### Zahlen

| | |
|---|---|
| entfernter Methodenaufruf mit Objektparameter<br>7 Integer Werte ~ 28 Bytes über Ethernet | 3350us |
| Socket ping-pong, ParaStation, 28 bytes | 25us |

***RESH: Projekt T2*** ***Bernhard Haumacher***

JavaParty wurde bereits im Rahmen einer vergleichenden Studie (Java versus Fortran90, HPF) erfolgreich zur parallelen Implementierung einer geophysikalischen Applikation[13] eingesetzt.

Bei Problemen mit feingranularerer Parallelität spielt dagegen die Geschwindigkeit des entfernten Methodenaufrufs eine immer größere Rolle. Mit der derzeitigen ParaStation-Implementierung [35] erreicht man Zeiten für den Nachrichtenaustausch zwischen zwei Knoten von ca. 25$\mu$s. Ein leerer entfernter Methodenaufruf mit RMI über ein Ethernet-Netzwerk liegt dagegen im Bereich von einigen Millisekunden und damit um gut zwei Größenordnungen höher. Daraus ergibt sich die dringende Notwendigkeit die Ursachen dieser Diskrepanz zu untersuchen und Ansätze für eine Optimierung von RMI zu entwickeln.

Der entfernte Methodenaufruf über RMI in obigem Beispiel wurde unter folgenden Randbedingungen durchgeführt:

- Methodenaufruf mit Objektparameter (Objekt mit 7 Instanzvariablen vom Typ `int`)
- RMI-Server und RMI-Client auf zwei UltraSparc Workstations mit 300 bzw. 167Mhz
- Netzwerkverbindung über 10Mbit Ethernet im selben Segment
- Java Version 1.2beta3

## Werte im Detail

| Ethernet | | | ParaStation |
|---|---|---|---|
| Socket ping-pong | 870us | 870us | → 25us |
| Objekt ping-pong | +1480us für Serialisierung | 2350us | |
| RMI Aufruf | +1000us für Methodenaufruf | 3350us | |

Haupthindernis für effizienten entfernten Methodenaufruf:

Objekt - Serialisierung



Erst nach einer nähere Aufschlüsselung des von RMI erzeugten Zusatzaufwands beim Methodenaufruf können sinnvolle Angriffspunkte für eine Optimierung ausfindig gemacht werden. Ein entfernter Methodenaufruf über RMI läuft grob folgendermaßen ab:

- Auf der Senderseite werden die Parameter des Aufrufs zusammen mit einer Kennung des entfernten Objekts und der aufzurufenden Methode in Netzwerkrepräsentation überführt. Der Prozeß des Hin- und Rückwandelns eines Objekts in seine Netzwerkrepräsentation heißt in der Java-Terminologie Serialisierung und Deserialisierung.
- Der resultierende Byte-Strom wird über die Java-Netzwerkabstraktion (Socket-Klassenbibliothek) letztendlich an die zugrundeliegende Netzwerkhardware zur Übertragung übermittelt.
- Auf die Serverseite übertragen muß die Netzwerkrepräsentation der Methodenparameter wieder in Java-Objekte zurückverwandelt werden. Ferner wird von RMI das entfernte Objekt und dessen aufzurufende Methode identifiziert.
- Mit den aktuellen Parametern kann jetzt auf der Serverseite die entsprechende Methode des entfernten Objekts ausgeführt und das Resultat berechnet werden.
- Mit dem Ergebnis wird anschließend entsprechend verfahren. Auf der Serverseite wird es serialisiert und zum Aufrufer zurückgeschickt. Beim Aufrufer kann daraufhin nach dem Deserialisieren des Ergebnisses weitergerechnet werden.

Die 3350$\mu$s für den betrachteten RMI-Aufruf schlüsseln sich demnach folgendermaßen auf. 870$\mu$s dauert die Übertragung von von rohen Daten (ohne Objektserialisierung) über das Netzwerk. Daraus ergibt sich direkt der zeitliche Anteil, den man beim Übergang von Ethernet auf ein ParaStation Netzwerk einsparen kann. Überträgt man stattdessen komplette Objekte (inklusive Serialisierung), so erhöht sich die Zeit auf 2350$\mu$s. Die Serialisierung kostet insgesamt auf beiden Seiten also ca. 1480$\mu$s. Weitere 1000$\mu$s länger dauert dann ein kompletter entfernter Aufruf über RMI.

Objektserialisierung ist ein wichtiger Baustein in RMI und trägt, wie obige Zahlen zeigen, entscheidend zur schlechten Laufzeit von entfernten Methodenaufrufen bei. Eine optimierte Version von RMI muß daher insbesondere eine schnelle Objektserialisierung beinhalten.

## Serialisierung

- Netzwerkrepräsentation für Objekte
- Objekte als Parameter in entfernten Aufrufen
- Persistente Speicherung von Objekten
- Mächtiges allgemeines Konzept
  - verzeigerte zyklische Strukturen
  - Kompatibitität zwischen unterschiedlichen Programmversionen
  - Ausführliche Analyse möglicher Kompatibilitätsprobleme
- Keine zusätzliche Zeile Code im Standardfall
- Spezialbehandlung für bestimmte Klassen möglich
  - Robust gegenüber fehlerhafter/bösartiger Implementation des zu serialisierenden Objekts
- Nicht auf maximale Geschwindigkeit getrimmt



Objektserialisierung in Java ist ein sehr allgemeines Konzept, das außer in RMI auch zur persistenten Speicherung von Objekten auf Datenträgern eingesetzt wird. Die Serialisierung im JDK[2] muß insbesondere mit Fällen umgehen können, wo Sender und Empfänger nicht mit exakt gleichem Programmcode ausgestattet sind. Sei es, daß der Aufruf von einem Programm stammt, das beim Aufrufer nicht in der aktuellen Version vorliegt, oder aber daß gespeicherte Objekte in einer weiterentwickelten Version desselben Programms viel später wieder von einem Datenträger geladen werden sollen. Aus diesem Grund muß bei der Serialisierung von Objekten ausführliche Typinformation übermittelt werden, um die korrekte Deserialisierung eines Objektes auch dann zu gewährleisten, wenn sich z.B. das Speicherlayout beim Empfänger von dem des Senders unterscheidet [30]. Unüberwindbare Kompatibilitätsprobleme müssen bei der Deserialisierung erkannt werden, um dann die Wiederherstellung des Objekts kontrolliert abzubrechen. Dadurch wird verhindert, daß Objekte mit undefiniertem Zustand erzeugt werden können. Dies ist insbesondere auch unter dem Gesichtspunkt Sicherheit bei der Kommunikation über das Internet zu sehen.

Die Standard-Serialisierung im JDK bietet alle diese Eigenschaften, ohne vom Programmierer zu verlangen, eine einzige Zeile Programmtext zu schreiben. Klassen können einfach mit der Schnittstelle `java.io.Serializable` markiert werden und erhalten damit die Eigenschaft, serialisierbar zu sein. Die notwendige Funktionalität ist allgemein über Abfrage des Laufzeittypsystems von Java in den Klassen `java.io.ObjectOutputStream` und `java.io.ObjectInputStream` implementiert.

Beim Einsatz von RMI in Rechnerbündeln können diese Voraussetzungen etwas abgemildert werden, ohne die Grundfunktionalität der Serialisierung zu verlieren. So greifen alle beteiligten Rechner auf ein gemeinsames Dateisystem zu, was einen Versionskonflikt von Klassendateien ausschließt. Auch liegt die Lebensdauer der Objekte bei der Kommunikation innerhalb der Grenzen des Programmlaufs, so daß auch hier keine Inkompatibilitäten zu befürchten sind.

[2] Java Development Kit

## Ablauf des Serialisierungs- Deserialisierungsprozesses

- Typinformation
  Objekterzeugung auf der Gegenseite
- Lesen des persistenten Zustands
  im Standardfall durch den Stream
- Auflösung von Referenzen
  zyklische Strukturen
- Überführung in Netzwerkrepräsentation
  Byte-Reihenfolge, Fließkommazahlen, Zeichensätze
- Pufferung
  Aufbau der zu versendenden Nachricht
- Netzwerktransfer



Bei jedem Serialisierungsansatz muß dabei Funktionalität für mindestens folgende Punkte implementiert werden:

- Es muß auf jeden Fall ein Mindestmaß an Typinformation übertragen werden, da auf der Gegenseite ein Objekt desselben Typs neu angelegt werden soll. In der JDK-Serialisierung werden neben dem Klassennamen auch die Namen aller Felder der Klassen, deren Typ, die Namen der Oberklassen, deren Felder, usw. übertragen. Die schnelle Serialisierung verwendet lediglich den voll qualifizierten Klassennamen (mit Angabe des Pakets), um die Klasse zu identifizieren.
- Der persistente Zustand des Objekts muß ausgelesen werden. Das übernimmt im JDK-Fall die Serialisierungsklasse. Sie untersucht das zu serialisierende Objekt per Introspektion und liest so die zu übermittelnden Felder aus. Dieses Vorgehen hat den Vorteil, daß der Programmierer den Serialisierungscode nicht explizit aufschreiben muß, ist aber insofern ineffizient, da die Interpretation der Laufzeittypinformation bei der Übertragung jedes Objekts wieder neu durchgeführt werden muß. Weiter hat ein JIT-Übersetzer bei dieser Vorgehensweise keine Chance, irgendwelche Optimierungen vorzunehmen.

  Bei der schnellen Serialisierung stellt jedes Objekt seinen Serialisierungscode in speziellen Methoden zur Verfügung. Dieser Code muß vom Programmierer implementiert werden. Genausogut könnte man den Serialisierungscode aber auch mit einem Werkzeug automatisch generieren lassen.
- Da Objekte neben einfachen Typen auch Referenzen auf andere Objekte enthalten können, ist es nicht ausreichend, ein einzelnes Objekt zu übertragen. In den Serialisierungsprozeß werden alle Objekte mit einbezogen, die über Referenzen von dem Ausgangsobjekt erreichbar sind. Serialisiert wird also nicht ein einzelnes Objekt, sondern ein ganzer Objektgraph, von dem lediglich die Wurzel zu Anfang bekannt ist. Da dieser Objektgraph Zyklen oder Mehrfachverweise auf dasselbe Objekt enthalten kann, ist es folglich notwendig, während der Serialisierung eine Zyklenauflösung durchzuführen und bei mehrfachen Verweisen auf dasselbe Objekt nur eine Kopie zu übertragen.
- Werte eines primitiven Typs müssen in eine Darstellung als Abfolge von Bytes überführt werden. Wegen der Abwesenheit von direkten Zeigern auf Speicherbereiche in Java, können ganzzahlige Werte nur durch arithmetische Operationen in eine Byte-Darstellung zerlegt werden.

Einen Integer-Wert durch eine Abfolge von Bitverschiebungen, Konjunktionen bzw. Disjunktionen in eine Byte-Darstellung zu überführen bzw. aus dieser wieder zu rekonstruieren, sieht auf den ersten Blick zwar nach einem entsetzlichen, komplett überflüssigen Aufwand aus, ist aber innerhalb der Sprache Java nicht anders möglich. Verfügbare JIT-Übersetzer scheinen auf die Optimierung dieser Operationen auch speziell getrimmt zu sein.

Anders steht es bei Fließkommazahlen. Fließkommazahlen lassen sich nicht durch (Fließkomma-) Operationen in eine Byterepräsentation zerlegen. Da in Java die Konversion von Fließkommazahlen in ihre binäre Repräsentation nicht vorgesehen ist, muß hier auf eine native Implementierung zurückgegriffen werden. Die Java-Klassenbibliothek bietet solche Konversionsroutinen zwar an, allerdings nur für einzelne Fließkommazahlen. Da der Ausbruch aus der virtuellen Maschine in derzeitigen Java-Implementierungen mit einem erheblichen Zusatzaufwand verbunden ist, sind einer schnellen Serialisierung von Fließkommazahlen in reinem Java enge Grenzen gesetzt.

- Pufferung wird in der JDK-Implementierung teilweise in eine eigene Klasse verlagert. Diese übertriebene objektorientierte Dekomposition zieht zum einen eine Fülle von Methodenaufrufen nach sich, zum anderen verhindert sie den effizienten direkten Zugriff auf das zugrundeliegende Byte-Feld und macht ein angepaßtes Puffermanagement unmöglich.

## Hindernisse für Höchstgeschwindigkeit

Ausgelegt für Client/Server Kommunikation

Übertragung kompletter Typinformation stellt Kompatibitiltät der Klassen sicher

Benutzerfreundlichkeit

Serialisierung über Laufzeittypinformation ohne zusätzlichen Code

Plattformunabhängigkeit

Umformatierung in standardisierte Netzwerkrepräsentation

Sicherheit (Schutz gegen nicht vertrauenswürdigen Code)

SecurityManager, Kopieren statt Referenzweitergabe

Ausgiebige objektorientierte Dekomposition

Kapselung von Serialisierung und Pufferung in unterschiedlichen Klassen

## Serialisierung "light"

- Typinformation aufs Notwendigste beschränken, Cache für Typinformation
- Spezialisierte Methoden für jede Klasse lesen und schreiben den persistenten Zustand eines Objekts
  - ermöglicht JIT Unterstützung/Übersetzung
- Option für native Umwandlung von float-Arrays in ihre Netzwerkrepräsentation
  - spart teure JNI Eintritte (Konversion float↔int in Java nicht möglich)
- Enge Zusammenarbeit der Serialisierungsroutinen des Objekts mit der Pufferverwaltung des Streams
  - spart teure virtuelle Methodenaufrufe, überflüssige Abfragen auf Pufferüberlauf und ermöglicht (theoretisch) komplettes Inlining der Transformation in die Netzwerkrepräsentation pro Klasse



Die schnelle Serialisierung beschränkt die übertragene Typinformation auf ein vertretbares Maß und bietet gleichzeitig die Option, einen separaten Cache für Typinformation anzulegen. Zum Zweck der Zyklenauflösung müssen alle bereits übertragenen Objekte in einer Streutabelle gehalten werden. Stößt die Serialisierung bei der Abarbeitung des zu übertragenden Objektgraphen auf ein bereits übertragenes Objekt, wird statt des Objekts lediglich ein Verweis auf dieses Objekt übertragen. In der Standardserialisierung wird dieselbe Streutabelle auch verwendet, um die die Typen aller Objekte des Graphen zu halten.

Wird mit einer solchen Serialisierung die Parameterliste eines entfernten Aufrufs übertragen, so muß vor dem nächsten Aufruf die Streutabelle gelöscht werden, um Objekte, deren Zustand sich in der Zwischenzeit geändert haben kann, neu zu übermitteln. Da die Typinformation standardmäßig in der selben Streutabelle abgelegt wird, geht mit dem Löschen der Streutabelle auch die gesammte Typinformation verloren. Da sich während des laufenden Programms zwischen zwei Aufrufen die Klassen aber nicht verändert haben können, ist es überflüssig, Typinformation bei jedem Aufruf neu zu übermitteln. Die schnelle Serialisierung kann daher die Typinformation separat von den übertragenen Objekten in einer eigenen Streutabelle halten, die das Löschen der bereits übertragenen Objekte überlebt.

Leider geht die derzeitige Implementierung von RMI einen noch radikaleren Weg. Anstatt zwischen zwei Aufrufen nur die Serialisierung durch Löschen der Streutabelle zurückzusetzen, wird für jeden Aufruf ein komplett neues Serialisierungsobjekt angelegt. RMI kann damit aus dieser Eigenschaft der schnellen Serialisierung keinen Gewinn ziehen, aber Vorschläge zur Verbesserung von RMI, die während dieser Arbeit entstanden sind, werden über das JavaGrande-Forum [14] in die weitere Entwicklung von RMI eingehen.

Wie schon erwähnt, gibt es in Java keine Möglichkeit, Fließkommazahlen in ihre Binärdarstellung zu wandeln. Die Wandlung muß in einer nativen Hilfsroutine durchgeführt werden. Speziell bei Feldern von Fließkommazahlen ist der für jedes Element notwendige langsame Ausstieg aus der virtuellen Maschine besonders schmerzlich. Für eine schnelle Serialisierung von Feldern von Fließkommazahlen wurde daher eine native Spezialroutine implementiert, die die Wandlung gleich für einen ganzen Bereich eines solchen Feldes durchführt. Bei großen Feldern erreicht man dadurch einen erheblichen Geschwindigkeitsgewinn [26]. Da es sich dabei aber um keine reine Java-Lösung mehr handelt, ist diese Option in der endgültigen

Version der schnellen Serialisierung nicht mehr vorgesehen. Es steht zu hoffen, daß sich Sun Microsystems entschließt, eine Konversionsfunktion für Fließkommafelder in die Standard-Klassenbibliothek für Java aufzunehmen.

Soll ein Objekt von der schnellen Serialisierung profitieren, muß es eine neu definierte Schnittstelle implementieren und zwei spezialisierte Methoden zum Schreiben und Lesen seines persistenten Zustandes bereitstellen. Durch kooperative Pufferverwaltung von Serialisierungklasse und spezialisierter Serialisierungsroutine schaft man zum einen gute Voraussetzungen für einen optimierenden JIT-Übersetzer und spart die meisten Überprüfungen auf Pufferüberlauf ganz ein, die neben einer Unmenge von Methodenaufrufen bei einer vollständigen objektorientierten Dekomposition notwendig werden.

## Integration ins Gesamtsystem

### Keine "steckbare" Serialisierung für RMI vorgesehen

- sun.rmi.server.MarshalOutputStream extends ObjectOutputStream
- RMI erzeugt explizit mit **new** Instanzen von ObjectOutputStream

### Gleicher Name für neue Serialisierungsklasse

- Überdecken der Standardserialisierung im CLASSPATH
- Standardserialisierung nicht mehr zugreifbar
- Patch der Original-Klasse und Neuübersetzung
- Serialisierung ist heißer Code im JDK, Änderungen mit jeder Version

### Anderer Name für die Serialisierungsklasse

- Standardserialisierung bleibt benutzbar, Trennung vom JDK-Code
- Wird von RMI nicht benutzt (auch Erweitern hilft nicht !)
- Patch der RMI Klassen (möglich in ihrer binären Form)

***RESH: Projekt T2*** ***Bernhard Haumacher***

Die Herausforderung bei der Implementierung einer schnellen Serialisierung besteht darin, eine Spezialösung für die Kommunikation in enggekoppelten Rechnernetzen zu schaffen, die trotzdem noch mit Objekten umgehen kann, die von der neuen Serialisierung nichts wissen. Damit die schnelle Serialisierung auch mit RMI zusammenarbeiten kann, müssen auch Objekte serialisierbar bleiben, die nicht die Spezialschnittstelle für schnelle Serialisierung implementieren. Da die Standardserialisierung im JDK nicht in reinem Java implementiert ist, kann nicht die gesammte Funktionalität der JDK-Serialisierung nachimplementiert werden. Die schnelle Serialisierung muß also auf den Code der Standardserialisierung zurückgreifen können.

Es bleiben zwei Möglichkeiten: Der erste Ansatz integriert die schnelle Serialisierung direkt in die Klassen der Standardserialisierung. Der zweite Ansatz kapselt die schnelle Serialisierung in einer eigenen Klasse, die nur im Notfall eine Instanz der Standardserialisierung verwendet (wenn ein Objekt serialisiert werden muß, das nicht auf die schnelle Serialisierung vorbereitet ist).

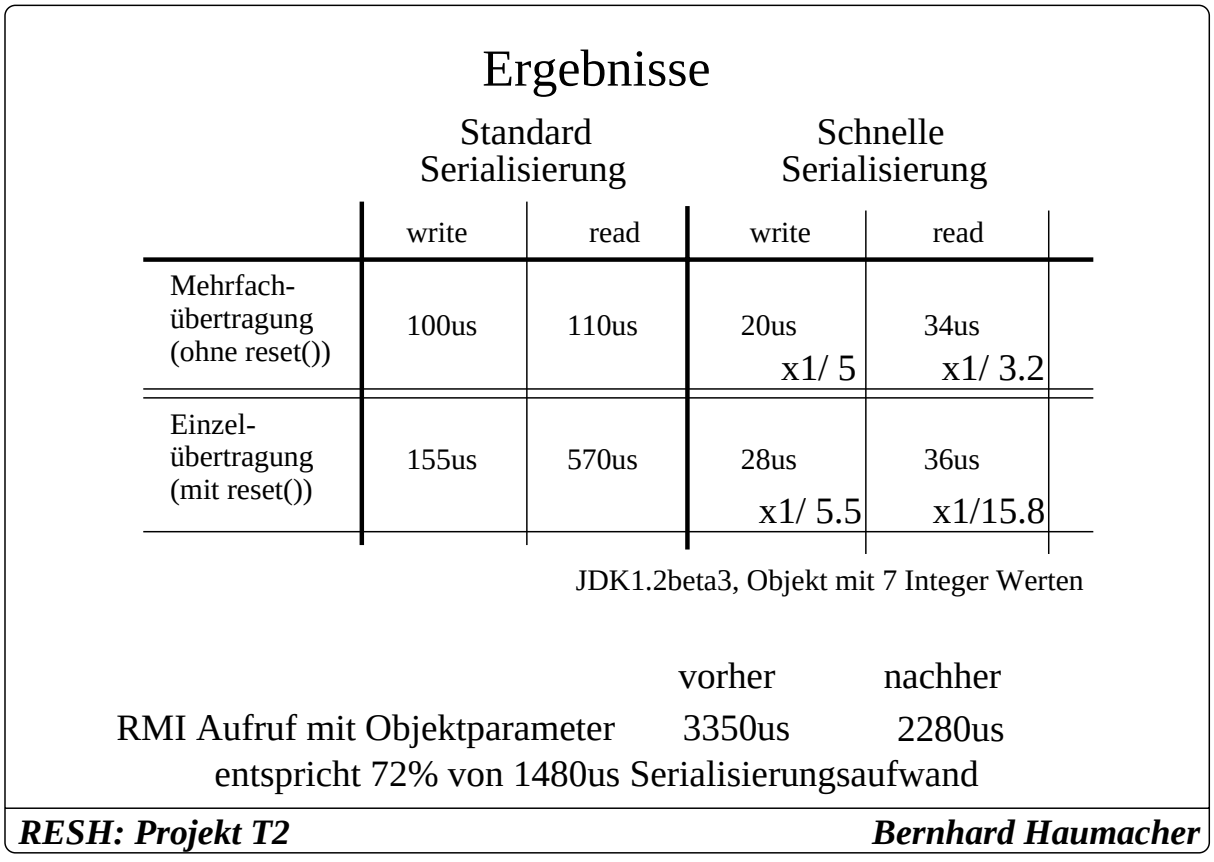
Bei der direkten Integration wird der Code der Standard-Serialisierung nachgebessert und anschließend neu übersetzt. Die resultierenden Klassendateien stehen dann unter demselben Namen zur Verfügung. Damit beim Programmablauf die veränderten Serialisierungsklassen auch verwendet werden, müssen sie im `CLASSPATH` vorangestellt werden. Diese Vorgehensweise hat mehrere Nachteile:

- Die echte Standardserialisierung ist nicht mehr zugreifbar und steht folglich auch nicht mehr für die persistente Speicherung von Objekten zur Verfügung.

- Die notwendigen Änderungen erstrecken sich praktisch über den gesammten Code der Standard-Serialisierung. Da an der Implementierung der Serialisierungsklassen im JDK von Java-Version zu Java-Version Änderungen vorgenommen werden, muß die selbe Arbeit ständig neu durchgeführt werden.

Speziell der letzte Grund hat dazu geführt, daß wir vom Nachbessern der Standard-Serialisierung Abstand genommen haben und die schnelle Serialisierung in einer eigenen Klasse unter anderem Namen zur Verfügung stellen. Aber auch bei diesem Ansatz sind mehrere Hürden zu nehmen:

- Die Klassen für die schnelle Serialisierung müssen die Klassen der Standard-Serialisierung erweitern, damit bestehender Code überhaupt mit der neuen Serialisierung zusammenarbeiten kann. Da die Schnittstelle `java.io.ObjectOutput` nicht alle öffentlichen Methoden der standard-Serialisierungsklasse `java.io.ObjectOutputStream` anbietet, wird sie meist nicht ausschließlich verwendet. Das Überschreiben von `java.io.ObjectOutputStream` muß zum einen explizit durch den installierten Security-Manager erlaubt werden, was beim RMI-Security-Manager der Fall ist. Zum anderen sind beim Erweitern der Serialisierungsklassen einige technische Hindernisse zu überwinden, die ebenfalls mit Sicherheitsrestriktionen in Java zusammenhängen.
- Im RMI ist keine auswechselbare Serialisierung vorgesehen. D.h. es gibt keine Fabrik-Klasse, durch deren Austausch man auf eine andere Serialisierung umschalten könnte. Stattdessen werden im RMI Instanzen von `java.io.ObjectOutputStream` und `java.io.ObjectInputStream` direkt erzeugt. Dies beschränkt sich aber auf sehr wenige Stellen, so daß hier eine Nachbesserung vertretbar ist. Die notwendige Modifikation der RMI-Klassen wird durch ein Werkzeug automatisch vorgenommen.




Obige Tabelle gibt Meßergebnisse von reiner Serialisierung und von einem kompletten RMI Aufruf wieder. Man erkennt, daß das Schreiben des Objekts generell schneller geht, als das Lesen. Dies liegt bei der Standard-Serialisierung daran, daß das Schreiben der ausführlichen Typinformation relativ schnell geht, beim Lesen die Typinformation aber rekonstruiert und gegen die aktuelle Klasse verglichen werden muß. Daß auch bei der schnellen Serialisierung das Lesen deutlich langsamer geht, liegt daran, daß während

des Einlesens die Klasse des gelesenen Objekts gefunden und ein Objekt des passenden Typs angelegt werden muß.

Insgesamt bleibt zu bemerken daß sich gegenüber der Standard-Serialisierung Geschwindigkeitssteigerungen bis um Faktor 15 erzielen lassen und die absoluten Werte (20-36$\mu$s) durchaus in interessante Regionen vorstoßen. Daß die schnelle Serialisierung auch in Verbindung mit RMI einen deutlichen Effekt zeigt, sieht man anhand der Zeit für den entfernten Methodenaufruf mit und ohne schnelle Serialisierung.

## Zusammenfassung und folgende Arbeiten

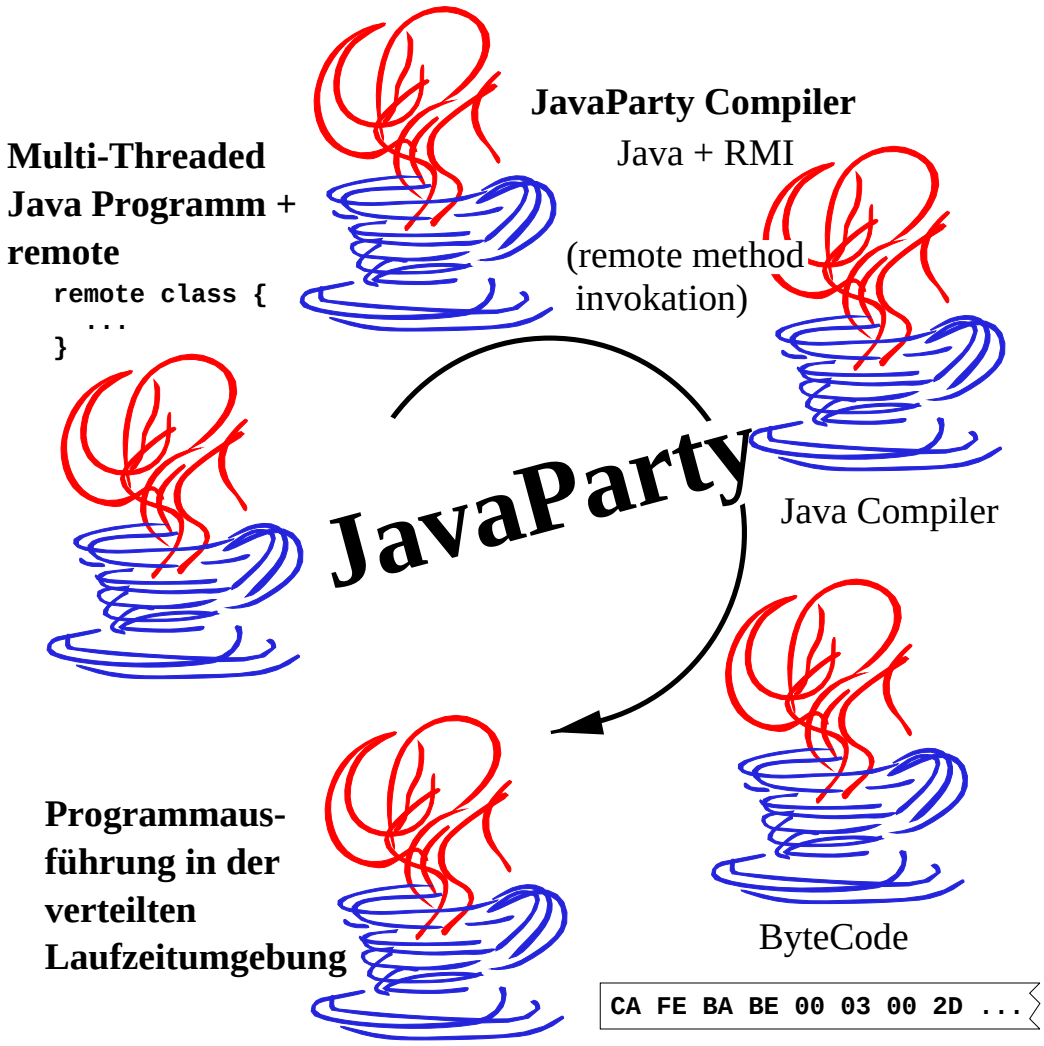
- Verteiltes Rechnen benötigt schnelle Objekt-Serialisierung
  - realisierbar in "100% pure" Java (Ausnahme: float[])
  - bessere Integrationsmöglichkeiten in RMI wünschenswert
  - Vorschläge für das JavaGrande Forum
- Messungen auf ParaStation
- Optimierung des Zusammenspiels von RMI und der ParaStation Bibliothek
- Schnelles eigenes RMI
- JavaParty ganz ohne RMI (JavaParty II)
- Semantik lokaler Objekte in JavaParty

# Forschergruppe RESH

Rechnernetze als Superrechner und Hochleistungsdatenbanken

**Gekoppelte Workstations als**
**Virtuelle Multiprozessor Java Maschine (JVM)**

**Transparente entfernte Objekte + Objektmobilität**




**Ansprechpartner / Kontakt:**

IPD Prof. Tichy
Universität Karlsruhe
Am Fasanengarten 5
D-76131 Karlsruhe

Tel: 0721/608-4067 Fax: 0721/608-7343
Email: javaparty@ira.uka.de
URL: http://wwwipd.ira.uka.de/RESH/
URL: http://wwwipd.ira.uka.de/JavaParty/

# 4 L1: Paralleles Data-Mining

## 4.1 Ausgangssituation

Das Wachstum der Datenbestände in Wirtschaft, Verwaltung und Wissenschaft, zukünftig aber auch im Konsumbereich (Stichwort Internet) scheint ungebrochen. Damit einher gehen eine wachsende Vielfalt von Anwendungen und Verfahren zur Informationsgewinnung aus den gesammelten Datenbeständen. Die Auswirkungen auf die Nutzung von Datenbanksystemen läßt sich wie folgt charakterisieren: Dominierte in der Vergangenheit das Anwendungsprofil des OLTP (Online Transaction Processing), so tritt ihm mehr und mehr das des OLAP (Online Analytical Processing) zur Seite.

OLTP geht von hohen Lasten einfacher, sog. Debit-/Credit-Transaktionen aus, die mit ihrer kurzen Dauer, einfachen Verarbeitungsvorgängen und dem vergleichweise geringen Datenvolumen dem ACID-Paradigma unterliegen. OLAP stellt demgegenüber nicht mehr die Transaktion in den Mittelpunkt der Betrachtung, sondern das Durchforsten umfangreicher Datenbestände, die sich über lange Zeit oder im Rahmen datenintensiver Erfassungen und Experimente angesammelt haben und als wertvolle „Goldmine“ von Langzeitwissen angesehen werden, die es nach vielfältigen Kriterien zu Zwecken der Entscheidungsunterstützung auszuschlachten gilt.

Sind die Geschäftsprozesse hinreichend stabil, so kennt man die Kenngrößen, nach denen die Datenbestände auszuwerten sind, im allgemeinen recht gut. Das Gebiet “Data Warehousing”, das in den letzten Jahren sprunghaft an Bedeutung gewonnen und eine Vielzahl kommerzieller Produkte hervorgebracht hat, bündelt heute seine Kräfte auf leistungssteigernde Maßnahmen in diesem Umfeld. Leistungssteigerung ist auch das Gebot beim “Data Mining” oder Knowledge Discovery in Databases, in dem allerdings die Datenauswertung eher experimentell und nach zunächst nur vage umrissenen Kriterien erfolgt. Kommerzielle Durchbrüche sind hier seltener. Tatsächlich erscheint jedoch die strikte Trennung zwischen beiden Gebieten, zumindest aus Anwendersicht, eher willkürlich. Bevor die stabile Situation eines Data Warehouse erreicht ist, müssen die geeigneten Auswertkriterien und -strategien erst einmal bestimmt werden. Und sind die Kriterien stabil, so muß diese Stabilität laufend hinterfragt werden. Das sogennante “Ausreißer”-Problem charakterisiert die Abweichungen vom Gewohnten oder Erwarteten. Das Projekt L1 hat die Unterstützung der hochiterativen, von Leistungsengpässen begrenzten Data-Mining-Phase zum Gegenstand, dieses insbesondere mit der Vorbereitung des Data Warehousing und der laufenden Überprüfung auf Ausreißer im Visier. Es konzentriert sich auf die Überwindung der Leistungsengpässe des Data Mining durch Parallelisierung der Verarbeitung umfangreicher Datenbestände.

Parallelisierung beim Zugriff auf große Datenbasen nach flexiblen Kriterien sowie bei deren Verwaltung, also bei OLTP, ist nichts Neues. Dabei entstanden unterschiedlichste Architekturen, die von hohen Zugriffsbreiten auf den Hintergrundspeicher mittels sog. Disk Arrays bis hin zur Intertransaktionsparallelität (gleichzeitige Bearbeitung ganzer Transaktionen auf verschiedenen Prozessoren) und Intratransaktionsparallelität (parallele Bearbeitung der Operatoren innerhalb einer Transaktion) durch Vervielfachung von Prozessoren sowie der Kombination all dieser Techniken reichen. Charakteristisch für all diese Ansätze ist eine hohe Grobkörnigkeit der Parallelverarbeitung: Auf jedem Knoten wird (zumindest zeitweilig) ein umfangreicher Datenbestand vorgehalten, der von einem abgeschlossenen, meist umfangreichen Programm lokal abgearbeitet wird, während sich der Nachrichten- und Datenaustausch zwischen den Knoten auf vergleichsweise wenige Zeitpunkte beschränkt. Die Referenzarchitektur für diese Vorgehensweise ist die verteilte Datenbank.

OLAP geht von ganz anderen Voraussetzungen aus. OLAP rechnet nur mit niedrigen Raten gleichzeitig zu bearbeitender Aufträge an das Datenbanksystem, erwartet aber umgekehrt von diesen Aufträgen sehr hohe Anforderungen an das zu untersuchende Datenvolumen bei hoher Rechenintensität der Analyseprozesse. Beim Data Mining kommt aufgrund des experimentellen Charakters des Wissensgewinnungsprozesses eine interaktive Arbeitsweise hinzu.

Bei Data Warehousing gehorchen die Aufträge noch wie bei OLTP einer beschränkten Zahl vordefinierter Anfragemuster. Beim Data Mining variieren diese mit dem Fortschritt der Entscheidungsprozesse der Anwender und sind damit schlecht vorhersagbar. Es stellt sich daher die Frage, ob unter diesen Umständen nicht eine grobkörnige Parallelität, die von vorgegebenen Anfragemustern ausgeht, ihren Zweck einbüßt und stattdessen feinkörnigere Parallelität auch für den Datenbankbereich an Bedeutung gewinnt.

## 4.2 Ziele

Feinkörnigkeit läßt sich natürlich mit dedizierten Parallerechnern erreichen. Nur erhebt sich die Frage, ob nicht abschnittsweise auch Grobkörnigkeit noch ihre Vorteile besitzt oder der Bedarf an Rechenkapazität Schwankungen unterliegt. Dieser Frage nachzugehen und das Potential einer gleitenden Körnigkeit zusammen mit einer dynamisch skalierbaren Rechenkapazität auszuloten, ist das genaue Ziel des Vorhabens. Mit der ParaStation2 liegt eine ideale Plattform vor, um das Parallelisierungspotential von OLAP auszuloten, da dort die Lücke zwischen Prozessor- und Datenaustauschgeschwindigkeit um Größenordnungen verringert wird.

Daß Grobkörnigkeit und Feinkörnigkeit irgendwie auszubalancieren sind, erscheint schon allein deshalb notwendig, weil der Ausgangspunkt, die Rohdatenbasis, zunächst einen grobkörnigen Verarbeitungsbeginn erzwingt, andererseits an ihm aber so früh als möglich rechenintensive Verfahren ansetzen sollten. Die Balancierung ist im wesentlichen für die Skalierbarkeit des Ansatzes verantwortlich. Aus diesen Untersuchungen folgen dann Aussagen zur statischen und dynamischen Datenverteilung sowie für Abarbeitungsprinzipien.

Ausgangspunkt unserer Untersuchungen soll die Architektur verteilter Datenbanken sein. Diese wird in Richtung Feinkörnigkeit entwickelt. Dieser Übergang von der Grob- zur Feinkörnigkeit folgt der Entwicklung des Wissensgewinnungsprozesses im Data-Mining ausgehend vom Zugriff auf die Rohdatenbestände hin zu immer stärkeren Verdichtungen. Dabei gehen wir nach unseren Erfahrungen davon aus, daß Leistungsoptimierung im Data-Mining sich vor allem auf die frühen Stufen der Informationsverdichtung konzentrieren muß. Dort sehen wir gute Chancen, daß sich spezielle, über die klassischen Operationen hinausgehende Mengenoperatoren identifizieren lassen, die einen lohnenden Ansatzpunkt für eine feinkörnige Parallelisierung bieten.

## 4.3 Aktueller Stand

Überblick

KDD und Data Mining

Ziele

Alternativen und Entscheidungen

Projekte

Algebramaschine- Parallelisierung

Ergebnisse

Paralleles Data-Mining-Verfahren

Ausblick

***RESH: Projekt L1*** ***Matthias Gimbel***

## KDD und Data Mining

KDD = Vorverarbeitung (Datenbereinigung und -auswahl)

(Selektion, Projektion, Join, Sampling, ....)

+ Analyse-(Lern-)verfahren

***RESH: Projekt L1*** ***Matthias Gimbel***

KDD (Knowledge Discovery in Databases) besteht nicht allein aus der Anwendung der verschiedenen Lernverfahren, sondern der Einsatz dieser Verfahren setzt eine geeignete Vorverarbeitung der Daten voraus. Diese dient u.a. der Verknüpfung der Analysedaten aus verschiedenen Relationen, der Reduktion des Datenumfangs, da die Komplexität der Analyseverfahren ansonsten zu inakzeptablen Antwortzeiten führen würde, sowie der Säuberung der Daten. Neben den aus der relationalen Technik bekannten Operatoren wie Selektion, Projektion, Join und Aggregation finden sich auch Data-Mining-spezifische Operatoren wie z.B. Sampling.

Am Institut wurde im Rahmen eines früheren Projekts ein eigenes objektorientiertes Datenmodell, das sogenannte Informationsmodell (IM) entwickelt [5], das auf die spezifischen Belange des Data Mining zugeschnitten ist und neben zusätzlichen Strukturierungsmöglichkeiten entsprechend zugeschnittene - überwiegend algebraische - Operatoren anbietet. Eine Besonderheit dieses Modells ist die Möglichkeit, die Prozeßhistorie in Form eines Objektmengenschemas fortzuschreiben, deshalb auch jederzeit nachvollziehen und somit insbesondere auch feststellen zu können, inwieweit aktuelle Analyseschritte bereits in der Vergangenheit vorkamen.

Bei der Modellierung werden die Operatoren, wie in der Abbildung dargestellt, zu Datenflußdiagrammen verknüpft, welche die Abfolge der einzelnen Verarbeitungsschritte beschreiben. Die Anfragebearbeitung verfeinert diese zu Operatorgraphen in einer erweiterten relationalen Algebra. Mit diesen Operatorgraphen kann nun auf zweierlei Weise umgegangen werden. Liegt als Basismaschine ein relationales DBMS vor, so kann der Operatorgraph weiter in einen Operatorgraphen mit rein relationalen Operatoren überführt werden. Tatsächlich erfolgt die Überführung besser in eine SQL-Anfrage, da man dann das Optimierungspotential des DBMS nutzen kann (Query Shipping). Man kann aber auch eine virtuelle Ausführungsmaschine definieren, die unmittelbar den IM-Operatorgraph ausführt. Zur Untersuchung des Parallelisierungspotentials scheint dieser Ansatz attraktiver. In beiden Fällen kann im übrigen eine weitere Form der Optimierung, die in einer Dissertation derzeit untersuchte Materialisierung von Zwischenergebnissen, Verwendung finden.

## Ideen und Ziele, Vorgehensweise

Unterstützung von Vorverarbeitung und Data Mining durch das DBMS

- Integrierte Optimierung, angepaßte Operatoren

Nutzung feingranularer Parallelität

- bessere Lastverteilung, Skalierbarkeit, möglicherweise bessere Lokalität

Steuerung der Parallelität von DB-Operatoren und Lernverfahren

Vorgehensweise:

- Zunächst Konzentration auf die Vorverarbeitung: parallele Algebramaschine
- Testen feingranularer Verarbeitung durch intensive Nutzung von Pipelining

***RESH: Projekt L1*** ***Matthias Gimbel***

Beim Verfolgen des zweiten Ansatzes ist eine wesentliche Idee die Integration von Vorverarbeitungsoperatoren mit den Datenzugriffsoperatoren des nachgeschalteten Analyseverfahrens. Durch die Zusammenfassung in einem gemeinsamen Operatorgraphen ergibt sich die Möglichkeit, bei der Anfrageoptimierung Zusatzwissen aus dem jeweils anderen Teil des Graphen auszunutzen. So lassen sich beispielsweise aufwendige Joins in der Vorverarbeitung einsparen, wenn das Analyseverfahren ohnehin nur einzelne (oder wenige) Dimensionen innerhalb einer Anfrage betrachtet und die Zusammenfassung der Dimensionen effizienter selbst vornehmen kann.

Bei der Parallelisierung der Anfragebearbeitung wird besonderes Gewicht auf die Nutzung feingranularer Parallelität gelegt, wie sie durch die schnelle Kommunikationshardware möglich wird. Feingranular bedeutet hierbei, daß die Menge der von einem Knoten isoliert (d.h. ohne Kommunikation mit anderen Knoten) bearbeiteten Daten verringert wird. In diesem Zusammenhang soll auch Pipelining intensiv genutzt werden. Feingranulare Verarbeitung hat dabei vor allem in Bezug auf die Lastbalancierung und die Skalierbarkeit Vorteile.

Da der Anfragebearbeitung eine Anfrage mit objektorientierten Operatoren zugrundeliegt, ergeben sich aber noch weitere Optimierungsmöglichkeiten: Anstatt zuerst auf rein relationale Operatoren abzubilden, können die objektorientierten Operatoren auch direkt ausgeführt werden. Dabei ergeben sich Änderungen durch den Übergang von der datenflußbasierten zur navigierenden Verarbeitung, wie sie beim Verfolgen von Objektreferenzen auftritt. Solch navigierende Verarbeitung führt aber auch bei der Parallelisierung zu neuartigen Verarbeitungsmustern und bewirkt eine extrem feingranulare Parallelität.

Mit den beschriebenen Erweiterungen der Konzepte traditioneller relationaler Verarbeitung ist auch die Notwendigkeit zur Erweiterung der parallelen Anfrageoptimierung verbunden. Hier sind sowohl die technischen Gegebenheiten von Workstation-Clustern als auch die neuen Freiheitsgrade des navigierenden Zugriffs zu berücksichtigen.

Die für den zurückliegenden Projektabschnitt gewählte Vorgehensweise ist in obiger Folie zusammengefaßt. Wir haben uns auf die Vorverarbeitung konzentriert, um in einem ersten Schritt zunächst im Umfeld rein relationaler Anfragebearbeitung die Möglichkeiten der ParaStation sowie der darauf aufbauenden Umgebungen auszuloten. Feingranulare Parallelität sollte hier vor allem durch die intensive Nutzung von Pipelining erzeugt werden.

## Alternativen und Entscheidungen

### Technische Basis:

kommerzielles DBMS, Erweiterungen oberhalb

industrielle Akzeptanz, allerings kein Einfluß auf die Basisoperationen, harter Schnitt durch SQL-Interface

SQL-DB mit erweiterbarem Optimierer

Zugang zur Codebasis, aber hohe Komplexität, viel für DM-Aufgaben unnötige Funktionalität

DB-Basisbibliothek + JavaParty

volle Kontrolle, nur die benötigten Funktionen, Performance ??



Zur Realisierung wurden die gezeigten Alternativen betrachtet und bewertet.

Die erste Alternative, die Verwendung eines kommerziellen DBMS, scheint nach den zuvor getroffenen Entscheidungen zwar zu entfallen. Sie sollte trotzdem zunächst nicht außer Acht gelassen werden, da sie den Vorteil böte, zu industriell akzeptierten Lösungen zu führen. Jedoch existiert gegenwärtig kein System, das auf der vorhandenen Hardware direkt aufsetzen kann. Für eine Portierung wäre man also auf die Zusammenarbeit mit dem Hersteller angewiesen. Keine Einflußnahme wäre möglich bei der Wahl der Parallelisierungsstrategie: Man ist auf die SQL-Schnittstelle festgelegt, die auch die Möglichkeiten des Übergangs von mengenorientiertem zu navigierendem Zugriff beschneidet.

Mit dem Ansatz einer objektorieniterten Verarbeitung eher verträglich wäre die Verwendung einer (im Quellcode vorliegenden) frei verfügbaren Datenbank mit erweiterbarem Optimierer. Hierbei handelt es sich aber im allgemeinen um sehr komplexe Systeme, die für Data-Mining-Zwecke viel unnötige Funktionalität mitbringen (z.B. Locking, Logging, Transaktionsmechanismen). Alles in allem also um eine sehr umfangreiche Codebasis, die eine Anpassung bzw. Parallelisierung nicht einfach macht.

Die dritte und von uns beschrittene Lösung ist daher eine Eigenentwicklung. Hier sind alle gewünschten Einflußmöglichkeiten (Parallelisierung, Operatoren) gegeben. Um den Entwicklungsprozeß zu beschleunigen, wurde Java als Implementierungssprache gewählt. Damit verbunden ist allerdings die Performance-Problematik von Java als interpretierter Sprache, die im Auge behalten werden muß.

Als erstes Projekt wurde also zunächst eine für OLAP-Aufgaben geeignete parallele Algebramaschine in Java-Party konzipiert und implementiert.

## Prototypen und Projekte

File-Input:

| | |
|---|---|
| Objekte | 160K/s |
| Einzelbytes: | 1.2M/s |

Prototyp-Pipeline, Datentransport über Sockets

| | | |
|---|---|---|
| Objektstream: | 39K/s | (auf BS- und PS-Sockets) |
| Bytes: | 1.1M/s | |

Einfach-DB-Pipeline (2 Knoten, Datentransport durch Methodenaufruf):

| | Ultra1/167, JDK1.2b3 | Ultra10/300,JDK1.2PR | Alpha/500,JDK1.1.6 |
|---|---|---|---|
| norm.-Stream | 30s | 7s | 17s |
| Faststream | 14s | 4s | 7s |

***RESH: Projekt L1*** ***Matthias Gimbel***

Bevor mit der eigentlichen Algebramaschine begonnnen wurde, wurden zunächst einige Voruntersuchungen durchgeführt, um Aussagen über die Leistungsfähigkeit der ParaStation-Plattform sowie die Java-Party-Umgebung zu erhalten. Die dabei gewonnenen Erkenntnisse flossen auch in die Konzeption der Algebramaschine ein.

Startpunkt und historischer Flaschenhals der Anfragebearbeitung ist der I/O, d.h der Zugriff auf die auf dem Externspeicher liegenden Daten [7]. Mit dem ersten Test sollte herausgefunden werden, welche Leistung mit Java auf diesem Gebiet zu erzielen ist. Dabei wurden die Freiheitsgrade, die Java bei der Speicherung von Daten bietet, ausgelotet: Java bietet Mechanismen an, Objekte persistent zu speichern und später darauf zuzugreifen. Dies eröffnet prinzipiell die Möglichkeit, Datentupel in Objektform zu speichern, und vereinfacht die Programmierung erheblich gegenüber der Alternative, der low-level-Verarbeitung von Byte-Arrays (in Java existiert kein Record-Typ).

Der erste Test sollte nun Maßzahlen für die Verwendung von Objekten im Vergleich zu Byte-Arrays für das Einlesen der Datensätze liefern. Gemessen wurde auf einer Sun Ultra1/167 mit jdk1.2beta3. Das Ergebnis zeigt, daß der vereinfachten Programmierung mit Datenobjekten eine Verschlechterung um den Faktor 7.5 beim Zugriff auf die Daten gegenübersteht.

Ein zweiter wesentlicher Faktor bei der verteilten Anfragebearbeitung ist die Kommunikation. Mit der zweiten Testreihe sollte diese Komponente untersucht werden. Gemessen wurde die Datenübertragung durch eine Pipeline über 8 Knoten, welche über Sockets mit ihrem Vorgänger- und Nachfolgerknoten verbunden waren und die Daten unverändert an ihren Nachfolger weitergaben. Diese Knoten waren auf den vorhandenen 8 Alpha-Rechnern des RESH-Clusters verteilt. Die Messungen wurden mit Objekten und mit Byte-Arrays sowie auf normalem 100Mbit-Ethernet und auf der ParaStation durchgeführt.

Es zeigt sich wiederum, daß die Verwendung von Objekten, die über Objekt-Streams übertragen werden, nicht konkurrenzfähig ist. Bei der Verwendung von Byte-Strömen traten signifikante Verbesserungen auf. Die Verwendung der ParaStation gegenüber der für die Werte in der Folie eingesetzten 100MBit-Ethernet-Verbindung brachte für die Objekte fast nichts (die Kosten waren durch die Verarbeitung auf den Knoten dominiert), bei der Übertragung von Einzelbytes jedoch eine Geschwindigkeitssteigerung um mehr als den Faktor 2 auf 2,4 MB/s. Bei diesen Messungen zeigte sich aber, daß in einer längeren Pipeline der Datenfluß durch die Sockets nicht sich selbst überlassen werden kann, da sich ansonsten durch die

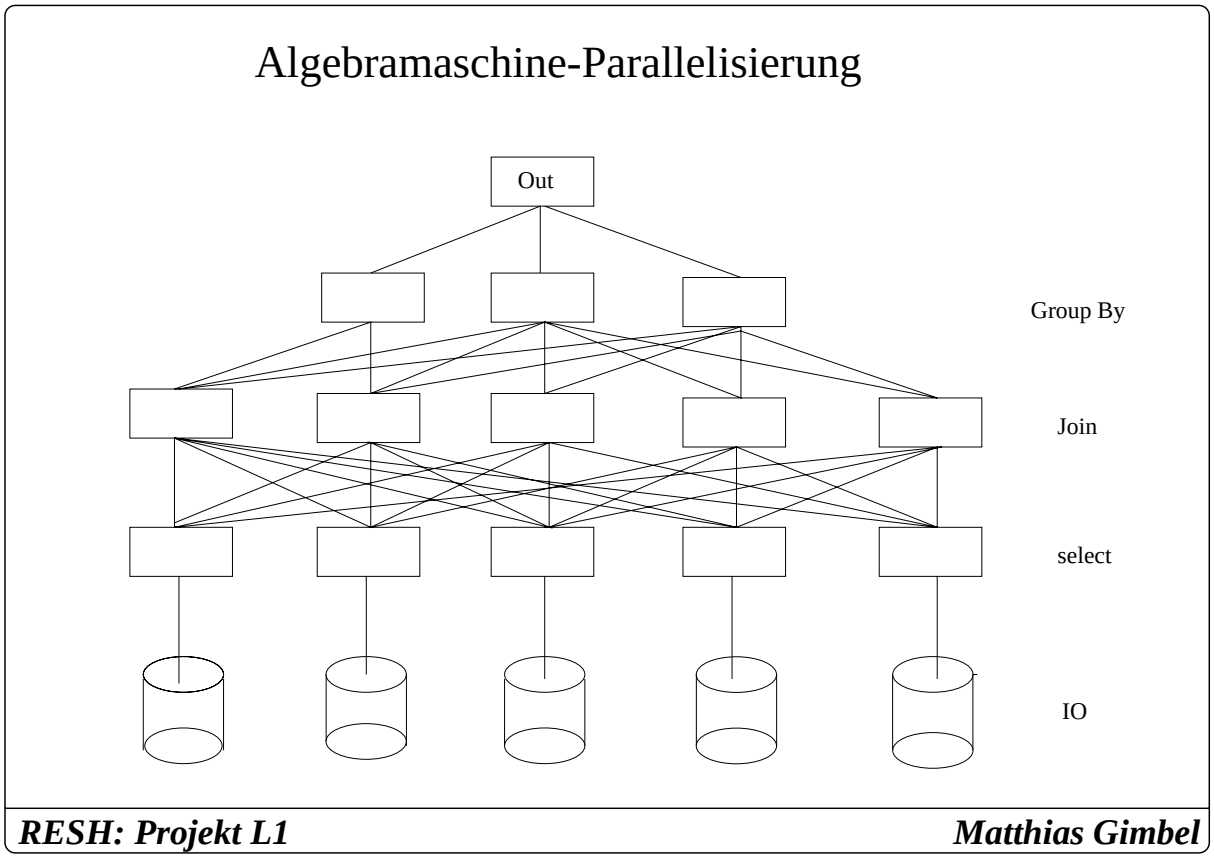
Pufferung Engpässe und Ungleichverteilungen aufschaukeln und insgesamt zu Geschwindigkeitsverlusten führen.

Aus diesen Beobachtungen, der einfacheren Programmierung und der besseren Skalierbarkeit (keine eigenen n-zu-n-Sockets, damit auch weniger Threads = effizientere Parallelisierung) heraus, wurde der Übergang auf den Datentransport durch Methodenaufrufe beschlossen. Die Infrastruktur dafür stellt JavaParty bereits zur Verfügung.

Mit dieser Methode wurden nun weitere Messungen durchgeführt. Der letzte dargestellte Test beschreibt eine Kombination von File-I/O und Datentransport über zwei Knoten, der Transport erfolgt über entfernte Aufrufe.

Hier zeigen sich bereits hier die Auswirkungen der im Teilprojekt T2 durchgeführten Arbeiten zur schnellen Serialisierung. In der unteren Zeile ist erkennbar, daß sich durch die verbesserte Serialisierung der als Parameter entfernter Methodenaufrufe übergebenen Daten (in einem schnellen Stream) in der Anwendung Geschwindigkeitssteigerungen um den Faktor 2 erzielen lassen.

Der Vergleich der Werte, die sich für unterschiedliche Java-Versionen ergeben, zeigt darüberhinaus, daß die Entwicklung von Java hin zu höheren Ausführungsgeschwindigkeiten stetig fortschreitet und Java auch im Bereich des Hochleistungsrechnens eine sehr vielversprechende Entwicklung nimmt.




Aufbauend auf diesen Erfahrungen wurde nun die parallele Algebramaschine entwickelt. Sie bietet im jetzigen Ausbauzustand die Operatoren Selektion, Projektion, Join und Gruppierung. Als Join-Implementierung wird, wie bereits in der am Institut entwickelten Algebra-Maschine CEE [12], der pipelined Hash-Join benutzt, der ein hohes Maß an (Pipelining-)Parallelität ermöglicht.

Im Bild ist beispielhaft ein Operatorgraph dargestellt, wie er bei der parallelen Ausführung von OLAP-Anfragen innerhalb der Algebramaschine auftritt. Unten sind die Platten der einzelnen Knoten dargestellt, darüber folgen die benötigten Operatoren, der Datenfluß geht also von unten nach oben. (Dieses Bild stellt den durchschnittlichen Ausbau für 2 gejointe Relationen dar, bei komplexeren bzw. einfacheren Anfragen werden entsprechend mehr/weniger Stufen benötigt.

Die Algebramaschine ermöglicht dabei sowohl horizontale Parallelität, die durch die statische Partitionierung der Daten zustandekommt, als auch Pipelining-Parallelität, die durch die gleichzeitige Arbeit der

im Bild dargestellten verschiedenen Stufen entsteht. Dabei ist der Grad der Parallelität, d.h. die Anzahl der parallelen Operatoren, die dieselbe Operation ausführen, sowie die Plazierung der Operatorknoten auf den einzelnen Rechnern einstellbar.

Mit dieser flexiblen Parallelisierung wird es möglich, die Auswirkungen verschiedener Parallelitätsgrade sowie verschiedener Anordnungen der Operatorknoten (mit entsprechenden Auswirkungen auf die Kommunikation) auf die Laufzeit zu bestimmen und somit das Potential für weitergehende Formen feingranularer Parallelisierung auszuloten.

## Erste Ergebnisse

Einfache Anfrage : Select BR, COUNT(BR) from Fahrzeug groupby BR

| Knoten | 1 | 2 | 4 | 8 |
|---|---|---|---|---|
| Zeit | 32s | 37s | 38s | 40s |

Pro Knoten 100000 Tupel, etwa 4MB lokale Daten

Startup-Time bei 8 Knoten : etwa 4 Sekunden

was fehlt: Vergleich der verschiedenen Parallelisierungsstrategien

Vergleich mit kommerziellem DBMS (wird nachgeliefert)

***RESH: Projekt L1*** ***Matthias Gimbel***

Mit dieser Algebramaschine wurden bereits erste Messungen durchgeführt. Dazu wurde die im Bild oben gezeigte Anfrage auf einem Testdatenbestand (100000 Tupel, 4MB Daten) ausgeführt. Hardwarebasis war wiederum das RESH-Cluster, bestehend aus 8 Alpha-Rechnern mit jdk 1.1.6.

Zu dieser Messung ist zu bemerken, daß hier jeder Knoten seinen lokalen Datenbestand von 4MB in die Verarbeitung einbringt, das insgesamt bearbeitete und übertragene Datenvolumen also proportional mit der Anzahl der beteiligten Rechnerknoten auf bis zu 32 MB (bei 8 Knoten) wächst. Es wurde also der Scale-Up bewertet. Die dargestellten Zahlen lassen dabei auf eine recht gute Skalierung des Ansatzes schließen.

Darüberhinaus lag zum Testzeitpunkt für diese Plattform noch keine optimierte Version des Streams vor. Die Fortentwicklung von Java im allgemeinen und die im Teilprojekt T2 im Gange befindlichen Arbeiten an schneller Serialisierung und optimiertem RMI lassen also noch wesentlich bessere Ergebnisse erwarten.

Auf diesem Gebiet sind allerdings noch umfangreichere Untersuchungen notwendig. Um die hierbei gewonnenen Ergebnisse vergleichbar zu machen, sollen dazu verschieden komplexe Queries aus der TPC-D-Suite [6] auf dem entsprechenden Datenbestand herangezogen werden. Daran kann dann auch mit verschiedenen Strategien zur Wahl des Parallelitätsgrades sowie der Verteilung der Operatoren auf die Rechnerknoten experimentiert werden.

## Paralleles Data Mining - Verfahren

### Assoziations- und Klassifikationsregeln

hier Evaluation der Güte gefundener Regeln interessant, meist datenparallel

### Entscheidungsbäume

Inter-Node-, Inter-Attribut- und Datenparallelität möglich

### Neuronale Netze

Netz entweder verteilt oder global, Problem sind globale Gewichtsupdates

### Genetische Algorithmen

entweder mehrere, auf die Knoten verteilte Populationen oder reine Datenparallelität

### Instance-Based Learning

Klassifikation aller Tupel auf partitionierten Daten oder verschiedene Tupel parallel

***RESH: Projekt L1*** ***Matthias Gimbel***

Parallel zu den Arbeiten an der Algebramaschine wurden auch verschiedene Analyseverfahren und deren Parallelisierungsmöglichkeiten untersucht. Das Ergebnis der 4 erstgenannten Analyseverfahren ist dabei ein Modell (z.B. eine Menge von Regeln oder ein neuronales Netz etc.), welches das gelernte "Wissen" darstellt. Beim Instance-based-Learning dienen die gesamten vorhandenen Daten als Modell und werden zur Klasifikation anderer Datensätze benutzt. Bezüglich der Parallelisierung werden grob zwei Kategorien unterschieden [8]:

- Datenparallel bedeutet in diesem Zusammenhang: Jeder Knoten hat seine eigenen Daten und berechnet bzw. überprüft das von allen Knoten gemeinsam erstellte Modell daran.
- Im Gegensatz dazu ist bei kontrollparallelen Ansätzen das Modell auf die Rechnerknoten verteilt und jeder Knoten hat entweder alle Daten zur Verfügung oder erstellt nur ein lokal gültiges Modell.

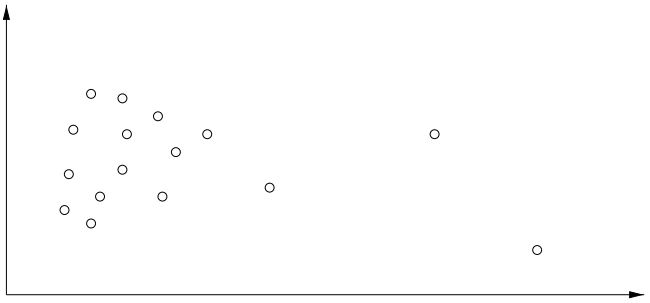
An dieser Stelle nur eine kurze Einordnung der Verfahren:

- Assoziationsregeln sind ein klassisches und weitverbreitetes Beispiel für Data-Mining insgesamt. Sie werden z.B. in der Warenkorbanalyse eingesetzt und liefern Zusammenhänge der Form: $X \Rightarrow Y$, also: wenn die Produkt in X gekauft werden, dann auch die in Y usw...
- Ein weiteres wichtiges Paradigma sind die Entscheidungsbäume. Sie dienen dazu, Datensätze zu klassifizieren, indem beginnend bei der Wurzel in jedem Knoten der Wert eines einzelnen Attributs geprüft und in den diesem Wert entsprechenden Zweig weitergegangen wird, bis man an einem Blatt angelangt ist, welches über die entsprechende Klasse Auskunft gibt.
- Neuronale Netze und genetische Algorithmen spielen im KDD-Bereich eine eher untergeordnete Rolle. Auch für diese Verfahren existieren aber bereits parallele Ansätze. Eine Besonderheit dieser Verfahren ist, daß hier aus Laufzeitgründen oft von der Vorgabe, das optimale sequentielle Modell (also das Modell, das der entsprechende sequentielle Algorithmus generiert hätte) zu finden, abgewichen wird.

- Bei Instance-Based-Learning werden die zu klassifizierenden Datensätze anhand der Klasse der nächsten Nachbarn klassifiziert. Dazu wird eine Metrik eingeführt, die anhand der Attributwerte die Distanz zweier Datensätze liefert.

Zu all diesen Verfahren existieren bisher nur einzelne Implementierungen, die auf eine bestimmte Rechnerarchitektur zugeschnitten sind und nur eine bestimmte Parallelisierungsstrategie verfolgen. Variable oder gar auf die zugrundeliegende Architektur optimierende Verfahren fehlen.

Das Outlier-Problem



Anwendungen: Finanzwesen (Credit Card Fraud),
Verkehrsüberwachung, Qualitätskontrolle

neues Gebiet: erweiterte Konsistenzprüfung in Datenbanksystemen
(Schutz vor "schleichender" Sabotage an Datenbeständen)

Im allgemeinen hohe Dimension (5-100), Komplexität $d*n^2$

***RESH: Projekt L1*** ***Matthias Gimbel***

Ein neues und interessantes Problem ist die Suche nach Ausreißern in großen Datenbeständen [17]. Ein Ausreißer in einem Datenbestand ist dadurch charakterisiert, daß seine Merkmale von denen der allermeisten anderen Einträge abweichen. Hinsichtlich der genauen Fassung des Begriffs existieren verschiedene Definitionen. Ausreißer lassen sich beispielsweise als unwahrscheinliche Elemente einer als bekannt vorausgesetzten statistischen Verteilung (distribution based) oder durch große Abweichungen im Sinne einer Metrik im Datenraum (distance-based) charakterisieren.

Waren die bisherigen Analyseverfahren alle darauf ausgelegt, die Effekte solcher Ausreißer möglichst zu eliminieren, um zu kompakten, aussagekräftigen Modellen zu kommen (s. z.B. [16]), so werden hier effiziente Verfahren benötigt, um gerade diese Ausreißer zu entdecken. Das Problem ist hierbei, daß die Suche nach Ausreißern in Räumen der Dimension 20 bei Datenbeständen mit mehreren Millionen Tupeln sehr aufwendig ist und herkömmliche Verfahren nur sehr eingeschränkt brauchbar sind.

Diese Problematik wird bei der weiteren Arbeit als weiteres Analyseverfahren berücksichtigt werden.

## Zusammenfassung, Ausblick

Zusammenfassung:

- Vorverarbeitungsstufen sind parallelisiert
- JavaParty durchaus geeignet

Ausblick:

- abschließende Untersuchung der Algebramaschine
- Konzentration auf Data-Mining-Verfahren
- Schwerpunkt: Parallelisierungsmethoden, angepaßte DBMS-Operatoren
  - zunächst: Outlier-Problem !



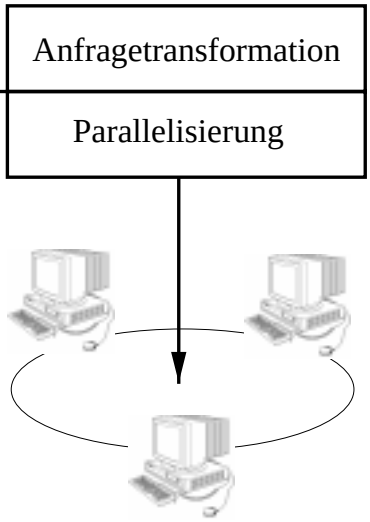
Abschließend noch ein kurzer Ausblick auf die Arbeit des nächste Halbjahres:

Zunächst sind noch Verbesserungen (vor allem im Bereich der Anfrageoptimierung und der Leistung) sowie die Durchführung der im Abschnitt “Erste Ergebnisse” angekündigten ausgiebigen Messungen an der Algebramaschine vorgesehen. Im weiteren Verlauf ist dann ihre Erweiterung um bzw. Anpassung an die objektorientierten IM-Operatoren sowie die Anbindung eines parallelen Data-Mining-Verfahrens geplant.

# Forschergruppe RESH

Rechnernetze als Superrechner und Hochleistungsdatenbanken

## Paralleles Data-Mining




Vernetzte Workstations bieten:

- enormen Hauptspeicher
- hohe I/O-Bandbreite
- effiziente Kommunikation durch ParaStation-Kopplung

=> großes Potential für preiswerte und skalierbare Hochleistungsdatenbankanwendungen

Effizienzsteigerung im Wissensgewinnungsprozeß durch:

- Nutzung feingranularer Parallelität
- Integration von Vorverarbeitungs- und Data-Mining-Operatoren in die Anfragebearbeitung des DBMS

**Ansprechpartner / Kontakt:**

IPD Prof. Lockemann
Universität Karlsruhe
Am Fasanengarten 5
D-76131 Karlsruhe
GERMANY

Tel: +49/721/608-3495
Fax: +49/721/608-7343
Email: gimbel@ira.uka.de
URL: http://wwwipd.ira.uka.de/RESH/L1

# 5 N1 & N2: Bildfolgenauswertung als Anwendungsprobe zur quantitativen Untersuchung von Parallelisierungsansätzen

## 5.1 Einführung

Die Aufgabe der Bildauswertung besteht darin, durch ein (digitisiertes) Bild eingefangene Informationen in Aussagen über die abgebildete Szene zu überführen. Entsprechend befaßt sich die Bild*folgen*auswertung insbesondere mit der algorithmischen Transformation von digitisierten Bildfolgen in Aussagen über *zeitliche* Veränderungen in der abgebildeten Szene.

Im allgemeinen kann man mindestens vier verschiedene Auswertungsebenen unterscheiden:

1. Auswertungsoperationen im *Bildbereich*, beispielsweise die Bestimmung von Kantenelementen aus örtlichen Grauwertvariationen.
2. Die Extraktion von geometrischen Beschreibungen im *Szenenbereich*.
3. Die Umsetzung geometrischer Beschreibungen in *begriffliche* Beschreibungen, die beispielsweise durch Logik-Operationen verknüpft werden können, um nicht (primär) geometrische Konsistenzprüfungen vorzunehmen.
4. Die Transformation begrifflicher Beschreibungen in *natürlichsprachlichen Text*.

Alternativ können bereits geometrische Zwischenergebnisse im Bild- oder Szenenbereich einem Benutzer grafisch dargeboten werden. Eine weitere Möglichkeit besteht darin, geometrische Zwischenergebnisse einer Bildfolgenauswertung zur Regelung der Bewegungen beispielsweise eines stationären oder mobilen Roboters heranzuziehen.

Der Umfang des auszuwertenden Datenstromes – eine normale Video-Kamera gibt mehr als 10 MByte pro sec aus – sowie die Komplexität der für eine Bildfolgenauswertung erforderlichen Berechnungen beanspruchen beträchtliche Rechenkapazitäten. Dies gilt selbst dann, wenn man keine mit der Bildaufnahme schritthaltende Auswertung fordert, wie sie beispielsweise bei der Integration der Bildfolgenauswertung in den Regelkreis eines Roboters angestrebt wird.

Da die Rechenleistung von normalerweise in einem Labor zur Verfügung stehenden Arbeitsplatzstationen noch weit unter der für ein flüssiges Experimentieren wünschbaren Größe bleibt, liegt es nahe, durch Verteilung der Berechnungen auf mehrere Rechner die Durchführung von Experimenten zu beschleunigen. Abhängig von den jeweilig zu bearbeitenden Aufgaben werden verschiedene Ansätze verfolgt, um die für Experimente erforderlichen Rechenkapazitäten zur Verfügung zu stellen. So wird man zur Einhaltung der Echtzeitanforderungen bei sichtsystemgestützten Regelungsexperimenten anders vorgehen als in Fällen, bei denen komplexe Verfolgungsvorgänge an längeren Bildfolgen interaktiv untersucht werden müssen. Im zweiten Beispiel wird man Wert auf die rasche Bereitstellung leicht inspizierbarer bildlich und graphisch dargebotener Informationen legen, um dem Experimentator die Formulierung und Überprüfung von Hypothesen zur Erklärung fehlerhafter oder noch nicht befriedigender Resultate zu erleichtern.

Der Umfang und die Variationsbreite der zu verarbeitenden Daten, die Komplexität und Verschiedenartigkeit der durchzuführenden Berechnungen sowie die Notwendigkeit, bei einzelnen Auswertungsschritten bestimmte Rechenzeitgrenzen einzuhalten, lassen die Bildfolgenauswertung zu einer Rechenlast werden, die zur Beurteilung der Möglichkeiten und Grenzen kleinerer Parallelrechneranordnungen geeignet erscheint.

Wir werden zunächst Erfahrungen mit dem Einsatz von Parallelrechnern zur Bildfolgenauswertung umreißen, weil unseres Erachtens dadurch besser verständlich wird, weshalb die Adaptation eines Bildfolgenauswertungssystems als Rechenlast zur Beurteilung der Möglichkeiten und Grenzen der 'Parastation' [2] mehr Fragen als ursprünglich vermutet aufwirft. Daran anschließend gehen wir auf Überlegungen ein, durch welche Experimente sich nach unserem gegenwärtigen Verständnis sinnvolle Meßwerte über den Einsatz der Parastation zur Bildfolgenauswertung gewinnen lassen.

## 5.2 Verwendung von Parallelrechnern zur Bildfolgenauswertung

Wie bereits angedeutet, greifen wir auf Erfahrungen aus unterschiedlichen Einsatzbereichen der Bildfolgenauswertung zurück. Dabei handelt es sich einerseits um die – teilweise noch interaktiv erfolgende – Auswertung monokularer Bildfolgen von Verkehrsszenen mit Hilfe des Programmsystems `Xtrack`. Auf der anderen Seite wurden unsere Überlegungen auch durch über zehnjährige Erfahrungen beim Aufbau einer Demontagezelle geprägt, in deren Rahmen zahlreiche Experimente zur sichtsystemgestützten Roboterführung konzipiert und durchgeführt worden sind.

### 5.2.1 Zur sichtsystemgestützten Roboterführung

Die Dissertation von Gengenbach [9] gibt einen guten Überblick über die Bemühungen, an zunächst einfachen Aufgaben verschiedene Teilfragen der sichtsystemgestützten Roboterführung zu untersuchen. Um dies zu ermöglichen, hat Gengenbach unter Verwendung zahlreicher Transputer einen Spezialrechner zur schnellen Bestimmung von Kantenelementen und von Optischen-Fluß-Vektoren konzipiert, realisiert und in ein experimentelles Gesamtsystem integriert. Die signalnahen Auswertungsschritte lassen sich damit sehr effizient und dennoch für damalige Verhältnisse erstaunlich preiswert durchführen. Zusätzlich zu den Transputern wurden noch mehrere 'Digitale Signalprozessoren (DSP)' in dieses inhomogene Rechnernetz integriert, um die anstehenden Aufgaben bearbeiten zu können. Wie in [9] gezeigt worden ist, ließen sich auf diesem Wege elementare sichtsystemgestützte Manipulationen studieren.

Sobald aber mehr Erfahrungen vorlagen, zeigte sich immer deutlicher die Notwendigkeit, komplexere Operationen auf einem noch *leistungsfähigeren* Rechner(netz) zu realisieren. Dabei sollte nach Möglichkeit in einer höheren Programmiersprache programmiert werden, um unverhältnismäßig lange Ausprüfzeiten zu vermeiden. Glücklicherweise gelang es im Laufe des Jahres 1995, einen strukturierten Parallelrechner – die sogenannte 'Gigamaschine' – zu erwerben und in unsere experimentelle Demontagezelle zu integrieren. Die in unserer Demontagezelle eingesetzte Gigamaschinen-Version ist aus 16 SPARC-Prozessoren aufgebaut und kann in C sowie C++ programmiert werden. Die durch die Gigamaschine verfügbar gewordene Rechenleistung gestattete die Untersuchung wesentlich komplexerer Fragestellungen. Die aus diesen Untersuchungen hervorgegangene Dissertation von Tonko umfaßt auch eine ausführlichere Diskussion verschiedener zur Bildauswertung eingesetzter Systemarchitekturen [32].

Allerdings zeigte sich bereits im Laufe des Jahres 1997, daß selbst die Rechenleistung der Gigamaschine und die bei ihrer Nutzung zu beachtenden Randbedingungen es nicht erlaubten, den inzwischen erreichten Stand der Systementwicklung zügig weiter auszubauen. Daher wurde im Spätherbst 1997 eine Sun Ultra-2 Arbeitsplatzstation mit zwei 300 MHz ULTRA-II Prozessoren bestellt. Die Migration des inzwischen entstandenen Programmsystems für Versuche mit der Demontagezelle von der Gigamaschine auf die Ultra-2 beanspruchte wesentlich mehr Zeit als ursprünglich erwartet. Eine Analyse der Gründe dafür ergab zu erwartende, aber auch unerwartete Einsichten.

Da die beiden Mitarbeiter, die über mehrere Jahre hinweg das System von Programmen auf der Gigamaschine zum Experimentieren mit der Demontagezelle entwickelt hatten, kurz hintereinander ausschieden, trat ein deutlicher Bruch im verfügbaren Wissen über den aktuellen Umgang mit der gesamten Anlage ein. Angesichts der Tatsache, daß dieses Programmsystem im Rahmen von Dissertationsprojekten entwickelt worden war, war verständlicherweise wichtiges Praxiswissen vorzugsweise in den Köpfen dieser Mitarbeiter und nicht in einer umfangreichen Dokumentation gespeichert und daher nicht mehr unmittelbar zugänglich. Mit solchen Situationen sehen sich universitäre Institutionen mehr oder weniger regelmäßig konfrontiert, d. h. damit mußte prinzipiell gerechnet werden.

Unerwarteter war dagegen, daß die Einarbeitung neuer Mitarbeiter durch folgende Problematik erschwert wurde. Im Rahmen eines EU-Projektes unter Nutzung der jeweils verfügbaren Version unserer Demontagezelle wurde unter anderem untersucht, welche Lösungsansätze sich für eine noch stärkere örtliche Verteilung von Rechenkapazitäten erarbeiten lassen, sofern man eine Vernetzung über mittelschnelle ATM-Verbindungen ins Auge faßte. Damit sollte u. a. der Frage nachgegangen werden, wie man dezentral verfügbare Lösungsansätze für bestimmte Teilprobleme der Bildfolgenauswertung zur Untersuchung sichtsystemgestützter Regelungsaufgaben ohne zeitaufwendige Adaptation von Programmsystemen an eine neue Rechnerumgebung nutzbar machen könnte. Angestrebt wurde insbesondere, Zwischenergebnisse *über öffentliche ATM-Verbindungen* auf entfernte Rechner zu übertragen. Die durch dort verfügbare

Programme berechneten Auswertungsergebnisse sollten über öffentliche ATM-Verbindungen an das lokale Rechnernetz zurückübertragen werden, das dann die eigentlichen Stellbefehle an die involvierten Roboter absetzte. Durch mehrere Demonstrationen konnte die prinzipielle Begehbarkeit eines solchen Lösungsweges nachgewiesen werden.

Allerdings zeigte sich bei den vorbereitenden Untersuchungen, daß – zumindest zum damaligen Zeitpunkt im Sommer 1996 – die Umlaufzeiten zwischen lokalem und abgesetztem Rechner(netz) stärker schwankten, als dies ohne Kompensation für eine Regelung tolerierbar war (siehe auch [33]). Ein Ausweg bestand darin, alle Daten mit Zeitstempeln zu versehen und den dezentral eingesetzten Teilprogrammen durch entsprechende Teilprozesse eine für die gesamte Anwendung *globale Uhrzeit* mit brauchbarer Genauigkeit zur Verfügung zu stellen. Die Einführung und Nutzung solcher globaler Zeitangaben erforderte die Einfügung weitgehend ähnlicher Anweisungsfolgen in zahlreiche Routinen des bereits vorher umfangreichen Programmsystems. Diese und andere, ähnlich gelagerte Teilaufgaben wurden mit Hilfe eines 'Vorübersetzers' bewältigt, der die Einfügung und Verwaltung solcher Programm-Modifikationen – u. a. mit Hilfe von Schablonen – übernahm [28].

Davon abgesehen wurden auch andere Programm-Modifikationen notwendig, um den speziellen Randbedingungen der Gigamaschine Rechnung zu tragen. Durch konsequent 'objektorientiertes' Programmieren wurde versucht, die Zerlegung eines umfangreichen Programmsystems in zahlreiche Teilprozesse und deren Verteilung auf die einzelnen Prozessoren der Gigamaschine zu unterstützen.

Im Laufe der Zeit führte jedoch die Wechselwirkung dieser und weiterer Programm-Modifikationen zu einem Zustand, der für Außenstehende nicht mehr ohne Weiteres nachvollziehbar war. Da die Nachfolgeanlage der Gigamaschine nur noch zwei – allerdings wesentlich leistungsfähigere – Prozessoren aufwies gegenüber den 16 Prozessoren der Gigamaschine, war die an die Gigamaschine adaptierte Prozeßzerlegung keineswegs mehr als optimal für die Ultra-2 zu betrachten. Obwohl die eigentliche Migration des in der Dissertation [32] behandelten Programmsystems auf die Ultra-2 im Winter 1997/98 nach Auslieferung und Installation dieser Anlage noch erstaunlich rasch bewerkstelligt werden konnte, warf die Optimierung dieses Systems unter den geänderten Randbedingungen Probleme auf, die noch immer nicht als vollständig gelöst angesehen werden.

Nach unserem gegenwärtigen Verständnis lassen sich aus den inzwischen gesammelten Erfahrungen zwei Schlußfolgerungen ziehen:

- Selbst eine zum Entwurfszeitpunkt gut durchdachte 'objektorientierte' Programmierung kann sich bei ständigen Änderungen, die durch laufendes Experimentieren erzwungen werden, innerhalb verhältnismäßig kurzer Zeit als Hemmnis für den Versuch herausstellen, solcherart entstandene Programme rasch zu durchschauen, um sie an veränderte Verhältnisse anzupassen.
- Der Arbeitsaufwand für eine System-Modifikation, von der man sich eine – vielleicht kleine, aber dennoch wünschenswerte – Verkürzung der Ausführungszeit verspricht, läßt sich praktisch nicht im voraus abschätzen.

Insbesondere die zweite Beobachtung bedingt, daß man bei knappen Ressourcen nicht zuverlässig entscheiden kann, ob der mit einer Modifikation erwartete Gewinn bei der Gesamtsystem-Leistung den mit der Modifikation verknüpften Zeit- (und damit Kosten-) Aufwand rechtfertigt.

> *Selbst wenn es gelingen sollte, den mit einer Adaptationsmaßnahme an die speziellen Bedingungen eines Parallelrechners zu erwartenden Rechenzeitgewinn abzuschätzen, läßt sich noch keine vertretbare Kosten-/Nutzen-Analyse durchführen. Dazu muß auch bekannt sein, welche Zeit für die Abklärung, Konzeption, Implementation und Ausprüfung einer Modifikation benötigt wird.*

Hinzu kommt eine weitere Erwägung: angesichts der Tatsache, daß sich zur Zeit die Rechenkapazität einer Arbeitsplatzstation etwa alle anderthalb Jahre verdoppelt, müssen Kosten-Nutzen-Analysen im Prinzip nach ein bis zwei Jahren grundsätzlich überprüft werden. Was vor zwei Jahren noch die Lösung 'aller' Probleme darstellte, kann heute selber zum Problem geworden sein. Daraus folgt, daß man

> *im Prinzip nicht nur den Aufwand abschätzen müßte, der zum aktuellen Zeitpunkt für eine – rechenzeitverkürzende – Adaptation an eine spezielle Parallelprozessorstruktur anfällt, sondern auch noch den in Zukunft eventuell anfallenden* **Aufwand für einen Rückbau** *von solchen Programm-Modifikationen.*

Wie die oben angedeuteten Erfahrungen belegen sollen, ist dieser Aspekt vor mehreren Jahren noch gar nicht gesehen worden.

Auf der anderen Seite ist noch einmal auf die Diskrepanz hinzuweisen, die immer noch besteht zwischen der für ein 'normales universitäres' Labor erschwingbaren Rechenkapazität und derjenigen, die für eine Echtzeitauswertung von Videobildfolgen erforderlich ist. Letzteres gilt insbesondere, wenn Videobildfolgen nicht unter stark vereinfachenden Annahmen ausgewertet werden sollen, da dann ein analoges Dilemma zu demjenigen auftaucht, das gerade für die 'Überadaptation' an die Gegebenheiten einer speziellen Parallelrechneranordnung diskutiert worden ist.

> *Wird ein Verfahren zur Bildfolgenauswertung zu stark darauf ausgerichtet, bei einer gegebenen Rechenkapazität mit einer vorgegebenen Wiederholungsrate oder Latenzzeit ausgeführt zu werden, so beobachtet man häufig, daß die Verfahrensentwicklung bei Verfügbarwerden einer etwa um den Faktor zwei größeren Rechenkapazität weitgehend neu zu konzipieren und zu realisieren ist.*

Mit anderen Worten, die im Hinblick auf beschränkte Rechenkapazitäten getroffenen Abwägungen zwischen Rechenzeit, Flexibilität des Einsatzbereiches und Robustheit – um nur einige Gesichtspunkte zu nennen – resultieren *beim gegenwärtig verfügbaren Niveau an Rechenkapazität* in Lösungsansätzen, die sich oft als Sackgassen herausstellen.

Die Berücksichtigung zusätzlicher, über die Einhaltung von Rechenzeiten hinausgehender, Gesichtspunkte sowie die Veränderung der Gewichtung zwischen den verschiedenen Aspekten, die bei zusätzlich verfügbarer Rechenkapazität einsetzt, ist im Hinblick auf die sichtsystemgestützte Roboterregelung in [23] ausführlicher diskutiert worden.

### 5.2.2 Das `Xtrack`-System

Die Auswertung von Bildfolgen, die mit einer *stationären* Videokamera von innerstädtischen Straßenverkehrsszenen aufgenommen worden sind, steht beim gegenwärtigen Stand der Entwicklung vor anderen Anforderungen als die sichtsystemgestützte Führung von Robotern in einer Demontagezelle. Die Komplexität der Szene, die Vielfalt zu berücksichtigender Objekte und Bewegungen sowie die Variation der Beleuchtungs- und Verdeckungsverhältnisse lassen selbst zum gegenwärtigen Zeitpunkt eine Echtzeitauswertung nicht zu. Dies galt in den zurückliegenden Jahren noch sehr viel ausgeprägter.

Es gibt zwar Aufgabenstellungen, bei denen eine Echtzeitauswertung von Videobildfolgen angestrebt wird, die aber nur unter gravierenden Einschränkungen im Hinblick auf den Einsatzbereich sowie unter Ausnutzung zahlreicher Zusatzannahmen aufgegriffen werden. Als Beispiel mag etwa die Zählung von Fahrzeugen auf nur in einer Richtung befahrenen geradlinigen Straßenabschnitten wie Autobahnen oder Schnellstraßen dienen.

Auch hier gilt eine bereits im vorangehenden Abschnitt angeführte Überlegung, daß praktisch jede Lösung bei Verfügbarwerden signifikant größerer Rechenkapazitäten bei vergleichbaren Preisen grundlegend in Frage gestellt werden muß. Angesichts dieses Sachverhaltes wurde im Bereich Kognitive Systeme das Schwergewicht der Entwicklung auf Robustheit und nicht auf Schnelligkeit bei der Auswertung gelegt. Das aus langjährigen Untersuchungen hervorgegangene Bildfolgenauswertungssystem `Xtrack` – siehe etwa [18, 19, 20, 21, 22, 11] – stellt daher auch andere Herausforderungen an Versuche, durch Übergang auf eine Parallelrechneranordnung eine signifikante Beschleunigung der Auswertungszeiten zu erzielen.

Charakteristisch für `Xtrack` sind u. a. die sehr differenzierten Ausgabemöglichkeiten, die es gestatten, praktisch die meisten Programm-Entscheidungen bei Bedarf am konkreten Einzelfall zu überprüfen. Nur dadurch wurde es möglich, selten auftretende Schwierigkeiten rasch daraufhin zu untersuchen, ob es sich um einen Kodierungsfehler eines grundsätzlich richtigen Lösungsansatzes handelte, um eine unerwartete Verletzung von Annahmen, die dem Lösungsansatz zu Grunde liegen, um unerwartete Auswirkungen von gestörten Rohdaten oder aber um Artefakte des Auswertungsprozesses, d. h. unwartete Auswirkungen irgendwelcher Vereinfachungen, die urspünglich für unkritisch gehalten worden waren. Die Problematik wird richtig deutlich, wenn man sich vergegenwärtigt, daß Schwierigkeiten der genannten Art sich oft erst Hunderte von Aufnahmen nach der eigentlichen Ursache in auffälligen Abweichungen vom erwarteten Programmverhalten bemerkbar machen können.

In vielen Fällen lassen sich solche Fehler nur durch sorgfältiges Eingrenzen verschiedener Zwischenphänomene diagnostizieren, die zahlreiche Wiederholungen bestimmter Programmläufe erfordern. Dauert ein solcher Programmlauf zu lange, so droht die Gefahr, daß der Beobachter ermüdet und den geringfügigen Hinweis auf erste aufkeimende Schwierigkeiten übersieht, die sich 'in voller Schönheit' erst sehr viel später zeigen können.

Ein weiterer Problemkreis ergibt sich durch die inzwischen vertretbare Forderung, auch seltener auftretende Fehlerkonstellationen durch umfangreichere Versuchsläufe zu detektieren. Da solche Versuchsläufe durchaus Tage kosten können, zahlen sich selbst Beschleunigungen um einen Faktor 1,5 oder zwei sehr spürbar aus.

`Xtrack` bietet gegenüber einer sichtgestützten Regelung als Testlast für einen Parallelrechner den großen Vorteil, daß die Daten von einer Platte gelesen werden können und daher nicht die Verfügbarkeit einer komplexen mechanischen Apparatur wie einer Demontagezelle voraussetzen.

Allerdings hat auch hier die Erfahrung gezeigt, daß die Messung von Rechenzeiten für einen einzelnen Verfolgungsprozeß von vielen Einflußgrößen abhängen kann, beispielsweise der Größe und dem Kontrast des Abbildes eines zu verfolgenden Fahrzeuges, von der Textur des Fahrzeugabbildes oder von der Angemessenheit des eingesetzten Fahrzeug- und Bewegungsmodells. Im Prinzip kann bereits die Auswahl der aktivierten Testausgaben die Ausführungszeiten spürbar beeinflussen. Aus diesen Gründen sollte man nicht eine Einzelzeitmessung zur Grundlage der Beurteilung irgendeiner Verfahrensvariante machen, sondern die konsistente Variation einer Serie von Zeitmessungen unter genau kontrollierten Bedingungen auswerten.

Vor dem Hintergrund dieser Überlegungen wurde zunächst eine überschaubare Testbildfolge ausgewählt, mit deren Hilfe erste Zeitmessungen an einer aktuellen `Xtrack`-Version durchgeführt worden sind.

## 5.3 Vorbereitende Zeitmessungen an einer `Xtrack`-Version

Abbildung 1 zeigt die erste, eine mittlere und die letzte Aufnahme der Bildfolge 'Ettlinger-Tor-Platz'. In dieser Testbildfolge von 90 Aufnahmen sind elf Fahrzeuge unterschiedlicher Größe und Fahrtrichtung zu erkennen. In der zu ihrer Verfolgung herangezogenen Version von `Xtrack` wurden die grafischen Ausgaben soweit wie möglich – allerdings bisher aus technischen Gründen noch nicht vollständig –unterdrückt. Die Zeitmessungen wurden folgendermaßen durchgeführt:

1. Das Programm wurde für jedes der elf Fahrzeuge nacheinander gestartet und die für die einzelnen Fahrzeuge benötigte Rechenzeit und die Anzahl der verwendeten Halbbilder notiert.
2. Punkt 1. wurde fünfmal hintereinander auf jedem Prozessor durchgeführt. Aus allen Zeitmessungen wurde der mittlere Rechenzeitbedarf für ein Fahrzeug in Sekunden pro Halbbild errechnet.

Die Versuchserie wurde zunächst für eine Programmversion durchgeführt, die mit Hilfe des gcc-2.7.2.1 Compilers für die folgenden Sun-Rechner übersetzt worden war: Sun SPARCstation-10, Sun SPARCstation-20, Sun UltraSPARC-1, Sun UltraSPARC-2 (einmal mit 200 MHz Prozessoren und einmal mit 300 MHz Prozessoren).

Die damit erzielten Zeitmessungen wurden umgerechnet auf die mittleren *Bildwiederholraten* (Kehrwert der jeweiligen Zeitmessung) und sind in Abbildung 2 dargestellt. Man erkennt, daß die Rate, mit der im Mittel über die in der Testbildfolge auftretenden Fahrzeuge durch ein Halbbild verfolgt werden können, in guter Näherung proportional zur Taktfrequenz des Prozessors ansteigt. Diese Beobachtung überraschte insofern, als zunächst davon ausgegangen worden war, daß der Übergang auf eine neue Prozessorgeneration sich in einer spürbaren Änderung der mittleren Auswertungszeit hätte bemerkbar machen sollen.

Um sich gegen eine Fehldeutung der Messungen abzusichern, wurde der Versuch auf einer Alphastation wiederholt, die über einen mit 500 MHz getakteten Prozessor des Typs 21164A verfügte. Dazu mußte der Quelltext der `Xtrack`-Version geringfügig an die speziellen Konventionen des auf der Alphastation zur Verfügung stehenden Compilers adaptiert werden. Die Messung auf dieser Alphastation ergab eine Rate von 0,725 Halbbildern pro Sekunde für ein Fahrzeug, also überproportional schneller als auf der schnellsten (300 Mhz) zur Verfügung stehenden Sun Ultra-2, bei der allerdings nur einer von den beiden Prozessoren durch `Xtrack` genutzt worden war. Die Messungen wurden daraufhin auf einer Alphastation des Typs 21164A wiederholt, deren Prozessor mit 600 MHz getaktet wird. Die entsprechenden Meßergebnisse sind in Abbildung 3 wiedergegeben. Man erkennt, daß die Steigung der Auswertungsrate als Funktion der Frequenz deutlich steiler verläuft als auf den verfügbaren Sun-Anlagen.

Da der 'Hebelarm' zur Bestimmung dieser Steigung mit 100 MHz deutlich kleiner als bei den Sun-Anlagen ist, wurde die Messung auch noch auf einer Alphastation vom Typ 21064 wiederholt, deren Prozessor nur

Abbildung 1: Die erste (oben), eine mittlere (Mitte) und die letzte (unten) Aufnahme der Bildfolge 'Ettlinger-Tor-Platz'.

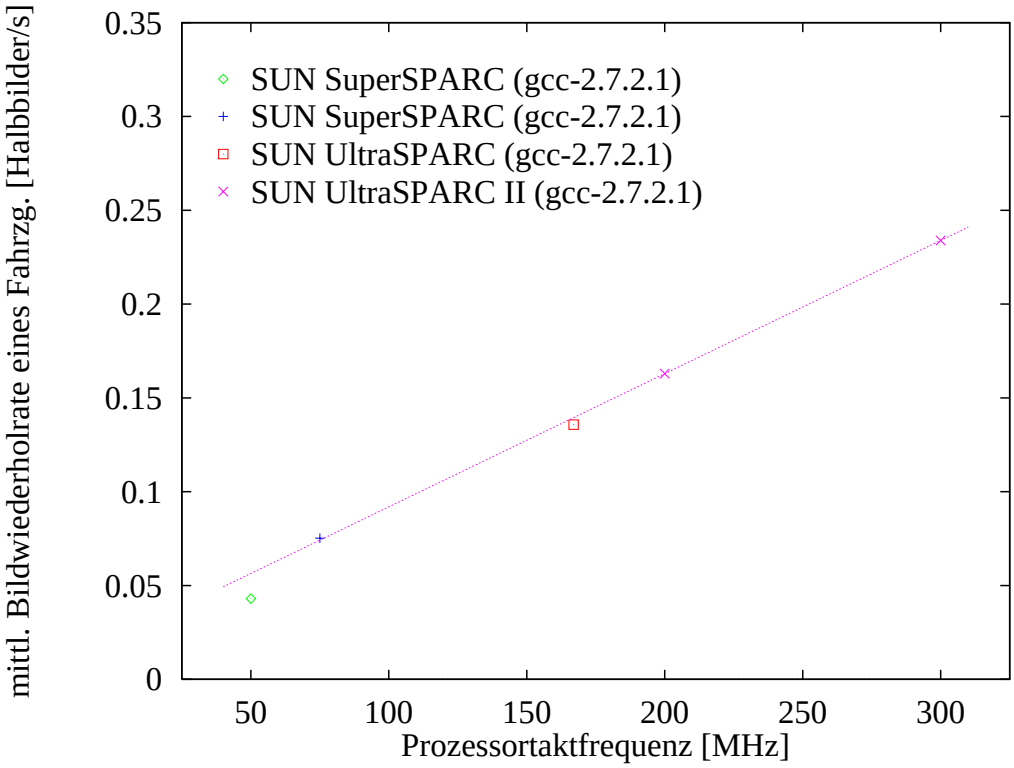



Abbildung 2: Mittlere Bildwiederholrate eines Fahrzeuges in Abhängigkeit von der Taktfrequenz der verwendeten Prozessoren der Firma Sun. Die Programme, die diese Ergebnisse erzielten, wurden mit dem gcc-2.7.2.1 Compiler übersetzt.

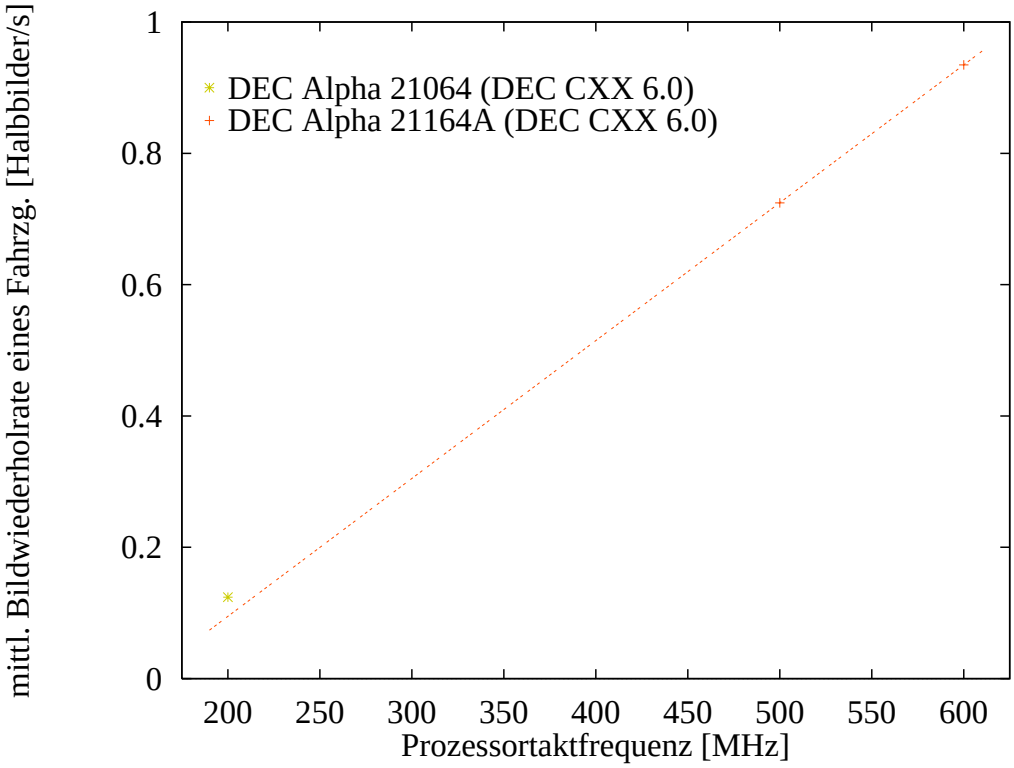



Abbildung 3: Mittlere Bildwiederholrate eines Fahrzeuges in Abhängigkeit von der Taktfrequenz der verwendeten Prozessoren der Firma Dec.

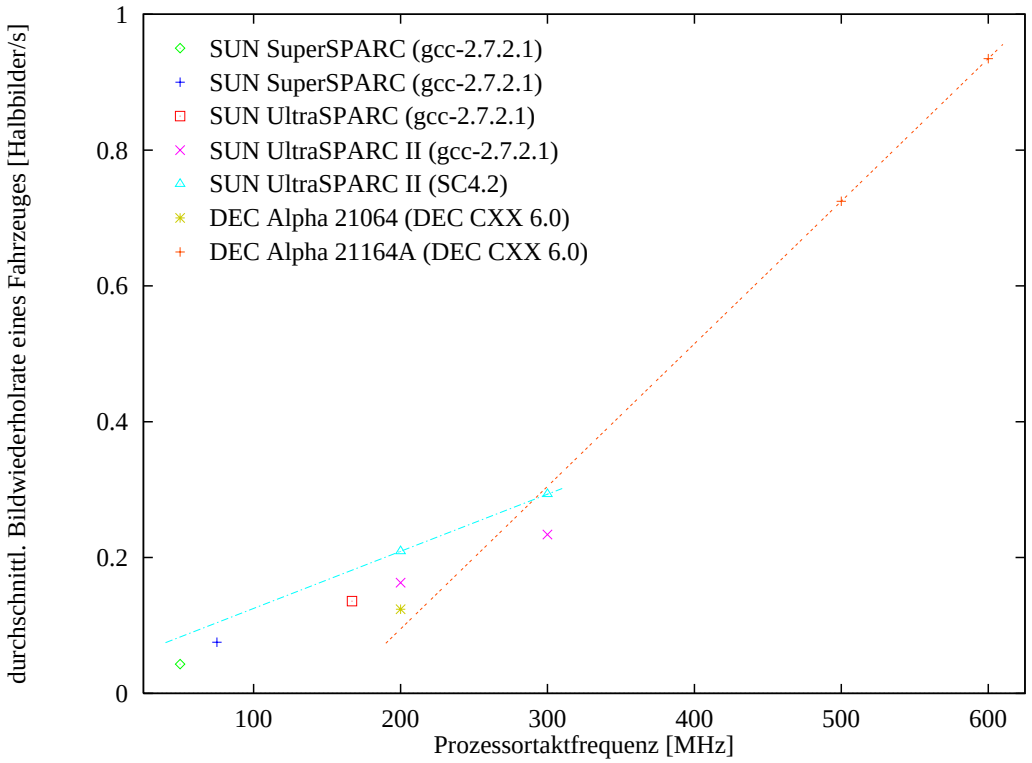



Abbildung 4: Mittlere Bildwiederholrate eines Fahrzeuges in Abhängigkeit von der Taktfrequenz aller verwendeten Prozessoren.

mit 200 MHz getaktet wird. Wie aus Abbildung 4 zu erkennen ist, liegt der so erhaltene Meßwert *unter* demjenigen einer mit 200 MHz getakteten Ultra-2. Auch im Fall der DEC Alphastationen, bei denen die Prozessortaktfrequenzen immerhin über einen Bereich von 300 MHz variieren, kann man die gemessenen Auswertungsraten in guter Näherung als lineare Funktion der Prozessortaktfrequenz beschreiben, wobei allerdings die Steigung der Geraden deutlich größer als für die Sun-Anlagen ist.

Da ein Ergebnis in dieser Deutlichkeit nicht erwartet worden war, wurde die Hypothese überprüft, ob zumindest ein Teil der für diese `Xtrack`-Version auf den Alphastationen gemessenen größeren Rechnerleistung auf den proprietären C/C++-Compiler der Firma DEC zurückzuführen sei. Dazu wurde die neueste Version des Sun-C/C++-Compilers (4.2) besorgt, um den Versuch auf den Ultra-2 Arbeitsplatzstationen nach Übersetzung von `Xtrack` mit diesem Compiler zu wiederholen.

Dabei stellte sich heraus, daß der Sun-4.2 Compiler ebenfalls eine in Details von den übrigen Compilern abweichende Sprachversion von C/C++ akzeptierte, was einen nicht unbeträchtlichen Adaptationsaufwand erforderte. Nachdem die `Xtrack`-Version an den Sun-4.2 Compiler adaptiert worden war, führten die analogen Messungen zu den in Abbildung 2 wiedergegebenen Meßwerten: es zeigte sich, daß nach Übersetzung mit dem proprietären Compiler von Sun auf den 200 MHz Ultra-2 Arbeitsplatzstationen nur noch 75 % der Rechenzeit benötigt wurde wie nach einer Übersetzung mit dem gcc-2.7.2.1 Compiler.

## 5.4 Ausblick

Es wird erwartet, daß Laufzeitmessungen mit Hilfe des `Xtrack`-Systems zu – möglicherweise nur geringfügigen, unter Umständen aber auch größeren – Unterschieden führen, je nachdem, wie eine Messung angesetzt wird. Dabei erscheint es nach unserem gegenwärtigen Verständnis sinnvoll, drei grundlegende Aspekte im Auge zu behalten:

1. Die Anzahl der nebenläufigen Prozesse, die innerhalb eines Meßlaufes gestartet werden und damit die ‘Verwaltungslast’ des Rechensystems festlegen.

2. Die Wechselwirkungsmöglichkeiten zwischen verschiedenen Prozessen, die wesentlich durch die Verteilung der Aufgaben auf die einzelnen Teilprozesse bedingt werden.

3. Die Art der Berechnungen, die innerhalb eines Teilprozesses auszuführen sind.

Wichtig erscheint es, diese Unterschiede in der erwarteten Last für das zu prüfende Rechensystem durch effizient realisierte Parameterwahl leicht und in hinreichend großem Umfange variieren zu können.

Gedacht wird daran, die Zahl der Prozesse beispielsweise durch die Anzahl der parallel in einer Bildfolge zu verfolgenden Fahrzeuge vorzugeben. Wechselwirkungsmöglichkeiten lassen sich zum Beispiel dadurch beeinflussen, daß zwei oder mehrere sich teilweise verdeckende Fahrzeuge parallel verfolgt werden, so daß Wechselwirkungen zwischen den Verfolgungs-Teilprozessen durch die Notwendigkeit der Auswertung von Verdeckungsbeziehungen erzwungen werden.

Von besonderem Interesse ist auch, ob sich Einflüsse durch einen grundsätzlich verschiedenen Charakter der jeweils durchzuführenden Berechnungen nachweisen lassen. Wir beabsichtigen, die folgenden vier Kategorien von Rechenlasten zu überprüfen:

- 'Signalnahe' Berechnungen, die in erheblichem Maße unter Inanspruchnahme einer jeweils *lokalen* Umgebung von Pixeln durchgeführt werden können, zum Beispiel die Berechnung von Kantenelementen oder von Optischen-Fluß-Vektoren (siehe [25]).

- Stark durch Gleitkommarechnungen charakterisierbare Berechnungen geometrischen Charakters bei der Prädiktion und Aktualisierung von Zuständen von sich in der Szene bewegenden Körpern.

- Logische Operationen der in das System zu integrierenden Schlußfolgerungsprozesse auf Basis einer 'unscharfen, metrisch-temporalen (Horn-)Logik' (siehe [29]).

- Die durch eine 'normale' Nutzung von `Xtrack` anfallende 'Mischung' dieser verschiedenen Berechnungen.

Vorbereitungen für die Durchführung solcher Messungen sind inzwischen weit gediehen.

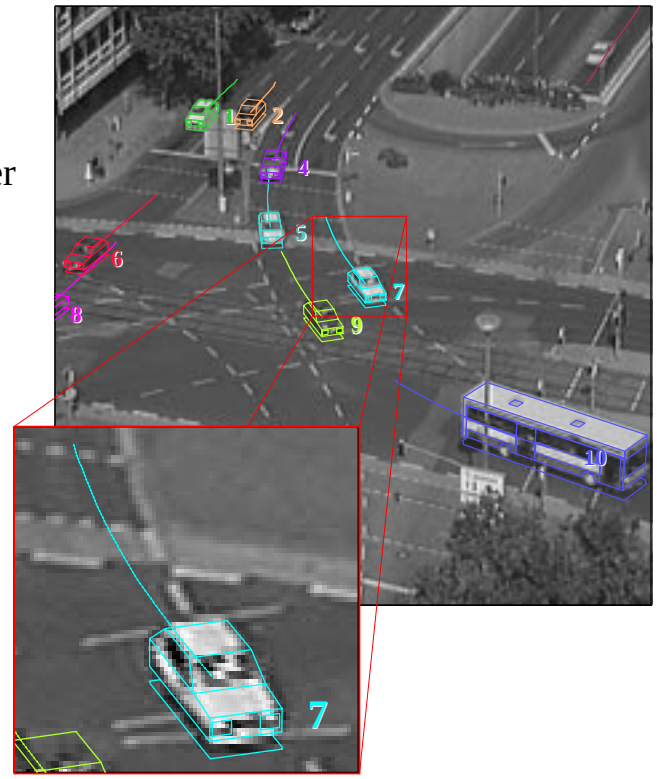
# Forschergruppe RESH

Rechnernetze als Superrechner und Hochleistungsdatenbanken

## Teilprojekt N1: Auswirkung sehr kurzer Latenzzeiten bei der Parallelisierung von Bildfolgenauswertungen

H. Leuck und H.-H. Nagel
Institut für Algorithmen und Kognitive Systeme
leuck@ira.uka.de
hhn@iitb.fhg.de

- Detektion und Verfolgung von Fahrzeugen in Videobildfolgen von Verkehrsszenen
- Untersuchungen von grob- bis feinkörniger Parallelisierung
- Quantitativer Vergleich verschiedener Parallelisierungs-alternativen
- enge Kooperation mit Entwicklern der ParaStation




**Ansprechpartner / Kontakt für RESH:**

IPD Prof. Tichy
Universität Karlsruhe
Am Fasanengarten 5
D-76131 Karlsruhe
GERMANY

Tel: +49/721/608-4317
Fax: +49/721/608-7343
Email: parastation@ira.uka.de
URL: http://wwwipd.ira.uka.de/RESH
URL: http://wwwipd.ira.uka.de/ParaStation

# Literatur


[1] Joachim M. Blum, Thomas M. Warschko, and F. Tichy. Parastation user level communication. In *Proceedings of the International Workshop on Distributed High Performance Computing and Gigabit Wide Area Networks*, Essen, September 1-5 1998. (to appear as Springer LNCS).

[2] Joachim M. Blum, Thomas M. Warschko, and Walter F. Tichy. The ParaStation Project. In *Proc. Third International ACPC Conference With Special Emphasis on Parallel Databases and Parallel I/O*, number 1127 in Lecture Notes in Computer Science, pages 215–218, Klagenfurt, Austria, 23–24 September 1996. Springer Verlag.

[3] Joachim M. Blum, Thomas M. Warschko, and Walter F. Tichy. Pulc: Parastation user-level communication. design and overview. In Jose Rolim, editor, *Parallel and Distributed Processing*, number 1388 in Lecture Notes in Computer Science, pages 498–509. Springer Verlag, March 1998.

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

[5] C.A. Breitner. *Ein Informationsmodell für Ableitungsprozesse und ihre Ergebnisse im Wissensgewinnungsproze*. PhD thesis, University of Karlsruhe, Fakultät für Informatik, 1998.

[6] Transaction Processing Council. http://www.tpc.org/dspec.html.

[7] J.Gray D.J. DeWitt. Parallel database systems: The future of high performance database systems. *Communications of the ACM*, 35(6):85–98, JUN 1992.

[8] A.A. Freitas and S.H. Lavington. *Mining very large Databases with parallel processing*. Kluwer Academic Publishers, 1998.

[9] V. Gengenbach. *Einsatz von Rückkopplungen in der Bildauswertung bei einem Hand–Auge–System zur automatischen Demontage*. PhD thesis, Institut für Algorithmen und Kognitive Systeme der Universität Karlsruhe, Sankt Augustin, Juli 1994.

[10] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. A high-performance, portable implementation of the MPI Message Passing Interface Standard. Technical report, Mathematics and Computer Science Division, Argonne National Laboratory, 1995.

[11] M. Haag. *Bildfolgenauswertung zur Erkennung der Absichten von Straßenverkehrsteilnehmern*. PhD thesis, Institut für Algorithmen und Kognitive Systeme der Universität Karlsruhe, Sankt Augustin, Juli 1998.

[12] U. Herzog. *Effiziente Konsistenzprüfung in Datenbanksystemen*. PhD thesis, University of Karlsruhe, Fakultät für Informatik, 1996.

[13] Matthias Jacob. Implementing Large-Scale Geophysical Algorithms with Java: A Feasibility Study. Master's thesis, University of Karlsruhe, Department of Informatics, November 1997.

[14] Java Grande Forum. http://www.javagrande.org.

[15] JavaParty. http://wwwipd.ira.uka.de/JavaParty.

[16] G.H. John. Robust decision trees: Removing outliers from databases. In *Proc. of the 1st International conference on Knowledge Discovery and Data Mining (KDD)*, pages 174–179, Montreal, Canada, 1995.

[17] E.M. Knorr and R.T. Ng. Algorithmis for mining distance-based outliers in large datasets. In *Proc. of the 24th VLDB Conference*, pages 392–403, New York, USA, 1998.

[18] D. Koller, K. Daniilidis, and H.-H. Nagel. Model-based object tracking in monocular image sequences of road traffic scenes. *International Journal of Computer Vision*, 10(3):257–281, 1993.

[19] H. Kollnig. *Ermittlung von Verkehrsgeschehen durch Bildfolgenauswertung*. PhD thesis, Institut für Algorithmen und Kognitive Systeme der Universität Karlsruhe, Sankt Augustin, Februar 1995.

[20] H. Kollnig and H.-H. Nagel. 3D Pose Estimation by Directly Matching Polyhedral Models to Gray Value Gradients. *International Journal of Computer Vision*, 23(3):283–302, 1997.

[21] H.-H. Nagel. Arbeitsbericht der Forschungsgruppe Bildauswertung. In Th. Beth, J. Calmet, and H.-H. Nagel, editors, *Forschung und Lehre am IAKS 1985–1995*, pages 13–67, Institut für Algorithmen und Kognitive Systeme der Universität Karlsruhe (TH), Karlsruhe, 1995.

[22] H.-H. Nagel. Zur Strukturierung eines Bildfolgen–Auswertungssystems. *Informatik – Forschung und Entwicklung*, 11(1):3–11, 1996.

[23] H.-H. Nagel, Th. Müller, V. Gengenbach, N. Andreff, A. Bachem, R. Horaud, and H. Leuck. Machine Vision Competence as a Function of Computing Power – Four Years Later. In *12th IAR Annual Meeting*, Mulhouse/France, 19–20 November 1998.

[24] Patrick Ohly, Joachim M. Blum, Thomas M. Warschko, and Walter F. Tichy. PSPVM2:PVM for ParaStation. In *Proc. of 1st Workshop on Cluster Computing*, pages 147–160, Chemnitz, Germany, Nov.6-7, 1997.

[25] M. Otte and H.-H. Nagel. Estimation of Optical Flow Based on Higher-Order Spatiotemporal Derivatives in Interlaced and Non-Interlaced Image Sequences. *Artificial Intelligence Journal*, 78(1):5–43, 1995.

[26] Michael Philippsen and Bernhard Haumacher. More Efficient Object Serialization. *submitted to:* International Workshop on Java for Parallel and Distributed Computing, San Juan, Puerto Rico, April 12–16, 1999.

[27] Michael Philippsen and Matthias Zenger. JavaParty: Transparent remote objects in Java. *Concurrency: Practice and Experience*, 9(11):1225–1242, November 1997.

[28] K.H. Schäfer. PATTY, Preprocessor for Attributed Types. Technical report, Institut für Algorithmen und Kognitive Systeme der Universität Karlsruhe, 1996.

[29] K.H. Schäfer. *Unscharfe zeitlogische Modellierung von Situationen und Handlungen in Bildfolgenauswertung und Robotik*. PhD thesis, Institut für Algorithmen und Kognitive Systeme der Universität Karlsruhe, Sankt Augustin, Juli 1996.

[30] Sun Microsystems Inc., Mountain View, CA. *Java Object Serialization Specification, beta draft*, 1996. ftp://ftp.javasoft.com/docs/jdk1.2/serial-spec-JDK1.2.pdf.

[31] Sun Microsystems Inc., Mountain View, CA. *Java Remote Method Invocation Specification, beta draft*, 1998. ftp://ftp.javasoft.com/docs/jdk1.2/rmi-spec-JDK1.2.pdf.

[32] M. Tonko. *Zur sichtsystemgestützten Demontage am Beispiel von Altfahrzeugen*. PhD thesis, Institut für Algorithmen und Kognitive Systeme der Universität Karlsruhe, Sankt Augustin, Juni 1997.

[33] M. Tonko, K.H. Schäfer, V. Gengenbach, H.-H. Nagel, R. Benner, and N. Pohle. Wie gut ist der 34-Mbit/s-ATM-Dienst der Deutschen Telekom? *Nachrichtentechnische Zeitschrift (Informationstechnik und Telekommunikation) ntz*, 50(7):58–60, 1997.

[34] Thomas M. Warschko. *Effiziente Kommunikation in Parallelrechnerarchitekturen*. PhD thesis, Universität Karlsruhe, Fakultät für Informatik, December 1997. In: VDI Fortschritt-Berichte, Reihe 10, Nr. 525, VDI-Verlag, ISBN: 3-18-352510-0.

[35] Thomas M. Warschko, Joachim M. Blum, and Walter F. Tichy. The ParaStation Project: Using Workstations as Building Blocks for Parallel Computing. In *Intl. Conf. on Parallel and Distributed Processing, Techniques and Applications (PDPTA'96)*, pages 375–386, Sunnyvale, CA, August 9–11, 1996.

[36] Thomas M. Warschko, Joachim M. Blum, and Walter F. Tichy. ParaStation: Efficient Parallel Computing by Clustering Workstations: Design and Evaluation. *Journal of Systems Architecture*, pages 241–260, 12 1997. Elsevier Science Inc., New York, NY 10010.

[37] Thomas M. Warschko, Joachim M. Blum, and Walter F. Tichy. Design and evaluation of parastation2. In *Proceedings of the International Workshop on Distributed High Performance Computing and Gigabit Wide Area Networks*, Essen, September 1-5 1998. (to appear as Springer LNCS).

[38] Thomas M. Warschko, Joachim M. Blum, and Walter F. Tichy. Kommunikationshürde speichersubsystem. *PARS (GI) Mitteilungen*, 17:23–29, September 1998.