Das Multiprozessorsystem MEMSY
Programmiermodell, Betriebssystemarchitektur, Kommunikation und dynamische Rekonfiguration

C-U. Linster et al.

November 1993

Publikation

Institut für
Mathematische Maschinen
und Datenverarbeitung
der
Friedrich-Alexander-Universität
Erlangen-Nürnberg

Lehrstuhl für Informatik IV
(Betriebssysteme)

This paper was also published as:

Claus-Uwe Linster, Wolfgang Stukenbrock, Thomas Thiel, Stefan Turowski:
"Das Multiprozessorsystem MEMSY: Programmiermodell, Betriebssystemarchitektur, Kommunikation und dynamische Rekonfiguration"; In: Wedekind, H. (Hrsg.): *Verteilte Systeme, Grundl. und zukünft. Entw. aus d. Sicht d. SFB 182.* Mannheim : BI-Wissenschafts Verlag, 1993, S. 206-228

Das Multiprozessorsystem MEMSY

Programmiermodell, Betriebssystemarchitektur, Kommunikation und dynamische Rekonfiguration

Claus-Uwe Linster, Wolfgang Stukenbrock, Thomas Thiel, Stefan Turowski
E-Mail: {linster, stukenbrock, thiel, turowski}@informatik.uni-erlangen.de

Teilprojekt B2
Entwurf und Implementierung eines an Hardwarearchitektur und Aufgabenklassen adaptierbaren Multiprozessorsystems

Zusammenfassung

MEMSY ist ein speichergekoppeltes Hochleistungsmultiprozessorsystem mit verteiltem, lokal gemeinsamen Speicher. Da sich die Hardware-Architektur von MEMSY von anderen Systemen wesentlich unterscheidet, wurde ein Programmiermodell entworfen, das den Benutzern in einfacher Weise gestattet, die Hardware-Eigenschaften des Systems effizient zu nutzen. Basierend auf einer Unix-Implementierung ist das Betriebssystem MEMSOS erstellt worden, das insbesondere an die, im Rahmen dieses Projekts konzipierte, Speicherkopplung angepaßt wurde. Es werden die in MEMSOS eingesetzten Kommunikationsmechanismen beschrieben, da diese generell einen besonderen Einfluß auf die Leistungsfähigkeit eines Multiprozessorsystems haben. Außerdem wird auf das bei Multiprozessorsystemen wichtige Problem der dynamischen (Re-)Konfiguration eingegangen und eine Lösung für MEMSY diskutiert. In diesem Zusammenhang spielt der Systemstart und das Verhalten des Systems in Ausnahmesituationen eine bedeutende Rolle.

1 Einführung

In der Vergangenheit wurden rechenintensive Problemstellungen, die auf herkömmlichen Monoprozessorsystemen nicht in vertretbarer Zeit lösbar waren, auf leistungsfähigen Vektorrechnern zur Ausführung gebracht, die ihre überlegene Rechenleistung durch Ausführung der gleichen Operation auf einer Vielzahl von Daten, einem Vektor und durch den Einsatz des jeweils technologisch Machbaren erreichten. Diese Rechner gehören der SIMD-Klasse (Single Instruction Multiple Data) an. Die technologische Spitzenstellung und der begrenzte Markt machten (und machen) diese Systeme extrem teuer.

Schon früh - in Erlangen seit Beginn der 70er Jahre [Hän76] - wurde über kostengünstigere Alternativen nachgedacht. Die Idee war, möglichst viele Standardprozessorkomponenten als ein paralleles Rechensystem geeignet zusammenzuschalten. Da die Einzelprozessoren natürlicherweise den vollen Funktionsumfang eines Monoprozessorsystems hatten, entstand so die Klasse der MIMD-Rechner (Multiple Instruction Multiple Data).

Die prinzipielle Realisierbarkeit solcher Parallelrechnersysteme wurde bereits in den 70er Jahren durch einige „akademische” Prototypen, wie z.B. das EGPA-System ([Hän76]) oder das C.mmp-System ([Wul72],[Wul74]), aufgezeigt. Aus technologischen Gründen war die Zeit für kommerzielle MIMD-Rechner als numerische Hochleistungsrechner noch nicht reif. Wesentliche Gründe zum Nachteil der Parallelrechner waren einerseits die relativ großen Baugruppen der Systeme und dadurch bedingt große topologische Ausdehnungen, was wiederum einer hohen Kommunikationsleistung entgegenstand, und andererseits die noch nicht voll ausgereizten Möglichkeiten der Vektorrechnertechnologie. Lediglich in der kommerziellen Datenverarbeitung (File-Server, Datenbank-Server) konnten Shared-Memory-Systeme wie z. B. Sequent-Symmetry, Concurrent Computer MPS3280 oder Encore-Systeme erfolgreich sein, wobei die Anzahl der Prozessoren relativ klein blieb.

Erst in der 2. Hälfte der 80er Jahre „wandelten” sich die technologischen Voraussetzungen zugunsten der Parallelrechner der MIMD-Klasse. Die Entwicklung der Vektorrechner als Einzelprozessoren näherte sich immer mehr den physikalischen Grenzen bei erheblichem Kostenaufwand. Die RISC-Mikroprozessortechnologie erreichte eine Leistungsfähigkeit, durch die sie mit Vektorprozessoren ernsthaft konkurrieren konnte. Hinzu kam die Erkenntnis, daß viele Anwendungen nur unzureichend vektorisierbar sind und dadurch bedingt das Leistungsvermögen eines Vektorrechners nur zu einem Bruchteil ausnutzen können. Das war noch solange hinnehmbar, wie es im skalaren Bereich keine Alternative gab. Heute jedoch ist bereits ein Hochleistungs-Einzelmikroprozessor im Skalarbereich leistungsfähiger als ein herkömmlicher Vektorprozessor.

Diese oben geschilderten Voraussetzungen führten in Erlangen dazu, die Ideen des EGPA-Projekts erneut aufzugreifen[1] und an den Stand der Technik anzupassen [Fri89]. Die Tragfähigkeit der Architektur sollte durch den Aufbau eines Experimentalsystems im Rahmen des Querschnitthemas MEMSY im Sonderforschungsbereich untermauert werden.

Die Klasse der MIMD-Rechner läßt sich nach Art der Kopplung der einzelnen Prozessoren in speicher- und nachrichtengekoppelte Systeme unterteilen.

Systeme, die über einen globalen Arbeitsspeicher (globally-shared-memory) gekoppelt sind, sind nur begrenzt ausbaubar, da mit der Anzahl der Prozessoren die Rate der Speicherzugriffskonflikte steigt. Diese Art der Kopplung ist für kleine Prozessoranzahlen jedoch optimal. Da jeder Prozessor auf alle Daten zugreifen kann, sind keine Konzepte zur Verteilung von Daten im Arbeitsspeicher erforderlich. Prozeßverlagerungen sind einfach lösbar. Dies sind die wesentlichen Vorraussetzungen für eine dynamisch ausba-

1. Genau genommen sind die Ideen aus dem EGPA-Projekt laufend weiterentwickelt worden, jedoch war man chancenlos, wenn es um die Förderung solcher innovativen Ideen ging. Bis zum Ende der 80er Jahre (ausgenommen SUPRENUM) wurden in Deutschland Anstrengungen in diesem Feld der Informatik - insbesondere auch von der einschlägigen Industrie - schlicht nicht zur Kenntnis genommen.

lancierte Lastverteilung des Systems, so daß auch bei schwankender Auslastung jederzeit die „optimale” Rechenleistung dem Multiprozessorsystem entnommen werden kann.

Nachrichtengekoppelte Systeme, wie z.B. Intel Paragon, vernetzte Systeme (Workstation-Netze) und Transputersysteme, sind im Prinzip unbegrenzt bezüglich der Anzahl der Prozessoren erweiterbar, wenn die lokale Komplexität des Verbindungsnetzwerkes konstant ist. Bei keinem größeren nachrichtengekoppelten System ist eine vollständige Vermaschung der Knoten möglich. Diese Vermaschung hat zwar im Hinblick auf das Ausfallverhalten des Systems Vorteile, ist jedoch physikalisch nicht realisierbar. Nachrichtengekoppelte Systeme unterscheiden sich im wesentlichen in der Topologie der Vernetzung der einzelnen Prozessoren (z.B. Hypercube oder Matrix). Prozeßverlagerungen, und damit dynamischer Lastausgleich, sind bei nachrichtengekoppelten Systemen aufgrund des Kommunikationsoverheads nicht sinnvoll und gestalten sich aus Betriebsystemsicht als schwierig. Im allgemeinen wird man eine statische Lastverteilung anstreben, d.h. zum Prozeßstart entscheiden, auf welchem Prozessor der Prozeß zur Ausführung kommen soll.

2 Aufbau von MEMSY

Die Architektur von MEMSY wurde mit Hinblick auf die folgenden Konstruktionsziele gewählt.

- *Wirtschaftlichkeit*. Das System soll sich soweit als möglich aus käuflichen Komponenten zusammensetzen.
- *Erweiterbarkeit*. MEMSY soll beliebig erweiterbar sein, ohne theoretische Begrenzung. Das Kommunikationsnetzwerk soll dabei mit der Anzahl der Prozessoren, bei konstanter Komplexität, wachsen.
- *Flexibilität*. MEMSY soll für eine breite Palette von numerischen Problemstellungen verwendbar sein.
- *Leistungsfähigkeit*. Das System soll auf leistungsfähige (state-of-the-art) Mikroprozessoren aufbauen und genügend Leistung zur Lösung realer wissenschaftlicher numerischer Probleme bereitstellen.

2.1 Topologie von MEMSY

MEMSY hat eine pyramidenförmige Struktur und wird in zwei Ebenen unterteilt (A- und B-Ebene). Der schematische Aufbau des 20-Knoten-Experimentalsystems ist in Abb. 2.1 dargestellt. Die Knoten der beiden Ebenen sind jeweils in einem Gitter angeordnet, das zum Torus geschlossen ist. Jedem Knoten der B-Ebene sind genau vier Knoten aus der A-Ebene zugeordnet. Diese fünf Knoten bilden eine *Elementarpyramide*. Die Kopplung zwischen benachbarten Knoten erfolgt über gemeinsame Spei-

chermodule. Dazu ist jedem Knoten ein zusätzlicher Speicher, der Kommunikationsspeicher, zugeordnet, auf den seine direkten Nachbarn zugreifen können (Nearest-Neighbour-Prinzip). Eine ausführliche Beschreibung des Speicherverbindungsnetzwerks kann in [Hil92] gefunden werden.
Zusätzlich sind die Knoten über optische Verbindungsnetzwerke verbunden. Zur Unterstützung globaler Synchronisationsmechanismen wurde ein N:N-Netzwerk realisiert [Sch93]. Zur Einbindung von MEMSY in die Netzumgebung des Instituts dient ein FDDI-Ring. Darüberhinaus ist beabsichtigt mit diesen Bussystemen alternative Kommunikationsmechanismen zu untersuchen.

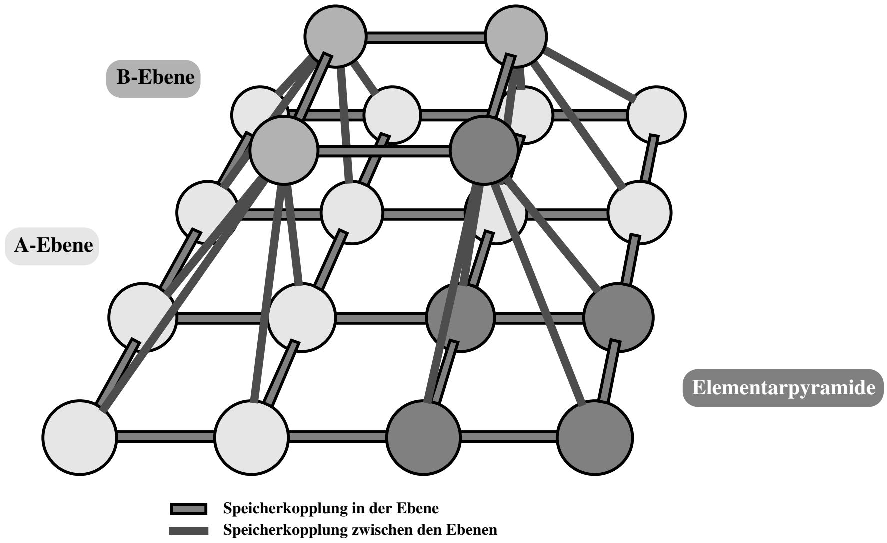


Abb. 2.1 20-Knoten-Experimentalsystem (ohne Torusverbindungen)

2.2 Struktur eines einzelnen MEMSY-Knotens

Jeder Knoten setzt sich aus einem 4fach-Multiprozessor-Modul mit integriertem Cache, dem lokalen Hauptspeicher und spezieller Zusatzbaugruppen zusammen, die im Rahmen des MEMSY-Projekts entwickelt wurden. Diese dienen zur Ansteuerung der Kommunikationsspeicher, zur Erzeugung von Unterbrechungen bei Nachbarknoten über dedizierte Verbindungen und der Realisierung des optischen Busses (N:N-Verbindung). Darüberhinaus ist pro Knoten ein privater Plattenspeicher und eine FDDI-Schnittstelle vorhanden. Der prinzipielle Aufbau eines Knotens ist in Abb. 2.2 dargestellt.

Der Kommunikationsspeicher (KS) wird wie der normale Hauptspeicher angesprochen. Im gekoppelten System entscheidet lediglich die Adressierung, auf welchen der verfügbaren Speicher (Hauptspeicher, eigener Kommunikationsspeicher oder Nachbar-Kommunikationsspeicher) zugegriffen wird. Die Zugriffe auf die Kommunikationsspeicher werden über die KS-Ansteuerschaltung realisiert. Durch eine Koppelhardware läßt sich die Anzahl der Speicheranschlüsse pro Knoten und Kommunikati-

LST+93.doc: 27.03.95 12:28

onsspeicher auf drei beschränken, unter Beibehaltung der angestrebten Topologie [Hil92]. Das Unterbrechungs-Subsystem ermöglicht es, Unterbrechungen auf anderen Knoten auszulösen. Dazu müssen separate Knoten-zu-Knoten Verbindungen hergestellt werden. In MEMSY liegen diese Verbindungen parallel zur Speicherkopplung, das heißt ein Knoten hat zu dem Nachbarn eine Unterbrechungsverbindung, mit dem er einen gemeinsamen Kommunikationsspeicher hat.

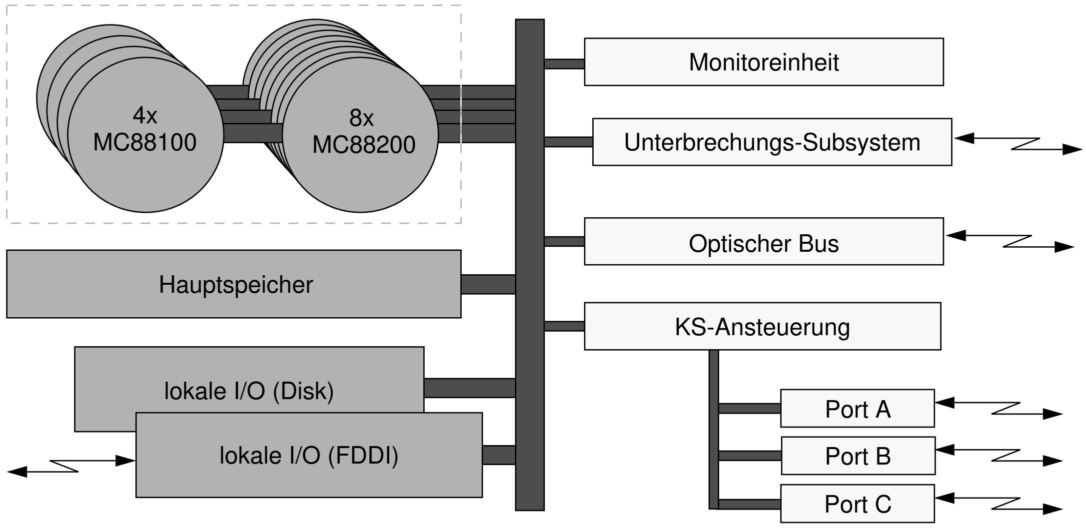


Abb. 2.2 Prinzipieller Aufbau eines MEMSY-Knotens

2.3 Einsatzgebiete

Entgegen aller Versprechen der Anbieter paralleler Rechensysteme ist eine Anpassung des Anwenderproblems an Architektureigenschaften des Rechensystems dann unabdingbar, wenn man einen hohen Grad an Effizienz anstrebt. Dabei sind unter anderem Programmsystemeigenschaften wie:

- Parallelitätsgrad,
- Rechenintensität auf beschränkte Datenbereiche (Lokalität),
- Topologieabhängigkeit der Datenpartitionierung,
- Kommunikations-/Rechenzeitverhältnis (grobe oder feine Granularität),
- typische Größen von Kommunikationspaketen,
- lokale Kommunikation,
- globale Kommunikation und
- E/A-Bindungen

geeignet abzubilden auf die Eigenschaften des Rechensystems, wie z.B.:

- Rechenleistung des Prozessors im Verhältnis zu seiner Cachegröße,
- Rechenleistung des Prozessors im Verhältnis zur Größe des lokalen Speichers,
- Synchronisationsmechanismen; Typen und Komplexität,
- Aufsetzzeiten der Kommunikationsmechanismen,
- lokale/globale Transferleistung des Kommunikationssystems,
- Kommunikationstopologie und
- E/A-Strukturen

Durch seine Topologie und seinen Hardwareaufbau eignet sich MEMSY vorwiegend für hochparallelisierbare Aufgaben mit überwiegend Nearest-Neighbour-Datenabhängigkeit. Dies trifft zum Beispiel für Aufgaben aus den Klassen numerischer Gitteraufgaben (z.B. Mehrgitterverfahren), Vielteilchenmodelle und quantenmechanische Berechnungen zu.

3 Das Programmiermodell

Wichtig für den Erfolg oder Mißerfolg eines Multiprozessorsystems sind nicht allein seine technischen Leistungsdaten, sondern auch die Programmierumgebung und das Programmiermodell, das den Anwendern zur Verfügung gestellt wird.

Für MEMSY wurden die folgenden Schwerpunkte bei der Erstellung des Programmiermodells gesetzt:

- *Effizienz.* Die Eigenschaften der Hardware sollen dem Anwender weitestgehend zugänglich gemacht werden, damit die Anwendung die besonderen Fähigkeiten der Hardware (lokal gemeinsame Speicher) optimal nutzen kann.
- *Vielseitigkeit.* Es sollen verschiedene Mechanismen zur Kommunikation und Koordinierung angeboten werden, von denen die jeweils geeignetsten ausgewählt werden können.
- *Erweiterbarkeit.* Das Programmiermodell soll so gestaltet sein, daß Erweiterungen, die aus Anregungen der Anwender oder Analysen von Programmen gewonnen werden, jederzeit umgesetzt werden können.
- *Implementierung mit vertretbarem Aufwand.* Die Implementierung der Funktionen des Programmiermodell soll nicht zu aufwendig sein. Es erfolgen keine Spracherweiterungen.

LST+93.doc: 27.03.95 12:28

3.1 Architektur von MEMSY aus der Sicht des Anwenders

Die Sicht des Anwenders auf die Architektur entspricht im wesentlichen der tatsächlichen Architektur, wie sie in Abschnitt 2 beschrieben wurde. Unsichtbar sind lediglich einige Besonderheiten der Hardwareimplementierung, wie zum Beispiel die teilweise diagonale Kopplung der Knoten und die durch das Verbindungsnetzwerk möglichen Mehrwegzugriffe auf die Kommunikationsspeicher.

3.1.1 Speicher

Jedem Knoten ist ein Kommunikationsspeicher zugeordnet. Auf Daten, die in diesem Speicher abgelegt werden, können außer dem Knoten selbst auch die vier direkten Nachbarn innerhalb der Ebene zugreifen. Auf den Kommunikationsspeicher eines Knotens der A-Ebene kann zusätzlich auch der übergeordnete Knoten der B-Ebene zugreifen.

3.1.2 Knoten

Jeder Knoten des MEMSY-Systems ist selbst ein 4-fach-Shared-Memory-Multiprozessorsystem. Die Adressierung der Knoten erfolgt durch Angabe der relativen Position, wodurch Verbindungen zu den Nachbarknoten uniform für alle Knoten programmierbar sind (Torustopologie). Eine Adressierung durch absolute Koordinaten ist ebenfalls möglich.

3.2 Komponenten des Programmiermodells

Nachfolgend sollen kurz die wesentlichen Komponenten des Programmiermodells vorgestellt werden. Eine ausführlichere Beschreibung kann in [Kar91] gefunden werden.

3.2.1 Gemeinsamer Speicher

Einzelne Abschnitte (Segmente) des gemeinsamen Speichers können von einem Prozeß angefordert werden (*create*), die dann von anderen Prozessen (auch auf benachbarten Knoten) in ihren Adreßraum eingeblendet werden können (*attach*).

Die Identifikation der Segmente erfolgt über eine Identifikationsnummer, die für einen Knoten und eine Applikation eindeutig ist.

Als Ergebnis der Systemaufrufe zur Verwaltung dieser Segmente erhalten die Prozesse Zeiger, die auf den Anfang der Segmente verweisen. Die Adressen, unter denen ein Segment bei verschiedenen Prozessen sichtbar wird, sind vollkommen unabhängig voneinander. Ein direkter Austausch der Zeiger zwischen verschiedenen Prozessen ist daher nicht möglich.

LST+93.doc: 27.03.95 12:28

Ein Wiederfreigeben der Segmente ist ebenfalls möglich (*detach*, *destroy*), wobei zuerst die durch *attach* erzeugten Bindungen aufgelöst werden müssen und danach das Segment wieder freigegeben werden kann.

Zugriffe auf den Kommunikationsspeicher werden normalerweise am Cache vorbeigeleitet, da sonst die Konsistenz der Daten zwischen den einzelnen Knoten nicht gesichert wäre. Wenn eine Anwendung sicher ist, daß es beim Zugriff auf ein bestimmtes Segment für eine gewisse Zeit nicht zu Konsistenzproblemen kommen kann (weil z.B. nur lesend oder nur von einem Knoten aus darauf zugegriffen wird), so kann sie für diese Zeit den Cache für dieses Segment aktivieren. Als weitere Geschwindigkeitssteigerung sind damit auch burst-mode[2] Zugriffe auf den Kommunikationsspeicher möglich.

3.2.2 Spinlocks

Für die Koordinierung kurzer kritischer Abschnitte werden Spinlocks, gemeinsame Variablen im Kommunikationsspeicher, zur Verfügung gestellt. Diese können nur die Zustände „frei” und „belegt” annehmen, welche durch entsprechende Operationen eingestellt werden.

Auf das Freiwerden eines Spinlocks wird aktiv gewartet, daher sollten diese Wartezeiten klein gehalten werden. Damit Koordinierungsfehler, die zu einem längeren Warten führen, begrenzte Auswirkungen auf das Gesamtsystem haben, muß beim Anlegen von Spinlocks eine Zeitschranke für das Belegtsein angegeben werden, nach deren Ablauf der Prozeß terminiert wird.

3.2.3 Semaphore

Als weitere Koordinierungsmöglichkeit bietet das System globale binäre Semaphore an, welche die üblichen P- und V-Operationen unterstützen.

3.2.4 Signale

Es ist möglich, einem Prozeß auf einem Nachbarknoten ein Signal zustellen zu lassen, worauf dann dessen normaler Ablauf unterbrochen und eine vorher definierte Signalbehandlungsroutine aufgerufen wird. Auf diese Weise kann schnell auf asynchrone oder außergewöhnliche Ereignisse reagiert werden.

3.2.5 Messages

Messages sind kurze Nachrichten, die ein Prozeß an Prozesse der selben Applikation senden kann. Die Nachrichten werden beim Empfänger gepuffert. Ein Empfang der Nachrichten kann entweder blockierend oder nicht blockierend erfolgen. Dem Empfänger wird neben dem Inhalt der Nachricht auch die Identität des Absenders mitgeteilt.

2. Im burst-mode werden atomar vier aufeinanderfolgende Datenworte übertragen, wobei nur eine Adreßinformation benötigt wird (sonst für jedes Wort eine).

Die Implementierung der Messages ist nicht auf den Transport größerer Datenmengen optimiert, ihr Einsatz ist vor allem zur Koordinierung und zum Austausch kleiner Datenmengen sinnvoll.

3.2.6 Ein-/Ausgabe

Jeder Knoten hat Zugriff auf einen lokalen und auf einen, für das Gesamtsystem globalen Plattenbereich. Der Zugriff erfolgt durch Varianten der UNIX-Bibliotheksfunktionen für gepufferte Ein-/Ausgabe.

Jeder Knoten im Experimentalsystem ist mit einer lokalen Platte ausgestattet, so daß die Zugriffe auf die knotenlokalen Datenbereiche schnell und unabhängig von anderen Knoten abgewickelt werden können.

Die Zugriffe auf die globalen Datenbereiche sind aufwendiger. Bei der derzeitigen Implementierung wird dazu NFS über das FDDI-Netz genutzt. In Entwicklung befindet sich die Realisierung über die Speicherkopplung.

3.3 Programmiersprachen

Das Programmiermodell für MEMSY wurde durch eine Reihe von Systemaufrufen und Bibliotheksfunktionen implementiert. Der Zugriff auf die Datenbereiche im gemeinsamen Speicher erfolgt ausschließlich über Zeiger. Damit ist die Verwendung von Programmiersprachen, die keine Zeigerzugriffe unterstützen, nicht möglich. Als Programmiersprachen werden C und C++ unterstützt.

4 Die Betriebssystemarchitektur

Bei der Auswahl des Betriebssystems wurde aus Aufwandsgründen, aber auch aufgrund der Erkenntnis, daß ein System nur dann angenommen wird, wenn es mit seiner Entwicklungsumgebung bekannten Mechanismen bzw. Schnittstellen nahekommt, auf das UNIX SYSTEM V/88 Release 3 von Motorola aufgesetzt, das bereits an die Knotenarchitektur von MEMSY angepaßt ist [Rus90].

Erweiterungen am Betriebssystem sind notwendig gewesen, um die zusätzliche Hardware zu integrieren. Für Aktionen[3], die innerhalb eines Knotens wirken, kann auf die bereits vorhandenen Mechanismen zurückgegriffen werden. Für knotenübergreifende Aktionen sind neue Mechanismen implementiert worden, die sich des Kommunikationsspeichers bedienen. Hier sind unter anderem die Kommunikationsspeicherverwaltung, der Unterbrechungsmechanismus und ein einfacher Nachrichtenmechanismus zu

3. Eine Aktion kann zum Beispiel das Versenden einer Nachricht oder das Anlegen eines gemeinsamen Speichersegmentes sein.

nennen. Damit MEMSY verschiedene Anwendungen im Time- und Space-Sharing-Betrieb abarbeiten kann, ist das Applikationskonzept eingeführt worden. Standard-Unix Anwendungen sind weiterhin ohne Einschränkungen ablauffähig.

4.1 Anwendungsunterstützung

Von Seiten der Anwender werden an ein Hochleistungsrechensystem verschiedene Anforderungen gestellt. Hier sind unter anderem hohe Verfügbarkeit des Systems (Ausfallsicherheit und einfaches Aufsetzen nach einem Knotenausfall), einfache Handhabbarkeit, kurze Aufsetzphasen von Anwendungen, gewohnter oder gar erweiterter Funktionalitätsumfang und die Möglichkeit zu interaktivem Testen von Anwendungen, ohne den Produktionsbetrieb zu stark zu stören, zu nennen.

Damit die oben angeführten Anforderungen erfüllt werden können, ist in MEMSOS das *Applikationskonzept* eingeführt worden. Die Gesamtheit aller Prozesse, die durch eine einzige Anwendung auf dem System erzeugt werden, wird als eine Einheit gesehen. Diese Einheit wird als *Applikation* bezeichnet. Damit mehrere parallele Applikationen gleichzeitig auf dem System ablaufen können, ist jeder Prozeß einer Applikation durch das Tripel (ApplikationsId, KnotenId, ApplikationsprozeßId) global eindeutig identifizierbar. Jede Applikation hat ihren eigenen Namensraum. Beeinflussungen treten somit nur beim konkurrierenden Zugriff auf Betriebsmittel (Rechenzeit, Speicher, usw.) auf.

Beim Systemstart wird initial eine ausgezeichnete (System-)Applikation erzeugt, welche die Rolle eines Koordinators übernimmt. Sie hat zunächst die Aufgabe, auf Anforderung hin, neue (Benutzer-)Applikationen zu erzeugen und diese zu überwachen. Dazu wird durch das Anwenderprogramm eine entsprechende Anforderung zur Erzeugung einer neuen Applikation gestellt. Die System-Applikation vergibt eine noch freie ApplikationsId und erzeugt auf jedem zugewiesenen Knoten je einen *Leader-Prozeß* für diese neue Applikation. Dieser Leader-Prozeß stellt die benötigte Umgebung für die Applikation bereit, übernimmt Koordinierungsaufgaben und überwacht den Ablauf und die ordnungsgemäße Terminierung der von ihm gestarteten Applikationsprozesse.

Mit Hilfe dieses Applikationskonzept sind globale Mechanismen, zum Beispiel ein globales Scheduling, integriert worden. Mehrere Applikationen können sowohl im Space-Sharing-Betrieb, als auch im Time-Sharing-Betrieb ablaufen, da diese nun einzeln erfaßt und differenziert behandelt werden können.

Das Anwendungsprofil für MEMSY läßt sich nicht eindeutig festlegen, da es sich hier um ein Experimentalsystem und nicht um ein reines Produktionssystem handelt. Es können also Anwendung zur Ausführung kommen, die völlig unterschiedliche Anforderungen stellen. Anwendungen mit langer Lauf- und Rechenzeit müssen anders behandelt werden, als etwa solche, mit vielen Benutzer-Interaktionsphasen. Zudem soll die Lastverteilung immer möglichst optimal sein, um eine gute Auslastung des Systems zu erzielen. In diesem Zusammenhang ist die Realisierung der Prozeß-Prozessor Bindung und die eingesetzte Prozessorvergabestrategie von Bedeutung.

Die Prozeß-Prozessor-Bindung innerhalb eines Knotens wird durch Prozeßwarteschlangen ausgedrückt. Jeder Prozessor hat eine Referenz auf die Prozeßwarteschlange, die er bevorzugt abarbeiten soll. Jeder Prozeß wiederum besitzt eine Referenz auf eine bestimmte Warteschlange, in die er eingetragen wird, wenn er lauffähig ist. Sowohl die Bindung eines Prozessors, als auch die eines Prozesses, an eine Warteschlange kann dynamisch geändert werden. Alle Prozessoren können ihre Warteschlange parallel zueinander abarbeiten, wobei eine einzelne Warteschlange jeweils nur exklusiv von einem Prozessor verändert werden darf.

In MEMSOS wird knotenlokal eine Trennung in Applikationsprozesse und „andere" Prozesse (diese werden hier als *Systemprozesse* bezeichnet) vorgenommen. Dazu wird eine Systemwarteschlange und für jede Applikation eine Applikationswarteschlange eingeführt. In die Systemwarteschlange werden alle Systemprozesse eingetragen. Das sind zum Beispiel die typischen Unix-Daemons für NFS und Nameserver. Sie wird einem Prozessor des Knotens fest zugeordnet, der diese vorrangig abarbeitet. Dieses Vorgehen hat den Vorteil, daß Systemprozesse immer bearbeitet werden können. Dadurch wird das Verhalten des Systems auch bei großer Last, die durch laufende Applikationen hervorgerufen wird, deutlich verbessert. Die Applikationswarteschlangen werden von den verbleibenden Prozessoren bedient. Die Applikationsumschaltung, d.h. welche Applikationswarteschlange ausgewählt wird, wird über Applikationsprioritäten gesteuert.

Die Applikationsprioritäten können sinnvollerweise nur global für das Gesamtsystem verändert werden. Für die globale Prozeß- und Applikationsumschaltung kann ein verteilter Server-Prozeß eingesetzt werden. Er wertet die Laufzeiten der Applikationen aus und ist in der Lage die Applikationsprioritäten nach globalen Gesichtspunkten zu verändern. So können zum Beispiel Langläufer in ihrer Priorität zurückgestuft und interaktive Applikationen bevorzugt werden. Zudem ist es möglich, verschiedene Umschaltstrategien einzusetzen und ihre Eignung für die Applikation zu überprüfen. In [Rot93] wird näher auf diese Thematik eingegangen.

4.2 Kommunikationsprimitive in MEMSY

Wie bereits dargestellt kann jeder Knoten in MEMSY nur mit seinen direkten Nachbarn unmittelbar über Shared-Memory oder Nachrichten interagieren. Aufbauend auf eine solche elementare Kommunikation wurden globale Kommunikationsmechanismen implementiert. Ist eine Kommunikation, die über die Reichweite einer elementaren Kommunikation hinausgeht, erforderlich, muß die zu übertragende Information weitergeleitet werden. Um die Richtung der Weiterleitung festzulegen werden Routingtabel-

LST+93.doc: 27.03.95 12:28

len genutzt, in denen dynamisch festgelegt wird zu welchem direkten Nachbarn, in Abhängigkeit vom endgültigen Empfänger, die Information weitergeschickt werden soll.

Neben den Kommunikationsprimitiven, die auf Kommunikation über die Kommunikationsspeicher basieren, gibt es zwei weitere Kommunikationssubsysteme, den FDDI-Ring und einen optischen N:N Bus.

4.2.1 Unterbrechungsmechanismus

Pollingmechanismen als Basis von Synchronisationsmechanismen sind nur bei hinsichtlich der Kommunikation einfach strukturierten Programmsystemen im Single-User-Betrieb oder bei Space-Sharing-Betrieb sinnvoll. Der zum Polling verwendete Spinlock-Mechanismus kann durch das Betriebssystem nicht kontrolliert werden und ist daher im Multi-User- bzw. Multi-Applikationsbetrieb nicht vernünftig einsetzbar. Hinzu kommt, daß durch die häufigen unproduktiven Kommunikationsspeicherzugriffe der Durchsatz stark beeinträchtigt werden kann. „Nichtpollende” Kommunikationsmechanismen setzen daher einen Unterbrechungsmechanismus voraus, der zwischen Knoten wirkt.

Mit jeder Unterbrechung, die bei einem direkten Nachbarn ausgelöst wird, wird ein Datenwort im Kommunikationsspeicher des auslösenden Knotens für den unterbrochenen Nachbarknoten abgelegt. Im Datenwort sind Absenderinformation, Unterbrechungstyp und Nutzdaten kodiert. Die Interpretation des Nutzdatenanteils ist abhängig vom Unterbrechungstyp. Kernmodule[4], welche den Unterbrechungsmechanismus verwenden wollen, müssen während der Systeminitialisierung einen oder mehrere Unterbrechungstypen reservieren und für jeden reservierten Typ eine Behandlungsroutine, welche die Nutzdaten verarbeiten kann, registrieren lassen. Die reservierten Typen müssen systemweit eindeutig sein.

Das Auslösen einer Unterbrechung erfolgt durch einen einfachen Funktionsaufruf. Als Parameter werden die Identifikation des Empfängers, der Unterbrechungstyp und der Nutzdatenanteil benötigt. Erkennt ein Knoten eine Unterbrechung, so wird das bereitgestellte Datenwort ausgelesen und bestätigt. Danach wird die, für den erkannten Typ registrierte, Behandlungsroutine aufgerufen, welche die Weiterverarbeitung übernimmt.

4.2.2 Nachrichtenmechanismus

Der Nachrichtenmechanismus setzt auf den Unterbrechungsmechanismus auf. Durch ihn können Nachrichten auch zu weiter entfernt liegenden Knoten geschickt werden. Dies erfolgt mit Unterstützung eines Routingmechanismus (siehe Abschnitt 4.2.4). Anhand der aktuellen Routingtabelle wird festgestellt, an welchen direkten Nachbarn

4. Kernmodule sind Teile des Kerns, die eine bestimmte Funktionalität realisieren.

die Nachricht geschickt werden muß, um möglichst effizient zum Ziel weitergeleitet zu werden. Dann wird der Inhalt der Nachricht im eigenen Kommunikationsspeicher hinterlegt, der für die Kommunikation mit dem ausgewählten direkten Nachbarn verwendet wird, und mittels Unterbrechungsmechanismus mit dem Typ „Nachrichtenübermittlung” dem Nachbarn zur Kenntnis gebracht. Dieser stellt fest, ob er der endgültige Empfänger ist. Wenn ja, gibt er die Nachricht an höhere Schichten weiter, wenn nein, wird erneut die Routingtabelle befragt und die Nachricht weitergeleitet.

Analog zum Unterbrechungsmechanismus müssen Kernmodule, die den Nachrichtenmechanismus verwenden wollen, systemweit eindeutige Nachrichtentypen reservieren und für jeden Typ eine Verarbeitungsroutine registrieren lassen.

Zum Verschicken einer Nachricht an den nächsten direkten Nachbarn werden, im eigenen Kommunikationsspeicher liegende, Nachrichtenpuffer benutzt. Ein freier Puffer wird initialisiert und die Daten in ihn kopiert. Bei der dann erfolgenden Unterbrechung wird der Index des benutzten Nachrichtenpuffers übertragen. Beim Empfänger wird mit dieser Referenz der Puffer lokalisiert und gemäß des Nachrichtentyps die entsprechende Verarbeitungsroutine aufgerufen. Ist keine solche Routine auf diesem Knoten registriert, wird der Puffer, entsprechend markiert, zurückgeschickt. Verbrauchte Puffer werden vom Empfängerknoten nicht sofort wieder an den Sender zurückgegeben. Sie werden zurückgegeben, wenn eine direkte Anforderung des Nachbarknotens erfolgt, wenn eine bestimmte Anzahl überschritten wird oder wenn selbst eine Nachricht in Gegenrichtung zu schicken ist, der man die Puffer mitgeben kann. Mit diesem Vorgehen kann die Anzahl der Unterbrechungen zwischen den Knoten gesenkt werden.

Besondere Beachtung muß der Ressource-Konsistenz geschenkt werden, da Nachrichtenpuffer, die an einen Empfängerknoten übergeben wurden solange nicht wiederverwendet werden dürfen, bis eine entsprechende Mitteilung des Empfängers erfolgt. Daher wird von jedem versandten Puffer der Status und der Empfänger vermerkt. Die Rückgewinnung abgeschickter Puffer beim Ausfall eines Empfängers wird dadurch erheblich erleichtert.

Auf diesem Mechanismus bauen (Benutzer-)Nachrichten, Semaphore und Signale auf, die von einer Applikation benutzt werden können.

4.2.3 Kommunikationsspeicher und Shared-Memory-Mechanismus

Jedem Knoten ist ein Kommunikationsspeicher zugeordnet, für dessen Verwaltung er zuständig ist. Die Verwaltung erfolgt seitenorientiert[5]. Der Kommunikationsspeicher wird als Menge von Seiten betrachtet, die über eine Indextabelle verkettet sind. Diese Tabelle ist im Kommunikationsspeicher abgelegt, um schnellen Zugriff von Nachbarknoten zu ermöglichen. Angeforderter Speicher wird nicht sofort in den virtuellen Adreßraum eingeblendet, stattdessen werden Funktionen bereitgestellt, die das Ein- und Ausblenden ermöglichen. Eine Speicheranforderung liefert somit nur eine Refe-

5. Es werden nur Seiten mit einer Granularität von 4k Byte unterstützt.

renz auf den Anfang der verketteten Seiten. Dieses Vorgehen ist notwendig, da der Speicher in verschiedene Adreßräume (Kern oder Benutzer) eingeblendet werden kann und sich diese von Knoten zu Knoten unterscheiden.

4.2.4 Routing in MEMSY

In Netzwerken gibt es im allgemeinen mehrere Wege um von A nach B zu gelangen. Die Auswahl des optimalen Wegs obliegt dem Routing. Dabei kann diese Entscheidung entweder aufgrund lokaler vollständiger Information über den Systemzustand erfolgen, oder von einem ausgezeichneten Knoten aufgrund vollständiger Information diktiert werden. Beide Verfahren erfordern das Sammeln der vollständigen Information, welches bei größeren Systemen einen hohen Kommunikationsaufwand bedingt.

In MEMSY wurde ein Verfahren gewählt, daß nur die lokale Zustandsinformation und Teile der Zustandsinformation der direkten Nachbarn benötigt. Dieses Verfahren kann daher nur aus lokaler Sicht optimale Entscheidungen treffen. Der geringere Aufwand bei der Erstellung der lokalen Routingtabellen rechtfertigt dieses Vorgehen. Die Wahrscheinlichkeit, daß ein größerer Umweg gewählt wird, ist bei dem im folgenden beschriebenen Verfahren gering.

Im Experimentalsystem MEMSY ist jeder Knoten für seine eigene Routingtabelle zuständig. Die Routinginformation setzt sich aus einer lokalen Routingtabelle und den Zustandsinformationen der direkten Nachbarn zusammen. Die lokale Routingtabelle wird vom Betriebssystem, alle weiteren Daten werden von einem Prozeß, dem Routingprozeß, verwaltet. Nach dem Systemstart enthält die lokale Routingtabelle nur die Einträge der direkten Nachbarn, die auch nicht verändert werden können. Damit ist bereits Kommunikation möglich, ohne daß der Routingprozeß aktiv werden muß.

Der Routingprozeß kommuniziert mit denen der direkten Nachbarn und erstellt aus der Routinginformation eine lokale Sicht des Routings. Diese wird in einer weiteren Tabelle, die im Kommunikationsspeicher liegt, abgelegt. Die Einträge bestehen aus zwei Wegangaben, dem besten und dem nächstbesten Weg. Ist ersterer gestört, wird automatisch der Alternativweg gewählt. Die Auswertung der Routinginformation wird periodisch immer wieder erneut vorgenommen. Ergeben sich große Änderungen, benachrichtigt der Routingprozeß die Routingprozesse seiner direkten Nachbarn, damit diese die Änderungen sofort in ihre eigenen Tabellen übernehmen können. Die Schwelle, wann eine Änderung als groß zu betrachten ist, muß noch ermittelt werden.

In IP-Netzwerken werden Routingentscheidungen aufgrund einer Metrik getroffen. Die Metrik beschreibt Eigenschaften der konkreten Netzwerkverbindung. In MEMSY wird ein verwandtes Verfahren eingesetzt. Als Entscheidungsgrundlage für die Routinginformation dient die Anzahl der elementaren Kommunikationsschritte auf dem gewählten Weg bis zum Ziel und die Auslastung der an diesem Weg beteiligten Knoten.

LST+93.doc: 27.03.95 12:28

5 Kommunikation und dynamische Rekonfiguration

Kommunikation und Konfiguration eines verteilten Systems sind eng miteinander verbunden. Kommunikation ist ohne Kenntnis der Konfiguration nicht möglich. Andererseits ist eine Systemkonfiguration ohne Kommunikation zwischen einzelnen Teilen überflüssig. Soll ein verteiltes System den Ansprüchen der Fehlertoleranz genügen, sind Konzepte und Verfahren zur dynamischen Rekonfiguration erforderlich. Von selbständiger dynamischer Rekonfiguration spricht man, wenn ein verteiltes System sich dynamisch, ohne Intervention eines Administrators, an veränderte Betriebsbedingungen anpaßt.

Der günstigste Kommunikationsweg in einem verteilten System um eine Nachricht von A nach B zu transportieren ist im allgemeinen nicht eindeutig feststellbar. Faktoren wie die Auslastung der an dem Transport beteiligten Knoten, die Anzahl der erforderlichen elementaren Kommunikationsschritte sowie die Menge der zu übertragenden Daten machen eine effiziente Auswahl zum Zeitpunkt des Kommunikationswunsches sehr aufwendig.

Je größer ein verteiltes System ist, um so wahrscheinlicher wird der Ausfall einzelner Komponenten. Führt der Ausfall einzelner Komponenten zum Ausfall des gesamten Systems, sinkt die Verfügbarkeit auf nicht mehr tolerierbare Werte ab. Deshalb darf der Ausfall von Teilkomponenten den Rest des Systems nicht – oder nur so wenig wie möglich – beeinflussen. Auf Anwendungsebene sind hierfür Mechanismen der Fehlertoleranz (z.B. Sicherungspunkte) einzusetzen. Auf Betriebssystemebene ist dieses Vorgehen nicht möglich. Ein Rücksetzen auf einen Sicherungspunkt ist bei permanentem Ausfall eines Knotens nicht durchführbar, da dieser jetzt defekte Knoten in diesem Sicherungspunkt mit enthalten sein würde. Auch würde ein solches Vorgehen bedeuten, daß das System kurzfristig komplett angehalten werden müßte, was einem Totalausfall gleich käme. Das Betriebssystem muß folglich den Ausfall registrieren, verwalten und geeignete Maßnahmen zur Eingrenzung des Schadens treffen. Das System muß dynamisch auf die geänderte Betriebssituation reagieren, es muß sich dynamisch rekonfigurieren.

In Abschnitt 4.2 wurde bereits auf die wesentlichen Kommunikationsprimitive eingegangen. In den folgenden Abschnitten sollen nun die für das Experimentalsystem MEMSY entwickelten Verfahren zur Konfiguration und Rekonfiguration dargestellt werden. Ein Verfahren zur Erkennung und Verwaltung von Teilausfällen wird diskutiert und dessen Eignung zur dynamischen Rekonfiguration erläutert, wobei nur Aspekte des Betriebssystems betrachtet werden.

LST+93.doc: 27.03.95 12:28

5.1 Die Bootphase in MEMSY – initiale Konfiguration

Um die Konfiguration auch großer Systeme einfach zu halten, sollte mit einem Minimum an erforderlicher Anfangsinformation die Gesamtkonfiguration ermittelt werden können. Bei MEMSY genügt in der Regel die systemweit eindeutige Identifikationsnummer des Knotens (KnotenId).

5.1.1 Fehlermöglichkeiten beim Systemaufbau

Die einzelnen Knoten von MEMSY können gemäß der gewünschten Topologie manuell miteinander verkabelt werden. Da auch alternative Topologien im Rahmen des Sonderforschungsbereiches untersucht werden sollen, muß das System selbsttätig die momentan aufgebaute Struktur, d.h. die aktuellen Nachbarschaftsbeziehungen, erkennen (Konfiguration). Anhand der vom System erkannten Topologie kann der Administrator die Korrektheit des Aufbaus verifizieren.

5.1.2 Phasen des Bootvorgangs und der Konfiguration

Beim Experimentalsystem MEMSY gliedert sich die Bootphase in mehrere Teilphasen. In einer ersten Phase werden die erreichbaren Kommunikationsspeicher ermittelt und die eigenen markiert. In einer zweiten Phase werden die weiteren Kommunikationsspeicher direkten Nachbarn zugeordnet. Dieses ist möglich, wenn die Nachbarknoten die erste Phase abgeschlossen haben. Sobald zu einem Zeitpunkt keine weitere Information gewonnen werden kann, wird in einer dritten Phase das Betriebssystem geladen. Da zwischen dem Systemstart einzelner Nachbarknoten kein fester zeitlicher Zusammenhang besteht, kann auch nach dem Einsprung in das Betriebssystem, die zweite Phase fortgeführt werden. Hierzu ist ein Anstoß seitens des betroffenen Knotens notwendig.

5.1.2.1 Finden des eigenen Kommunikationsspeichers

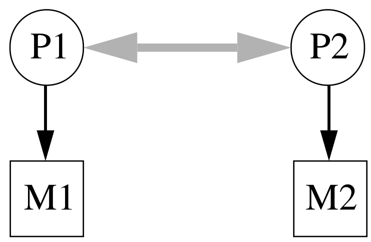
Nach dem Feststellen der physikalischen Präsenz der Kommunikationsspeicher wird zunächst der eigene Kommunikationsspeicher identifiziert. Bei Matrizen mit einer Kantenlänge größer zwei ist der eigene Kommunikationsspeicher derjenige, welcher auf genau zwei unterschiedlichen physikalischen Adressen sichtbar ist (vergleiche [Dal92]).

5.1.2.2 Finden der direkten Nachbarn

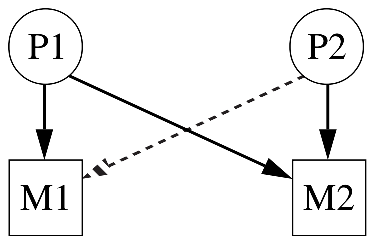
Im nächsten Schritt wird versucht, zu den bestehenden Kommunikationsspeicherverbindungen die Unterbrechungsverbindungen zu selektieren und so die direkten Nachbarn zu ermitteln. Ein Nachbar ist ein direkter Nachbar, wenn es einen gemeinsamen Kommunikationsspeicher und eine Unterbrechungsverbindung zwischen beiden Knoten gibt. Zum Ermitteln der Verbindungen wird im eigenen Kommunikationsspeicher eine, den Knoten identifizierende, Statusinformation hinterlegt und auf allen noch nicht zugeordneten Unterbrechungsverbindungen eine Unterbrechung ausgelöst. Stellt ein Knoten eine Unterbrechung fest, wird versucht die Statusinformation aus den Kommunikationsspeichern auszulesen. Um bei diesem Vorgehen eine zuverlässige Entscheidung treffen zu können, sind möglicherweise Wiederholungen notwendig, da mehrere Knoten gleichzeitig aktiv sein können. Konnte ein Knoten in der ersten Phase keinen eigenen Kommunikationsspeicher und keine statischen Konfigurationsdaten finden, so reagiert er in dieser Phase lediglich auf eingehende Unterbrechungen.

Als Ergebnis der Suchphase ergeben sich unterschiedliche Arten von Verbindungen.

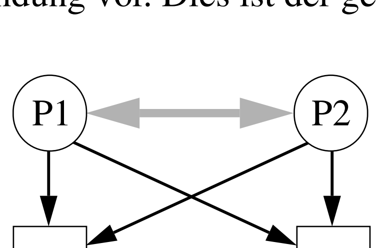
- Es existiert eine Unterbrechungsverbindung, aber keine Kommunikationsspeicherverbindung. Diese Möglichkeit wird als Aufbaufehler gewertet.



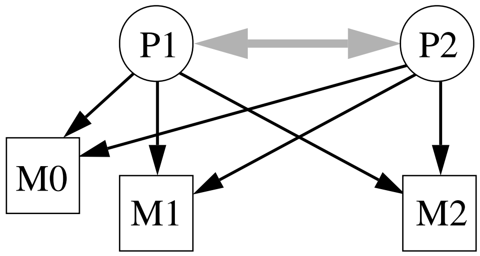
- Es existiert eine Kommunikationsspeicherverbindung, jedoch keine Unterbrechungsverbindung. Hierbei handelt es sich entweder um einen Aufbaufehler oder um eine durch den Aufbau der Kommunikationsspeicherverkabelung zusätzlich entstandene Verbindung, die vom System nicht genutzt werden soll.



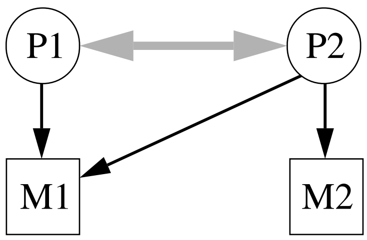
- Es existiert eine Unterbrechungsverbindung und beide Knoten haben auf genau einen eigenen Kommunikationsspeicher des anderen Knotens Zugriff. Es liegt eine symmetrische Verbindung vor. Dies ist der gewünschte Zustand.



- Es existiert eine Unterbrechungsverbindung und mindestens ein Knoten hat auf mehr als einen eigenen Kommunikationsspeicher des anderen Knotens Zugriff. In diesem Fall müssen sich die Knoten auf einen der Kommunikationsspeicher einigen oder es muß auf statische Konfigurationsdaten zurückgegriffen werden.



- Es existiert eine Unterbrechungsverbindung, aber nur einer der Knoten hat Zugriff auf einen eigenen Kommunikationsspeicher des anderen Knotens. Es liegt eine asymmetrische Verbindung vor. In MEMSY sind zum Beispiel die Verbindungen zwischen den Ebenen asymmetrisch. Um hier eine echte Kommunikation ermöglichen zu können, muß diese Verbindung durch Aufteilen des physikalischen Kommunikationsspeichers in zwei logische Teile symmetrisiert werden. Dazu wird von dem Knoten, der keinen Kommunikationsspeicher besitzt, auf den der andere Knoten zugreifen kann, über das Konfigurationsprotokoll eine spezielle Speicheranforderung gestellt. Die Verwaltung des dann zugeteilten Speichers wird vom antragstellenden Knoten übernommen. Es ist zu beachten, daß zum Beispiel bei einem Ausfall des Nachbarn der zugeteilte Speicher unter Umständen wieder neu angefordert werden muß.



5.1.2.3 Laden des Betriebssystems

Derzeit wird das Betriebssystem von der lokalen Platte geladen. In Zukunft sollen alle Knoten des Systems durch die Kommunikationsspeicher eine „Basis"-Version des Betriebssystems laden. Dieses ist aus Konsistensgründen zweckmäßig um sicherzustellen, daß alle Knoten unter der gleichen Version laufen.

Zum Laden werden in allen eigenen Kommunikationsspeichern die Ladewünsche hinterlegt und zyklisch alle direkten Nachbarn per Unterbrechung aufgefordert ein Betriebssystem zur Verfügung zu stellen. Im Erfolgsfall wird das Betriebssystem über einen definierten Pufferbereich durch den Kommunikationsspeicher transportiert und geladen (vergleiche Abschnitt 5.1.3).

5.1.3 Auswirkungen des Bootvorgangs auf den Betrieb des Systems

Teilphasen des Bootvorgangs können sich, auch während das Betriebssystem vollständig läuft, wiederholen. Dies ist eine Grundvoraussetzung für die Fähigkeit der dynamischen Rekonfiguration des Systems. Da die entsprechenden Codeteile der Bootsoftware nicht im Betriebssystemkern liegen sollten, müssen sie speicherresident und für das Betriebssystem zugreifbar vorhanden sein. Zum Beispiel in den Festwertspeichern der Knoten.

Muß ein MEMSY Knoten während des Betriebs erneut booten, dann darf er, zur Wahrung der Speicherkonsistenz, nicht willkürlich auf die Kommunikationsspeicher zugreifen, sondern nur auf einen definierten Bereich. Diese Bedingung hängt damit zusammen, daß ein Nachbarknoten den Ausfall erst nach dem erneuten Booten mit Sicherheit als solchen erkennen kann. Die Daten in den Kommunikationsspeichern dürfen also nicht modifiziert und die eigenen Kommunikationsspeicher erst initialisiert werden, wenn sichergestellt ist, daß kein anderer Knoten mehr irgendwelche Teile des Kommunikationsspeichers nutzt. Daher darf auch beim Laden des Betriebssystems über den Kommunikationsspeicher nur der dafür vorgesehene Bereich verwendet werden.

5.2 Das Verhalten bei Betriebsstörungen – Rekonfiguration

Eine Betriebsstörung in MEMSY liegt vor, wenn irgendeine Komponente während des Betriebs ihre Funktion nicht mehr wahrnehmen kann. Dabei kann es sich um einen Ausfall oder um eine beabsichtigte Betriebsunterbrechung handeln.

Zwei Arten von Ausfällen sind zu unterscheiden. Der Ausfall eines Knotens und der Ausfall von Kommunikationshardware. Letzterer hat schwerwiegende Auswirkungen auf die Konfiguration des Gesamtsystems. Besonders problematisch ist die Tatsache, daß Fehler in der Kommunikationshardware nicht zwangsläufig von allen Knoten, welche die defekte Komponente nutzen, bemerkt werden müssen. In der Bootphase können unter Umständen Kommunikationsspeicher nicht mehr als eigene erkannt werden. Durch den Ausfall einer Komponente kann zusätzliche feste Konfigurationsinformation für die Dauer des Ausfalls erforderlich sein. In einem solchen Fall ist eine automatische Rekonfiguration des Systems nicht ohne Eingriff eines Administrators möglich.

Im folgenden werden einige Aspekte des Ausfalls eines Knotens näher betrachtet.

5.2.1 Das Feststellen einer Betriebsstörung

Der Ausfall eines Knotens kann nur durch Kommunikationsversuche mit ihm festgestellt werden. Reagiert ein Knoten nicht auf Unterbrechungen, gilt er als ausgefallen. Ein Ausfall wird vom Nachbarn spätestens dann erkannt, wenn der in der Bootphase befindliche Knoten eine entsprechende Kommunikation bezüglich der Konfiguration anstößt.

Beim Neustart eines Knotens sind auf dessen Nachbarknoten eine Reihe von Aktionen durchzuführen. Zunächst werden alle Informationen über den Zustand des ausgefallenen Knotens gelöscht, damit in jedem Fall eine neue Verbindung aufgebaut werden muß. Dann werden alle Speichersegmente, die von einem Kommunikationsspeicher des ausgefallenen Knotens stammen, aus den Adreßräumen der Applikationen entfernt. Den betroffenen Applikationen wird zuvor ein UNIX-Signal zugestellt, damit sie entsprechend reagieren und sonst zu Fehlern führende Speicherzugriffe vermeiden können.

Ist ein Knoten als ausgefallen erkannt, so muß auch das Routing angepaßt werden (siehe Abschnitt 5.2.2). Weiterhin sind alle Segmente des privaten Kommunikationsspeichers, auf die der Nachbarknoten Zugriff hatte, auf ihre Konsistenz hin zu überprüfen. Dies ist aber lediglich bei Segmenten möglich, die Verwaltungsinformation enthalten. Schließlich muß der systemglobalen Applikationsverwaltung der Ausfall gemeldet werden, damit diese ihrerseits geeignete Maßnahmen treffen kann.

5.2.2 Auswirkungen einer Betriebsstörung auf das Routing

Der Ausfall eines Knotens erfordert Änderungen im Routing, da bestimmte Wege jetzt nicht mehr benutzbar sind. Das heißt der lokale Routingprozeß muß die Routingtabellen neu bestimmen und gegebenenfalls Änderungen an die direkten Nachbarn melden (vergleiche Abschnitt 4.2.4). Erfolgt eine Benachrichtigung der direkten Nachbarn, wiederholt sich dort dieser Vorgang.

Auf den ersten Blick scheint dieses Verfahren sehr viele Neubestimmungen der lokalen Routingtabellen anzustoßen. Bedingt durch die Tatsache aber, daß es in MEMSY sehr viele Wege von A nach B gibt, tritt bereits nach wenigen Schritten der Fall ein, daß ein bisher zweitbester Weg weiterhin verwendbar ist und der Abstand seiner Metrik zu der des bisher besten Wegs nicht mehr zu einer Benachrichtigung der direkten Nachbarn führt. Zu beachten ist, daß sich die Routinginformation zu diesem Knoten hin nicht verändert. Änderungen betreffen immer nur Wege. Einzige Ausnahme ist, wenn ein Knoten eine neue KnotenId erhält. Nur in diesem Falle muß systemweit jegliche Routinginformation zum vorhergehenden Knoten gelöscht und die neue Routinginformation von allen Knoten aufgenommen werden.

Der beschriebene Mechanismus ermöglicht neben dem Austausch von Knoten zu Wartungszwecken auch die Erweiterung des Gesamtsystems im laufenden Betrieb, wobei dafür die kurzfristige Stillegung aller Knoten entlang einer Geraden durch das Gesamtsystem am günstigsten erscheint.

Ein besonders kritischer Fall einer Betriebsstörung liegt bei einer Netzwerkpartitionierung vor. Aufgrund der lokal beschränkten Sicht des Routings würde es zu einem unendlichen Anwachsen der Metriken für die nicht mehr erreichbaren Knoten kommen. Daher ist es erforderlich einen Maximalwert systemweit festzulegen. Wird dieser erreicht, bedeutet das, daß der entsprechende Knoten nicht erreichbar ist. Der Routingprozeß entfernt den Knoten aus seiner lokalen Routingtabelle. Nach einer kurzen Einschwingphase wird auf diese Weise veraltete oder ungültige Routinginformation aus dem System entfernt.

5.2.3 Auswirkungen einer Betriebsstörung auf die laufenden Applikation

Fällt ein Knoten aus, so sind bei Applikationen drei Fälle zu unterscheiden. Der ausgefallene Knoten wurde für Rechenaufgaben eingesetzt, der ausgefallene Knoten wurde von der Applikation zu Datentransporten genutzt oder der ausgefalle Knoten wurde in keiner Weise genutzt. Im letzten Fall gibt es keine Auswirkungen auf die Applikation.

In den beiden anderen Fällen muß die Applikation selbst auf die Betriebsstörung aktiv reagieren. Ein fehlertolerantes Verhalten im System ohne entsprechende Unterstützung durch die Applikation ist nicht möglich. Wurde der ausgefallene Knoten zum Rechnen genutzt, erscheint der Abbruch und das Wiederaufsetzen an einem Sicherungspunkt sinnvoll.

Wurde der ausgefallene Knoten nur für Datentransporte genutzt, muß die Applikation auf die, durch den Ausfall auftretenden Störungen, reagieren. Einfach zu erkennen sind dabei Zusammenbrüche von Datenverbindungen durch die Kommunikationsspeicher. Beide Endpunkte können den Zusammenbruch erkennen und eine neue Verbindung aufbauen. Alle Daten ab einem letzten Kontrollpunkt müssen erneut übertragen werden, da bei dem Zusammenbruch Datenverluste aufgetreten sein können. Schwieriger zu behandeln ist der Verlust von Nachrichten. Diese sind verloren, ohne daß eine Fehlerindikation erfolgt. Die Applikation muß hier selbst ein Sicherungsprotokoll realisieren.

6 Zusammenfassung und Ausblick

Mit MEMSY ist ein speichergekoppeltes Parallelrechnersystem entwickelt worden, das aufgrund seiner Architektur die Vorteile der Speicherkopplung nutzt, ohne theoretische Beschränkung der Ausbaubarkeit.

Die vermeintliche Beschränkung durch die „Nearest-Neighbour-Verbindungsstruktur" trifft für ein weites Feld von numerischen Problemstellungen nicht zu, da die Datenstrukturen dieser Problemklassen mit der Architektur des MEMSY-Systems korrespondieren.

Im, für MEMSY definierten, Programmiermodell besteht, im Gegensatz zu Programmiermodellen anderer Multiprozessorsysteme, eine engere Verzahnung mit der Architektur, die auch ganz bewußt dem Anwender offengelegt ist. Wir versprechen uns dadurch effizientere Anwendungen, die trotzdem leicht auf andere Architekturen portierbar sein sollten. Umgekehrt ist es dem Anwender, durch die Unterstützung vielfältiger Kommunikationsmechanismen, erleichtert worden, bestehende Anwendungen auf MEMSY zu portieren.

LST+93.doc: 27.03.95 12:28

Das aufgebaute Experimentalsystem mit 80 Prozessoren zeigt, daß man auch mit relativ bescheidenen Mitteln in der Umgebung einer Universität solche Parallelrechensysteme entwickeln kann. Die Konstruktion des Systems unterstützt den Aufbau alternativer Topologien und die Integration weiterer Kommunikationsmechanismen. So wird z.B. am Lehrstuhl für Rechnerstrukturen an neuen globalen Synchronisationseinheiten gearbeitet und die optischen Kommunikationselemente werden am Lehrstuhl für Angewandte Optik weiterentwickelt.

Um auch objektorientierte Ansätze bei der Lösung numerischer Aufgabenstellungen besser verfolgen zu können, wird zur Zeit am Lehrstuhl für Betriebssysteme die Entwicklung eines Betriebssystems auf der Basis des Mach-Kernels vorangetrieben.

Literatur

[Dal92] M. Dal Cin, „Rechnerbaukasten für funktionsorientierte Multiprozessoren mit Fehlertoleranzeigenschaften"; in: *Arbeits- und Ergebnisbericht des SFB 182 Multiprozessor- und Netzwerkkonfigurationen 1990 - 1992*, Seite 114 ff.

[Fri89] G. Fritsch, W. Henning, H. Hessenauer, R. Klar, C.-U. Linster, C.-W. Oehlrich, P. Schlenk, J. Volkert, Distributed Shared Memory Multiprocessor Architecture MEMSY for High Performance Parallel Computations; in: *Computer Architecture News*, Vol. 17, No. 6, Dec. 1989

[Hän76] W. Händler, F. Hofmann, H. J. Schneider, A General Purpose Array with a Broad Spectrum of Applikations; in: W. Händler: Computer Architecture, *Informatik Fachberichte*, Springer Verlag Berlin Heidelberg NewYork, Nr. 4, Seiten 311-335, 1976

[Hed88] C. Hedrick, Routing Information Protocol; RFC 1058, June 1988

[Hil92] U. Hildebrand, *Konzeption, Bewertung und Realisierung einer dynamischen Netzwerkkomponente für speichergekoppelte Multiprozessoren*; Dissertation, Arbeitsberichte des IMMD, Univ. Erlangen-Nürnberg, Band 25, Nr. 5, 1992

[Kar91] F. Kardel, W. Stukenbrock, T. Thiel, S. Turowski, *Anleitung zur Benutzung des MEMSY-Programmiermodells für Anwender*; interner Bericht, IMMD 4, Univ. Erlangen-Nürnberg, Oktober 1991

[Rot93] B. Rotzoll, Dynamischer Multiprogrammbetrieb von Parallelrechnern; in diesem Band, 1993

[Rus89] Karl J. Rusnock, *Multiprocessor SYSTEM V/88 Release 3 Design Specification*; Motorola, Tempe, Confidential Proprietary, May 1989

[Rus90] K. Rusnock, P. Raynoha, „Adapting the Unix operating system to run on a tightly coupled multiprocessor system"; *VMEbus Systems*, Oct. 1990, Vol. 6, No. 5, pp. 8-28

[Sch93] J. Schwider, N. Streibl, K. Zürl, Optoelektronische Verbindungen in Multiprozessoren; in diesem Band, 1993

[Tan88] A. S. Tanenbaum, *Computer Networks*; Prentice-Hall International, Englewood Cliffs, NJ u.a., 1988

[Wul72] W. A. Wulf, C. G. Bell, C.mmp: A multi-miniprocessor; in *1972 AFIPS Fall Jt. Computer Conf.*, AFIPS Pres, Arlinghton, Va., Dec. 1972

[Wul74] W. A. Wulf, E. Cohe, W. Corwin, A. Jones, R. Levin, C. Pierson, F. Pollack, HYDRA: The kernel of a multiprocessor system; Communications of the ACM, Vol. 17, No. 6, pp. 337-345, June 1974