Universität Karlsruhe

Fakultät für Informatik

Institut für Betriebs- und Dialogsysteme

Portierung des COSY–Betriebssystemkerns auf PowerPC

cand.-inform. Roland Weber

Diplomarbeit
an der
Universität Karlsruhe (TH)
Fakultät für Informatik
Institut für Betriebs- und Dialogsysteme

Betreuer: Prof. Dr.-Ing. H. Wettstein
Dipl.-Inform. W. Burke

Ich versichere, daß ich die vorliegende Arbeit ohne fremde Hilfe angefertigt habe. Die verwendeten Quellen sind im Literaturverzeichnis vollständig aufgeführt.

Karlsruhe, den 23. September 1996

We will call a program *portable* if adapting it to a new environment is easier than rewriting it for that environment.

„Notes on Writing Portable Programs in C“

Die Durchführung dieser Diplomarbeit war nur durch die direkte und indirekte Hilfe vieler verschiedener Personen möglich. Stellvertretend für alle, die mich auf die eine oder andere Art unterstützten, möchte ich die folgenden Personen erwähnen:

Professor Wettstein schuf durch jahrzehntelange Forschungsarbeit im Bereich Betriebssysteme die Grundlagen für die Entwicklung von COSY und damit auch für diese Diplomarbeit.

Wolfgang Burke investierte viel Zeit in die Korrektur dieser Ausarbeitung. Eventuelle Schwächen in der Darstellung einzelner Sachverhalte sind mit Sicherheit darauf zurückzuführen, daß ich seine Bemerkungen ignoriert habe.

Roger Butenuth entfwarf und implementierte COSY für Transputer. Bei der Einarbeitung in die vorhandenen Quelltexte erwiesen sich seine Kommentare als große Hilfe, egal ob telefonisch, per EMail oder im Quelltext.

James McCuller gelang es trotz widrigster Umstände, die sich in Form von Netzwerk-, Festplatten- und Stromausfällen bemerkbar machten, ein stabiles Rechnersystem zur Verfügung zu stellen und mich vor Datenverlusten zu bewahren.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Aufgabenstellung

In den führenden Industrienationen zeichnet sich ein Wandel von der Industrie- zur Informationsgesellschaft ab, dessen Auswirkungen ähnliche Ausmaße annehmen könnten wie die der Industrialisierung [Cam92] seit der zweiten Hälfte des 19. Jahrhunderts. Im Zuge des Wandels zu einer Informationsgesellschaft, unterstützt durch die Globalisierung der Wirtschaft, ergibt sich ein wachsender Bedarf an leistungsfähigen Rechnern zur Bearbeitung der resultierenden Informationsflut. Die Fortschritte in der Prozessortechnologie [SGI96] können den steigenden Bedarf an Rechenleistung nicht decken. Eine offensichtliche Lösung dieses Problems besteht darin, Aufgaben von mehreren Prozessoren gleichzeitig bearbeiten zu lassen. Eine Möglichkeit dazu bieten *Parallelrechner*.

Die Prozessoren eines Parallelrechners können auf zwei grundsätzlich verschiedene Arten zusammenarbeiten: *speichergekoppelt* und *nachrichtengekoppelt*. Ein speichergekoppeltes System besitzt einen gemeinsamen Speicher, auf den jeder Prozessor zugreifen kann. Diese Systeme gelten als leicht programmierbar, weshalb sie sich zur Zeit großer Beliebtheit erfreuen. Der Zugriff mehrerer Prozessoren auf einen gemeinsamen Speicher setzt allerdings der *Skalierbarkeit* Grenzen, da nur wenige Zugriffe gleichzeitig durchgeführt werden können. Mit zunehmender Anzahl an Prozessoren steigt auch die Wartezeit jedes Prozessors bei Speicherzugriffen. Die zusätzliche Rechenleistung kann deshalb nur teilweise genutzt werden.
In nachrichtengekoppelten Parallelrechnern ist jeder Prozessor mit einem eigenen, lokalen Speicher ausgestattet. Der Austausch von Daten zwischen den Prozessoren erfolgt durch das Versenden von Nachrichten. Hier ergeben sich Probleme durch die Kommunikation zwischen den Prozessoren. Bei Zugriffen auf Daten, die nicht im lokalen Speicher eines Prozessors vorliegen, müssen erhebliche Wartezeiten in Kauf genommen werden.
Durch ihre bessere Skalierbarkeit, in Kombination mit den für die Zukunft zu erwartenden Fortschritten in der Kommunikationstechnologie, scheinen nachrichtengekoppelte Parallelrechner langfristig mehr Entwicklungspotential zu besitzen. Ziel der Forschung sollte es sein, den Nachteil der im Vergleich zu speichergekoppelten Systemen schwierigeren Programmierung zu vermindern. Ein

besonderes Augenmerk gilt in diesem Zusammenhang den *Betriebssystemen* für Parallelrechner, da sie die Basis für die Entwicklung paralleler Anwendungen bilden.

Cosy ist ein Betriebssystem, das speziell für den Einsatz auf hochparallelen, nachrichtengekoppelten Rechner entwickelt wurde und auch auf Parallelrechnern mit über 1000 Prozessoren effizient arbeitet. Die erste Implementierung von Cosy basiert auf Parallelrechnern mit Transputer-Prozessoren [IMS92]. Die Rechenleistung dieser Prozessoren entspricht jedoch nicht mehr dem aktuellen Stand der Technik. Deshalb ist es notwendig, Cosy auf andere Parallelrechner mit moderneren Prozessoren zu portieren. Als Basis für die erste Portierung wurde ein Parallelrechner des Typs PowerXplorer [PXP93] ausgewählt, der mit PowerPC-Prozessoren [PPC94] ausgestattet ist.
Aufgabe dieser Diplomarbeit ist es, den Kern des Cosy-Betriebssystems auf PowerPC-Prozessoren zu portieren. Dabei soll die konzeptionelle Gestaltung des Kerns überprüft und gegebenenfalls korrigiert werden. Eine solche Überprüfung ist notwendig, da erfahrungsgemäß bei einer Erstimplementierung die verwendete Plattform einen erheblichen Einfluß auf die Gestaltung eines Systems ausübt.

Die Gliederung der vorliegenden Ausarbeitung ist an das Vorgehen bei der Durchführung der Diplomarbeit angelehnt. Kapitel 2 beschreibt das Cosy-Betriebssystem, insbesondere den zu portierenden Kern. In Kapitel 3 ist die Plattform für die vorhandene Implementierung von Cosy beschrieben, Kapitel 4 beschäftigt sich mit der Zielplattform. Der Überblick über die verfügbaren Entwicklungswerkzeuge in Kapitel 5 beschließt die Beschreibung der Grundlagen für die vorzunehmende Portierung.
In Kapitel 6 wird eine Methodik zur Durchführung von Betriebssystemportierungen entwickelt. Die darauf folgenden Kapitel beschäftigen sich mit einigen konzeptionellen Schwerpunkten der Portierung: die Speicherverwaltung in Kapitel 7, der Speicherschutz in Kapitel 8, die Prozeßumschaltung in Kapitel 9 und die knotenübergreifende Kommunikation in Kapitel 10. Dabei wird jeweils die Realisierung auf der Ausgangs- und der Zielplattform gegenüber gestellt. Eine Bewertung der Implementierung findet sich in Kapitel 11. Abschließend faßt Kapitel 12 die Resultate der Diplomarbeit zusammen und gibt einen Ausblick auf weitere Möglichkeiten im Zusammenhang mit Cosy.
Grundlage der gesamten Ausarbeitung ist die *Systemarchitektur* [Wett93]. Grundbegriffe, wie etwa *Prozeß* und *Programm*, werden entsprechend der dort festgelegten Bedeutung ohne explizite Einführung verwendet.

Kapitel 2

COSY

Ein Betriebssystem hat eine Vielzahl von Aufgaben zu bewältigen. Wer ein Betriebssystem benutzt erwartet von diesem, daß es Programme ausführen kann. Zum Ausführen eines Programms müssen Betriebsmittel, zum Beispiel Speicher, bereitgestellt werden. Das Programm muß Zugriff auf Eingabedaten erhalten und seine Resultate ausgeben können. Nach der Ausführung sollen alle belegten Betriebsmittel wieder freigegeben werden, damit sie anderen Programmen zur Verfügung stehen. Laufen mehrere Programme gleichzeitig ab, so sollen diese untereinander Daten austauschen können, jedoch vor unerwünschter Beeinflussung durch Fehler oder Störversuche in anderen Programmen geschützt sein.
Schon dieser kurze Überblick zeigt, daß ein Betriebssystem eine Vielzahl verschiedener Aufgaben zu bewältigen hat. Bei der Realisierung eines Betriebssystems ist es deshalb zweckmäßig, die einzelnen Aufgaben in eigenen Systemteilen zu kapseln, um ein überschaubares Gesamtsystem zu erhalten. Ein erster Ansatz hierzu ist die Aufteilung in einen *Instanzenbereich* und einen *Infrastrukturbereich*. Letzterer wird auch als *Kern* des Betriebssystems bezeichnet.

Wie die meisten modernen Betriebssysteme ist COSY nach dem Prinzip der *Mikrokernarchitektur* entworfen. Ein Mikrokern enthält alle für den Betrieb unbedingt notwendigen Teile. Zusätzliche Funktionalität wird nur dann angeboten, wenn eine Realisierung im Instanzenbereich zu unverhältnismäßig hohen Effizienzverlusten des Beriebssystems führen würde. Bei diesem Vorgehen bleibt das Verhalten des Kerns überschaubar. Der Nachteil ist eine erhöhte Komplexität des Instanzenbereichs, in dem die für eine komfortable Anwendungsentwicklung zusätzlich benötigte Funktionalität durch Dienstgeber und Bibliotheken zur Verfügung zu stellen ist.

Beispiel. *Ein Mikrokern enthält keine Treiber für Peripheriegeräte. An der Kernschnittstelle wird nur die Funktionalität angeboten, die Treiber im Instanzenbereich benötigen, um Peripheriegeräte anzusprechen. Ein willkommener Nebeneffekt dieser Aufteilung ist, daß eventuell vorhandene Peripheriegeräte die Realisierung des Mikrokerns nur in geringem Umfang beeinflussen.*

Der folgende Abschnitt 2.1 beschreibt den im Rahmen dieser Diplomarbeit zu portierenden Mikrokern des COSY-Betriebssytems. Abschnitt 2.2 geht auf die Besonderheiten des COSY-Kerns ein, die den Einsatz auf nachrichtengekoppelten Parallelrechner ermöglichen. Die Beschreibung der wichtigsten COSY-Dienstgeber in Abschnitt 2.3 ergänzt den Überblick über das Betriebssystem.

2.1 Kern

Wie bereits erwähnt sollte der Aufgabenbereich eines Kerns möglichst klein und klar abgegrenzt sein, damit er sowohl für den System- als auch den Anwendungsentwickler überschaubar bleibt. Der COSY-Kern beschränkt sich auf die Unterstützung von sechs Kernobjekttypen, die in den folgenden Abschnitten 2.1.1 bis 2.1.4 beschrieben werden. Abschnitt 2.1.5 behandelt die Schnittstelle des COSY-Kerns.

2.1.1 Speicherverwaltung

Der Speicher gehört zu den grundlegenden Betriebsmitteln, denn jedes Programm, das ausgeführt werden soll, benötigt Speicher. Dieser dient sowohl dazu, das Programm abzulegen, als auch zur Aufbewahrung von Daten, die zum Abarbeiten benötigt werden oder bei der Abarbeitung entstehen.
Sowohl abzuarbeitende Programme als auch der Kern benötigen Speicher. Dieser kann also nicht in einem Stück vergeben werden, eine Verwaltung als Mehrexemplarbetriebsmittel ist erforderlich. Erschwerend kommt hinzu, daß mehrere Instanzen auf Teile des Speichers gemeinsam zugreifen können sollen. So ist es bei gleichzeitiger Ausführung mehrerer Instanzen eines Programms sicher angebracht, das Programm nur einmal in den Speicher zu laden, während die Daten für die ausführenden Prozesse in getrennten Teilen des Speichers liegen. Aus Effizienzgründen ist in diesem Fall ein direkter Zugriff auf den Speicher angebracht, der somit nicht exklusiv vergeben werden darf. Es ergibt sich eine $n : m$–Beziehung zwischen Teilen des Speichers und den darauf zugreifenden Instanzen.

Viele Rechner bieten technische Möglichkeiten, um Speicherzugriffe zu kontrollieren und zu manipulieren. Der Einsatz dieser Möglichkeiten muß durch eine privilegierte Komponente des Betriebssystems erfolgen, um einen verläßlichen Speicherschutz anbieten zu können. Es liegt nahe, diese Aufgabe dem Kern zuzuteilen, dessen Zuverlässigkeit ohnehin Grundlage für den Einsatz jedes Betriebssystems ist.
Dieses Vorgehen widerspricht dem Prinzip der Mikrokernarchitektur nicht, da *Umgebungswechsel* zu den Aufgaben des Kerns gehören. Im Zuge eines Umgebungswechsels ist es oftmals erforderlich, die Rechte für Speicherzugriffe zu verändern. Müßte der Kern dabei auf eine Komponente im Instanzenbereich zurückgreifen, so wäre ein erheblicher Effizienzverlust kaum zu vermeiden.

Die Vergabe von Rechten für Speicherzugriffe ist unteilbar mit der Bereitstellung von Speicher verbunden. Beides gehört in COSY zu den Aufgaben des Kerns, der dazu die folgenden drei Kernobjekttypen unterstützt:

Adreßräume sind Umgebungen, in denen Speicher benutzt werden kann. Zum Beispiel existieren Prozesse innerhalb von Adreßräumen. Tatsächlich ist der Adreßraum[1] in COSY der wichtigste Teil der *Ablaufumgebung* jedes Prozesses.

Adreßbereiche definieren Teile des verfügbaren Speichers. Ein Adreßbereich besteht aus einer Anzahl aufeinanderfolgender Speicherworte und deren Inhalt. Da COSY zur Zeit keinen virtuellen Speicher unterstützt, kann jeder Adreßbereich mit einem Teil des physikalischen Speichers identifiziert werden.

Einblendungen modellieren $1:1$–Beziehungen zwischen Adreßräumen und Adreßbereichen. Eine Einblendung definiert die Adresse, an der das erste Speicherwort eines Adreßbereichs in einem Adreßraum angesprochen werden kann. Die restlichen Speicherworte des Adreßbereichs sind über die unmittelbar folgenden Adressen zugänglich.

Zwischen Adreßräumen und Einblendungen besteht eine $1:n$–Beziehung, das heißt, ein Adreßraum kann mehrere Einblendungen enthalten. Adreßbereiche können ihrerseits in mehrere Adreßräume eingeblendet werden, auch hier besteht also eine $1:n$–Beziehung. Insgesamt ergibt sich daraus die benötigte $n:m$–Beziehung zwischen Adreßräumen und Adreßbereichen.
Damit Speicherzugriffe eindeutig sind, dürfen sich die Einblendungen in einem Adreßraum nicht überlappen. Dies wird vom Kern bei der Erzeugung und Manipulation von Einblendungen garantiert.

2.1.2 Prozesse

Prozesse sind Träger von Aktivitäten. Da in modernen Betriebssystemen üblicherweise mehr Aktivitäten als Prozessoren existieren, ist eine dynamische Vergabe von Prozessoren an Aktivitäten, beziehungsweise eine dynamische Zuordnung von Aktivitäten zu Prozessoren, notwendig. Diese Zuordung ist eine zentrale Aufgabe eines Betriebssystemkerns.

Der COSY-Kern unterstützt zu diesem Zweck Kernobjekte des Typs *Prozeß*. Ein Prozeß im Sinne der Systemarchitektur besteht aus einem Programm, Daten und einer Aktivität. Programm und Daten werden in COSY in einem geeignet initialisierten Adreßraum hinterlegt und bei der Erzeugung eines Prozesses als Adressen in diesem Adreßraum angegeben. Ein COSY-Prozeß ist an den Adreßraum gebunden, für den er erzeugt wurde.

[1] Die vollständige Bezeichnung in der Systemarchitektur lautet *Programmadreßraum*.

Die Vergabe von Prozessoren an Prozesse erfordert eine Strategie. In COSY beruht diese auf einem Zeitscheibenverfahren mit Verdrängung. Bei Vergabe eines Prozessors an einen Prozeß wird eine maximale Nutzungsdauer, die *Zeitscheibe*, festgelegt. Ist diese Nutzungsdauer abgelaufen oder wird ein weiterer Prozeß in die Menge der sich um einen Prozessor bewerbenden Prozesse aufgenommen, so erfolgt eine Überprüfung und eventuelle Änderung der Zuordnung. Bei der Überprüfung werden aus der Menge der konkurrierenden Prozesse diejenigen mit der höchsten Dringlichkeit ausgewählt, um ihnen Prozessoren zuzuordnen. Diese Menge schließt auch Prozesse ein, denen bereits Prozessoren zugeteilt sind, eine Änderung der Zuordnung erfolgt also nicht zwingend.

Zur Feststellung der Dringlichkeit eines Prozesses dienen Prioritäten. Jedem Prozeß ist eine Priorität aus einer endlichen, total geordneten Menge zugeordnet. Bewerbende Prozesse mit gleicher Priorität sammeln sich in einer Warteschlange für diese Priorität. Innerhalb einer solchen Warteschlange hat derjenige Prozeß die höchste Dringlichkeit, der sich am längsten in der Warteschlange befindet. Prozesse, denen ein Prozessor zugeordnet ist, besitzen eine höhere Dringlichkeit als die Prozesse gleicher Priorität, die sich in der entsprechenden Warteschlange befinden. Durch dieses Vorgehen bleibt die Zuordnung eines Prozessors zu einem Prozeß für eine volle Zeitscheibe erhalten, sofern dieser Prozeß nicht von einem Prozeß mit höherer Priorität verdrängt wird.

Die Priorität eines COSY-Prozesses verändert sich bei festgelegten Ereignissen: Nach dem Ablauf einer Zeitscheibe erniedrigt sich die Priorität des entsprechenden Prozesses, bei Deblockiervorgängen erhöht sie sich. Durch diese *dynamischen Prioritäten* nehmen Prozesse, die langwierige Berechnungen durchführen, eine niedrige Priorität an. Prozesse, die häufig blockieren und nach der Deblockierung nur kurze Berechnungen durchführen, erhalten eine hohe Priorität. Dies tritt etwa bei der Reaktion auf Benutzereingaben oder bei periodischen Vorgängen wie dem regelmäßigen Ermitteln von Auslastungsdaten auf. Diesen Prozessen würde man auch in einem System mit festen Prioritäten Vorrang einräumen, was dort jedoch ein explizites Setzen erfordert. Dynamische Prioritäten ermöglichen zudem eine automatische Anpassung bei Prozessen, deren Verhalten sich während des Ablaufs ändert.

2.1.3 Kommunikation

Kommunikation bezeichnet das Übergeben von Daten eines *Sender*–Prozesses an einen *Empfänger*–Prozeß. Die zu übergebenden Daten werden als *Nachricht* bezeichnet. Die Übergabe erfolgt in COSY durch Kopieren der Nachricht in den Adreßraum des empfangenden Prozesses, also per *Wertübergabe*.

Der COSY-Kern unterstützt Kernobjekte vom Typ *Kanal* als Treffpunkte von Sendern und Empfängern. Jeder COSY-Kanal besitzt einen begrenzten Stauraum zur Aufnahme von Nachrichten. Die Operationen **Senden** und **Empfangen** laufen an einem unbenutzten Kanal nach folgendem Grundschema ab:

Senden: Im Kanal wird eine Sendebereitschaft hinterlegt, in der unter anderem die Identität (Prozeßkennung) des Senders enthalten ist. Die zu versendende Nachricht ist im Adreßraum des Senders hinterlegt. Steht genügend Stauraum zur Verfügung, so kann sie in den Kanal kopiert werden.

Empfangen: Im Kanal wird eine Empfangsbereitschaft hinterlegt.

Ist der Kanal nicht leer, sondern enthält bereits mindestens eine Bereitschaft für eine Operation der gleichen Art, so wird die neue Bereitschaft an die bereits vorliegenden angehängt. Im Kanal entsteht eine Warteschlange von Sende- oder Empfangsbereitschaften.
Enthält der Kanal eine Bereitschaft für eine komplementäre Operation, also eine Empfangsbereitschaft bei einer Sendeoperation beziehungsweise eine Sendebereitschaft bei einer Empfangsoperation, so initiiert dies eine Kommunikation. Die erste im Kanal hinterlegte Bereitschaft wird entnommen, sie identifiziert den Kommunikationspartner. Sende- und Empfangsbereitschaften befinden sich also niemals gleichzeitig in einem Kanal.

Ist eine Kommunikation initiiert, das heißt, ein Sender und ein Empfänger haben sich an einem Kanal getroffen, kann die Nachricht aus dem Adreßraum des Senders angefordert werden, falls sie nicht unmittelbar verfügbar ist. Eine Kopie der Nachricht wird im Adreßraum des Empfängers abgelegt. In Verbindung mit der Kommunikation erfolgt meist auch eine zeitliche Abstimmung von Sender und Empfänger. Dazu kann noch eine Mitteilung an den sendenden oder empfangenden Prozesses erforderlich sein. Die Art der zeitlichen Abstimmung wird beim Senden beziehungsweise Empfangen festgelegt. Es gibt vier grundsätzliche Arten, die sich beliebig kombinieren lassen:

Versuchend: Diese Variante weicht vom Grundschema des Sendens und Empfangens dahingehend ab, daß auf keinen Fall eine Bereitschaft im Kanal hinterlegt wird. Eine Kommunikation findet nur dann statt, wenn bereits eine komplementäre Bereitschaft im Kanal hinterlegt war. Der ausführende Prozeß erhält als Resultat der versuchenden Kommunikationsoperation die Information, ob eine Kommunikation stattfand oder nicht.

Asynchron: Eine asynchrone Kommunikationsoperation beschränkt sich auf das Hinterlegen der Bereitschaft und eventuell der Nachricht im Kanal. Der ausführende Prozeß erhält keine Information darüber, daß die Kommunikation stattgefunden hat.

Umlenkend: Nach erfolgter Kommunikation wird der Prozeß an einer beim Aufruf der Operation angegebenen Stelle fortgesetzt. Dazu ist eine Mitteilung an den Prozeß erforderlich.

Synchron: Der Prozeß blockiert beim Aufruf der Kommunikationsoperation, sofern keine Nachricht vorliegt. Nach erfolgter Kommunikation wird er an der Aufrufstelle fortgesetzt. Auch hier ist eine Mitteilung erforderlich.

Anmerkung. *Durch eine versuchende Kommunikationsoperation wird keine Bereitschaft im Kanal hinterlegt. Die Kombination von versuchendem Senden und versuchendem Empfangen ist deshalb wirklungslos, da keine Kommunikation zustande kommen kann.*

Der COSY-Kern unterstützt nicht alle Sende- und Empfangssemantiken. Die Implementierung ermöglicht es jedoch, problemlos weitere Sende- und Empfangssemantiken hinzuzufügen, die mit beliebigen komplementären Semantiken zu verwenden sind. Bei Bedarf kann somit auf alle Möglichkeiten zurückgegriffen werden. Zur Verfügung stehen das synchrone und versuchende Empfangen sowie synchrones und asynchrones Senden. COSY bietet zwei Varianten des asynchronen Sendens, deren Verhalten sich unterscheidet, wenn die Nachricht nicht im Kanal abgelegt werden kann. Bei einer Variante geht die Nachricht verloren. Die andere blockiert den sendenden Prozeß, bis die Nachricht im Kanal abgelegt ist oder aufgrund einer initiierten Kommunikation von einem Empfänger angefordert wurde. In beiden Fällen ist garantiert, daß der Speicherbereich des Senders, in dem die Nachricht hinterlegt war, nach Beendigung der Sendeoperation verändert werden kann, ohne die versendete Nachricht zu verfälschen. Zusätzlich existiert eine prototypische Implementierung des umlenkenden Empfangens.

Jeder COSY-Kanal existiert innerhalb eines Adreßraums. Dies schränkt jedoch die Benutzung des Kanals nicht ein, sowohl Senden als auch Empfangen ist über Adreßraumgrenzen hinweg möglich. Des weiteren kann ein Kanal, im Gegensatz zu einem Prozeß, in einen anderen Adreßraum verlegt werden. Die Bezeichnung *Kanal* steht in der Systemarchitektur für Kommunikationsobjekte *zwischen* Instanzen und ist in diesem Fall korrekt gewählt, da COSY-Kanäle nicht im Adreßraum eines Senders oder Empfängers liegen müssen.

Anmerkung. *Daß* COSY*-Kanäle in einem Adreßraum existieren erleichtert die Beseitigung von Programmen nach deren Ausführung. Kanäle werden meist in der Form von* Eingangstoren *verwendet. Sie befinden sich dann im Adreßraum des (einzigen) Empfängers, was eine einfache Ermittlung der zu beseitigenden Kanäle ermöglicht.*

2.1.4 Koordination

Koordination bezeichnet die zeitliche Abstimmung von Prozessen ohne zusätzlichen Informationsaustausch. Der vom COSY-Kern zu diesem Zweck unterstützte Kernobjekttyp ist die *Zählsperre*. Zählsperren sind eine Verallgemeinerung der von E. W. Dijkstra eingeführten *Semaphore*[2] [GeMc88].

Eine Zählsperre ist eine Variable, die Werte aus den natürlichen Zahlen annehmen kann. Auf dieser sind zwei Operationen definiert, **Erhöhen** und **Vermindern**. Ist der Wert einer Zählsperre kleiner als der Wert, um den sie vermindert

[2] Singular: der, das Semaphor; Plural: die Semaphore [Duden]

werden soll, so blockiert diese Operation. Die Deblockierung erfolgt, sobald die Zählsperre durch andere Prozesse soweit erhöht wurde, daß eine Verminderung ohne Verlassen des zulässigen Wertebereichs möglich ist. Der initiale Wert einer Zählsperre wird bei der Erzeugung angegeben.

Anmerkung. *Die von Dijkstra eingeführten Operationen* **P** *(*Passieren*) und* **V** *(*Verlassen*) entsprechen einem Vermindern beziehungsweise Erhöhen um den Wert 1. Der Begriff Semaphor stammt aus dem Bereich des Schienenverkehrs. Eingleisige Abschnitte, die in beiden Richtungen benutzt werden, sind durch ein Semaphor geschützt. Der Wert 0 (Semaphor gesetzt) zeigt an, daß sich ein Zug auf dem Gleisabschnitt befindet. Später ankommende Züge müssen warten, bis der Gleisabschnitt wieder frei ist. In der Informatik wird dieser häufig auftretende Fall als* Binärsemaphor *bezeichnet, da nur die Werte 0 und 1 auftreten. Binärsemaphore dienen dem Schutz kritischer Bereiche von Programmen.*

Das Erhöhen einer Zählsperre ist ein asynchrones Signalisieren oder Entsperren. Das Erniedrigen ist ein synchrones Warten oder Sperren. Prinzipiell können auch bei Zählsperren Operationen mit allen für Kanäle vorgestellten Semantiken angeboten werden.

Ebenso wie Kanäle existieren auch die COSY-Zählsperren innerhalb von Adreßräumen, ohne daß dies die Benutzung einschränkt. Auch Zählsperren können in einen anderen Adreßraum verlegt werden. Kanäle und Zählsperren sind die einzigen COSY-Kernobjekte mit dieser Eigenschaft.

2.1.5 Zugang

Anwendungen und Systeminstanzen für COSY sind, ebenso wie der überwiegende Teil des Kerns, in einem C-Dialekt implementiert, der weitgehend dem internationalen Standard [ISO90] entspricht. Die Kernaufrufe stehen in Form einer *Bibliothek* von Prozeduren zur Verfügung. In dieser Bibliothek befinden sich allerdings nicht die eigentlichen Prozeduren sondern *Aufrufstummel*. Ein Aufrufstummel ist eine Prozedur, in der die übergebenen Argumente gebündelt über eine festgelegte Schnittstelle an eine andere Instanz, in diesem Fall den Kern, übergeben werden. Diese Instanz führt die eigentliche Prozedur aus und übergibt deren Resultate, wiederum gebündelt, an den Aufrufstummel. Der aufrufende Prozeß erhält die Resultate vom Aufrufstummel.

Der Einsatz von Aufrufstummeln ist in mehrfacher Hinsicht sinnvoll. Zum einen ist die Funktionalität der Prozeduren nicht in den Programmen enthalten, sondern davon abgesetzt. Die Programme können deshalb auch nach Änderungen am Kern unverändert ausgeführt werden, wenn das funktionale Verhalten der abgesetzten Prozeduren gleich bleibt. Zum anderen sind Aufrufstummel im allgemeinen wesentlich kürzer als die auszuführenden Prozeduren, was den Speicherbedarf reduziert.

Der wichtigste Punkt ist jedoch die Betriebssicherheit des Kerns. Im Zuge von Manipulationen sind Kerndaten inkonsistent. Bis zum Abschluß einer Manipulation darf nur die manipulierende Instanz Zugriff auf die zu verändernden Daten erhalten. Die von allen Aufrufstummeln benutzte, einheitliche Schnittstelle zu den Kernprozeduren erlaubt eine effiziente Kontrolle des Zugriffs auf Kerndaten. Die Konsistenz der Kerndaten bei jedem Zugriff kann mit einfachen Mechanismen, zum Beispiel durch wechselseitigen Ausschluß von Kernaufrufen, sichergestellt werden.

Beispiel. *Angenommen, der Zugriff auf die Kerndaten sei nicht abgesichert. Ein Prozeß könnte die Kerndaten verändern, um einen neuen Prozeß zu erzeugen. Wenn im Verlauf der Manipulation ein Prozeßwechsel durchzuführen ist, bestimmt der Kern die Menge der sich bewerbenden Prozesse. Diese befindet sich jedoch in einem inkonsistenten Zustand. Ein sicherer Betrieb des Systems wäre nicht möglich.*

Die vom COSY-Kern angebotenen Prozeduren lassen sich anhand ihrer Funktionalität in die folgenden Gruppen einteilen:

1. Das Erzeugen und Beseitigen von Kernobjekten.
2. Das Bestimmen von Bezugsobjekten.
3. Das Senden und Empfangen von Nachrichten an Kanälen.
4. Das Erhöhen und Vermindern von Zählsperren.
5. Das Lesen und Schreiben von Attributen der Kernobjekte und des Kerns.
6. Das Blockieren bis zu einem gegebenen Zeitpunkt.
7. Die Kommunikation mit Nachbarknoten. (siehe Abschnitt 2.2)

Die den ersten fünf Gruppen zuzuordnenden Prozeduren arbeiten mit Kernobjekten. Für jedes Kernobjekt existiert ein Identifikator, die *Kernobjektkennung*. Der Aufbau einer Kernobjektkennung ist unabhängig vom Typ des Kernobjektes. Um eine einheitliche Schnittstelle zur Kernbibliothek zu gewährleisten, existiert ein Pseudo-Objekt *Kern* mit festgelegtem Identifikator. Dieses dient als *Behälter* für Adreßräume und Adreßbereiche, so wie ein Adreßraum als Behälter für Prozesse, Einblendungen, Kanäle und Zählsperren fungiert. Bei allen Aufrufen zum Erzeugen von Objekten muß ein Behälterobjekt angegeben werden. Der Kernidentifikator ermöglicht es zudem, auf Attribute des Kerns, etwa die Systemzeit, über die gleichen Prozeduren zuzugreifen, die auch Attribute von echten Kernobjekten auslesen oder setzen.

Abbildung 2.1 zeigt die resultierenden Beziehungen zwischen den Kernobjekttypen. Pfeile symbolisieren $1:n$–Beziehungen zwischen Ausgangs- und Endtyp. Sie können als „enthält“ gelesen werden. Die Prozeduren zum Bestimmen

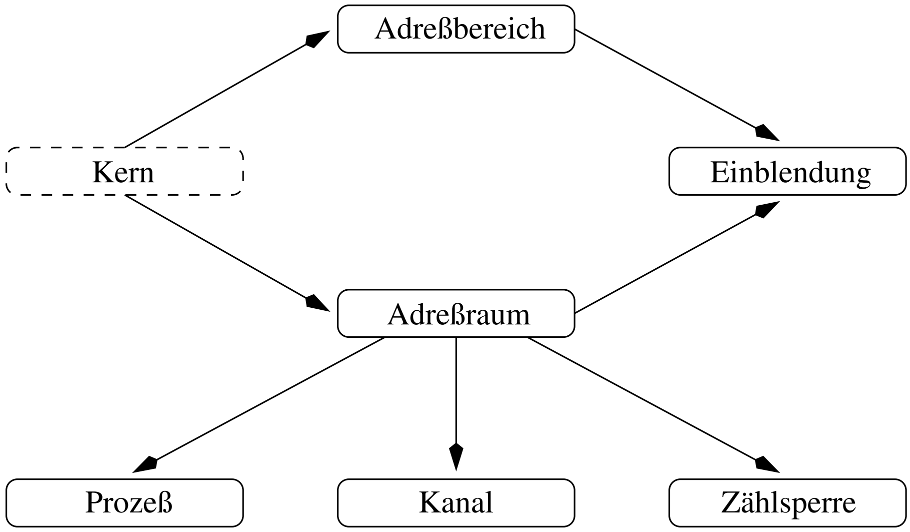

Abbildung 2.1: COSY-Kernobjekttypen

von Bezugsobjekten (Gruppe 2) erlauben es, Partnerobjekte im Sinne dieser Beziehungen zu ermitteln, zum Beispiel den Adreßbereich, auf den über eine bestimmte Einblendung zugegriffen wird, oder alle Prozesse, die sich in einem gegebenen Adreßraum befinden.

2.2 Verbund

COSY ist für den Einsatz auf homogenen, nachrichtengekoppelten Parallelrechnern konzipiert. Diese bestehen aus einer Anzahl von *Knoten* mit lokalem Speicher und mindestens einem Prozessor. Jeder Knoten ist über Kommunikationsleitungen mit Nachbarknoten verbunden. Auf allen Knoten befinden sich Prozessoren gleichen Typs. Meist ist auch die Anzahl der Prozessoren und die Größe des lokalen Speichers auf allen Knoten identisch. Die Kommunikationsleitungen zwischen den Knoten bilden einen zusammenhängenden Graphen.
Während des Betriebs ist auf jedem Knoten ein lokaler Kern installiert. Prozesse werden auf verschiedenen Knoten eingerichtet und interagieren, um die Rechenleistung der einzelnen Knoten zur Lösung eines gemeinsamen Problems einzusetzen. Die Interaktion über Knotengrenzen hinweg geschieht durch das Versenden und Empfangen von Datenpaketen. Auf diese Art wird aus der Ansammlung einzelner Knoten ein *Verbund*. Abhängig davon, ob dieser Verbund im Instanzen- oder Infrastrukturbereich hergestellt wird, handelt es sich um einen *Instanzen-* oder einen *Kernverbund* [Wett94].

Bei einem Instanzenverbund wird der Zugang zu den Kommunikationsleitungen durch Dienstgeber realisiert. Abbildung 2.2 zeigt eine vereinfachte Darstellung. Die Leitungen werden über Kommunikationshardware angesteuert. Diese Hardware kann im Speicher abgelegte Pakete zu anderen Knoten transferieren und Pakete von anderen Knoten im Speicher ablegen. Sie greift dabei unmittelbar

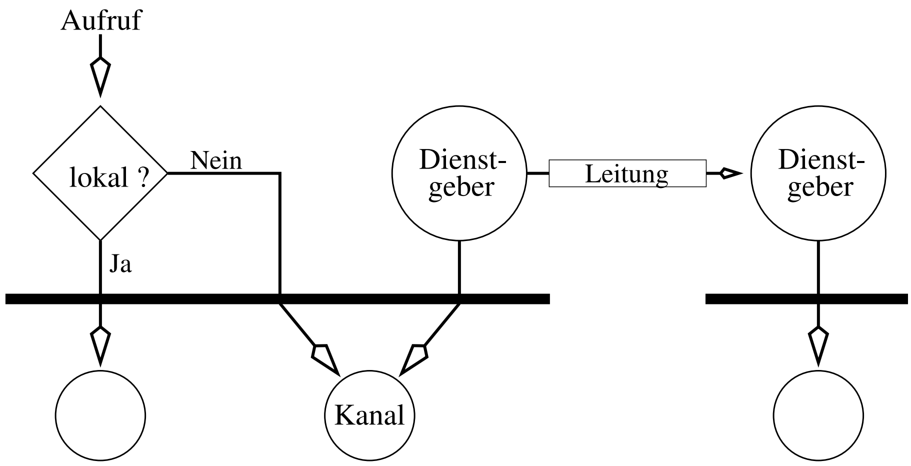

Abbildung 2.2: Instanzenverbund

auf den physikalischen Speicher zu. Ein Dienstgeber-Prozeß hingegen arbeitet in einem Adreßraum und greift unter Umständen nur indirekt über einen Adreßtransformationsmechanismus[3] auf den physikalischen Speicher zu. Die Interaktion von Dienstgeber und Kommunikationshardware erfolgt über *Adreßraumausbrüche*, die in den Zuständigkeitsbereich des Kerns fallen. Eine Kommunikation mit geringem Rechenzeitaufwand erfordert eine Umlenkung oder Deblockierung des Dienstgeber-Prozesses durch Signale der Hardware. Auch dies kann nur mit Unterstützung des Kerns geschehen.
Die Realisierung eines Instanzenverbundes erfordert also eine Erweiterung des Kerns um die beschriebene Funktionalität. Die Nutzung der knotenübergreifenden Interaktionsmöglichkeiten erfolgt über lokale Kommunikation mit den Dienstgebern. Der zusätzliche Protokollaufwand erhöht die Laufzeit von Nachrichten. Ein Instanzenverbund eignet sich besonders dann, wenn diese Erhöhung keine entscheidende Rolle spielt.

Beispiel. *Eine prozentual kleine Erhöhung der Paketlaufzeit ergibt sich bei der Übertragung großer Pakete über langsame Leitungen, zum Beispiel in verteilten Dateisystemen. Auch in Netzen mit hohen Latenzzeiten, etwa dem Internet, ist der Einsatz von Instanzenverbünden sinnvoll.*

In einem Kernverbund werden Kernprozeduren auch dann lokal aufgerufen, wenn sie sich auf Objekte entfernter Kerne beziehen. Abbildung 2.3 zeigt den Ablauf. Der lokale Kern leitet den Aufruf zum Kern auf dem entfernten Knoten weiter, wo er ausgeführt wird. Knotenübergreifende Kommunikation und Koordination erfolgt ebenso wie im lokalen Fall direkt über Kernobjekte auf entfernten Knoten. Ein Kernverbund ermöglich es, die Unterstützung der Kommunikationshardware vollständig im Kern zu kapseln. Ebenso wie die Kommunikationshardware kann der Kern unter Umgehung eines Adreßtransformationsmechanismus auf den physikalischen Speicher zugreifen. Das Deblockieren oder

[3] engl.: memory management unit, MMU

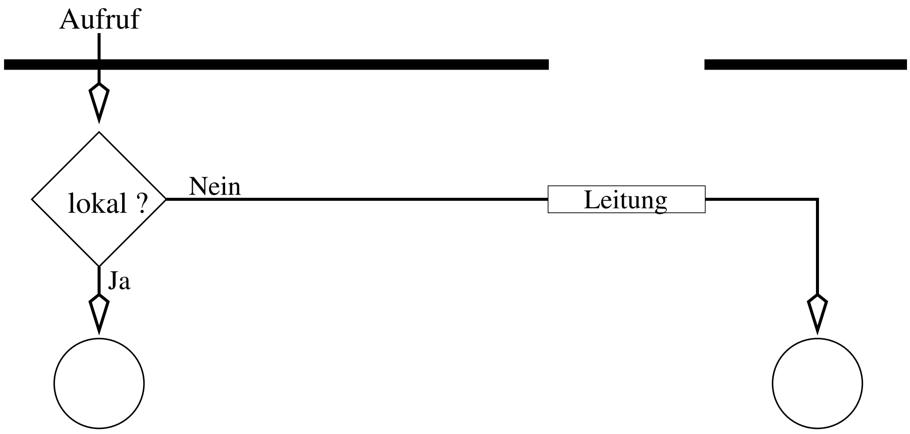

Abbildung 2.3: Kernverbund

Umlenken von Prozessen durch Hardwaresignale erfolgt innerhalb des Kerns. Bei einem Instanzenverbund muß diese Funktionalität an der Kernschnittstelle zur Verfügung stehen. Entfernte Kernaufrufe können ohne lokale Kommunikation mit Dienstgebern ausgeführt werden. Erfordert die eingesetzte Kommunikationshardware allerdings die Abwicklung komplexer Protokolle, etwa zur Fehlererkennung und -korrektur, so steht ein Kernverbund in Konflikt mit dem Prinzip der Mikrokernarchitektur.

Eine schnelle Interaktion zwischen Prozessen ist von entscheidender Bedeutung für den effizienten Einsatz von Parallelrechnern. Um die Rechenleistung eines hochparallelen Rechners mit eventuell hunderten von Knoten zu nutzen, müssen Aufgaben in eine große Anzahl von Teilaufgaben zerlegt werden. Jede Teilaufgabe besteht aus einem Teil der gesamten Daten und einem Prozeß, der diese Daten bearbeitet. Die Teilaufgaben können jedoch nicht unabhängig voneinander sein, da sie alle zur Lösung der Gesamtaufgabe beitragen. Die Interaktion von Prozessen ist somit unumgänglich. Mit steigender Zahl an Prozessen steigt auch die Zahl der Interaktionen, während der Umfang der pro Prozeß zu bearbeitenden Daten abnimmt, sofern der Umfang der Gesamtaufgabe gleich bleibt. Als Folge davon muß zur Durchführung der Interaktionen eine Vielzahl kurzer Datenpakete zu anderen Knoten transportiert werden. Der zusätzliche Protokollaufwand eines Instanzenverbundes würde die Geschwindigkeit der knotenübergreifenden Interaktion erheblich beeinträchtigen. Aus diesem Grund ist Cosy als Kernverbund realisiert.

2.2.1 Betrieb

Ein zentrales Problem beim Betrieb eines Verbundsystems ist die *Orientierung*. Im Fall des COSY-Kernverbundes muß bei Aufruf einer Kernprozedur der Zielknoten des Aufrufs bestimmt werden. Zur Identifikation der Knoten sind diese durchnumeriert.
Der überwiegende Teil der Kernaufrufe erwartet mindestens ein Kernobjekt als Argument. Kernobjekte befinden sich immer auf dem Knoten, auf dem sie erzeugt wurden. Der COSY-Kern unterstützt keine Beziehungen zwischen Objekten auf verschiedenen Knoten. Aufgrund dieser Gegebenheiten bietet sich die Orientierung über *Ortspräfixe* an. Das bedeutet, daß die Knotennummer eines Kernobjektes einen Teil der Kernobjektkennung bildet. Kernobjektkennungen in COSY sind Tripel der Form:

$$< \text{Knotennummer, Objektnummer, Kontrollwert} >$$

Die Objektnummer ist eine lokal eindeutige Kennung in der Menge der existierenden Kernobjekte eines Knotens. Nach dem Löschen eines Objektes kann die freigewordene Objektnummer wieder vergeben werden. Der Kontrollwert unterscheidet die Kennungen existierender Objekte von denen gelöschter Objekte mit gleichen Objektnummern. Für die Pseudo-Objekte des Typs *Kern* sind Objektnummer und Kontrollwert festgelegt. Die Kennung des Kerns eines bestimmten Knoten kann also mit Kenntnis der Knotennummer konstruiert werden.

Muß ein Kernaufruf auf einem entfernten Knoten ausgeführt werden, so erstellt der Kern auf dem Knoten des Aufrufers ein Auftragspaket. Dieses enthält den Zielknoten, eine Kennung der auszuführenden Prozedur, die Argumente für diese Prozedur und die Kennung des aufrufenden Prozesses. Das Auftragspaket muß nun zum Zielknoten transportiert werden. Dazu wird es über eine Kommunikationsleitung zu einem Nachbarknoten geschickt, der den Weitertransport übernimmt. Es stellt sich das Problem der *Wegwahl*, also die Frage:

> Zu welchem Nachbarknoten soll ein Paket mit einem gegebenen Zielknoten geschickt werden ?

In COSY erfolgt die Wegwahl deterministisch. Auf jedem Knoten existiert eine *Wegwahltabelle*, in der für jeden Zielknoten die Leitung vermerkt ist, über die Pakete zu transportieren sind. Die Berechnung der Wegwahltabelle basiert auf der *Knotenkonfiguration*, die auf jedem Knoten zur Verfügung steht. Die Knotenkonfiguration ist ein zusammenhängender Graph, in dem jeder Knoten einen Knoten des Parallelrechners repräsentiert. Die Kanten des Graphen entsprechen den vorhandenen Leitungen. Die Knotenkonfiguration muß bereits vor dem Laden des Betriebssystems bekannt sein (siehe 2.2.2).

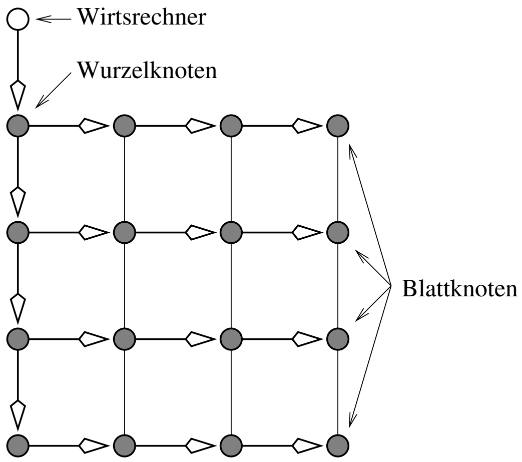

Abbildung 2.4: Inbetriebnahme eines Parallelrechners

2.2.2 Inbetriebnahme

Um unter Cosy auf einem Parallelrechner arbeiten zu können, muß auf jedem Knoten ein lokaler Kern installiert werden. Zusätzliche Funktionalität wird von Systemdienstgebern erbracht (siehe Abschnitt 2.3). Auch die Systemdienstgeber sowie die Knotenkonfiguration müssen auf allen Knoten verfügbar sein. In einer Initialisierungsphase werden diese Informationen über die Knoten des Parallelrechners verteilt.

Ausgangspunkt der Inbetriebnahme ist zur Zeit ein *Wirtsrechner*. Auf diesem ist ein Betriebssystem vorhanden. Der Wirtsrechner ist über eine Kommunikationsleitung mit einem Knoten des Parallelrechners, dem *Wurzelknoten*, verbunden. Kern, Dienstgeber und die Knotenkonfiguration liegen als Dateien vor, die von einem Programm auf dem Wirtsrechner ausgelesen werden können.
Auf jedem Knoten kann, ausgehend von einem Nachbarknoten beziehungsweise dem Wirtsrechner, ein erstes Programm gestartet werden. Dieses muß ohne ein Betriebssystem ablaufen. Als Aufrufargument erhält es die Leitung, über die es gestartet wurde. Unter Umständen existieren Beschränkungen, zum Beispiel eine maximale Länge des Programms oder eine festgelegte Startadresse. In diesen Fällen wird ein *Lader* verwendet, ansonsten kann unmittelbar der Kern gestartet werden. Das Starten des ersten Programms ist die Grundlage für den Einsatz eines Parallelrechners, es muß vom Hersteller unterstützt werden.

Zur Inbetriebnahme wird, ausgehend vom Wurzelknoten, ein spannender Baum in die Knotenkonfiguration eingebettet. Jeder Knoten wird von seinem Vorgänger in diesem Baum in Betrieb genommen. Eine Ausnahme bildet der Wurzelknoten, der vom Wirtsrechner in Betrieb genommen wird.
Abbildung 2.4 zeigt als Beispiel eine Knotenkonfiguration mit 16 Knoten, die in einem zweidimensionalen Gitter verschaltet sind. Der eingebettete, spannende Baum ist durch Pfeile eingezeichnet, die von einem Vaterknoten zu einem Kindknoten zeigen. Leitungen, die während der Inbetriebnahme nicht genutzt werden, sind durch einfache Striche dargestellt.

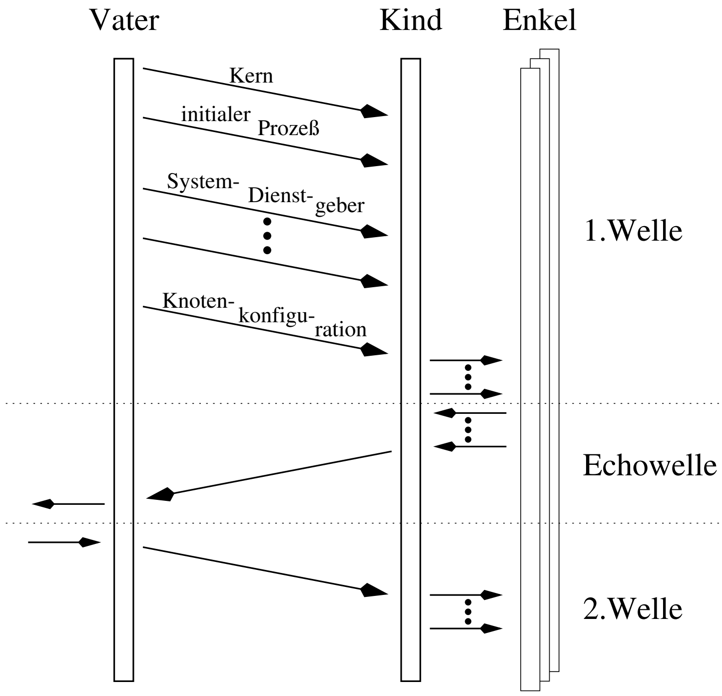

Abbildung 2.5: Inbetriebnahme eines Knotens

Der Ablauf der Inbetriebnahme eines Knotens ist in Abbildung 2.5 dargestellt. Zunächst wird der COSY-Kern, bei Bedarf mit Hilfe eines Laders, auf den Kindknoten übertragen. Der Kern erhält als Aufrufargument die Leitung, über die er mit dem Vaterknoten kommunizieren kann. Nachdem er seine internen Daten initialisiert hat, lädt er ein Programm vom Vaterknoten und erzeugt den *initialen Prozeß*, der es ausführt. Wiederum ist die Leitung zum Vaterknoten ein Aufrufargument.
Der initiale Prozeß kann den COSY-Kern lokal nutzen. Da er ebenfalls mit dem Vaterknoten kommunizieren muß, benötigt er Zugriff auf die Kommunikationsleitung. Dazu kann er mit Sonderrechten ausgestattet werden, die ihm den direkten Zugriff auf die Kommunikationshardware und die Umgehung eines Adreßtransformationsmechanismus erlauben. Alternativ dazu kann auch für die Dauer der Inbetriebnahme eine Möglichkeit zur Kommunikation mit Nachbarknoten an der Kernschnittstelle angeboten werden. Letzteres ist die in COSY verwendete Variante (siehe Abschnitt 2.1.5).

Der initiale Prozeß lädt nun alle Systemdienstgeber und die Knotenkonfiguration auf den lokalen Knoten. Die Systemdienstgeber initialisieren sich, nehmen aber ihre Arbeit noch nicht auf. Aus der Knotenkonfiguration bestimmt der initiale Prozeß die Menge der Kindknoten (Enkel, aus Sicht des Vaterknotens) durch eine Breitensuche. Daraufhin wird der lokale Knoten zum Vaterknoten und überträgt die bisher empfangenen Daten zu seinen Kindknoten. In Abbildung 2.5 ist dies durch Pfeile mit weißen Spitzen dargestellt. Die Initialisierung der Knoten erfolgt durch dieses Vorgehen in Form einer *Welle* durch den Parallelrechner.

Auf einem *Blattknoten*, also einem Knoten ohne Kind, wird das Auslaufen der Welle durch eine Bestätigung zum Vaterknoten gemeldet. Knoten mit Kindern sammeln erst alle Bestätigungen der Kinder auf, bevor sie ihre Bestätigung weitergeben. Es entsteht eine *Echowelle*, die auf dem Wurzelknoten verebbt. Die Echowelle kann benutzt werden, um Informationen zu sammeln, die im Verlauf der Initialisierungswelle entstanden sind.
Sobald die Echowelle auf dem Wurzelknoten eintrifft ist sichergestellt, daß auf jedem Knoten ein Kern und die notwendigen Dienstgeber zur Verfügung stehen. Ausgehend vom Wurzelknoten wird nun in einer zweiten Welle der Verbund hergestellt. Ein Knoten, der die Benachrichtigung für die zweite Welle erhält, gibt diese zuerst an seine Kinder weiter. Danach übergibt der initiale Prozeß auf diesem Knoten die Knotenkonfiguration an den lokalen Kern. Der Kern berechnet daraus die Wegwahltabelle. Ab diesem Zeitpunkt werden die Kommunikationsleitungen dieses Knotens für den Kernverbund benötigt. Ein direkter Zugriff auf die Leitungen ist also nicht mehr möglich. Auch die zuvor initialisierten Dienstgeber nehmen ihre Arbeit auf.

Die Aufgabe der initialen Prozesse, nämlich die Inbetriebnahme des COSY-Betriebssystems, ist mit dem Auslaufen der zweiten Welle erledigt. Die Beseitigung dieser Prozesse und Freigabe der belegten Resourcen kann von den nun nutzbaren Dienstgebern erledigt werden. Ebenso kann dies in einer zweiten Echowelle geschehen. Dies erfordert jedoch eine Sonderbehandlung des Wurzelknotens, da dort kein Vaterknoten existiert, von dem aus der initiale Prozeß beseitigt werden kann. Außerdem muß eine zweite Echowelle durch Interaktion mit COSY-Kernobjekten realisiert werden, da der direkte Zugriff auf die Leitungen zu den Vaterknoten nicht mehr möglich ist.

2.3 Instanzen

Von einem Betriebssystem erwartet der Benutzer mehr Möglichkeiten als ein Mikrokern anbieten kann. Der Zugriff auf Dateien, das Starten von Programmen und die Interaktion mit dem Anwender sind in einem vollwertigen Betriebssystem selbstverständlich. Prozeduren für diese Zwecke werden in COSY, ebenso wie die Kernaufrufe, in einer Bibliothek angeboten. Um den Umfang der Bibliotheksprozeduren klein zu halten, stehen Dienstgeber zur Verfügung. In der Bibliothek befinden sich, wie schon bei der Kernschnittstelle, Aufrufstummel, die die Nutzung der Dienstgeber ermöglichen. In den folgenden Abschnitten werden einige grundlegende COSY-Systemdienstgeber beschrieben.

2.3.1 Katalogdienst

Die Interaktion zwischen Instanzen beruht auf der gemeinsamen Nutzung von Kernobjekten. Dazu muß jeder an einer Interaktion beteiligten Instanz die Kernobjektkennung des gemeinsam benutzten Objektes bekannt sein. Da Kernobjektkennungen dynamisch vergeben werden, müssen sie den interagierenden Instanzen zur Laufzeit mitgeteilt werden. Dieses Problem tritt insbesondere dann auf, wenn Instanzen unabhängig voneinander entstehen. Zum Beispiel erzeugt ein Systemdienstgeber seinen Auftragskanal während der Inbetriebnahme. Ein Programm, das einen Dienstgeber benutzt, muß die Kennung des Auftragskanals während der Ausführung bestimmen. Zu diesem Zweck steht ein Katalogdienst [Matt91] zur Verfügung.

Einträge im Katalog sind Tupel aus einem *Namen* und einem *Wert*. In COSY bestehen Namen aus einer Zeichenkette, der ein Ortspräfix vorangestellt sein kann. Als Werte unterstützt der COSY-Katalogdienst zur Zeit nur Kernobjektkennungen. Der COSY-Katalog ist segmentiert, jeder Teilkatalog wird von einer *Kataloginstanz* verwaltet. Auf jedem Knoten befindet sich eine lokale Kataloginstanz. Zudem existiert ein systemweites, globales Katalogsegment. Dieses kann auf mehreren Knoten repliziert sein. Der Zugriff auf Einträge im globalen Katalogsegment erfolgt analog zu dem auf Einträge in den lokalen Segmenten.
Namen mit Ortspräfix werden in der Kataloginstanz auf dem vom Ortspräfix angegebenen Knoten nachgeschlagen. Namen ohne Ortspräfix werden zuerst in der Kataloginstanz auf dem Knoten des Aufrufers nachgeschlagen. Falls dort kein entsprechender Eintrag vorhanden ist, erfolgt eine Suche im globalen Katalogsegment.

Im Gegensatz zu den dynamisch erzeugten Kernobjektkennungen werden die im Katalogdienst verwendeten Namen *von Hand* vergeben. Für jeden Systemdienstgeber ist ein Name festgelegt, unter dem der Auftragskanal nachgeschlagen werden kann. Im Verlauf der Inbetriebnahme hinterlegen alle Systemdienstgeber die Kennung ihres Auftragskanals im Katalog. Dienstgeber, deren Aufgabenbereich auf einen Teil der Knoten beschränkt ist, tragen ihren Auftragskanal in den Kataloginstanzen dieser Knoten ein. Auftragskanäle globaler Dienstgeber sind im globalen Katalogsegment verzeichnet.

Auch Kataloginstanzen sind Dienstgeber mit einem dynamisch erzeugten Auftragskanal. Die Auftragskanäle von Kataloginstanzen werden in drei Situationen benötigt:

1. Systemdienstgeber müssen ihren Auftragskanal im Katalog eintragen.
2. Kataloginstanzen müssen zum Nachschlagen von Namen auf andere Kataloginstanzen zugreifen, wenn Namen einen Ortspräfix enthalten oder im globalen Katalogsegment nachzuschlagen sind.
3. Vom Anwender ausgeführte Programme müssen die Auftragskanäle von Systemdienstgebern im Katalog nachschlagen.

Die Verbreitung der Auftragskanalkennungen der Kataloginstanzen für die beiden ersten Situationen erfolgt im Rahmen der Inbetriebnahme (siehe 2.2.2). Während der ersten Welle initialisieren sich die Kataloginstanzen. Sie teilen dem jeweiligen initialen Prozeß die Kennung des neu erzeugten Auftragskanals mit, der sie an die anderen lokalen Dienstgeber übermittelt. Die Kennungen werden in der ersten Echowelle aufgesammelt und mit der zweiten Welle an alle Kataloginstanzen verteilt. Die nach der Inbetriebnahme erzeugten Anwender-Prozesse erhalten die Kennung des Auftragskanals der lokalen Kataloginstanz von der Programmverwaltung (siehe 2.3.3) als Aufrufargument. Damit ist auch die dritte Situation abgedeckt.

2.3.2 Gruppenkommunikation

Der Cosy-Kern unterstützt nur Kanäle zur Einzel:Einzel–Kommunikation, also zur Weitergabe einer Nachricht von *einem* Sender- zu *einem* Empfänger-Prozeß. Bei der Entwicklung paralleler Programme erweist es sich jedoch hin und wieder als notwendig, die gleiche Nachricht an mehrere Prozesse zu versenden, also eine Einzel:Und–Kommunikation durchzuführen. Auch der Fall, daß mehrere Sender Teilnachrichten schicken, die *zusammen* eine Nachricht an einen Empfänger bilden, ist denkbar. Dies entspricht einer Und:Einzel–Kommunikation.

Eine mögliche Realisierung der Einzel:Und–Kommunikation ist das sequentielle Verschicken der Nachricht vom Sender zu allen Empfängern. Dabei muß die Nachricht jeweils vom Knoten des Senders zum Knoten des Empfängers transportiert werden. In vielen Fällen verringert sich jedoch die Zahl der Transportoperationen zu Nachbarknoten, wenn die Nachricht nur *einmal* zu Zwischenknoten transportiert und dort von *Verteilern* an mehrere Empfänger oder andere Zwischenknoten weitergegeben wird.

Beispiel. *Abbildung 2.6 zeigt eine Situation, in der eine Nachricht von einem Sender (S) zu vier Empfängern (E) verschickt werden soll. Die Kreuzungspunkte des Gitters stehen für die Knoten eines Parallelrechners, die Linien für Kommunikationsleitungen. Beim sequentiellen Versenden muß die Nachricht acht*

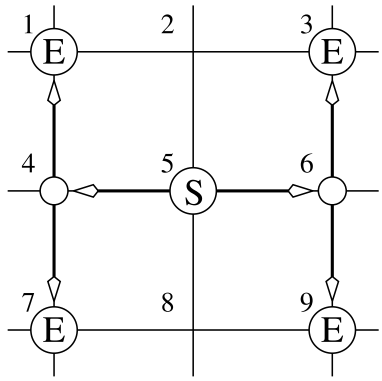

Abbildung 2.6: Einzel:Und–Kommunikation

mal über eine Kommunikationsleitung transportiert werden, dabei je zweimal über die Leitungen vom Knoten 5 zu den Knoten 4 und 6. Durch Verteiler auf den Knoten 4 und 6 reduziert sich diese Zahl auf sechs Transportvorgänge. Befinden sich auf den Knoten 1, 3, 7 oder 9 mehrere Empfänger, so erhöht sich die Zahl der Transportvorgänge im sequentiellen Fall um zwei pro zusätzlichem Empfänger. Mit zusätzlichen Verteilern auf den Knoten mit mehreren Empfängern bleibt sie dagegen konstant.

Bei der Und:Einzel–Kommunikation werden die Teilnachrichten der Sender aufgesammelt und kombiniert. Die Kombination kann auf verschiedene Arten erfolgen. In COSY werden Teilnachrichten gleicher Größe *verknüpft*. Die Verknüpfungsoperation ist so gewählt, daß als Ergebnis eine Nachricht entsteht, die ebenso groß ist wie jede der verknüpften Teilnachrichten. Zudem muß sie assoziativ und kommutativ sein, damit sie bereits auf den Zwischenknoten angewendet werden kann, ohne das Resultat zu verändern. Bei dieser Realisierung von Und:Einzel–Kanälen ergibt sich ein Gewinn nicht nur durch die verringerte Zahl der Transportoperationen, sondern auch durch die Parallelisierung des mit der Verknüpfung verbundenen Rechenaufwands.

Anmerkung. *Auch andere Arten von Und:Einzel–Kanälen sind denkbar. So könnten Nachrichten* zusammengesetzt *werden. Dabei steigt zwar die Größe der transportierten Nachrichten, aber die Anzahl der Transportoperationen verringert sich. Ein Gewinn ergibt sich also durch den geringeren Protokollaufwand. In* COSY *steht diese Art von Und:Einzel–Kanälen bisher jedoch nicht zur Verfügung.*

Gruppenkommunikation wird in COSY durch lokale Dienstgeber ermöglicht. Nach Angabe einer Menge von Sendern oder Empfängern berechnet der Dienstgeber auf dem Knoten des Aufrufers einen in die Knotenkonfiguration eingebetteten Baum zu den Knoten der Kommunikationspartner. Die Dienstgeber auf Knoten, die Verzweigungen des Baums entsprechen, werden benachrichtigt und erzeugen je einen Unterprozeß, der Nachrichten entweder verteilt oder aufsammelt und verknüpft. Als Verknüpfungsoperationen stehen die Minimum-, Maximum- und Summenbildung mit den aus C bekannten elementaren Datentypen zur Verfügung. Datentyp und Verknüpfungsoperation werden bei der Erzeugung eines Und:Einzel–Kanals angegeben. Die über den Kanal versendeten Nachrichten enthalten Felder mit Elementen des angegebenen Typs, die elementweise verknüpft werden.

Das Senden oder Empfangen an einem Kanal zur Gruppenkommunikation erfolgt über die vom Cosy-Kern angebotenen Kanäle. Kommunikationspartner eines Senders oder Empfängers ist ein vom Gruppenkommunikationsdienstgeber erzeugter Prozeß, der Nachrichten verknüpft beziehungsweise vervielfältigt und weiterleitet. Solche zwischengeschalteten Prozesse verhindern den direkten Kontakt zwischen Sender- und Empfänger-Prozessen. Zwar sind weiterhin alle Sendesemantiken (siehe 2.1.3) verwendbar, diese beziehen sich jedoch auf die Kanäle des Kerns. Eine Bestätigung erfolgt also bereits nach dem Empfang durch einen Unterprozeß des Dienstgebers. Am Instanzen-Objekt „Gruppenkanal“ erfolgt das Senden deshalb immer asynchron. Der Sender erhält keine Information darüber, ob ein Empfänger die Nachricht entgegengenommen hat.

2.3.3 Programmverwaltung

Programme für Cosy werden in C formuliert und übersetzt. Das Ergebnis der Übersetzung ist eine *Programmdatei*, die den vom Prozessor auszuführenden Code enthält. Zudem sind in der Programmdatei Informationen darüber abgelegt, wieviel Speicher für Code und Daten anzufordern ist. Die Ausführung eines Programms kann in sieben Schritte gegliedert werden:

1. Bestimmen des Betriebsmittelbedarfs aus der Programmdatei.
2. Einrichten einer Ablaufumgebung mit den benötigten Betriebsmitteln.
3. Übertragen der Aufrufargumente in die Ablaufumgebung.
4. Übertragen des Codes aus der Programmdatei in die Ablaufumgebung.
5. Erzeugen eines Prozesses in der Ablaufumgebung.
6. Abarbeiten des Programms.
7. Freigeben der Ablaufumgebung mit den belegten Betriebsmitteln.

Das vom Benutzer gewünschte Ziel ist die Abarbeitung des Programms in Schritt 6. Zur Bewältigung der notwendigen Verwaltungsaufgaben in den restlichen Schritten steht in Cosy eine integrierte Programm- und Prozeßverwaltung zur Verfügung, die der besseren Lesbarkeit wegen im folgenden nur als *Programmverwaltung* bezeichnet wird. Die Funktionalität der Cosy-Programmverwaltung wird durch die Zusammenarbeit von drei Komponenten erbracht: *Bibliotheksprozeduren*, *Programmverwalter* und *Lader*.

Im Gegensatz zu den bisher vorgestellten Diensten enthalten die Bibliotheksprozeduren der Programmverwaltung nicht nur Aufrufstummel. Das Lesen von Programmdateien und das Verteilen der darin enthaltenen Informationen wird von den Bibliotheksprozeduren übernommen. Dies umfaßt die Schritte 1 und 4. Auch die Übertragung der Aufrufargumente in Schritt 3 ist Aufgabe der Bibliotheksprozeduren.
Programmverwalter sind für das Einrichten und Beseitigen von Ablaufumgebungen zuständig. Wie ihr Name schon andeutet, beschränken sie sich dabei nicht nur auf die Ausführung dieser Tätigkeiten, sondern verwalten auch Informationen über die eingerichteten Ablaufumgebungen, die geladenen Codes und die erzeugten Prozesse. Diese Informationen können bei Bedarf erfragt werden. Bei der Ausführung eines Programms sind die Programmverwalter für die Schritte 2, 5 und 7 zuständig. Programmverwalter sind nicht auf jedem Knoten vorhanden. Statt dessen ist die Menge der Knoten des Parallelrechners *partitioniert.* Für jede Partition existiert genau ein Programmverwalter.
Die Aufgabe eines Laders ist es, Codes und Aufrufargumente entgegenzunehmen und in den Ablaufumgebungen abzulegen. Er unterstützt also die Bibliotheksprozeduren in den Schritten 3 und 4. Auf jedem Knoten ist ein Lader installiert, da seine Aufgaben aus Effizienzgründen lokal ausgeführt werden sollen.

Das Zusammenspiel der drei Komponenten der COSY-Programmverwaltung ist beim *Mehrfachladen* am deutlichsten sichtbar. Beim Mehrfachladen sollen mehrere Instanzen eines Programms gleichzeitig erzeugt werden. Die Instanzen können auf verschiedenen Knoten eingerichtet werden, aber auf jedem Knoten können auch mehrere Instanzen zu liegen kommen. Die Programmdatei sollte dabei nur einmal gelesen werden, da die darin enthaltene Information für jeden zu erzeugenden Prozeß identisch ist und Dateizugriffe zeitaufwendig sind.

Anmerkung. *Nützlich ist das Mehrfachladen vor allem bei der Verwendung des als* SPMD *(engl.: Single Program, Multiple Data) bezeichneten Programmierstils. Dieser wird häufig bei der Parallelisierung von Algorithmen eingesetzt. Ein Beispiel aus dem Bereich der numerischen Anwendungen ist die Matrixmultiplikation. Um ein Element der Resultatsmatrix zu berechnen muß ein Zeilenvektor der ersten Eingabematrix mit einem Spaltenvektor der zweiten multipliziert werden. Die Berechnung ist für jedes Element der Resultatsmatrix identisch* (SP)*, nur die zu multiplizierenden Vektoren unterscheiden sich* (MD).

Zu Beginn eines Mehrfachladevorgangs liest eine der Bibliotheksprozeduren die für eine Ausführung benötigten Informationen über anzufordernde Betriebsmittel aus der Programmdatei (Schritt 1). Für jeden Knoten, auf dem das Programm auszuführen ist, ermittelt die Prozedur den Auftragskanal des zuständigen Programmverwalters durch Nachschlagen im Katalogdienst. Der Verwalter wird beauftragt, auf dem jeweiligen Knoten die vorgegebene Anzahl von Ablaufumgebungen einzurichten.
Ein Programmverwalter, der einen Auftrag zum Einrichten von Ablaufumgebungen erhält, erzeugt auf dem im Auftrag angegebenen Zielknoten für jede

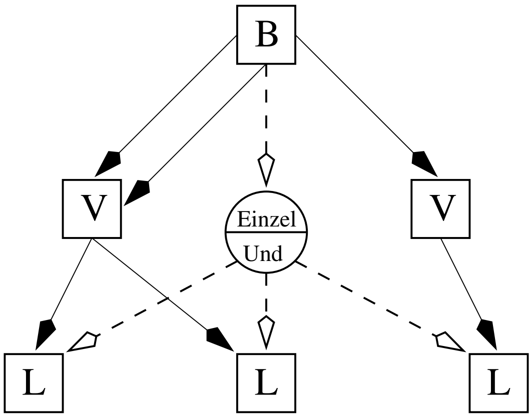

Abbildung 2.7: Mehrfachladen

Ablaufumgebung einen Adreßraum. Für jeden Adreßraum wird ein Daten- und Stapelbereich angefordert. Zudem erzeugt der Verwalter für alle Ablaufumgebungen auf dem Zielknoten einen gemeinsamen Codebereich. In jedem Stapelbereich wird die Auftragskanalkennung der Kataloginstanz des Zielknotens abgelegt. Wenn der verwendete C-Übersetzer es erfordert, können auch weitere Werte, etwa die Anfangsadresse des Datenbereichs, abgelegt werden. Anschließend benachrichtigt der Verwalter den Lader auf dem Zielknoten. Im Adreßraum des Laders wird ein Kanal sowie ein Unterprozeß erzeugt, der an diesem Kanal auf den Code und eventuelle Aufrufargumente wartet. Die Bibliotheksprozedur erhält die Kennung des erzeugten Kanals vom Verwalter. Damit ist Schritt 2 ausgeführt.
Nachdem alle Ablaufumgebungen eingerichtet sind, erzeugt die Bibliotheksprozedur einen Einzel:Und–Kanal, der Nachrichten in die Kanäle der Lader-Unterprozesse ausliefert. In Schritt 3 sind zwei Fälle zu unterscheiden: Sollen für jede Ausführung die gleichen Aufrufargumente verwendet werden, so können diese über den Einzel:Und–Kanal verschickt werden. Die Unterprozesse der Lader legen sie dann in den neu erzeugten Ablaufumgebungen ab. Anderenfalls werden die Aufrufargumente von einer Bibliotheksprozedur direkt in den Stapelbereich der jeweiligen Ablaufumgebung geschrieben.
In Schritt 4 muß nun der Code zu den Zielknoten übertragen werden. Dies geschieht über den Einzel:Und–Kanal, der danach beseitigt wird. Die Verwalter werden von den Ladern benachrichtigt, wenn der Code vollständig übertragen ist. Daraufhin erzeugen sie in jeder Ablaufumgebung einen Prozeß, der das geladene Programm abarbeitet (Schritt 5).

Beispiel. *Abbildung 2.7 zeigt ein Mehrfachladen auf drei Knoten, die von zwei Verwaltern betreut werden. Pfeile mit schwarzen Spitzen stehen für eine Beauftragung. Es ist zu erkennen, daß die Bibliotheksprozedur* (B) *für jeden Zielknoten einen Verwalter* (V) *mit der Einrichtung der Ablaufumgebungen beauftragt. Die Anzahl der Ablaufumgebungen ist im Auftrag vermerkt und aus der Abbildung nicht ersichtlich. Der Verwalter beauftragt den Lader* (L) *mit dem*

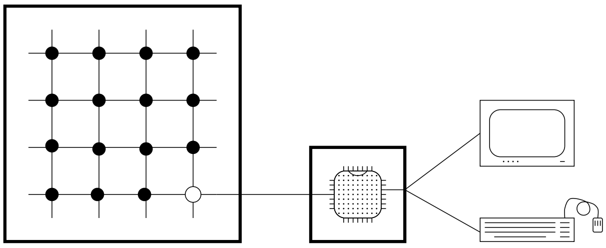

Abbildung 2.8: Wirtsrechner-Anbindung

Empfangen des Codes und eventuell der Aufrufargumente. Das Übertragen des Codes über den erzeugten Einzel:Und–Kanal ist durch die Pfeile mit weißen Spitzen angedeutet.

Das Terminieren eines Programms erfolgt in C durch Ausführen einer *Nachspann-Prozedur*. In dieser Prozedur wird der zuständige Verwalter darüber informiert, daß die Ausführung abgeschlossen ist. Daraufhin löscht er alle im Adreßraum der Ablaufumgebungen enthaltenen Kernobjekte (Schritt 7). Im Adreßraum eingeblendete Adreßbereiche werden nur dann gelöscht, wenn keine Einblendungen in andere Adreßräume existieren. Insbesondere werden Codebereiche erst dann beseitigt, wenn auf einem Knoten die letzte Instanz eines Programms terminiert.
Programmverwalter beseitigen nur Kernobjekte. Externe Betriebsmittel wie zum Beispiel Gruppenkanäle müssen vom ausgeführten Programm bereits vor der Benachrichtigung des Verwalters freigegeben werden.

2.3.4 Benutzerschnittstelle

Meist verfügen Parallelrechner nicht über direkt angeschlossene Peripherie-Geräte wie Bildschirm, Tastatur oder Festplatten. Statt dessen werden die über den Wirtsrechner erreichbaren Geräte auch von Programmen auf dem Parallelrechner verwendet (Abb. 2.8). Um dies zu ermöglichen muß das Betriebssystem des Parallelrechners Aufrufe anbieten, die an den Wirtsrechner weitergeleitet und vom dort installierten Betriebssystem ausgeführt werden. Über diese Aufrufe können Programme auf dem Parallelrechner zum Beispiel Fenster oder Dateien auf dem Wirtsrechner erzeugen und benutzen.

Die in COSY verwendete Wirtsrechner-Anbindung ist als *Brücke* im Instanzenbereich realisiert. Der *Brückenkopf* der COSY-Seite ist ein Dienstgeber auf dem Wurzelknoten (Abb. 2.8, weißer Kreis), der über eine Kommunikationsleitung mit dem Wirtsrechner verbunden ist. Auf dem Wirtsrechner befindet sich der zweite Brückenkopf sowie mindestens eine Instanz zur Bearbeitung der eingehenden Aufträge. Zur Interaktion zwischen Brückenkopf und Instanzen auf dem Wirtsrechner werden die Möglichkeiten des auf dem Wirtsrechner eingesetzten Betriebssystems genutzt [Schä95].

Anmerkung. *Der Zugriff auf Peripherie-Geräte über einen Wirtsrechner ist eine Einschränkung der Hardware. Neuere Parallelrechner haben direkten Zugriff auf Peripherie. Auf diesen könnte* COSY *auch ohne Wirtsrechner betrieben werden, sobald geeignete Gerätetreiber implementiert sind.*

Die Brücke zwischen Parallel- und Wirtsrechner wird unmittelbar nach der Inbetriebnahme des Parallelrechners aufgebaut. Der initiale Prozeß auf dem Wurzelknoten startet den Brückenkopf der COSY-Seite. Auf dem Wirtsrechner wird der Brückenkopf von dem Prozeß gestartet, der auch die Inbetriebnahme des Parallelrechners ausgelöst hat.

Nach dem Aufbau der Brücke wird auf einem Knoten des Parallelrechners ein *Kommandointerpreter* gestartet. Es handelt sich dabei um ein Programm, das auf Benutzereingaben wartet und diese auswertet. Der Kommandointerpreter ermöglicht das Ausführen von Anwenderprogrammen. Erst durch ihn kann der Benutzer einen Parallelrechner für beliebige Zwecke einsetzen.

Kapitel 3

Ausgangsarchitektur

Die erste Implementierung von COSY ist auf der Grundlage der von Parsytec hergestellten *MegaCluster*-Parallelrechner entstanden. Ein Knoten dieser Parallelrechner besteht aus einem *Transputer*-Prozessor des Typs T805 und 4 Megabyte lokalem Speicher. In einem Transputer-Prozessor sind vier Kommunikationsleitungen integriert [IMS92].

Abbildung 3.1 zeigt eine schematische Darstellung eines Knotens in einem MegaCluster. Der Transputer-Prozessor greift über einen 32 Bit breiten Datenbus auf den Speicher zu. Die Kommunikationsleitungen sind als Linien mit zwei Spitzen dargestellt. Sie müssen mit Leitungen anderer Knoten verbunden werden, um eine Kommunikation zu ermöglichen.

In einem MegaCluster können die Leitungen der Transputer über Verschaltungseinheiten in verschiedenen Konfigurationen miteinander verbunden werden. Für COSY werden zweidimensionale Gitterstrukturen verwendet. Eine Änderung der Konfiguration während des Betriebs ist nicht vorgesehen, da die Verschaltungseinheiten für diesen Zweck nicht schnell genug arbeiten.

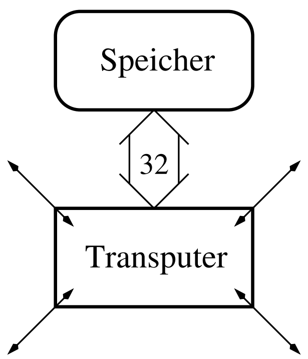

Abbildung 3.1: Knoten eines MegaCluster

3.1 Transputer

Transputer sind speziell für Mehrprozeßbetrieb entwickelte Prozessoren. Sie enthalten einen Hardware-Kern, der Prozesse und Kanäle unterstützt [IMS88]. Die vom Hardware-Kern angebotenen Objekttypen bieten allerdings nicht die im Cosy-Kern vorgesehenen Möglichkeiten. Statt dessen sind beide Objekttypen auf eine effiziente Handhabung durch den Hardware-Kern ausgelegt.

3.1.1 Prozesse

Transputer-Prozessen ist eine Priorität fest zugeordnet. Der Transputer-Kern unterstützt eine *niedrige* und eine *hohe* Priorität. Prozesse mit niedriger Priorität erhalten nur dann Rechenzeit, wenn kein Prozeß mit hoher Priorität fortgesetzt werden kann.

Prozesse mit niedriger Priorität werden nach dem Zeitscheibenverfahren abgearbeitet. Die Länge einer Zeitscheibe beträgt 2 Millisekunden. Nach Ablauf einer Zeitscheibe wird der aktive Prozeß noch solange weiter bearbeitet, bis er den ersten Befehl erreicht, der einen Entzug des Prozessors erlaubt. Zu diesen Befehlen gehören zum Beispiel Sprünge, wie sie bei Prozeduraufrufen oder am Ende von Programmschleifen auftreten. Nach der Ausführung eines solchen Befehls ist der Inhalt einiger Register undefiniert. Diese Register müssen also nicht gesichert werden. Die restlichen Register werden im Stapelbereich des Transputer-Prozesses abgelegt. Der Zeiger in diesen Stapelbereich ist die einzige Information, die der Hardware-Kern über einen Prozeß benötigt.

Prozesse mit hoher Priorität werden abgearbeitet, bis sie eine blockierende Operation ausführen. Wird ein hochpriorer Prozeß deblockiert, während ein Prozeß mit niedriger Priorität arbeitet, so unterbricht er diesen Prozeß. Der Zustand des unterbrochenen Prozesses muß in diesem Fall vollständig gespeichert werden. Es kann zu jedem Zeitpunkt nur ein Prozeß unterbrochen sein, deshalb ist der zur Sicherung des Zustandes notwendige Speicher begrenzt. Ein ausreichend großer Speicherbereich ist zu diesem Zweck auf dem Prozessor reserviert.

3.1.2 Kanäle

Transputer-Kanäle dienen zur Kommunikation zwischen Transputer-Prozessen. Sowohl das Senden als auch das Empfangen sind synchrone Operationen. Ein Transputer-Kanal ist durch ein Speicherwort realisiert, der Aufwand für das Einrichten eines Transputer-Kanals ist also verschwindend gering. In einem solchen Speicherwort ist ein Zeiger in den Stapelbereich eines blockierten Transputer-Prozesses eingetragen, oder ein festgelegter Wert, wenn kein Prozeß am entsprechenden Kanal blockiert ist.

Bei der Ausführung einer Kommunikationsoperation an einem Transputer-Kanal wird überprüft, ob an diesem ein anderer Prozeß blockiert ist. Ist dies der

Fall, so findet eine Kommunikation statt. Die Adresse des Nachrichtenpuffers des vorgefundenen Prozesses ist in dessen Stapelbereich abgelegt. Die Größe der Nachricht und die Richtung des Datentransfers sind Parameter der Kommunikationsoperation, die der zweite Transputer-Prozeß gerade ausführt.
Wird am Kanal kein Prozeß angetroffen, so blockiert die Operation. Die Adresse des Nachrichtenpuffers wird im Stapelbereich des Transputer-Prozesses abgelegt und der Stapelzeiger im Kanal eingetragen. Die Größe der Nachricht und die Richtung des Datentransfers gehen verloren.

Führen zum Beispiel zwei Transputer-Prozesse am gleichen Kanal eine Sendeoperation aus, ohne daß ein Empfänger vorhanden ist, so blockiert der Prozeß, dessen Operation zuerst ausgeführt wird. Bei der Sendeoperation des zweiten Prozesses findet der Transputer-Kern einen blockierten Prozeß im Kanal vor und interpretiert diesen als *Empfänger*. Ein analoges Verhalten ergibt sich beim Aufeinandertreffen zweier Empfangsoperationen. Die Nachrichtengröße wird ebenfalls aus der zuletzt ausgeführten Kommunikationsoperation bestimmt, auch wenn es sich um zwei komplementäre Operationen handelt.
Eine sichere Verwendung von Transputer-Kanälen ist deshalb nur dann möglich, wenn jeder Kanal von genau einem sendenden und genau einem empfangenden Transputer-Prozeß zur Übertragung von Nachrichten fester Größe verwendet wird.

Muß ein Transputer-Prozeß Nachrichten von mehreren Sendern empfangen können, so wird für jeden Sender ein eigener Kanal eingerichtet. Der empfangende Prozeß kann diese Kanäle zu einer Gruppe zusammenfassen und blockieren, bis an einem der Kanäle eine Nachricht vorliegt.

3.1.3 Leitungen

Die im MegaCluster eingesetzten Transputer sind mit jeweils vier Kommunikationsleitungen ausgestattet. Leitungen verschiedener Transputer können verbunden werden und erlauben die Übertragung von Daten in beide Richtungen. Die Übertragung erfolgt seriell, wobei der Empfang jedes Bytes bestätigt wird. Die Übertragung ist *sicher*[1], ein Protokoll zur Fehlererkennung und -beseitigung muß also nicht implementiert werden.

Für jede Leitung und jede Übertragungsrichtung existiert ein Transputer-Kanal, über den ein Transputer-Prozeß Nachrichten versenden beziehungsweise empfangen kann. Bei der Ausführung einer Kommunikationsoperation an einem dieser Kanäle ist die Richtung der Übertragung durch den verwendeten Kanal festgelegt. Nur die Adresse des Nachrichtenpuffers und die Länge der zu übertragenden Nachricht werden aus den Parametern der Operation bestimmt.

Das Verhalten für den Fall, daß auf einen dieser Kanäle zwei Prozesse gleichzeitig zugreifen, ist in der zur Verfügung stehenden Literatur [IMS88, IMS92] nicht spezifiziert.

[1] Die Fehlerwahrscheinlichkeit bei der Datenübertragung liegt in der gleichen Größenordnung wie die bei einem Speicherzugriff.

3.1.4 Weitere Komponenten

Transputer verfügen über zwei Zählregister, deren Inhalt in regelmäßigen Zeitabständen erhöht wird. Eines ist nur für Prozesse mit hoher Priorität zugänglich und wird in Abständen von einer Mikrosekunde erhöht. Das andere kann nur von Prozessen mit niedriger Priorität gelesen werden, sein Inhalt erhöht sich nach jeweils 64 Mikrosekunden. Es existiert ein Transputer-Befehl, mit dem ein Prozeß bis zum Erreichen eines bestimmten Zählerstandes blockiert werden kann.

In Rechnern mit zusätzlicher Hardware, zum Beispiel Kommunikationsprozessoren, werden externe Ereignisse über ein Hardware-Signal an den Prozessor gemeldet. Der Transputer bietet die Möglichkeit, einen Prozeß bis zum Auftreten eines solchen Hardware-Signals zu blockieren. Nach dem Deblockieren meldet der Transputer über ein Bestätigungs-Signal an die Hardware, daß das externe Ereignis erkannt wurde. Daraufhin löscht der Auslöser das Hardware-Signal, so daß es zum Signalisieren weiterer Ereignisse dienen kann.
Üblicherweise wird ein Prozeß mit hoher Priorität verwendet, um auf externe Ereignisse zu warten. Seine Deblockierung unterbricht laufende Prozesse mit niedriger Priorität, was eine schnelle Reaktion auf externe Ereignisse ermöglicht.

Eine Adreßtransformationseinheit, die den Schutz von Speicherbereichen vor unerwünschten Zugriffen ermöglicht, fehlt auf Transputern des Typs T805. Auch das Erkennen von *Ausnahmesituationen*, etwa einer Division durch 0 oder das Auftreten eines illegalen Befehlswortes, wird nicht unterstützt.

Kapitel 4

Zielarchitektur

Als Basis für die Portierung von COSY auf PowerPC-Plattformen dient der von Parsytec hergestellte PowerXplorer [PXP93]. Ein Knoten eines PowerXplorer enthält einen PowerPC-Prozessor des Typs MPC601 und einen Transputer-Prozessor des Typs T805 oder T425. Auf jedem Knoten befinden sich mindestens 8 Megabyte Speicher, auf den beide Prozessoren zugreifen können, sowie mindestens 1 Megabyte Speicher, auf den nur der Transputer-Prozessor zugreifen kann. Transputer des Typs T425 bieten, im Gegensatz zum Typ T805, keine Hardwareunterstützung für das Rechnen mit Gleitkommazahlen. Der auf jedem Knoten vorhandene Transputer ist also als reiner Kommunikationsprozessor vorgesehen.

Abbildung 4.1 auf Seite 32 zeigt eine schematische Darstellung eines Knotens in einem PowerXplorer. Der Transputer mit seinem lokalen Speicher entspricht einem Knoten in einem MegaCluster (Abbildung 3.1 auf Seite 27). Der gemeinsame Speicher ermöglicht den Austausch von Daten zwischen den Prozessoren. Der PowerPC-Prozessor greift über einen 64 Bit breiten Datenbus auf diesen Speicher zu. Zugriffe des Transputers über einen 32 Bit breiten Datenbus erfolgen wesentlich langsamer. Zur Koordination kann jeder der Prozessoren ein externes Ereignis an den jeweils anderen Prozessor melden. Dies ist durch die gestrichelte Linie angedeutet. Zusätzlich kann der PowerPC-Prozessor vom Transputer angehalten und zurückgesetzt werden, was für die Inbetriebnahme des Parallelrechners unerläßlich ist.

Der Aufbau der Knoten läßt darauf schließen, daß die einfache Portierung von Betriebssystemen des MegaClusters ein wichtiges Ziel bei der Entwicklung des PowerXplorers war. Durch den im Vergleich mit Knoten des MegaCluster hinzugekommenen Prozessor und Speicher kann eine „lokale“ Portierung vorgenommen werden. Das bedeutet, das vorhandene Betriebssytem kann auf den Transputern arbeiten und die Kommunikation innerhalb des Parallelrechners abwickeln. Gleichzeitig wird die neue Version des Betriebssytems auf den PowerPC-Prozessoren ausgeführt, wobei Transportaufträge für Datenpakete an den lokalen Transputer weitergeleitet werden. Mit dem auf Transputern lauffähigen Betriebssystem kann Software zur Fehlersuche eingesetzt werden, die im

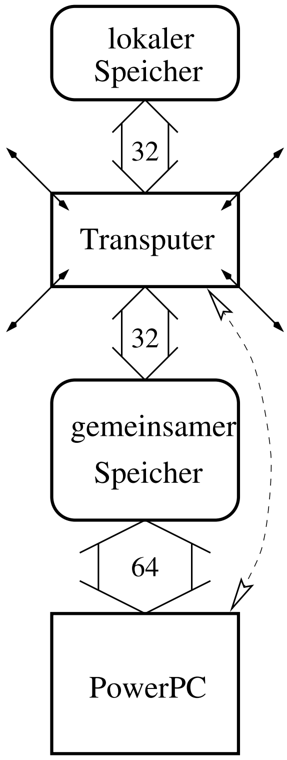

Abbildung 4.1: Knoten eines PowerXplorer

lokalen Speicher des Transputers vor Fehlfunktionen des PowerPC-Prozessors oder der dort laufenden Software geschützt ist. Wenn eine stabile PowerPC-Version des Betriebssystems entstanden ist, wird das auf den Transputern verwendete vollwertige Betriebssystem auf eine Minimalversion reduziert, die nur noch die Abwicklung der Kommunikation übernimmt. Auf diese Weise wurde die Portierung von Cosy durchgeführt.

Anmerkung. *Die ersten PowerXplorer wurden mit Transputern des Typs T805 und 4 Megabyte lokalem Transputer-Speicher ausgeliefert. Bei späteren Versionen wurde der lokale Speicher auf 1 Megabyte reduziert und der T805-Prozessor durch den billigeren Typ T425 ersetzt. Das beschriebene Vorgehen für die Portierung von Betriebssystemen spiegelt sich also auch bei der Entwicklung der Hardware wider: Nachdem eine stabile PowerPC-Version des von Parsytec eingesetzten Betriebssystems entstanden war, entfiel die Notwendigkeit, die MegaCluster-Version auf den Transputern einzusetzen. Somit konnten Komplexität und damit Herstellungskosten des Transputer-Teils der PowerXplorer-Knoten reduziert werden.*

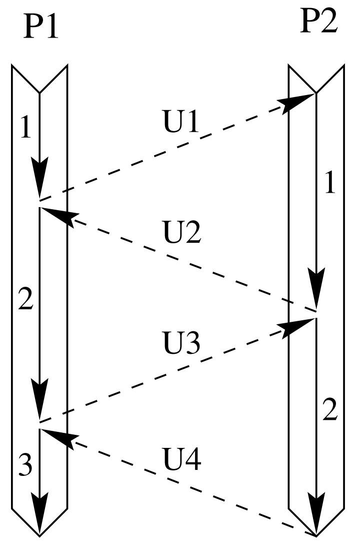

Abbildung 4.2: Mehrprozeßbetrieb

4.1 PowerPC

Die folgenden Abschnitte enthalten einen Überblick über die Eigenschaften und Möglichkeiten des auf dem PowerXplorer eingesetzten PowerPC-Prozessors. Genauere Beschreibungen einzelner Bereiche befinden sich in den Kapiteln, zu deren Verständnis sie benötigt werden.

Im Gegensatz zu Transputer-Prozessoren enthält der im PowerXplorer eingesetzte PowerPC-Prozessor keinen Hardwarekern [PPC93]. Er führt eine sequentielle Folge von Befehlen, den *Befehlsstrom*, aus. Um einen Mehrprozeßbetrieb zu ermöglichen müssen in diesen Befehlstrom Umschaltsequenzen eingefügt werden. Die Befehlssequenzen zwischen den Umschaltsequenzen werden aus den Befehlsströmen der auszuführenden Prozesse entnommen.
Abbildung 4.2 zeigt als Beispiel einen Ausschnitt aus einem Befehlsstrom, in dem zwei Prozesse abgearbeitet werden. Einer der Prozesse (P1) enthält drei Teilsequenzen, der zweite Prozeß (P2) zwei. Hinzu kommen vier Umschaltsequenzen (U1 bis U4). Der vom Prozessor abgearbeitete Befehlsstrom enthält dann die einzelnen Teilsequenzen in der Reihenfolge:

$$\ldots \text{ P1.1 U1 P2.1 U2 P1.2 U3 P2.2 U4 P1.3} \ldots$$

Der PowerPC-Prozessor verfügt über einen Mechanismus zur *Ausnahmebehandlung*, mit dem Befehlssequenzen, zum Beispiel Umschaltsequenzen, beim Auftreten vordefinierter Ereignisse in den Befehlsstrom des Prozessors eingefügt werden können. Diese Ereignisse können im Befehlsstrom auftreten und ein *freiwilliges* Umschalten herbeiführen. Aber auch externe Ereignisse, etwa das Ablaufen eines Zählers, können eine Umschaltsequenz aktivieren und damit ein *zwangsweises* Umschalten bewirken.

Anmerkung. *Die Bezeichnung* PowerPC *steht für eine Prozessorspezifikation [PPC94]. Die real existierenden PowerPC-Prozessoren, wie etwa der MPC601, entsprechen dieser Spezifikation nicht vollständig. Für einen Anwendungsprogrammierer spielt dies kaum eine Rolle, da die Unterschiede durch geeignete Systemsoftware verdeckt werden können. Für die Programmierung eines Betriebssystems ergeben sich jedoch teilweise gravierende Unterschiede. Die Angaben in den folgenden Abschnitten und Kapiteln beziehen sich, soweit nicht anders erwähnt, auf den im PowerXplorer verwendeten Prozessor des Typs MPC601.*

4.1.1 Sequentieller Betrieb

Der Befehlsstrom wird von mehreren Berechnungseinheiten abgearbeitet, die für verschiedene Befehlstypen zuständig sind. Zum Beispiel werden Gleitkomma- und Ganzzahlbefehle von verschiedenen Berechnungseinheiten ausgeführt. Prinzipiell können Berechnungseinheiten für einen Befehlstyp auch mehrfach vorhanden sein, um mehrere Befehle eines Typs gleichzeitig auszuführen. Beim MPC601 ist dies allerdings nicht der Fall.
Um die Berechnungseinheiten möglichst gut auszulasten, werden Befehle nicht immer in der Reichenfolge ausgeführt, in der sie im Befehlsstrom auftauchen. Falls Befehle nicht von vorausgehenden Befehlen abhängig sind, können sie *spekulativ*, also vorzeitig, ausgeführt werden, um freie Berechnungseinheiten zu nutzen. Datenabhängigkeiten zwischen Befehlen werden vom Prozessor erkannt, so daß sich das Ergebnis im Vergleich zu einer sequentiellen Ausführung nicht ändert.

Einige Befehle des PowerPC-Prozessors, zum Beispiel das Verändern des Prozessorzustandsregisters, beeinflussen die im Prozessor gespeicherte Ablaufumgebung und damit indirekt die Ausführung nachfolgender Befehle. Solche Einflüsse werden als *Seiteneffekte* bezeichnet und, im Gegensatz zu den Datenabhängigkeiten zwischen Befehlen, nicht vom Prozessor erkannt. Damit die Abarbeitung des Befehlsstroms zum erwarteten Ergebnis führt, müssen in diesem Fall Synchronisationsbefehle eingefügt werden. Nach einem Befehl, der einen Seiteneffekt bewirkt, stellt ein Synchronisationsbefehl sicher, daß alle folgenden Befehle in der veränderten Ablaufumgebung ausgeführt werden. Eine Beeinflussung von vorhergehenden Befehlen durch einen spekulativ ausgeführten Befehl mit Seiteneffekt ist nicht möglich [PPC93].

Um die Auswirkungen von Fehlern in Anwenderprogrammen zu begrenzen, unterscheidet der PowerPC-Prozessor zwei Betriebsarten: Einen *Anwender*-Modus und einen privilegierten *System*-Modus. Im Anwender-Modus stehen bestimmte Register und Befehle nicht zur Verfügung. Insbesondere ist es im Anwender-Modus nicht möglich, Befehle auszuführen, die Seiteneffekte bewirken. Die Betriebsart wird durch den Wert des Prozessorzustandsregisters bestimmt, der nur im System-Modus geändert werden kann. Ein Wechsel vom Anwender- in den System-Modus erfolgt durch das Auslösen einer *Ausnahmebehandlung*.

4.1.2 Ausnahmebehandlung

Ausnahmen sind Situationen, die die Abarbeitung des Befehlsstroms „stören", zum Beispiel das Auftreten eines privilegierten Befehls im Anwender-Modus oder eines illegalen Befehlswortes. Um eine frei definierbare Reaktion auf Ausnahmen zu ermöglichen, ruft der Prozessor *Ausnahmebehandlungsprozeduren* auf, die Teil der Systemsoftware sind.

Der PowerPC-Prozessor unterscheidet mehrere Arten von Ausnahmen. Tritt eine davon auf, so kopiert er das Prozessorzustandsregister und den Befehlszeiger in für diesen Zweck reservierte Register. In anderen reservierten Registern werden Informationen über den Grund für die Ausnahmesituation abgelegt. Danach werden Teile des Prozessorzustandsregister auf definierte Wert gesetzt und damit unter anderem die Betriebsart auf den System-Modus festlegt. Der Befehlszeiger wird auf eine Adresse gesetzt, die von der Art der Ausnahme abhängig ist. Ab dieser Adresse muß die Systemsoftware eine Ausnahmebehandlungsprozedur im Speicher abgelegt haben. Die in den reservierten Registern gespeicherten Daten erlauben eine genaue Bestimmung des Grundes für die Ausnahme und eine Fortsetzung des Befehlsstroms an der Stelle, an der sie aufgetreten ist.

Die vom PowerPC-Prozessor unterstützten Ausnahmesituationen lassen sich grob in zwei Klassen einteilen, die in der PowerPC-Spezifikation als *synchrone* und *asynchrone* Ausnahmen bezeichnet werden. Synchrone Ausnahmen sind an den abgearbeiteten Befehlsstrom gebunden. Sie entstehen beim Versuch, ein Befehlswort zu lesen, zu interpretieren oder den Befehl auszuführen. Die oben erwähnten Beispiele für Ausnahmen sind synchrone Ausnahmen. Asynchrone Ausnahmen werden durch Ereignisse ausgelöst, die nicht an den Befehlsstrom gebunden sind. Zum Beispiel führt die Signalisierung eines externen Ereignisses zu einer Ausnahmesituation. Die Ereignisse, die zu asynchronen Ausnahmen des PowerPC führen, werden in Abschnitt 4.1.5 beschrieben.

Da synchrone Ausnahmen an den Befehlsstrom gebunden sind, können sie sowohl vermieden als auch erzwungen werden. Das Vermeiden von Ausnahmen ist insbesondere in Ausnahmebehandlungsprozeduren wichtig. Informationen über eine erkannte Ausnahmesituation werden in reservierten Registern abgelegt, deshalb darf im Verlauf einer Ausnahmebehandlung keine weitere Ausnahme auftreten, bevor der Inhalt dieser Register gesichert wurde.
Das Erzwingen einer Ausnahme ist notwendig, wenn der Prozessor aus dem Anwender- in den System-Modus umgeschaltet werden soll. Der Befehlssatz aller PowerPC-Prozessoren enthält einen Befehl, der speziell zum Auslösen einer Ausnahmebehandlung vorgesehen ist. Die entsprechende Ausnahmebehandlungsprozedur wird also wie eine normale Prozedur explizit aufgerufen, aber im System-Modus ausgeführt.

Aufgrund der spekulativen Ausführung von Befehlen können die an Befehle gebundenen synchronen Ausnahmen auftreten, bevor alle vorhergehenden Befehle ausgeführt sind. In diesem Fall verzögert der Prozessor die Ausnahmebehandlung, bis die Ausführung vorhergehender Befehle abgeschlossen ist. Wurden bereits spekulativ Befehle ausgeführt, so verwirft der Prozessor die von diesen Befehlen bewirkten Änderungen und auch die aufgetretenen Ausnahmen.

4.1.3 Adreßtransformation

Zur Umsetzung von Programm-Adressen in physikalische Speicher-Adressen enthalten PowerPC-Prozessoren eine Adreßtransformationseinheit. Diese kann für Daten- und Codezugriffe getrennt ein- oder ausgeschaltet sein. Ist die Adreßtransformation für einen Zugriff ausgeschaltet, so wird die Programm-Adresse als Speicher-Adresse verwendet.

Die Adreßtransformationseinheit eines PowerPC-Prozessors bietet zwei Arten der Abbildung von Programm- in Speicher-Adressen an: eine *einstufige* und eine *zweistufige* Transformation. Mit der einstufigen Transformation können wenige, große Blöcke des Programmadreßraums direkt auf zusammenhängende Bereiche des physikalischen Speichers abgebildet werden. Die segment- und seitenbasierte zweistufige Transformation erlaubt es, kleinere Bereiche fester Größe aus dem Programm- in den Speicheradreßraum abzubilden. Ist für eine Programm-Adresse sowohl eine ein- als auch eine zweistufige Abbildung definiert, so benutzt der Prozessor die einstufige Abbildung.

Die Adreßtransformationseinheit des MPC601 kann vier Blöcke mit Größen zwischen 128 Kilobyte und 8 Megabyte einstufig abbilden. Als Blockgrößen sind nur Zweierpotenzen zugelassen. Die Anfangsadresse eines Blocks muß sowohl im Programm- als auch im Speicheradreßraum ein Vielfaches der Blockgröße sein. Bei der zweistufigen Abbildung ist die Seitengröße auf 4 Kilobyte festgelegt. Auch Anfangsadressen von Seiten sind auf ein Vielfaches der Seitengröße beschränkt.
Die Informationen über die einstufige Abbildung werden in speziellen Prozessor-Registern abgelegt. Informationen über die zweistufige Abbildung befinden sich nur zum Teil in Prozessorregistern. Zusätzlich wird im physikalischen Speicher eine *Seitentabelle* angelegt, auf die der Prozessor bei Bedarf zugreift. In der Seitentabelle können Abbildungen für mehrere Millionen Seiten gespeichert sein.

Für jede Abbildungseinheit, also jeden Block und jede Seite, werden *Zugriffsrechte* vergeben. Diese legen fest, ob ein lesender oder schreibender Zugriff auf den entsprechenden Speicherbereich erlaubt ist. Die Zugriffsrechte werden für den Anwender- und System-Modus getrennt festgelegt.
Eine Verletzung der Zugriffsrechte führt zu einer Ausnahmebehandlung. Gleiches gilt für einen Zugriff auf eine Programm-Adresse, die nicht in eine Speicher-Adresse umgesetzt werden kann. Dabei werden zwei Arten von Ausnahmen unterschieden, je eine für Daten- und Codezugriffe. Vor dem Aufruf einer Ausnahmebehandlungsprozedur wird die Adreßtransformationseinheit ausgeschaltet. Dies gilt für alle Ausnahmen, nicht nur für Ausnahmen im Zusammenhang mit Adreßtransformationen.

Der PowerPC-Prozessor greift über einen 64 Bit breiten Datenbus auf den physikalischen Speicher zu. Eine Programm- oder Speicheradresse bezieht sich jedoch traditionell auf ein Byte, also 8 Bit des physikalischen Speichers. Es gibt zwei übliche Methoden, die einzelnen Bytes innerhalb eines größeren Wortes zu adressieren, die als *HBF* (englisch: High Byte First) und *LBF* (englisch: Low Byte First) bezeichnet werden. Der PowerPC-Prozessor kann auf beide Arten betrieben werden und bietet zusätzlich Befehle an, um einzelne Zugriffe in der jeweils anderen Byte-Reihenfolge durchzuführen. Nach der Inbetriebnahme des Prozessors ist der HBF-Modus aktiviert.

Anmerkung. *Die auf den Knoten des PowerXplorer eingesetzten Transputer verwenden die LBF-Reihenfolge für Speicherzugriffe. Dies ist bei der Kooperation von PowerPC und Transputer zu berücksichtigen.*

4.1.4 Zwischenspeicher

Aus Kostengründen besteht der Hauptspeicher moderner Rechner aus Speicherbausteinen, die im Vergleich zur Geschwindigkeit etwa eines PowerPC-Prozessors nur langsam arbeiten [SGI96]. Zugriffe auf den Hauptspeicher „bremsen" also den Prozessor. Aus diesem Grund sind moderne Prozessoren mit Zwischenspeicher (englisch: Cache) ausgestattet.

Der Zwischenspeicher ist wesentlich schneller als der Hauptspeicher, aber auch kleiner. Er wird mit Teilen des Hauptspeicherinhalts gefüllt. Greift der Prozessor auf eine Speicheradresse zu, deren Inhalt sich nicht im Zwischenspeicher befindet, so wird dieser Teil des Hauptspeichers in den Zwischenspeicher geladen. Dazu muß ein Teil des Zwischenspeichers überschrieben werden, in dem meist bereits ein anderer Teil des Hauptspeichers enthalten ist. Die Wahl des zu ersetzenden Bereichs im Zwischenspeicher erfolgt beim MPC601 nach der LRU-Strategie (englisch: Least Recently Used) [Baum96].
Bei einem Zugriff auf eine Adresse, deren Inhalt sich im Zwischenspeicher befindet, ist kein Zugriff auf den langsamen Hauptspeicher notwendig. Da Programme meist *lokal* arbeiten, also auf einen kleinen Teil des Speichers sehr häufig zugreifen, ergibt sich durch den Zwischenspeicher eine erhebliche Geschwindigkeitssteigerung bei geringem Kostenaufwand.

Führt der Prozessor schreibende Zugriffe durch, die nicht sofort vom Zwischen- an den Hauptspeicher weitergegeben werden, so enthält der Hauptspeicher veraltete Daten. Dies führt zu Problemen, wenn andere Einheiten eines Rechnersystems, zum Beispiel Kommunikationsprozessoren, auf den Inhalt des Hauptspeichers zugreifen. PowerPC-Prozessoren unterstützen drei Methoden, um die Aktualität gelesener Daten zu sichern:

- Überwachung von Buszugriffen.
- Eingeschränkte Verwendung des Zwischenspeichers.
- Speichersynchronisation durch Befehle.

Die Überwachung von Buszugriffen erfolgt nach dem MESI-Protokoll[1], das in der Steuerlogik des PowerPC-Zwischenspeichers implementiert ist. Die Steuerlogik erkennt schreibende Zugriffe auf den Hauptspeicher und löscht Einträge im Zwischenspeicher, die ungültig werden. Bei einem lesenden Zugriff auf eine Adresse, deren Inhalt im Zwischenspeicher verändert wurde, signalisiert die Steuerlogik der zugreifenden Einheit, daß die gelesenen Daten ungültig sind. Dann überträgt sie den Inhalt des Zwischenspeichers in den Hauptspeicher, damit die Wiederholung des Zugriffs zum Erfolg führt.

Anmerkung. *In der zur Verfügung stehenden Dokumentation über den PowerXplorer [PXP93] findet sich keine Information darüber, ob die Zugriffe des Transputers ebenfalls mit dem MESI-Protokoll gesichert werden. Für die Portierung von* COSY *muß deshalb eine der anderen Methoden gewählt werden, die sich beide durch eine geeignete Programmierung des PowerPC-Prozessors auch ohne Hardware-Unterstützung realisieren lassen.*

Probleme mit der Datenkonsistenz treten nicht auf, wenn schreibende Zugriffe direkt auf den Hauptspeicher erfolgen. Dies ist der Fall, wenn der Zwischenspeicher entweder nicht oder nur für lesende Zugriffe benutzt wird. Beide Möglichkeiten stehen bei PowerPC-Prozessoren zur Verfügung. Die Arbeitsweise des Zwischenspeichers wird mit Hilfe der Adreßtransformationseinheit festgelegt. Ebenso wie die Zugriffsrechte auf Speicherbereiche wird die Arbeitsweise des Zwischenspeichers für jede Abbildungseinheit getrennt angegeben. Dabei läßt sich auch die Verwendung des MESI-Protokolls unterbinden.

Oftmals greifen Hardwarekomponenten nur nach einer Beauftragung durch den Prozessor auf den Hauptspeicher zu. Zum Beispiel wird ein Kommunikationsprozessor nur dann eine Nachricht aus dem Hauptspeicher lesen und versenden, wenn er vom Prozessor über die Adresse und Größe der Nachricht in Kenntnis gesetzt wurde. In einem solchen Fall muß der Zwischenspeicher des PowerPC-Prozessors nicht ausgeschaltet werden. Es genügt, wenn unmittelbar vor der Beauftragung alle Änderungen aus dem Zwischen- in den Hauptspeicher übertragen werden. Eine solche *Speichersynchronisation* kann bei allen PowerPC-Prozessoren durch einen Befehl erzwungen werden.

Auch ohne zusätzliche Hardware ergeben sich Probleme mit der Datenkonsistenz, da ein PowerPC-Prozessor intern aus mehreren Komponenten aufgebaut ist. Eine dieser Komponenten übernimmt die Abwicklung der Zugriffe auf Programmdaten, eine andere liest Befehlsworte aus dem Speicher und eine dritte greift auf die Seitentabelle des Adreßtransformationsmechanismus zu. Bei Zugriffen auf Programmdaten garantiert der Prozessor, daß die aktuellen Werte gelesen werden. Bei Zugriffen auf den Code oder die Seitentabelle ist dies nicht der Fall. Werden diese Speicherinhalte verändert, etwa beim Laden eines Programms oder dem Eintragen neuer Abbildungen in die Seitentabelle, so ist eine Speichersynchronisation erforderlich.

[1] MESI: Modified, Shared, Exclusive, Invalid. Dies sind die englischen Bezeichnungen für die vier Zustände, in denen sich ein Eintrag des Zwischenspeichers befinden kann.

4.1.5 Weitere Komponenten

Der MPC601 verfügt über eine *Echtzeituhr*. Diese ist in Form zweier Register implementiert. Eines der Register zählt Nanosekunden. Sein Wert erhöht sich alle 128 Nanosekunden, bis eine Sekunde abgelaufen ist. Danach wird es auf den Wert 0 gesetzt und der Wert des zweiten Registers um 1 erhöht. Das zweite Register zählt also die abgelaufenen Sekunden. Beide Register können nur im System-Modus modifiziert werden. Das Auslesen ist auch im Anwender-Modus möglich.

Ein zusätzliches *Zählregister* ermöglicht das Auslösen einer Ausnahmebehandlung nach Ablauf einer vorgegebenen Zeit. Der Wert dieses Registers wird, synchron mit der Echtzeituhr, alle 128 Nanosekunden um 1 vermindert. Ein Unterlauf des Registers, also ein Übergang vom Wert 0 zum Maximalwert, stellt eine Ausnahmesituation dar. Diese Ausnahme ist *maskierbar*, das heißt, sie führt nicht notwendigerweise zu einer Ausnahmebehandlung. Der Wert des Prozessorzustandsregisters legt fest, ob eine Ausnahmebehandlung erfolgen soll. Wenn nicht wird der Unterlauf des Registers gespeichert, bis die Ausnahmebehandlung erlaubt wird. Weitere Unterläufe gehen verloren. Beim Aufruf einer Ausnahmebehandlungsprozedur sind die Ausnahmen durch das Zählregister ausgeschaltet, um verschachtelte Ausnahmen zu verhindern. Der Zugriff auf das Zählregister ist nur im System-Modus möglich, somit kann es zur Programmierung eines Zeitscheiben-Verfahrens für die Prozessorvergabe benutzt werden.

Bei PowerPC-Prozessoren kann über ein Hardware-Signal eine Ausnahmesituation angezeigt werden. Auch diese Ausnahme ist maskierbar und bei der Ausführung einer Ausnahmebehandlungsprozedur ausgeschaltet. Im Gegensatz zu Transputern zeigt der PowerPC-Prozessor nicht über ein Bestätigungs-Signal an, daß er die Ausnahmesituation erkannt hat. Deshalb muß die Ausnahmebehandlungsprozedur die auslösende Hardware dazu veranlassen, das Signal zu löschen. Wird das Signal gelöscht, bevor die Ausnahmebehandlung beginnt, so ist nicht garantiert, daß es vom Prozessor zur Kenntnis genommen wurde.

Es existieren zwei weitere asynchrone Ausnahmen. Eine dient dazu, den Prozessor zurückzusetzen und somit in einen definierten Zustand zu bringen. Sie wird bei der Inbetriebnahme des Rechners benötigt. Die andere zeigt einen Fehler des Prozessors an und sollte im normalen Betrieb nicht auftreten. Diese beiden Ausnahmen sind nicht maskierbar. Sie können deshalb auch zu Beginn einer Ausnahmebehandlung auftreten und dabei den Inhalt reservierter Register überschreiben. Eine Fortsetzung des unterbrochenen Befehlsstroms ist in diesem Fall nicht mehr möglich.

Der PowerPC-Prozessor bietet zusätzlich Möglichkeiten an, die die Fehlersuche in Programmen erleichtern. So kann etwa nach jedem ausgeführten Befehl eine Ausnahmebehandlung ausgelöst werden, was einen *Einzelschrittbetrieb* ermöglicht. Während der Ausnahmebehandlungen ist diese Betriebsart selbstverständlich nicht aktiviert. Des weiteren kann je eine Daten- und Code-Adresse überwacht werden. Bei einem Zugriff auf diese Adresse wird eine Ausnahmebehandlung durchgeführt.

Die Möglichkeiten zur Fehlersuche wurden im Rahmen dieser Diplomarbeit nicht verwendet und sind an dieser Stelle nur aufgeführt, um den Überblick über die vom PowerPC-Prozessor erkannten Ausnahmesituationen abzurunden.

Kapitel 5

Werkzeuge

Der Cosy-Kern und die Systemdienstgeber sind eigenständige Programme. Um sie auf einem Rechner zu laden und auszuführen, müssen sie in Form von Programmdateien vorliegen. Eine Programmdatei enthält die Beschreibung eines Programms in einer für einen Prozessor „verständlichen“ Form. Programme werden mit Hilfe einer Programmiersprache formuliert, die es Menschen erleichtert, die abzuarbeitenden Algorithmen anzugeben. Eine solche Beschreibung eines Programms wird als *Quelltext* bezeichnet. Programme können in Module unterteilt sein. Dann existiert für jedes Modul ein eigener Quelltext. Quelltexte müssen mit Hilfe geeigneter Werkzeuge in Programmdateien transformiert werden.

Ein *Werkzeugsatz* ist eine Ansammlung aufeinander abgestimmter Hilfsmittel, mit denen Quelltexte in eine Programmdatei transformiert werden können. Üblicherweise enthält ein Werkzeugsatz zumindest einen *Übersetzer*, der einen Quelltext in ein Zwischenformat transformiert, und einen *Binder*, der die übersetzten Module eines Programms zu einer Programmdatei kombiniert. Sowohl das Übersetzen als auch das Binden kann in mehreren Schritten erfolgen, was einen größeren Werkzeugsatz erfordert.

Die Entwicklung oder Portierung eines Betriebssystems stellt besondere Anforderungen an den Werkzeugsatz. Programmiersprachen werden so entworfen, daß die damit formulierten Programme auf möglichst vielen, unterschiedlichen Rechnerarchitekturen abgearbeitet werden können. Jede Rechnerarchitektur besitzt jedoch bestimmte Eigenheiten, die nicht von einer allgemeinen Programmiersprache abgedeckt werden können.

Beispiel. *Der PowerPC-Prozessor besitzt reservierte Register, die für Ausnahmebehandlungen und Adreßtranformationen benötigt werden. Eine rechnerunabhängige Programmiersprache bietet keine Möglichkeit, auf solche Register zuzugreifen. Auch die Verwendung von prozessorspezifischen Befehlen, etwa zur Speichersynchronisation, ist in einer allgemeinen Programmiersprache nicht möglich.*

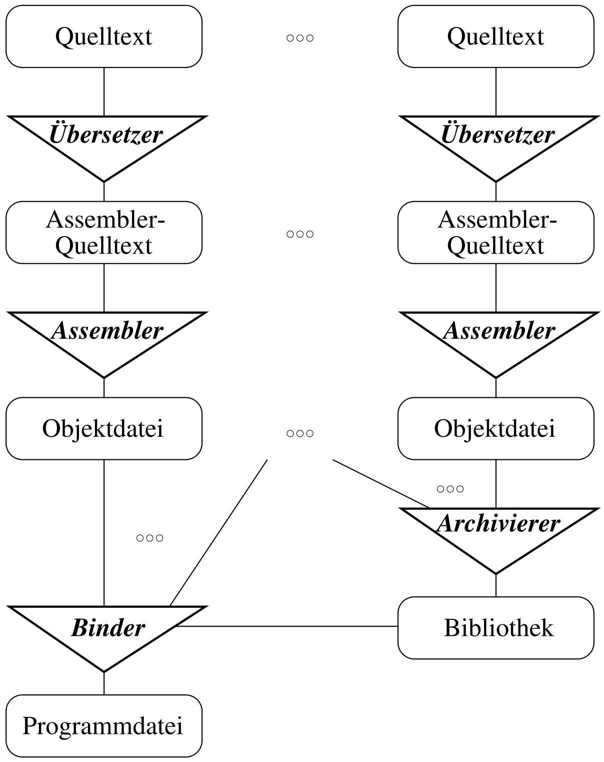

Abbildung 5.1: Transformation von C-Programmen

Ein Betriebssystem muß an diese Eigenheiten angepaßt werden. Dazu werden rechnerspezifische *Erweiterungen* der verwendeten Programmiersprache benötigt. Im Gegensatz zu Programmiersprachen sind deren Erweiterungen jedoch nicht standardisiert und unterscheiden sich deshalb bei verschiedenen Werkzeugsätzen. Die Wahl des für die Entwicklung eines Betriebssytems eingesetzten Werkzeugsatzes ist also eine Entwurfsentscheidung, die erheblichen Einfluß auf die spätere Arbeit und deren Resultat nehmen kann.

Dieses Kapitel gibt einen Überblick über die in dieser Diplomarbeit verwendeten Entwicklungswerkzeuge. In Abschnitt 5.1 werden die benötigten Werkzeuge und die daran gestellten Anforderungen beschrieben. Die Abschnitte 5.2 und 5.3 beschreiben die zwei in Frage kommenden Werkzeugsätze. In Abschnitt 5.4 sind die getroffenen Entscheidung festgehalten und begründet.

5.1 Anforderungen

Der Cosy-Kern, die Systemdienstgeber und auch Cosy-Anwenderprogramme sind und werden in einem gegenüber dem Standard leicht eingeschränkten Dialekt von C [ISO90] entwickelt. Für die Portierung von Cosy wird also ein Werkzeugsatz für die Transformation von C-Programmen benötigt. Abbildung 5.1 zeigt den Ablauf einer solchen Transformation.

Ausgehend von einem C-Quelltext erzeugt ein *Übersetzer* zunächst einen *Assembler-Quelltext*. Assembler-Sprachen sind speziell auf einen Prozessor zugeschnitten. Jeder Befehl einer Assembler-Sprache entspricht einem Befehl, der vom Prozessor abgearbeitet werden kann. Deshalb bieten Assembler-Sprachen nur relativ einfache Befehle an. Insbesondere gibt es keine Möglichkeit zur Definition von komplexen Datentypen wie Strukturen, Zeichenketten oder Feldern. Bei der Übersetzung des C-Quelltextes werden für eine C-Anweisung meist mehrere Assembler-Anweisungen erzeugt.
Im nächsten Schritt erzeugt ein Assembler-Übersetzer, kurz *Assembler*[1], aus dem Assembler-Quelltext eine sogenannte *Objektdatei*. Eine Objektdatei enthält vom Prozessor ausführbare Befehlsworte. Diese Befehlsworte enthalten jedoch noch Verweise auf Programmadressen, die bei der Erzeugung der Objektdatei unbekannt sind. Dies tritt zum Beispiel dann auf, wenn Verweise sich auf den Inhalt anderer Objektdateien beziehen. In der Objektdatei sind Informationen darüber enthalten, wo sich unbekannte Verweise befinden und worauf sie sich beziehen.
Im letzten Schritt kombiniert der *Binder* mehrere Objektdateien, um unbekannte Verweise aufzulösen und eine Programmdatei zu erzeugen. Die Programmdatei enthält ein „Bild“ des ausführbaren Programms, in dem Lage und Inhalt der verwendeten Speicherbereiche beschrieben sind.
Bibliotheken sind Sammlungen von Objektdateien, die mit einem *Archivierer* erzeugt werden. Der Archivierer faßt mehrere Objektdateien zu einer Bibliotheksdatei zusammen, in der auch eine Tabelle der Verweise gespeichert ist, die mit Hilfe dieser Objektdateien aufgelöst werden können. Beim Binden mit einer Bibliothek werden von den enthaltenen Objektdateien nur diejenigen in die Programmdatei aufgenommen, die der Binder zum Auflösen unbekannter Verweise benötigt. Die gespeicherte Tabelle mit Verweisen beschleunigt das Bestimmen der benötigten Objektdateien.

Die rechnerspezifischen Teile der Systemsoftware werden teilweise in der Assembler-Sprache des Prozessors formuliert. Assemblerbefehle lassen sich auf zwei Arten in ein Programm integrieren: als *Prozedur* oder als *Makro*.
Eine Prozedur kann direkt als Assembler-Quelltext definiert werden. Es entsteht eine separate Objektdatei, die als Teil eines Programms gebunden wird. Aus dem C-Quelltext heraus kann eine Assembler-Prozedur ebenso aufgerufen werden wie eine C-Prozedur. Um in der Prozedur auf Parameter zugreifen zu können muß jedoch bekannt sein, in welcher Weise der Übersetzer die Parameterübergabe bei Prozeduraufrufen handhabt. Nur in einfachen Fällen kann dies durch eine Analyse des vom Übersetzer erzeugten Assembler-Quelltextes in Erfahrung gebracht werden. Für die Systemprogrammierung ist deshalb eine Dokumentation der Parameterübergabe wünschenswert.
Ein Makro ist eine Sequenz von Assembler-Befehlen im C-Quelltext, die textuell in den vom Übersetzer erzeugten Assembler-Quelltext eingefügt wird. Dazu ist eine Erweiterung der vom Übersetzer akzeptierten Programmiersprache notwendig. Da kein expliziter Aufruf nach einem festgelegten Schema stattfindet,

[1] Die Kurzform *Assembler* ist sowohl für die Assembler-Sprache als auch für den Assembler-Übersetzer gebräuchlich.

erfordert die Übergabe von Werten oder der Zugriff auf Resultate des Makros eine spezielle Unterstützung durch den Übersetzer. Im Gegensatz zur Prozedur-Variante ist hier eine ausführliche Dokumentation der Parameterübergabe unumgänglich.

Eine Programmdatei definiert Lage und Inhalt von Speicherbereichen. Beim Laden einer Programmdatei wird im Rechner ein „Bild" der Programmdatei erzeugt, indem Speicherbereiche angefordert und mit dem definierten Inhalt gefüllt werden. Es ist jedoch möglich, daß ein Betriebssystem keine beliebige Plazierung von Speicherbereichen in einem Programmadreßraum erlaubt.

Beispiel. *In einem Rechner ohne Adreßtransformationseinheit kann ein Speicherbereich nur dort angefordert werden, wo sich zum Zeitpunkt des Ladens noch freier Speicher befindet. Aber auch wenn eine Adreßtransformation möglich ist, existieren Einschränkungen. Zum Beispiel muß die Anfangsadresse eines Speicherbereichs üblicherweise ein Vielfaches der Seitengröße sein. Außerdem kann durch das Betriebssystem ein Teil jedes Programmadreßraums reserviert sein, in dem zum Beispiel der Kern arbeitet.*

Erlaubt der Binder die Angabe von Speicherbereichen, in denen Programmteile zu liegen kommen dürfen, so kann vielen, im Idealfall sogar allen Einschränkungen schon beim Binden Rechnung getragen werden. Ist dies nicht der Fall, oder sind nicht alle Einschränkungen zum Zeitpunkt des Bindens bekannt, so muß die Lage der Speicherbereiche beim Laden verändert werden. Dazu müssen in der Programmdatei zusätzliche Informationen abgelegt sein, die es erlauben, die im Code enthaltenen Programmadressen beim Laden so zu modifizieren, daß der Code im zugeteilten Speicherbereich ablaufen kann.
Einen Sonderfall bilden *verschiebbare Programme*. Ihre Speicherbereiche lassen sich beliebig plazieren, ohne daß Änderungen am Code vorzunehmen sind. Statt dessen erhalten sie die Lage der Speicherbereiche als Aufrufargumente und berechnen Programmadressen erst zur Laufzeit.

Die Entwicklung von Systemsoftware kann durch Werkzeuge zur Analyse von Programm- und Objektdateien erleichtert werden. Die Rückübersetzung von Programmdateien in Assembler-Quelltext durch einen sogenannten *Disassembler* erlaubt bei Fehlern, die eine Ausnahme auslösen, eine genaue Bestimmung des Quelltextbereichs, in dem der Fehler auftrat. Außerdem wird hierdurch die Kontrolle des vom Übersetzer erzeugten Assembler-Quelltextes vereinfacht, da dieser üblicherweise unmittelbar nach der Transformation in eine Objektdatei gelöscht wird. Auch Angaben zur Lage der in einer Programmdatei definierten Speicherbereiche können sich als nützlich erweisen, wenn beim Laden eines Programms ein Fehler auftritt. Eine Fehlinterpretation des Dateiinhaltes ist durch den Vergleich mit der Interpretation eines Analysewerkzeuges leicht zu erkennen.

5.2 Werkzeuge von Motorola

Die Firma Motorola war maßgeblich an der Entwicklung der PowerPC-Architektur beteiligt und stellt auch die im PowerXplorer verwendeten Prozessoren vom Typ MPC601 her. Motorola bietet einen Satz von Entwicklungswerkzeugen für Standard-C-Programme an, der im Lieferumfang von PowerXplorer-Parallelrechnern enthalten ist.

Der von Motorola angebotene Übersetzer führt weitreichende Optimierungen durch, die mit ihm erzeugten Programme besitzen also sehr kurze Ausführungszeiten. Allerdings erzeugt er Assembler-Quelltexte für POWER-Prozessoren. Die POWER-Architektur ist ein von IBM entwickelter Vorgänger der PowerPC-Architektur. Der überwiegende Teil der POWER-Befehle kann von PowerPC-Prozessoren ausgeführt werden. Die Assembler-Sprachen von POWER und PowerPC unterscheiden sich jedoch erheblich. Außerdem besitzen PowerPC-Prozessoren Register, die mit POWER-Befehlen nicht zugänglich sind. Der Motorola-Übersetzer ist somit für die Systemprogrammierung nur bedingt geeignet. Für Prozeduraufrufe verwendet der Motorola-Übersetzer die in der PowerOpen-Spezifikation [PO94] festgelegten Aufrufkonventionen. Die Entwicklung und Verwendung von Assembler-Prozeduren ist somit leicht möglich. Über Möglichkeiten zur Verwendung von Assembler-Makros dagegen enthält die mit dem Übersetzer ausgelieferte Anwenderdokumentation keine Informationen.

Für den von Motorola gelieferten Assembler ist keine Anwenderdokumentation verfügbar. Die erzeugten Objektdateien enthalten Daten im XCOFF-Dateiformat, das in der PowerOpen-Spezifikation [PO94] festgelegt ist.
Der Binder verwendet ebenfalls das XCOFF-Dateiformat, um Programmdateien zu erzeugen. Er bietet keine Möglichkeit, die Lage der Speicherbereiche im Programmadreßraum zu beeinflussen. Dies macht unter Umständen eine Modifikation des gespeicherten Codes beim Laden notwendig.
Ein Archivierer ist im Motorola-Werkzeugsatz enthalten. Zur Rückübersetzung und Analyse von Programmdateien steht ein Werkzeug aus dem GNU-Werkzeugsatz zur Verfügung, der im nächsten Abschnitt beschrieben wird.

5.3 Werkzeuge von GNU

GNU ist ein Projekt mit dem Ziel, Software verschiedenster Art frei zugänglich zu machen. Im Rahmen des GNU-Projektes wurde unter anderem ein Übersetzer für C-Programme und ein Werkzeugsatz mit Assembler, Binder, Archivierer und Analysewerkzeug realisiert. Sowohl der Übersetzer als auch der Werkzeugsatz unterstützen eine Vielzahl von Prozessoren und Formaten für Objekt- und Programmdateien.
Alle Programme können sowohl für die POWER-Architektur mit dem XCOFF-Dateiformat als auch für die PowerPC-Architektur mit dem ELF-Dateiformat [ELF93] konfiguriert werden. Das Dateiformat beeinflußt in geringem Umfang

den Assembler-Quelltext. Zum Beispiel werden Verweise auf unbekannte Programmadressen schon im Assembler-Quelltext gekennzeichnet. Deshalb ist eine Kombination von POWER-Architektur mit dem ELF-Dateiformat nicht möglich.

Auch der GNU-Übersetzer ist ein optimierender Übersetzer. Die resultierenden Programme besitzen jedoch eine etwas größere Ausführungszeit als die mit dem Motorola-Übersetzer erzeugten. In der POWER-Konfiguration verwendet der GNU-Übersetzer für Prozeduraufrufe die gleichen Konventionen wie der Übersetzer von Motorola. Die in der PowerPC-Konfiguration verwendeten Aufrufkonventionen sind ähnlich, jedoch nicht identisch. Die Quelltexte des GNU-Übersetzers sind zugänglich, so daß bei Bedarf die Aufrufkonventionen nachgeschlagen oder verändert werden können. Die Verwendung von Assembler-Makros wird unterstützt. Die Übergabe von Parametern und Resultaten ist in der Dokumentation des Übersetzers ausführlich erläutert.

Der Assembler des GNU-Werkzeugsatzes ist ausführlich dokumentiert, so daß Assembler-Prozeduren bequem entwickelt werden können. Die Eigenheiten der Assembler-Sprache, die sich durch das Dateiformat ergeben, lassen sich durch Analyse der vom GNU-Übersetzer erzeugten Assembler-Quelltexte oder durch Nachschlagen in den Quelltexten des GNU-Assemblers ermitteln.
Der Binder verwendet eine eigene Skript-Sprache. Diese bietet umfangreiche Möglichkeiten zur Beeinflussung der Programmdatei. So kann die Lage von Speicherbereichen im Programmadreßraum beliebig festgelegt werden. Außerdem können Verweise auf Programmadressen in der Skript-Sprache definiert werden. Damit ist es möglich, im Programm auf Daten direkt zuzugreifen, die erst zum Zeitpunkt des Bindens bekannt werden, etwa die Größe der definierten Speicherbereiche. Auch der Archivierer unterstützt eine Skript-Sprache. Damit ist es möglich, mehrere Bibliotheken zu einer gemeinsamen Bibliothek zusammenzufassen. Dies wird von anderen Bindern meist nicht unterstützt.
Das Analysewerkzeug des GNU-Werkzeugsatzes kann eine Programm- oder Objektdatei in den Assembler-Quelltext zurückübersetzen und dabei Prozedurnamen rekonstruieren. Es dient auch dazu, Größe und Lage der Speicherbereiche in einer Programmdatei zu ermitteln.

5.4 Entscheidungen

Zur Portierung des Cosy-Betriebssystems wurden die GNU-Werkzeuge in der Konfiguration für die PowerPC-Architektur und das ELF-Dateiformat verwendet.

Bei Verwendung des Motorola-Übersetzers ist die Verwendung von Assembler-Makros nicht möglich. Auch bei kurzen Assembler-Sequenzen, etwa zur Speichersynchronistaion, muß deshalb der erhebliche Aufwand für einen Prozeduraufruf in Kauf genommen werden. Bei Verwendung des GNU-Übersetzers ist dies nicht der Fall.

Die Aufgabenstellung „Portierung auf PowerPC“ legt es nahe, den GNU-Übersetzer für die PowerPC-Architektur zu konfigurieren. Außerdem enthält die Dokumentation des im PowerXplorer verwendeten Prozessors vom Typ MPC601 [PPC93] ausführliche Beschreibungen der PowerPC-Befehle. Die POWER-Befehle werden nur in einer Tabelle aufgeführt, die zum Nachschlagen der entsprechenden PowerPC-Befehle dient. Eine Programmierung in der POWER-Assembler-Sprache wäre also mit erheblich höherem Zeitaufwand verbunden.

Die Wahl der PowerPC-Konfiguration legt auch das Dateiformat ELF fest, da der GNU-Übersetzer nicht ohne aufwendige Änderungen für die Kombination PowerPC/XCOFF konfiguriert werden kann. Damit ist auch die Verwendung anderer Motorola-Werkzeuge ausgeschlossen, da diese nur das XCOFF-Dateiformat unterstützen.
Für die Verwendung des ELF-Dateiformats spricht zudem, daß die Dokumentation [ELF93] bereits zu Beginn der Diplomarbeit zur Verfügung stand, während auf die XCOFF-Dokumentation [PO94] erst im Laufe der Diplomarbeit zurückgegriffen werden konnte.

Die Entscheidung zugunsten der GNU-Werkzeuge betrifft zunächst nur die Systemsoftware. Die Verwendung von Assembler-Makros ist nur mit dem GNU-Übersetzer möglich und die als Assembler-Quelltext formulierten Prozeduren sind an das Dateiformat ELF angepaßt. Anwenderprogramme hingegen können durchaus im XCOFF-Dateiformat entwickelt und ausgeführt werden. Dazu kann der Motorola-Übersetzer verwendet werden, dessen Optimierungsmöglichkeiten die des GNU-Übersetzers übertreffen. Die unterschiedlichen Konventionen für Prozeduraufrufe können an der Kernschnittstelle ausgeglichen werden. Da Quelltexte des GNU-Übersetzers verfügbar sind, ist es auch nicht ausgeschlossen, die in der PowerOpen-Spezifikation festgelegten Aufrufkonventionen in Kombination mit dem ELF-Dateiformat einzusetzen.

Kapitel 6

Vorgehensweise

Das Ziel einer Portierung ist es, ein System, das in einer bestimmten Umgebung funktionsfähig ist, so zu verändern, daß es in einer anderen Umgebung ebenfalls funktionsfähig ist. Dabei sollen die Veränderungen des Systems auf ein unumgängliches Mindestmaß beschränkt bleiben. Dies reduziert den Aufwand sowohl für die eigentliche Portierung als auch für die Weiterentwicklung des Systems.
Jeder Aspekt, in dem sich die Umgebungen unterscheiden, kann Änderungen am System erzwingen. Ein Aspekt betrifft aber meist nur einen Teil des Systems. Um zu entscheiden, in welchen Teilen des Systems Änderungen vorzunehmen sind, müssen Umgebungen anhand der veränderlichen Aspekte charakterisiert werden. Diese Charakterisierung erlaubt es, die von einem bestimmten Aspekt beeinflußten Teile eines Systems zu lokalisieren und gezielt zu verändern.

Die Umgebung eines Betriebssystems ist in erster Linie durch die Architektur des Rechners bestimmt, auf dem es benutzt wird. Aber auch die für die Entwicklung benutzten Werkzeuge beeinflussen insbesondere die Quelltexte des Betriebssystems. Auf dieser Grundlage kann eine erste, grobe Charakterisierung der Umgebung eines Parallelrechner-Betriebssystems anhand folgender Aspekte erstellt werden:

- Topologie
- Zugang
- Werkzeug
- Hardware

Als wichtigster topologieabhängiger Teil von Cosy ist das *Transportsystem* zu nennen, über das der Kernverbund realisiert wird. Des weiteren können sich Änderungen bei der Plazierung von Dienstgebern ergeben, die für mehrere Knoten zuständig sind. Da PowerXplorer jedoch als zweidimensionale Gitter verschaltet sind, die bereits von Cosy unterstützt werden, sind im Rahmen

dieser Diplomarbeit keine Änderungen aufgrund von Topologieabhängigkeiten vorzunehmen.
Der Zugang zu einem Parallelrechner kann über Wirtsrechner erfolgen. Als Alternative dazu bietet sich an, die Knoten des Parallelrechners mit Peripherie auszustatten, so daß der Zugang direkt über einen der Knoten erfolgen kann. Der Zugang zu einem MegaCluster erfolgt ebenso wie der zu einem PowerXplorer über einen Wirtsrechner. Auch hier sind also für die Portierung keine wesentlichen Änderungen notwendig.
Der für die Entwicklung von Cosy für MegaCluster verwendete Werkzeugsatz von Inmos unterscheidet sich vom GNU-Werkzeugsatz in mehreren Punkten, zu denen auch die Benutzung der Werkzeuge zählt. Änderungen aufgrund unterschiedlicher Werkzeuge sind deshalb unumgänglich.
Der Begriff „Hardware“ bezieht sich auf die Architektur eines Knotens des Parallelrechners. Hier ist eine Verfeinerung der Charakterisierung angebracht. Die Architektur eines Knotens kann durch die folgenden Aspekte umrissen werden:

- Prozessor
- Adreßtransformation
- Kommunikation

In den folgenden Abschnitten werden die Aspekte „Hardware“ und „Werkzeuge“ präzisiert und die von ihnen beeinflußten Teile von Cosy lokalisiert. Dabei wird auch auf die Frage nach der Orthogonalität der Aspekte eingegangen. Eine nähere Betrachtung der Aspekte „Topologie“ und „Zugang“ könnte im Zuge weiterer Portierungen auf andere Parallelrechner erfolgen, sofern ihnen dabei eine tragende Rolle zukommt.

6.1 Prozessor

Der Aspekt „Prozessor“ beschreibt, wie der eingesetzte Prozessor Programme abarbeitet. Er wird vor allem durch den Register- und Befehlssatz des Prozessors sowie dessen Betriebsarten geprägt. Auch die Ausnahmebehandlung bei PowerPC-Prozessoren und der von Transputern realisierte Mehrprozeßbetrieb fallen unter diesen Punkt. Erweiterungen des Prozessors wie etwa eine Adreßtransformationseinheit oder die bei Transputern vorhandenen Kommunikationsleitungen hingegen werden nicht vom Aspekt „Prozessor“ abgedeckt, da sie die Abarbeitung von Befehlsströmen nur indirekt betreffen. Im Gegensatz dazu ist eine Erweiterung eines Prozessors um eine Einheit zum Rechnen mit Gleitkommazahlen dem Aspekt „Prozessor“ zuzuordnen.

Der Registersatz eines Prozessors beeinflußt in erster Linie die Verknüpfungs- und Ablaufumgebung eines Prozesses. Die Verknüpfungsumgebung besteht aus den Registern, die bei jeder Prozeßumschaltung gesichert werden müssen, da sie von einem Prozeß verändert werden können. Die Ablaufumgebung enthält die

Register, die zwar nicht gesichert, aber neu geladen werden müssen. Ein Beispiel dafür ist das Prozessorzustandsregister in PowerPC-Prozessoren, durch das die Betriebsart des Prozessor festgelegt wird. Verknüpfungs- und Ablaufumgebung müssen bei der Erzeugung eines Prozesses zumindest teilweise initialisiert werden, auch hier ergibt sich also eine Prozessorabhängigkeit.

Der Befehlssatz beeinflußt sämtliche Teile der Systemsoftware, die in der Assembler-Sprache formuliert sind. Um die daraus resultierenden Änderungen auf ein Mindestmaß zu beschränken, sollte die Systemsoftware nur an den Stellen Assembler-Befehle enthalten, die auch von anderen Hardware-Abhängigkeiten betroffen sind.

Sowohl Befehls- als auch Registersatz beeinflussen die Realisierung des Kerneintritts. Der Kerneintritt kann durch spezielle Befehle eines Prozessors unterstützt werden, die zum Beispiel die Betriebsart ändern. Die Übergabe von Parametern bei Kernaufrufen kann in Registern erfolgen, falls davon genügend zur Verfügung stehen. Ist dies nicht der Fall, so müssen Parameter auf dem Stapel oder in Form von Nachrichtenpaketen übergeben werden.

Bei der Inbetriebnahme wird der Prozessor durch ein Hardware-Signal in einen definierten Zustand gebracht. Die Adresse, an der die Befehlsabarbeitung beginnt, sowie die Betriebsart, in der die ersten Befehle abgearbeitet werden, sind ebenfalls von Prozessor zu Prozessor unterschiedlich.

6.2 Adreßtransformation

Der Aspekt „Adreßtransformation“ (AT) ist nicht orthogonal zum Aspekt „Prozessor“, da für einen Prozessortyp meist nur eine bestimmte AT-Einheit angeboten wird. Allerdings muß diese nicht unbedingt verwendet werden. Für eine komplexe AT-Einheit wie die des PowerPC-Prozessors wiederum ergeben sich unterschiedliche Einsatzmöglichkeiten. Zudem existieren nur wenige grundlegende AT-Mechanismen, deren Realisierungen sich bei verschiedenen Prozessoren nur in Details unterscheiden. Zum Beispiel wurde die AT-Einheit des PowerPC-Prozessors von Motorola in ähnlicher Form bereits bei Prozessoren der 88k-Familie eingesetzt. Aus diesen Gründen scheint es gerechtfertigt, die Adreßtransformation als eigenen Aspekt einer Hardware-Umgebung zu betrachten.

Die Adreßtransformation beeinflußt die Ablaufumgebung von Prozessen und damit sowohl die Prozeßumschaltung als auch die Prozeßerzeugung. Eine AT-Einheit wird über Prozessor- oder abgesetzte Hardware-Register angesprochen. Der Inhalt dieser Register bildet meist den umfangreichsten Teil der Ablaufumgebung.
In den Aufgabenbereich der Adreßtransformation fällt auch der Schutz von Speicherbereichen vor illegalen Zugriffen. Bei einem illegalen Zugriff wird üblicherweise eine Ausnahmebehandlung eingeleitet, die dem Aspekt „Prozessor“ zugeordnet ist. Die Parameter der Ausnahmebehandlungsprozedur sind von beiden Aspekten abhängig.

Zur Realisierung von Adreßraumausbrüchen muß der Speicherschutz umgangen werden. Ein Adreßraumausbruch tritt zum Beispiel beim Kerneintritt auf, wenn die Parameter eines Kernaufrufs nicht in Prozessorregistern übergeben, sondern im Adreßraum des aufrufenden Prozesses hinterlegt werden.
Die Verwendung einer Adreßtransformationseinheit kann zu Problemen führen, wenn externe Hardware auf den Speicher zugreift. Diese Zugriffe erfolgen üblicherweise nicht über die AT-Einheit. In diesem Fall verwendet externe Hardware also *Speicheradressen*, keine *Programmadressen*.

Eine Adreßtransformationseinheit benötigt Informationen über die durchzuführenden Adreßabbildungen und die zu prüfenden Zugriffsrechte. Diese Informationen müssen bei der Inbetriebnahme zur Verfügung gestellt und bei Bedarf aktualisiert werden.

Die Verwaltung des physikalischen Speichers ist stark vom verwendeten Adreßtransformationsmechanismus abhängig. In Systemen ohne AT oder mit segmentbasierter AT kann externer Verschnitt auftreten. Um diesen klein zu halten, müssen aufwendige Speicherverwaltungsverfahren eingesetzt werden. Bei einer seitenbasierten AT tritt dagegen kein externer Verschnitt auf, so daß ein einfacheres Verfahren eingesetzt werden kann. Außerdem muß die Stückgröße bei der Vergabe von physikalischem Speicher an die Seiten- oder Segmentgrößen einer AT-Einheit angepaßt werden.

6.3 Kommunikation

Der Aspekt „Kommunikation“ betrifft die Leitungen zwischen den Knoten eines Parallelrechners und wie sie auf einem Knoten angesprochen werden können. Leitungen werden meist durch vom Prozessor abgesetzte Kommunikationshardware realisiert. Transputer mit ihren integrierten Kommunikationsleitungen bilden einen Sonderfall. Aber auch Transputer-Prozessoren können externe Kommunikationshardware ansprechen, die Leitungen auf eine andere Art realisiert. Deshalb kann der Aspekt „Kommunikation“ als orthogonal zum Aspekt „Prozessor“ betrachtet werden. Ein unmittelbarer Bezug zum Aspekt „Adreßtransformation“ besteht nicht, auch hier ist also Orthogonalität gegeben.

Während des Betriebs eines Parallelrechners müssen Datenpakete zwischen den Knoten verschickt werden. Der Mechanismus zum Abschicken oder Entgegennehmen dieser Datenpakete ist von der Realisierung der Leitungen abhängig. Sie kann auch den Aufbau der Pakete beeinflussen, wenn etwa Ziel und Länge eines Pakets in einer bestimmten Form gespeichert sein müssen.
Zu Beginn der Inbetriebnahme ist auf Nachbarknoten noch kein Kern installiert. Der Kommunikationspartner bei der Übertragung des ersten Programms ist also ein anderer als während des Betriebs. Auch bei der Inbetriebnahme können deshalb Änderungen notwendig sein. Gleiches gilt für die Kommunikation mit dem Wirtsrechner, falls ein solcher vorhanden ist.

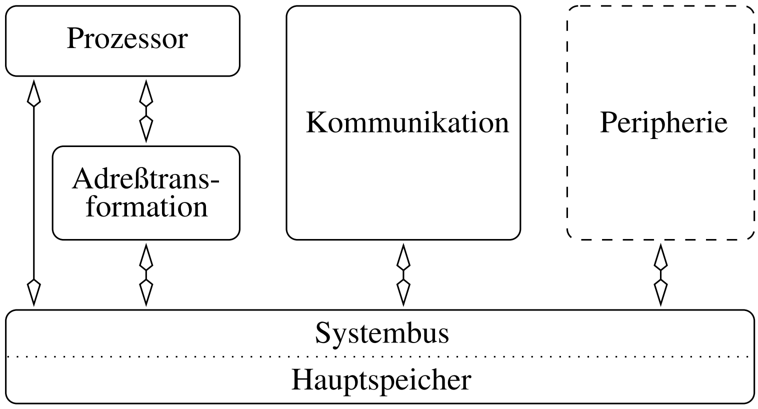

Abbildung 6.1: Aspekte im Zusammenhang

6.4 Werkzeuge

Der Aspekt „Werkzeuge“ ist sicher nicht orthogonal zum Aspekt „Prozessor“. Werkzeuge transformieren Programme so, daß sie von einem bestimmten Prozessor ausgeführt werden können. Sie dienen also dazu, den Einfluß des Aspekts „Prozessor“ zu vermindern. Für häufig eingesetzte Prozessoren existieren viele Werkzeugsätze, die verschiedene Programmiersprachen unterstützen. Aber auch bei Werkzeugsätzen mit gleicher Quellsprache ergeben sich Unterschiede, wie in Kapitel 5 erläutert wurde. Ein Aspekt „Programmiersprache“ wäre also zu allgemein.

Mit der Wahl der Entwicklungswerkzeuge für Programme wird auch ein Format für Programmdateien festgelegt. Dieses beeinflußt das Laden und Ausführen von Programmen, sowohl während des Betriebs als auch bei der Inbetriebnahme, denn Systemdienstgeber und der Systemkern sind ebenfalls Programme.

Die vom Übersetzer verwendeten Konventionen für Prozeduraufrufe haben nicht nur Auswirkungen auf den Aufruf von Assembler-Prozeduren. Es gibt zwei wichtige Bereiche, die ebenfalls davon betroffen sind: zum einen der Kerneintritt, zum anderen die Übergabe der Aufrufparametern beim Erzeugen eines Prozesses.

6.5 Überblick

Abbildung 6.1 zeigt den Zusammenhang zwischen einigen der vorgestellten Aspekte. Die Aspekte „Prozessor“, „Adreßtransformation“ und „Kommunikation“ beschreiben *aktive* Teile der Umgebung eines Betriebssystems, also Komponenten, die für die Abarbeitung von Programmen zuständig sind. Das verbindende Element zwischen diesen Komponenten ist der Systembus eines Rechners, über den externe Ereignisse an den Prozessor gemeldet und Zugriffe auf den Hauptspeicher durchgeführt werden.

Der Prozessor kann den Systembus entweder direkt oder über eine Adreßtransformationseinheit nutzen, während Kommunikationshardware immer direkt auf den Systembus zugreift. Wenn ein Rechner über zusätzliche Peripherie, zum Beispiel eine Grafikkarte oder eine Festplatte, verfügt, so wird auch diese Peripherie über den Systembus angesteuert.

Die Aspekte „Topologie“ und „Werkzeuge“ sind in der Abbildung nicht aufgeführt. Beides sind *passive* Aspekte, die zwar berücksichtigt werden müssen, aber die Konzeption eines Betriebssytems nur in geringem Ausmaß beeinflussen. Man könnte sie daher als *Meta-Aspekte* bezeichnen.
Die aktiven Aspekte dagegen beschreiben die ausführenden Einheiten, also die Knoten eines Parallelrechners. Deshalb ist die Ähnlichkeit von Abbildung 6.1 mit der Blockdarstellung einer Rechnerarchitektur durchaus kein Zufall. Die Rechnerarchitektur ist ein wichtiger Bestandteil der Umgebung eines Betriebssystems und muß sich folglich in der Charakterisierung dieser Umgebung widerspiegeln.

Abbildung 6.1 enthält zwei Komponenten, die bisher nicht als Aspekte in Erscheinung getreten sind, nämlich die Peripherie und das verbindende Element Systembus/Hauptspeicher. Die vorgestellte Charakterisierung ist also keineswegs vollständig. Sie erwies sich jedoch zur Bewältigung der gestellten Aufgabe als hinreichend. Ergänzungen oder Verfeinerungen können in Zuge weitere Portierungen vorgenommen werden, falls sich dies als notwendig erweist.

Kapitel 7

Speicherverwaltung

Die Aufgabe der Speicherverwaltung ist es, *logischen* und *physikalischen Speicher* zu verwalten. Wie in Kapitel 2 beschrieben ist, unterstützt COSY zur Verwaltung logischen Speichers drei Kernobjekttypen. Abbildung 7.1 verdeutlicht noch einmal den Zusammenhang zwischen diesen Objekttypen. COSY-Adreßbereiche (AB) entsprechen Teilen des Hauptspeichers. Adreßräume (AR) sind Umgebungen, in denen auf Speicher zugegriffen werden kann. Jede Einblendung (E) erlaubt in genau einem Adreßraum den Zugriff auf genau einen Adreßbereich.

Dieses Kapitel beschäftigt sich mit der Realisierung der genannten drei COSY-Kernobjekttypen und mit der Verwaltung des physikalischen Speichers. Abschnitt 7.1 gibt einen Überblick über die Realisierung mit Transputern. Abschnitt 7.2 beschreibt die Realisierung mit PowerPC-Prozessoren und die Änderungen, die an den Kernobjekttypen und der physikalischen Speicherverwaltung vorgenommen wurden.

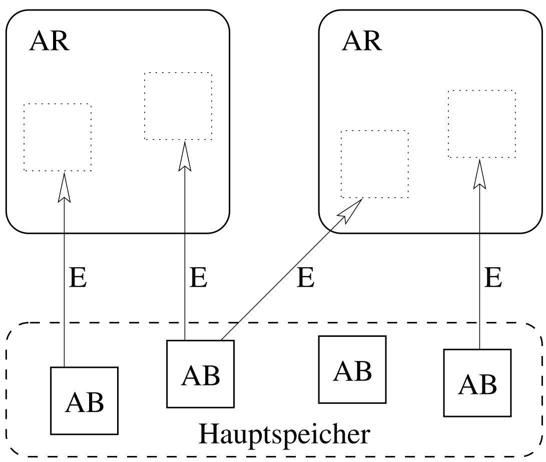

Abbildung 7.1: Speicherverwaltung in COSY

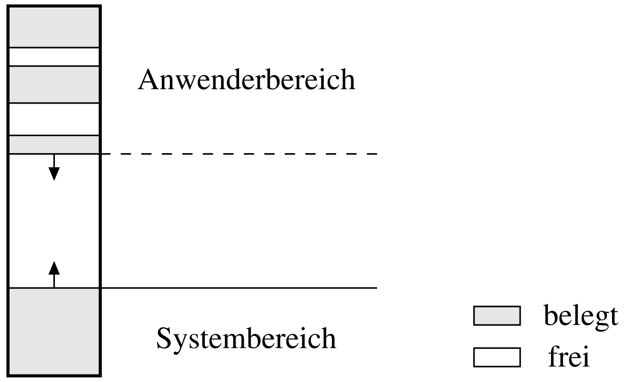

Abbildung 7.2: Speicherverwaltung ohne Adreßtransformation

7.1 Realisierung auf dem Transputer

Die Speicherverwaltung muß insbesondere an den Aspekt „Adreßtransformation“ angepaßt werden. Transputer verfügen nicht über eine Adreßtransformationseinheit. Alle COSY-Adreßräume müssen deshalb auf einen globalen Adreßraum, den *Speicheradreßraum*, abgebildet werden.

Der COSY-Kern unterteilt den Hauptspeicher in zwei Bereiche, einen Systembereich und einen Anwenderbereich. Der Systembereich ist ein zusammenhängendes Speicherstück, das sich am Anfang des Hauptspeichers befindet. In ihm liegen der Code und die Daten des Kerns. Bei Bedarf kann der Systembereich vergrößert werden, sofern der Speicher am Anfang des Anwenderbereichs nicht belegt ist.
Die unbelegten Stücke des Anwenderbereichs sind, nach ihrer Lage geordnet, in einer Freispeicherliste verzeichnet. Der Speicher des Anwenderbereichs wird in Einheiten von 128 Byte vom Ende her vergeben. Durch diese Belegungsstrategie wird bevorzugt Speicher belegt, der nicht für eine Vergrößerung des Systembereichs benötigt wird. Abbildung 7.2 zeigt eine Speicherbelegung, die für diese Verfahren typisch ist.

Jeder COSY-Adreßbereich ist mit einem zusammenhängenden Speicherstück im Anwenderbereich unterfüttert. Die Verwaltungsdaten für Adreßbereiche liegen, ebenso wie die aller anderen Kernobjekttypen, im Systembereich. Ein Adreßbereich besitzt in COSY die Attribute *Lage* und *Größe*, die den zugehörigen physikalischen Speicherbereich beschreiben. Keines dieser Attribute kann verändert werden.
Die Einblendung eines Adreßbereichs in einen Adreßraum besitzt nur ein einziges Attribut, nämlich die *Lage*. Da auf Transputern Programm- und Speicheradressen identisch sind, ist die Lage einer Einblendung in einem Adreßraum identisch mit der Lage des eingeblendeten Adreßbereichs im physikalischen Adreßraum. Sie kann demnach nicht verändert werden.

Ein zuverlässiger Speicherschutz kann auf Transputern nicht effizient realisiert werden. Programmadressen sollten nur in ihrem Adreßraum gültig sein. Da je-

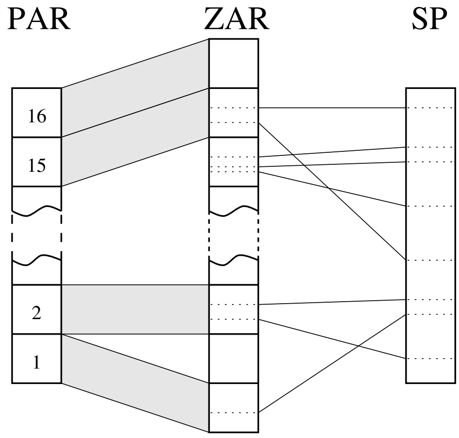

Abbildung 7.3: Zweistufige Adreßtransformation

doch alle logischen Adreßräume im Speicheradreßraum realisiert sind, werden unerlaubte Zugriffe auf Daten in anderen Adreßräumen nicht verhindert. Dies bringt schwerwiegende Nachteile mit sich. Zum Beispiel ist ein Mehrbenutzerbetrieb nur eingeschränkt möglich, da fehlerhafte Programme sowohl Programme anderer Benutzer als auch den Kern beeinträchtigen können.

7.2 Realisierung auf dem PowerPC

PowerPC-Prozessoren verfügen über eine Adreßtransformationseinheit, die sowohl ein- als auch zweistufige Transformationen durchführen kann. Als Grundlage für das weitere Vorgehen muß zunächst entschieden werden, auf welche Weise, wenn überhaupt, die Adreßtransformationseinheit eingesetzt werden soll.

Adreßräume dienen dazu, Ablaufumgebungen für Programme bereitzustellen. Transputer-Programme sind verschiebbar, es spielt also keine Rolle, an welcher Stelle im Adreßraum ein Adreßbereich eingeblendet ist. Im Gegensatz dazu erzeugt der verwendete GNU-Übersetzer PowerPC-Programme, die nicht verschiebbar sind. Das bedeutet, daß entweder die Adreßbereiche an vorgegebenen Stellen eines Adreßraums eingeblendet werden müssen oder beim Laden eines Programms Änderungen am Code vorzunehmen sind. Änderungen am Code würden jedoch das sogenannte *code sharing* verhindern, also das gemeinsame Benutzen des selben Codes von mehreren Instanzen eines Programms. Deshalb ist es angebracht, die Adreßtransformationseinheit des PowerPC-Prozessors einzusetzen.
Ein Programm benötigt mindestens drei Adreßbereiche, je einen für den Code, die statischen Daten und den Stapel. Zusätzlich kann Speicher für dynamische Daten angefordert werden, für den weitere Adreßbereiche benötigt werden. Die in Abschnitt 4.1.3 beschriebe einstufige Adreßtransformation ist nicht flexibel genug, um dynamische Speicheranforderungen effizient zu bewältigen. Deshalb wird für Cosy die zweistufige Adreßtransformation eingesetzt.

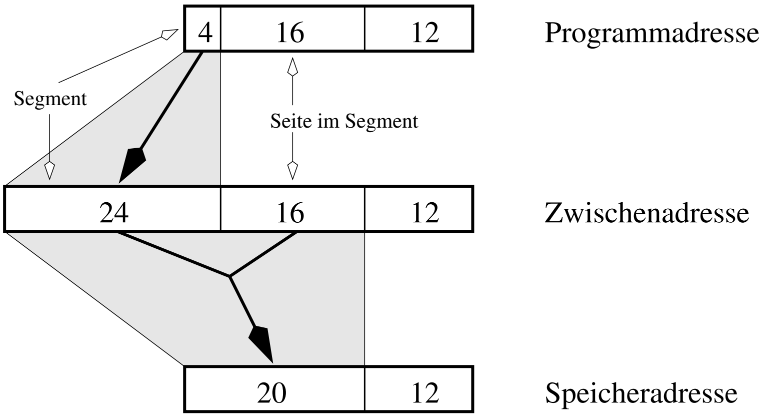

Abbildung 7.4: Zweistufige Adreßtransformation im Detail

Der folgende Abschnitt 7.2.1 beschreibt die zweistufige Adreßtransformation von PowerPC-Prozessoren. Im Abschnitt 7.2.2 werden die Möglichkeiten zum Einsatz der Adreßtransformation diskutiert. Die Realisierung der COSY-Kernobjekttypen mit PowerPC-Prozessoren ist in Abschnitt 7.2.3 erläutert. Abschließend beschäftigt sich Abschnitt 7.2.4 mit der Verwaltung des physikalischen Speichers.

7.2.1 Zweistufige Adreßtransformation

Abbildung 7.3 auf Seite 57 zeigt das Prinzip der zweistufigen Adreßtransformation, Abbildung 7.4 die konkrete Realisierung auf PowerPC-Prozessoren. Die Zahlen in Abbildung 7.4 geben die Breite der Adreßteile in Bit an.
Ausgangspunkt ist eine 32 Bit breite Adresse im Programmadreßraum (PAR). Der Programmadreßraum ist in 16 *Segmente* zu je 256 Megabyte unterteilt. Die höherwertigen 4 Bit der Adresse bestimmen das Segment, in das die Adresse fällt. Jedes Segment des Programmadreßraums wird auf ein frei wählbares Segment des Zwischenadreßraums (ZAR) abgebildet, der in 2^{24} Segmente unterteilt ist. Die relative Adresse innerhalb eines Segmentes, also der Wert der niederwertigen 28 Bit der Adresse, bleibt erhalten. Nach der ersten Stufe der Transformation liegt also eine 52 Bit breite Adresse im Zwischenadreßraum (Zwischenadresse) vor, die sich aus einer 24 Bit breiten Segmentnummer und einer 28 Bit breiten Adresse innerhalb des Segmentes zusammensetzt.
In der zweiten Stufe wird die Zwischenadresse in eine 32 Bit breite physikalische Adresse im Speicheradreßraum (SP) transformiert. Jedes Segment des Zwischenadreßraums ist in 2^{16} *Seiten* zu je 4 Kilobyte unterteilt. Die höherwertigen 16 Bit der Adresse innerhalb des Segmentes bestimmen die Seite des Segmentes, auf die die Adresse verweist. Zusammen mit der 24 Bit breiten Segmentnummer ergibt sich eine 40 Bit breite Seitennummer im Zwischenadreßraum. Jeder dieser 2^{40} Seiten kann eine beliebige Seite des Speicheradreßraums zugeordnet werden. Die relative Adresse innerhalb der Seite, also der 12 Bit breite, niederwertige Teil der Adresse, bleibt erhalten.

Die Informationen für die erste Stufe, also die Transformation von Programmadressen in Zwischenadressen, werden in speziellen Registern des PowerPC-Prozessors bereitgestellt. Für jedes der 16 Segmente des Programmadreßraums existiert ein *Segmentregister*, in dem die Nummer des entsprechenden Segmentes des Zwischenadreßraums abgelegt wird.
Die Abbildung vom Zwischen- in den Speicheradreßraum wird durch eine Seitentabelle definiert, die im Hauptspeicher abgelegt ist. Diese Seitentabelle ist als *Hashtabelle* organisiert. Ihre Einträge sind in Gruppen mit je 8 Einträgen aufgeteilt. Der PowerPC-Prozessor bestimmt aus der Segment- und Seitennummer einer Adresse des Zwischenadreßraums zwei Hashwerte. Jeder Hashwert legt eine Gruppe der Seitentabelle fest, in der die Transformation für die entsprechende Seite abgelegt sein kann. Ist in keiner dieser beiden Gruppen eine Transformation für die Seite des Zwischenadreßraums definiert, so erfolgt eine Ausnahmebehandlung.
Die Transformation für eine Seite kann also in einem von 16 verschiedenen Einträgen in der Seitentabelle abgelegt werden. Die Anzahl der Gruppen in der Seitentabelle ist eine Zweierpotenz im Bereich von 2^{10} bis 2^{19}. Der Speicherbedarf der Tabelle liegt zwischen 64 Kilobyte und 32 Megabyte.

Zugriffsrechte für Speicherbereiche können für jede Seite festgelegt werden. Dabei wird zwischen bevorzugten und nicht bevorzugten Zugriffen unterschieden. Die Zugriffsrechte können zum Beispiel so festgelegt werden, daß nur bevorzugte Schreibzugriffe erfolgen dürfen.
Mit dem Wert in einem Segmentregister wird festgelegt, ob Zugriffe bevorzugt sind oder nicht. Die Festlegung erfolgt pro Segment getrennt für den Anwender- und den Systemmodus des Prozessors. Üblicherweise werden Zugriffe im Anwendermodus als nicht bevorzugt und solche im Systemmodus als bevorzugt definiert. Dies ist jedoch nicht zwingend der Fall.

7.2.2 Einsatzmöglichkeiten

Um einen Überblick über die Einsatzmöglichkeiten der zweistufigen, segment- und seitenbasierten Adreßtransformation zu gewinnen, gilt es, die Extremfälle zu betrachten. Dies sind der Einsatz als segmentorientierte beziehungsweise rein seitenbasierte Adreßtransformation. Eine angemessene Einsatzmöglichkeit muß zwischen diesen Extremfällen gesucht werden.

In Abbildung 7.5 auf Seite 60 ist der Einsatz als segmentorientiertes Verfahren skizziert. Jedem Adreßbereich ist ein eigenes Segment des Zwischenadreßraums (ZAR) zugeordnet. Um einen Adreßbereich in einen Programmadreßraum (PAR) einzublenden wird ein Segment des Programmadreßraums auf das entsprechende Segment des Zwischenadreßraums abgebildet. Die Einträge in der Seitentabelle können beim Erzeugen des Adreßbereichs angelegt werden. Das gemeinsame Benutzen eines Adreßbereichs in mehreren Adreßräumen ist möglich, wenn je ein Segment der betroffenen Adreßräume auf das entsprechende Segment des Zwischenadreßraums abgebildet wird.

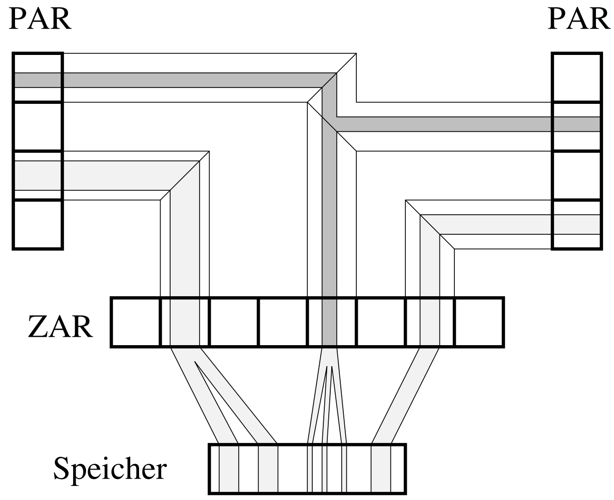

Abbildung 7.5: Segmentorientierter Einsatz

Der Vorteil dieses Verfahrens besteht darin, daß in der Seitentabelle nur wenige Einträge benötigt werden. Als Nachteil ist festzuhalten, daß in einen Adreßraum nur 16 Adreßbereiche eingeblendet sein können. Obwohl dieser Wert für viele Zwecke ausreicht, gibt es Situationen, in denen mehr Adreßbereiche in einem Adreßraum benötigt werden. Außerdem können Adreßbereiche bei diesem Verfahren keine Segmentgrenzen überschreiten.
Ein weiterer Nachteil ist, daß die Zugriffsrechte durch die Einträge in der Seitentabelle für Adreßbereiche festgelegt werden müssen. In COSY kann auf einen Adreßbereich über verschiedene Einblendungen zugegriffen werden. Deshalb sollte es möglich sein, Zugriffsrechte pro Einblendung festzulegen.
Beim segmentorientierten Verfahren kann pro Einblendung mittels des benutzten Segmentregisters festgelegt werden, ob ein Zugriff bevorzugt ist oder nicht. Wie in Abschnitt 7.2.1 beschrieben wurde, können Zugriffsrechte für Einträge in der Seitentabelle so eingerichtet werden, daß Schreibzugriffe bevorzugt sein müssen. In diesem Fall kann über eine Einblendung nur dann ein schreibender Zugriff erfolgen, wenn die Einblendung Zugriffe als bevorzugt definiert. Auf diese Weise wird das gewünschte Verhalten erzielt.

Anmerkung. *Die segmentorientiert eingesetzte, zweistufige Adreßtransformation unterscheidet sich in einigen Punkten von einer rein segmentbasierten Adreßtransformation. Zum Beispiel muß der Speicher eines Adreßbereichs nicht zusammenhängend angefordert werden. Andererseits ist die Lage eines Adreßbereichs im Programmadreßraum eingeschränkt, weil keine der festgelegten Segmentgrenzen überschritten werden dürfen. Bei einer rein segmentbasierten Adreßtransformation kann sowohl die Größe von Segmenten als auch ihre Lage im Programmadreßraum verändert werden.*

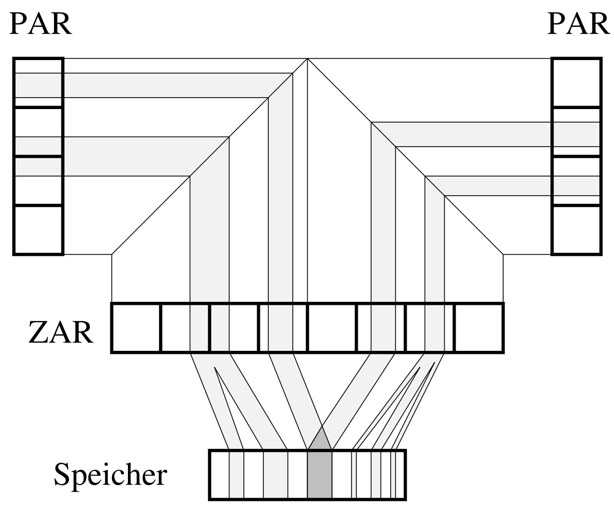

Abbildung 7.6: Seitenbasierter Einsatz

Beim Einsatz als seitenbasiertes Verfahren wird jedem Programmadreßraum ein gleich großer Teil des Zwischenadreßraums fest zugeordnet. Für jede Einblendung müssen neue Einträge in der Seitentabelle erzeugt werden. In Abbildung 7.6 ist das Vorgehen skizziert.
Der Vorteil dieser Variante besteht darin, daß beliebig viele Adreßbereiche in einen Adreßraum eingeblendet werden können. Die Lage von Adreßbereichen in einem Programmadreßraum unterliegt keinen Einschränkungen durch Segmentgrenzen. Für jede Einblendung lassen sich eigene Zugriffsrechte festlegen, da eigene Einträge für die eingeblendeten Seiten existieren.
Entscheidender Nachteil dieses Verfahrens ist, daß die Anzahl der notwendigen Einträge in der Seitentabelle potentiell unbeschränkt ist. Wenn mehr Einträge definiert sind, als die Seitentabelle aufnehmen kann, treten auch bei erlaubten Zugriffen Ausnahmen auf. In der Ausnahmebehandlungsprozedur muß dann ein Eintrag in der Seitentabelle durch den gerade benötigten Eintrag ersetzt werden. Obwohl dies technisch kein Problem darstellt, würde der Aufwand für die Implementierung dieses Verfahrens den zeitlichen Rahmen dieser Diplomarbeit sprengen.

Der Vorteil der segmentbasierten Arbeitsweise liegt in der einfachen Realisierung von gemeinsam benutzten Adreßbereichen. Das seitenbasierte Verfahren hingegen schränkt die Anzahl der Adreßbereiche pro Adreßraum nicht ein, was für exklusiv genutzte Adreßbereiche genutzt werden kann. Es gilt deshalb, gemeinsam genutzte, *öffentliche* von exklusiv genutzten, *privaten* Adreßbereichen zu unterscheiden und die Adreßtransformationseinheit entsprechend einzusetzen. Der nächste Abschnitt beschreibt die dazu notwendigen Änderungen an der COSY-Speicherverwaltung.

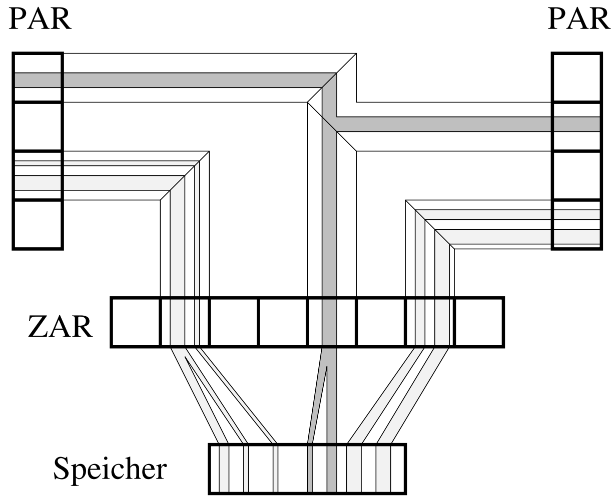

Abbildung 7.7: Einsatz durch COSY

7.2.3 Logischer Speicher

Grundlage für den Einsatz der PowerPC-Adreßtransformationseinheit ist die Unterscheidung zwischen *privaten* und *öffentlichen* Adreßbereichen. Private Adreßbereiche werden über die Seitentabelle in höchstens einen Adreßraum eingeblendet. Öffentliche Adreßbereiche hingegen können über Segmentregister in beliebig viele Adreßräume eingeblendet werden. Die Anzahl der in einen Adreßraum eingeblendeten öffentlichen Adreßbereiche ist jedoch durch die Anzahl der Segmentregister auf höchstens 16 beschränkt.
In Abbildung 7.7 ist diese Verfahren skizziert. Die heller dargestellten, privaten Adreßbereiche können nur in einen Programmadreßraum (PAR) eingeblendet werden, jedoch mehrere in das gleiche Segment. Auf den dunkleren, öffentlichen Adreßbereich wird über ein eigenes Segment des Zwischenadreßraums (ZAR) zugegriffen. Die Einblendung in mehrere Adreßräume erfordert deshalb keine zusätzlichen Einträge in der Seitentabelle.

Die Segmente des Zwischenadreßraums sind Betriebsmittel, die zur Realisierung von Einblendungen benötigt werden. Der COSY-Kern vergibt diese Segmente implizit bei der Erzeugung von Kernobjekten. Der größte Bedarf an Segmenten ergibt sich bei Kernobjekten des Typs Adreßraum und liegt bei höchstens 16 Segmenten. In der bisherigen Implementierung des Kerns können auf einem Knoten höchstens 2^{16} Kernobjekte existieren. Der Zwischenadreßraum besteht aus 2^{24} Segmenten und ist somit groß genug, um für jedes Kernobjekt 16 Segmente reservieren zu können. Für die Realisierung der meisten Kernobjekttypen werden keine oder nur wenige Segmente des Zwischenadreßraums benötigt. Trotzdem ist die Reservierung sinnvoll, da sich eine zusätzliche Verwaltung der Segmente des Zwischenadreßraums erübrigt.

Als Zwischensegment für den Zugriff auf *private* Adreßbereiche wird eines der Segmente verwendet, die für den jeweiligen *Adreßraum* reserviert sind. Die Adreßtransformation für den Zugriff auf *öffentliche* Adreßbereiche erfolgt dage-

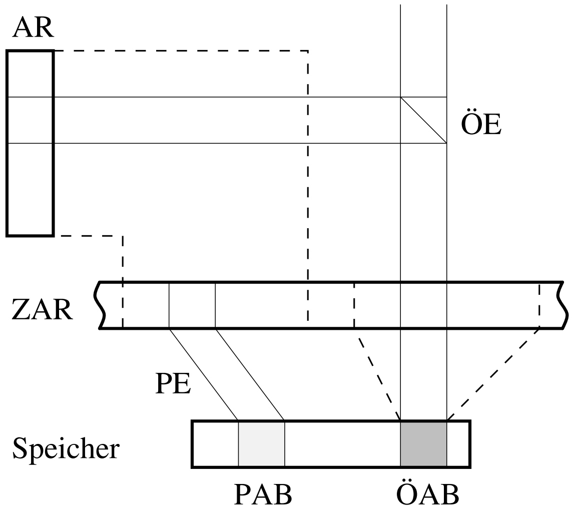

Abbildung 7.8: Nutzung von Zwischensegmenten

gen über ein Zwischensegment, das dem entsprechenden *Adreßbereich* zugeordnet ist. In Abbildung 7.8 ist das Vorgehen graphisch dargestellt.
Einem Adreßraum (AR) wird ein entsprechend großer Bereich des Zwischenadreßraums (ZAR) zugeordnet. Die Einblendung (PE) eines privaten Adreßbereichs (PAB) erfolgt über diesen Teil des Zwischenadreßraums. Für einen öffentlichen Adreßbereich (ÖAB) wird ebenfalls ein Bereich des Zwischenadreßraums reserviert. Einträge in der Seitentabelle erlauben den Zugriff auf den Adreßbereich über eines der dafür reservierten Zwischensegmente. Die Einblendung (ÖE) eines öffentlichen Adreßbereichs erfolgt über ein Segmentregister. Eines der Segmente des Cosy-Adreßraums wird auf das Zwischensegment abgebildet, über das der öffentliche Adreßbereich zugänglich ist. Dies kann in beliebig vielen Adreßräumen geschehen, ohne daß zusätzliche Einträge in der Seitentabelle notwendig werden.

Anmerkung. *Da öffentliche Adreßbereiche über* ein *Segment des Zwischenadreßraums eingeblendet werden, darf ihre Größe 256 Megabyte nicht überschreiten. Der Hauptspeicher auf einem Knoten eines PowerXplorer ist jedoch nur 8 Megabyte groß und* Cosy *unterstützt keinen virtuellen Speicher. Deshalb spielt diese Beschränkung keine Rolle. Daß öffentliche Adreßbereiche keine Segmentgrenze überschreiten dürfen, ist hingegen eine Einschränkung, der beim Erzeugen oder Laden von Programmdateien Rechnung getragen werden muß.*

Die Unterscheidung zwischen privaten und öffentlichen Adreßbereichen könnte implizit über die Anzahl der Einblendungen erfolgen. Dies würde es erforderlich machen, für die zweite Einblendung eines dann öffentlichen Adreßbereichs die vorhandenen Einträge aus der Seitentabelle zu löschen und neue zu erzeugen. Zudem müßte sich die Lage von Adreßbereichen im ersten Adreßraum ändern, wenn dort andere Adreßbereiche in das gleiche Segment eingeblendet sind.

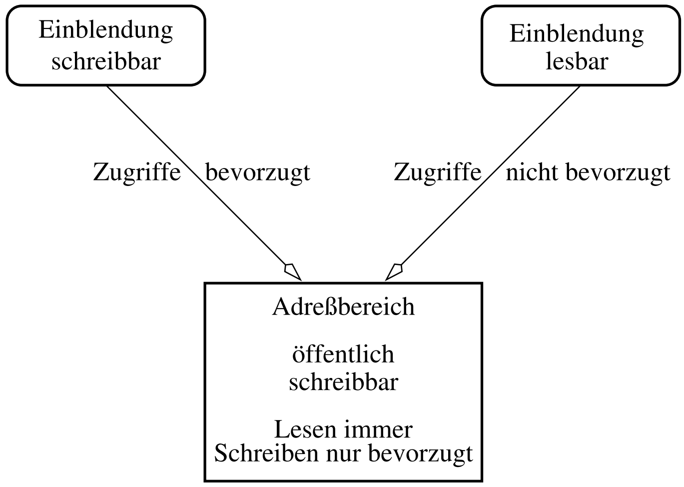

Abbildung 7.9: Speicherschutz öffentlicher Adreßbereiche

Um diesen Fall zu vermeiden, gelten Adreßbereiche solange als privat, bis sie über einen Kernaufruf *veröffentlicht* werden. Diese „Verfügbarkeit" eines Adreßbereichs wird in Form eines Attributs gespeichert, das zu den schon vorhandenen Attributen „Lage" und „Größe" hinzugefügt wird. Das neue Attribut kann die Werte „privat" und „öffentlich" annehmen.

Um den durch die Adreßtransformationseinheit ermöglichten Speicherschutz zu unterstützen, erhalten Adreßbereiche und Einblendungen ein weiteres Attribut, das Zugriffsrechte festlegt. Dieses kann die Werte „lesbar" und „schreibbar" annehmen. Der Inhalt eines schreibbaren Adreßbereichs kann sowohl ausgelesen als auch verändert, der eines lesbaren Adreßbereichs nur ausgelesen werden. Analog dazu sind schreibende Zugriffe auf einen Adreßbereich über eine Einblendung mit dem Attribut „lesbar" nicht erlaubt. Die Zugriffsrechte einer Einblendung können also die Zugriffsrechte des eingeblendeten Adreßbereichs einschränken, aber nicht erweitern.
Für Einblendungen privater Adreßbereiche existieren eigene Einträge in der Seitentabelle, mit denen die Zugriffsrechte festgelegt werden. Dagegen erfolgen Einblendungen öffentlicher Adreßbereiche über Segmentregister, wobei nur Zugriffsprivilegien festgelegt werden können. Abbildung 7.9 zeigt, wie der Speicherschutz mit diesen Vorgaben realisiert werden kann. Einträge in der Seitentabelle, die für schreibbare, öffentliche Adreßbereiche erzeugt werden, definieren schreibende Zugriffe als bevorzugt und lesende Zugriffe als nicht bevorzugt. Bei nur lesbaren öffentlichen Adreßbereichen werden auch bevorzugte Schreibzugriffe unterbunden. Auf diese Art lassen sich die Rechte für den Zugriff auf einen Adreßbereich für jede Einblendung separat durch die Wahl der Zugriffsprivilegien einschränken.

Adreßbereiche sind mit einem Attribut „Anfang" versehen, das auf Rechnern ohne Adreßtransformation den Anfang des Bereichs im Hauptspeicher angibt. Durch den Einsatz der Adreßtransformation geht diese Bedeutung verloren, da der Hauptspeicher nicht mehr in zusammenhängenden Stücken vergeben werden

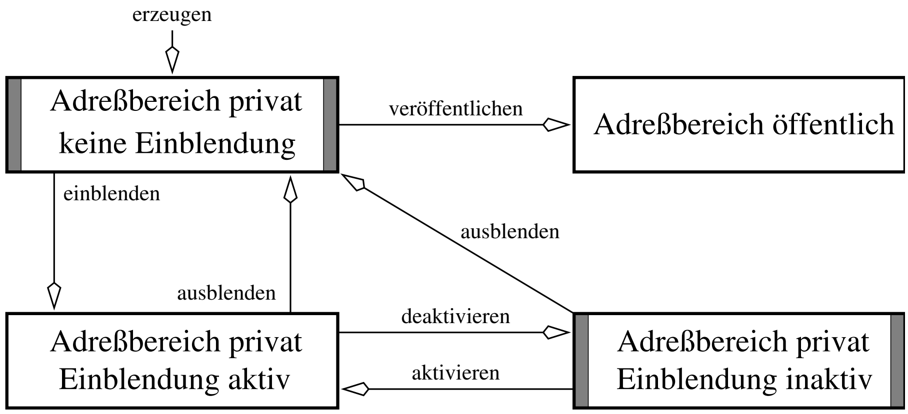

Abbildung 7.10: Zustände eines Adreßbereichs

muß. Statt dessen kann es verwendet werden, um die Stelle im Programmadreßraum anzugeben, an der der Adreßbereich eingeblendet werden soll. In Systemen ohne Adreßtransformation ist die Bedeutung implizit die gleiche, da auch dort eine Einblendung nur an der Stelle erfolgen kann, die von diesem Attribut festgelegt wird.
Bei einem neu erzeugten Adreßbereich besitzt das Attribut „Anfang“ den Wert „irgendwo“. Der Kern ermittelt dann beim Einblenden einen Bereich im jeweiligen Programmadreßraum, an dem eine Einblendung ohne Überlappungen mit anderen Einblendungen möglich ist.

Beispiel. *Das Festlegen der Anfangsadresse eines Adreßbereichs ist beim Laden von Programmen nützlich. Eine Programmdatei enthält Angaben über die Größe, den Inhalt und die Lage von Speicherbereichen im Programmadreßraum Der Codebereich eines Programms kann von mehreren Instanzen gemeinsam genutzt und dazu in mehreren Adreßräumen eingeblendet werden. Statt nun bei jeder Einblendung die Lage neu anzugeben, ist es möglich, sie durch das entsprechende Attribut des Adreßbereichs einmal für alle Einblendungen festzulegen.*

Einträge in der Seitentabelle der Adreßtransformationseinheit sind von den Attributen der Kernobjekte abhängig, mit denen sie assoziiert sind. Änderungen an den Attributen können deshalb Änderungen an Einträgen in der Seitentabelle erforderlich machen. Das Suchen und Ändern von Einträgen in der Seitentabelle ist jedoch mit erheblichem Aufwand verbunden. Aufgrund der nur begrenzt zur Verfügung stehenden Bearbeitungszeit für diese Diplomarbeit wurde der einfachste Weg zur Lösung diese Problems gewählt: Attribute dürfen nur geändert werden, wenn sich daraus keine Änderungen an den Einträgen in der Seitentabelle ergeben.
In Abbildung 7.10 sind die Zustände durch dunkle Balken markiert, in denen Attribute geändert werden können. Für private Adreßbereiche, die noch nicht oder nicht mehr in einen Adreßraum eingeblendet sind, existieren keine Einträge in der Seitentabelle. Die Attribute „Zugriffsrechte“, „Anfang“ und

Kernobjekttyp	Attribut	Werte	Bedeutung
Adreßbereich	Verfügbarkeit	privat	Kann nur in einen Adreßraum eingeblendet werden.
		öffentlich	Kann in beliebig viele Adreßräume eingeblendet werden.
	Zugriff	lesend	Ausschließlich lesende Zugriffe sind erlaubt.
		schreibend	Lesende und schreibende Zugriffe sind erlaubt.
Einblendung	Zugriff	lesend	Ausschließlich lesende Zugriffe sind erlaubt.
		schreibend	Schreibende Zugriffe sind erlaubt, sofern der Adreßbereich diese zuläßt.
	Schalter	aktiv	Zugriffe können erfolgen.
		inaktiv	Keine Zugriffe möglich.

Abbildung 7.11: Neue Attribute für Kernobjekte

„Verfügbarkeit" können also geändert werden. Bei einer Änderung des Attributs „Verfügbarkeit" auf den Wert „öffentlich" werden allerdings Einträge in der Seitentabelle erzeugt. Die anderen Attribute müssen also bei Bedarf vorher gesetzt werden.

Mit Einblendungen privater Bereiche sind Einträge in der Seitentabelle assoziiert. Um Attribute solcher Einblendungen ändern zu können, werden Einblendungen mit einem Schalter-Attribut versehen, das die Werte „aktiv" und „inaktiv" annehmen kann. Eine inaktive Einblendung ist ein reines Verwaltungsobjekt, für das keine Adreßtransformation definiert ist. In diesem Zustand können die anderen Attribute der Einblendung verändert werden. Speicherzugriffe über inaktive Einblendungen sind jedoch nicht möglich.
Neu erzeugte Einblendungen sind aktiv, das Verhalten entspricht somit dem auf Rechnern ohne Adreßtransformation. Beim Übergang von „aktiv" zu „inaktiv", der durch Aufruf einer Kernprozedur erfolgt, werden eventuell vorhandene Einträge in der Seitentabelle gelöscht. Die bei der Rückkehr in den Zustand „aktiv" erzeugten Einträge entsprechen den dann eingestellten Werten der Attribute.

In der Tabelle in Abbildung 7.11 sind die neuen Attribute zusammengefaßt. Für jedes Attribut sind die Werte angegeben, die es annehmen kann, sowie eine kurze Beschreibung der Bedeutung jedes Wertes.

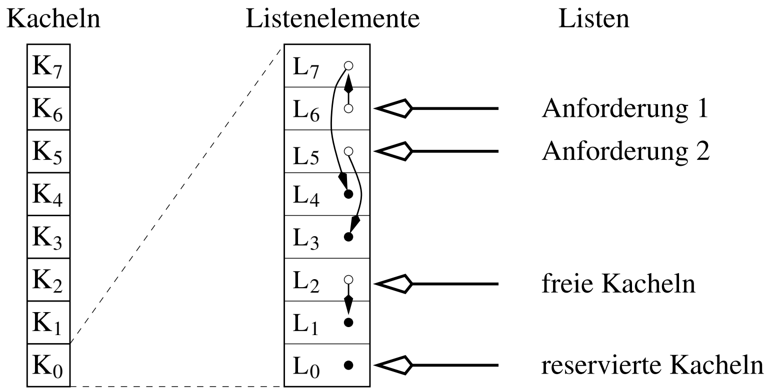

Abbildung 7.12: Physikalische Speicherverwaltung auf PowerPC

7.2.4 Physikalischer Speicher

Aufgrund der Verwendung der Adreßtransformation werden für Adreßbereiche in der Regel keine zusammenhängenden Speicherbereiche angefordert. Statt dessen ist der Hauptspeicher in *Kacheln* zu 4 Kilobyte unterteilt. Der Speicher eines Adreßbereichs wird durch eine hinreichend große Anzahl von Kacheln bereitgestellt. Dazu müssen die Kacheln im Hauptspeicher nicht benachbart sein.

Alle unbelegten Kacheln sind in einer Liste vermerkt. Bei einer Speicheranforderung werden entsprechend viele Kacheln aus dieser Liste entnommen. Sie bilden eine neue Liste. Für die Kacheln in einer solchen Liste können Einträge in der Seitentabelle erzeugt werden, die einen zusammenhängenden Speicherbereich in einem Zwischensegment bereitstellen. Angeforderte Kachellisten werden vollständig freigegeben. Dazu wird die Liste der freizugebenden Kacheln mit der Liste der freien Kacheln vereinigt.

Da bei allen Listen Zeiger auf das erste und das letzte Element gespeichert werden, kann die Speicherfreigabe in konstanter Zeit erfolgen. Der Zeitaufwand für die Anforderung von Speicher wächst linear mit der Anzahl der anzufordernden Seiten. Bei beiden Operationen entfällt der bei anderen Verfahren meist nötige Aufwand zum Zerteilen oder Verschmelzen von Speicherbereichen, da die Listen der Kacheln ungeordnet sind.
Jede Kachel befindet sich zu jedem Zeitpunkt in genau einer Liste. Deshalb können die Datenstrukturen zur Verwaltung der Listenelemente à priori eingerichtet werden. Somit entsteht kein zusätzlicher Verwaltungsaufwand für die Beschaffung und Freigabe der Listenelemente.

Abbildung 7.12 zeigt das Prinzip dieser Verwaltungmethode am Beispiel eines Speichers mit 8 Kacheln ($K_0 \ldots K_7$). Eine der Kacheln (K_0) wird verwendet, um die Listenelemente ($L_0 \ldots L_7$) für alle Kacheln abzulegen. Diese Kachel befindet sich in einer Liste von reservierten Kacheln. Des weiteren sind im Beispiel drei weitere Listen eingezeichnet. Die Liste der freien Kacheln enthält die Kacheln K_2 und K_1. Für eine Anforderung 1 von drei Kacheln wurden K_6, K_7 und K_4 belegt. Die zweite Anforderung über zwei Kacheln konnte mit K_5 und K_3 erfüllt werden.

Kapitel 8

Infrastruktur

Dieses Kapitel beschäftigt sich mit der Realisierung des COSY-Kerns als „black box“. Im Vordergrund steht also nicht die Implementierung bestimmter Kernprozeduren, sondern die Frage, auf welche Weise Kernprozeduren mit Programmen interagieren. In Abschnitt 8.1 wird der Kerneintritt und die Parameterübergabe an Kernprozeduren behandelt. Abschnitt 8.2 beschäftigt sich mit der Einbettung des Kerns in Adreßräume sowie mit Adreßraumausbrüchen des Kerns.

8.1 Kerneintritt

8.1.1 Realisierung auf dem Transputer

COSY-Kernprozeduren können sowohl von Prozessen auf dem lokalen Knoten als auch von Prozessen auf entfernten Knoten aufgerufen werden. Bei einem entfernten Aufruf arbeitet der Kern ohne Mitwirkung eines lokalen COSY-Prozesses. Um dennoch einen Träger für die Aktivität des Kerns bereitzustellen, ist der COSY-Kern auf Transputern als Transputer-Prozeß realisiert. Transputer verfügen nicht über einen Unterbrechungsmechanismus, mit dem zum Beispiel das Eintreffen eines Datenpakets über eine Kommunikationsleitung signalisiert werden könnte. Deshalb besteht das *Transportsystem*, über das die Kommunikation abgewickelt wird, aus mehreren Transputer-Prozessen. Die Prozesse des Transportsystems besitzen hohe Priorität, um Prozeßumschaltungen innerhalb kritischer Bereiche zu vermeiden. Der Kern-Prozeß ist ein Transputer-Prozeß mit niedriger Priorität, da sonst Verklemmungen auftreten könnten [Bute94].
COSY-Prozesse werden durch Transputer-Prozesse mit niedriger Priorität realisiert. Der COSY-Prozeß, der den Prozessor vom COSY-Kern erhält, wird vom Transputer-Kern verwaltet. Alle anderen COSY-Prozesse sind dem Transputer-Kern unbekannt. Bei einer Prozeßumschaltung durch den COSY-Kern wird der bisher rechnende COSY-Prozeß aus dem Transputer-Kern entfernt und statt dessen der neue COSY-Prozeß eingefügt. Durch dieses Vorgehen ist sichergestellt,

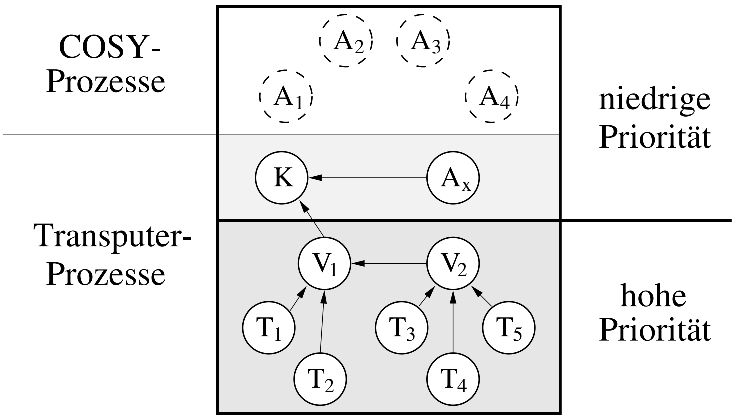

Abbildung 8.1: Transputer-Prozesse in COSY

daß dem Transputer-Kern zu jedem Zeitpunkt nur zwei Transputer-Prozesse mit niedriger Priorität bekannt sind. Einer davon ist der COSY-Kern, der andere der COSY-Prozeß, dem der Prozessor zugeteilt ist.
Abbildung 8.1 zeigt eine vereinfachte Darstellung der Situation. Im unteren Bereich sind die Transputer-Prozesse mit hoher Priorität eingezeichnet. T_1 bis T_5 stellen Prozesse dar, die Ereignisse, zum Beispiel das Eintreffen eines Datenpakets, erkennen. V_1 und V_2 sind Verwaltungsprozesse, deren Aufgabe sich darauf beschränkt, Mitteilungen von mehreren Prozessen entgegennehmen und an einen Prozeß weiterleiten. Die Pfeile zwischen Prozessen stehen für Transputer-Kanäle, über die Mitteilungen verschickt werden. Es ist zu erkennen, daß der Kern (K) nur von zwei Transputer-Prozessen Mitteilungen erhält, jeweils einem (V_1) mit hoher und einem (A_X) mit niedriger Priorität. Sämtliche Mitteilungen von Prozessen hoher Priorität werden über V_1 an den Kern vermittelt. A_X ist der Transputer-Prozeß, der einem COSY-Prozeß entspricht. Von den dargestellten COSY-Prozessen A_1 bis A_4 tritt jeweils einer an die Stelle von A_X. Eine Mitteilung von A_X entspricht demnach einem Kernaufruf.

Solange der COSY-Kern sich im Ruhezustand befindet, ist der entsprechende Transputer-Prozeß (K) blockiert. Die Blockierung erfolgt durch eine Empfangsoperation an einer Gruppe aus zwei Transputer-Kanälen, über die Mitteilungen an den Kern gesendet werden können. Sobald an einem dieser Kanäle eine Mitteilung eintrifft und der Transputer den Kern-Prozeß abzuarbeiten beginnt, entfernt der COSY-Kern den zweiten Transputer-Prozeß mit niedriger Priorität aus dem Transputer-Kern. Die dazu notwendige Sequenz von Transputer-Befehlen enthält keinen Befehl, der eine Prozeßumschaltung erlaubt. Nach der Abarbeitung dieser Sequenz kann der Kern-Prozeß nur noch von Transputer-Prozessen mit hoher Priorität verdrängt werden. Diese greifen ausschließlich auf ihre lokalen Daten zu, so daß die Konsistenz der Kerndaten gesichert ist. Nach Bearbeitung der eingegangenen Mitteilung wird wieder ein COSY-Prozeß als Transputer-Prozeß mit niedriger Priorität installiert. Durch dieses Vorgehen kann der COSY-Kern eine Prozeßumschaltung durch Änderung seiner privaten Daten vornehmen, ohne zusätzlich mit dem Transputer-Kern zu interagieren.

Wie in Abschnitt 3.1.2 beschrieben wurde, können über Transputer-Kanäle nur Nachrichten festgelegter Größe verschickt werden. Kernprozeduren erwarten jedoch unterschiedlich viele Eingabeparameter. Deshalb enthalten die Nachrichten für COSY-Kernaufrufe die folgenden drei Elemente:

- Den Identifikator für die auszuführende Kernprozedur.
- Die Adresse zur Rückgabe einer Fehlerkennung.
- Die Adresse der Eingabeparameter.

Prozeduren, die in C formuliert werden, können beliebig viele Aufrufparameter erhalten, aber nur einen direkten Rückgabewert liefern. Jede Kernprozedur liefert 0 als Rückgabewert, wenn sie ohne Fehler ausgeführt werden konnte. Ansonsten ist der Rückgabewert eine Fehlerkennung, die einen Hinweis auf die Ursache des aufgetretenen Fehlers bietet.
Der COSY-Kern für Transputer speichert den Rückgabewert an der Adresse, die in der Aufruf-Nachricht vermerkt ist. Wenn eine Kernprozedur zusätzliche Werte zurückgeben muß, zum Beispiel die Kennung eines neu erzeugten Kernobjektes, übergibt der Aufrufer der Kernprozedur als Aufrufparameter Adressen, an denen die Rückgabewerte abgelegt werden sollen.
Sämtliche Aufrufparameter werden vom Aufrufstummel einer Kernprozedur in einer Struktur auf dem Stapel abgelegt. Die Adresse dieser Struktur ist der dritte Wert in der Nachricht, die an den Kern geschickt wird. Der Zugriff auf diese Struktur erfordert, ebenso wie die Rückgabe von Werten, einen Adreßraumausbruch des COSY-Kerns. Da auf Transputern aber nur ein globaler Adreßraum realisiert werden kann, erfolgen Adreßraumausbrüche auf die gleiche Art wie Zugriffe im eigenen Adreßraum.

Mitteilungen des Transportsystems an den Kern sind ebenso aufgebaut wie die Nachrichten für den Aufruf von Kernprozeduren. Im Gegensatz zu Kernaufrufen gibt es aber keine Rückgabewerte. Der entsprechende Eintrag in den Nachrichten für Kernaufrufe bleibt ungenutzt. Der entscheidende Vorteil dieses Vorgehens ist die einheitliche Schnittstelle des COSY-Kerns, da sowohl Kernaufrufe als auch Mitteilungen auf die gleiche Art behandelt werden können.

8.1.2 Realisierung auf dem PowerPC

Im Gegensatz zu Transputern verfügt der PowerPC nicht über einen Hardware-Kern. Deshalb muß der COSY-Kern für PowerPC als *Prozedurgruppe* implementiert werden. Der Aufruf einer Kernprozedur erfolgt über eine Ausnahmebehandlung, was vom PowerPC-Prozessor durch einen speziellen Befehl (*sc: system call*) unterstützt wird.
Das Transportsystem auf dem PowerXplorer signalisiert externe Ereignisse über eine Hardware-Leitung. Der PowerPC-Prozessor reagiert darauf mit einer Ausnahmebehandlung. Allerdings wird eine andere Ausnahmebehandlungsprozedur ausgeführt als bei einem Kernaufruf.

Während der COSY-Kern auf Transputern als eigener Prozeß implementiert ist und deshalb einen eigenen Stapel benötigt, könnte der Kern auf PowerPC-Prozessoren die Stapel der aufrufenden Anwender-Prozesse benutzen. Allerdings bringt ein solches Verfahren einige schwerwiegende Nachteile mit sich:
Der Bedarf des Kerns an Stapelspeicher ist beschränkt, da im COSY-Kern keine rekursiven Prozeduraufrufe stattfinden. Bei Anwenderprozessen ist dies nicht der Fall. Beim Kerneintritt kann also nicht garantiert werden, daß auf dem Stapel des Anwenderprozesses noch genug Speicher frei ist. Außerdem kann der Stapelzeiger von Anwenderprozessen auf beliebige Werte gesetzt werden. Dies gefährdet den Speicherschutz, da der Stapelzeiger beim Kerneintritt einen Wert enthalten könnte, der nur im Adreßraum des Kerns einer gültigen Adresse entspricht. Dies müßte bei jedem Kerneintritt überprüft werden.
Einige Kernprozeduren führen schreibende Zugriffe in COSY-Adreßräume durch, etwa um eine empfangene Nachricht abzulegen. Da auch Anwenderstapel in solchen Adreßräumen zugänglich sind, könnte dabei der vom Kern verwendete Stapelbereich überschrieben werden. Des weiteren wären die von Kernprozeduren auf dem Stapel abgelegten Daten für Anwenderprozesse zugänglich. Auch dies beeinträchtigt den Speicherschutz.
Um die genannten Probleme zu vermeiden benutzt der COSY-Kern auch auf PowerPC-Prozessoren einen eigenen Stapelbereich fester Größe, der nur im Adreßraum des Kerns zugänglich ist. Bei jedem Kerneintritt wird der Stapelzeiger auf den Anfang des Kernstapels gesetzt. Dies erlaubt eine Optimierung beim Kernaustritt, da die auf dem Stapel abgelegten Prozedurrahmen nicht explizit freigegeben werden müssen.

Der PowerPC-Prozessor verfügt über eine hinreichend große Anzahl an Registern, um die meisten Parameter von Kernprozeduren darin zu übergeben. Auch die Rückgabe von Werten kann über Register erfolgen. Da der C-Standard letzteres nicht unterstützt, muß dies allerdings in Assembler-Prozeduren explizit programmiert werden.
Die Übergabe von Werten in Registern vermeidet Adreßraumausbrüche des Kerns in den meisten Fällen. Nur wenn größere Datenstrukturen übergeben werden, zum Beispiel Nachrichten bei Sendeoperationen, sind Adreßraumausbrüche notwendig. Diese sind bei PowerPC-Prozessoren aufwendiger als bei Transputern, da in COSY für PowerPC die Adreßtransformationseinheit eingesetzt wird. Abschnitt 8.2 beschreibt die Realisierung.

8.2 Einbettung in Adreßräume

Das Problem der Einbettung in Adreßräume stellt sich auf Transputern nicht, da dort nur ein physikalischer Adreßraum existiert. Die Realisierung von Programmadreßräumen mit Hilfe der Adreßtransformationseinheit eines PowerPC-Prozessors wurde bereits in Kapitel 7 ausführlich diskutiert. In diesem Abschnitt wird nun die Einbettung des COSY-Kerns und die damit zusammenhängende Realisierung von Adreßraumausbrüchen auf PowerPC-Prozessoren erläutert.

Als Grundlage für das weitere Vorgehen muß zunächst entschieden werden, ob während der Ausführung von Kernprozeduren die Adreßtransformation verwendet wird, oder ob der Kern im physikalischen Speicheradreßraum arbeitet.
Beim Versenden und Empfangen von Nachrichten muß aus Kernprozeduren heraus auf Daten in COSY-Adreßräumen zugegriffen werden. Der Einsatz der Adreßtransformation im Kern erleichtert solche Zugriffe, da die Umrechnung von Programm- in Speicheradressen entfällt. Die Umrechnung müßte sonst beim Zugriff auf Nachrichten eventuell sogar mehrfach erfolgen, wenn eine Nachricht in mehreren Kacheln gespeichert ist.
Der Kern verwaltet dynamisch angeforderte Daten über die Kernobjekte. Bei Einsatz der Adreßtransformation kann ein zusammenhängender Speicherbereich für diese Daten verwaltet und bei Bedarf vergrößert oder verkleinert werden. Ohne Adreßtransformation ist dies nicht möglich, da der physikalische Speicher in einer ungeordneten Kachelliste verwaltet wird. Die Suche nach freien, benachbarten Kacheln ist in diesem Fall mit einem hohen Aufwand verbunden und nicht immer erfolgreich.
Aus diesen Gründen wird in COSY auch bei der Ausführung von Kernprozeduren die Adreßtransformationseinheit des PowerPC-Prozessors eingesetzt.

Für Anwenderadreßräume ist eine hohe Flexibilität bei der Einblendung von Adreßbereichen wünschenswert, um Programmierer nicht unnötig einzuschränken. Der Kern eines Systems dagegen ist für einen klar abgegrenzten Aufgabenbereich zuständig. Dies kann bei der Entwicklung berücksichtigt werden, um durch eine effiziente Nutzung der Betriebsmittel Freiräume für Optimierungen zu schaffen.
Der Umfang der vom Kern gespeicherten Daten ist durch die Größe des physikalischen Speichers beschränkt. Ein Knoten eines PowerXplorer ist mit 8 Megabyte Speicher ausgestattet, die nur teilweise vom Kern belegt werden. Somit ist es möglich, alle Kerndaten in einem 256 Megabyte großen Segment der Adreßtransformationseinheit zur Verfügung zu stellen. In jedem COSY-Adreßraum wird ein solches *Kernsegment* reserviert.
Durch dieses Vorgehen können Kernprozeduren in jedem COSY-Adreßraum ausgeführt werden und dabei sowohl auf die Daten in diesem Adreßraum als auch auf die Daten des Kerns zugreifen. Der Adreßraumausbruch aus dem Kernadreßraum erfolgt durch direkten Zugriff. Auch der Wechsel von einem Adreßraum in einen anderen kann problemlos in einer Kernprozedur erfolgen.
Die Daten des Kerns werden mit Hilfe des Adreßtransformationsmechanismus vor Zugriffen von COSY-Prozessen geschützt. Der PowerPC-Prozessor führt COSY-Prozesse im Anwender-Modus, Kernprozeduren dagegen im System-Modus aus. Durch geeignete Wahl der Zugriffsrechte auf Kerndaten lassen sich Zugriffe von COSY-Prozessen unterbinden.

Bei der Kommunikation und bei direkten Lese- oder Schreibzugriffen auf Adreßräume kann es erforderlich werden, Daten von einem Adreßraum in einen anderen zu kopieren. Dies könnte über einen Puffer im Kernadreßraum geschehen. Dieses Vorgehen erfordert jedoch zwei Kopiervorgänge, zwischen denen ein Adreßraumwechsel stattfinden muß. Für COSY wurde deshalb ein anderer Weg gewählt. Neben dem Kernsegment wird ein weiteres Segment in jedem

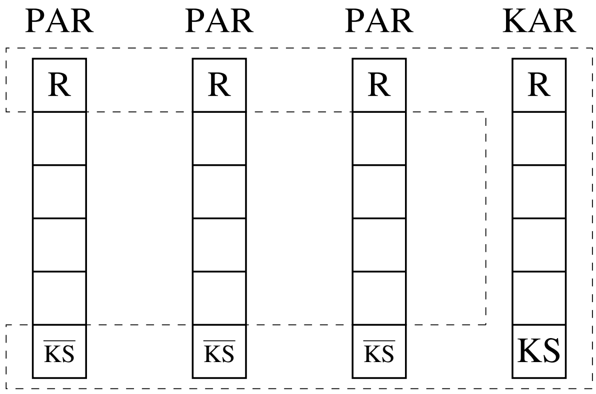

Abbildung 8.2: Adreßräume auf PowerPC

Adreßraum reserviert. In dieses reservierte Segment kann der Kern ein beliebiges Segment eines anderen Adreßraums einblenden, was ein direktes Kopieren ohne Adreßraumwechsel ermöglicht. Die resultierende Einschränkung, daß zu kopierende Datenbereiche keine Segmentgrenze überschreiten dürfen, spielt in der Praxis keine Rolle.
Statt ein Segment für das Kopieren zu reservieren, könnte auch für jeden Kopiervorgang ein geeignetes Segment gewählt werden. Eine Anzahl von 14 verfügbaren Segmenten pro COSY-Adreßraum im Falle der Reservierung scheint jedoch ausreichend in Anbetracht der vergleichsweise geringen Größe des physikalischen Speichers und der fehlenden Unterstützung von virtuellem Speicher.

Abbildung 8.2 zeigt das Prinzip der Einbettung des Kerns. Für den Kern wird ein Kernadreßraum (KAR) eingerichtet, in dem der Leerlaufprozeß arbeitet. Auf den Code und sämtliche Daten des Kerns wird über ein einzelnes Segment dieses Adreßraums, das Kernsegment (KS), zugegriffen. In jedem Programmadreßraum (PAR) ist dieses Kernsegment eingeblendet ($\overline{\text{KS}}$). Es handelt sich also um ein gemeinsames Segment[1] jedes Adreßraums. Zusätzlich ist in jedem Adreßraum ein Segment reserviert (R), um das Kopieren über Adreßraumgrenzen hinweg zu erleichtern. Der Kernadreßraum und die reservierten Segmente, also der umrandete Bereich in Abbildung 8.2, werden nur vom COSY-Kern und dem Leerlaufprozeß genutzt. Alle anderen Segmente der Adreßräume können für die Einblendung von Adreßbereichen benutzt werden.
In Abbildung 8.3 auf Seite 75 ist die Realisierung von Adreßraumausbrüchen und Speicherschutz dargestellt. Das Kopieren über Adreßraumgrenzen (1) kann nur vom Kern durchgeführt werden, da nur dieser ein Segment eines Adreßraums in das reservierte Segment eines anderen Adreßraums einblenden kann. Zugriffe des Kerns auf Daten in einem Adreßraum (2) können direkt erfolgen. Zugriffe eines COSY-Prozesses auf Daten des Kerns (3) werden durch den Speicherschutz der PowerPC-Adreßtransformation unterbunden.

[1] Im Sinne der Systemarchitektur sollte dieser Bereich als *Adreßbereich* bezeichnet werden. Um Verwechslungen mit dem COSY-Kernobjekttyp Adreßbereich zu vermeiden, wurde an dieser Stelle die Terminologie der PowerPC-Spezifikation verwendet.

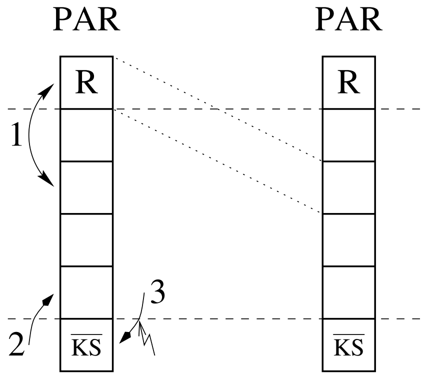

Abbildung 8.3: Adreßraumausbrüche auf PowerPC

Der physikalische Speicheradreßraum wird in Cosy für PowerPC nur an wenigen Stellen benutzt. Während der Inbetriebnahme ist dies unumgänglich, da die Verwaltungsinformationen für die Adreßtransformation noch nicht verfügbar sind. Auch zu Beginn einer Ausnahmebehandlung, also bei jedem Kerneintritt, ist die Adreßtransformation ausgeschaltet. Sie wird in diesem Fall nach wenigen Assembler-Befehlen wieder aktiviert, noch bevor die eigentliche Ausnahmebehandlung beginnt.
Die dritte Situation, in der eine Prozedur im Speicheradreßraum ausgeführt wird, ist nicht durch den Prozessor erzwungen. In neu angeforderten Cosy-Adreßbereichen enthält jede Speicherstelle den Wert 0. Auf diese Weise wird vermieden, daß Daten von vorherigen Nutzungen einer Kachel sichtbar werden. Das Löschen von neu angeforderten Kacheln erfolgt im Speicheradreßraum, da es zu aufwendig wäre, allein zu diesem Zweck Einträge in der Seitentabelle zu erzeugen oder die einstufige Adreßtransformation zu initialisieren.
Abgesehen von diesen Fällen kann die Verwendung des Speicheradreßraums vermieden werden, da die benötigten Bereiche des Speicheradreßraums in das Kernsegment eingeblendet sind. Der Einsatz der Adreßtransformation für die Arbeit des Kerns erhöht die Komplexität des Kerns also nur geringfügig.

Abbildung 8.4 auf Seite 76 zeigt den Aufbau des Kernsegmentes. Die Größenverhältnisse der eingezeichneten Bereiche entsprechen allerdings nicht den tatsächlichen Gegebenheiten. Im linken Teil ab Adresse 0 befinden sich die statischen Teile des Kerns. Diese umfassen den Code, die statischen Daten und den Kernstapel. Der dunkel gezeichnete Bereich deutet an, wieviel des vorgesehenen Bereichs tatsächlich genutzt wird. In diesem und auch den folgenden Bereichen stimmen die Speicher- und Programmadressen überein. Auf diese Weise ist es möglich, Kernprozeduren sowohl mit als auch ohne Adreßtransformation auszuführen.
Nach dem Kern ist jeweils ein Bereich für die Kachellisten zur Verwaltung des physikalischen Speichers und für die Kooperation von PowerPC und Transputer vorgesehen. Der folgende Bereich wird nur während der Inbetriebnahme genutzt. Er ist während des Betriebs nicht in das Kernsegment eingeblendet. Daran schließt sich die Seitentabelle an, die Informationen für die zweistufige Adreßtransformation enthält.

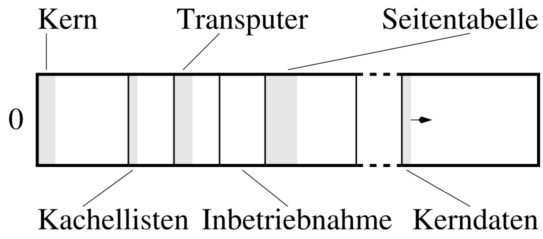

Abbildung 8.4: Kernsegment

Die dynamisch angeforderten Kerndaten sind im obersten Teil des Kernsegmentes abgelegt. Programmadressen in diesem Bereich entsprechen keinen gültigen Speicheradressen. Die Einblendung erfolgt über die zweistufige Adreßtransformation. Auf die dynamischen Kerndaten kann deshalb nur zugegriffen werden, wenn die Adreßtransformation eingeschaltet ist.
Der statische Teil des Kerns und die Seitentabelle werden mit Hilfe der einstufigen Adreßtransformation in das Kernsegment eingeblendet. Dies reduziert die Zahl der Einträge in der Seitentabelle und beschleunigt Zugriffe auf diese Bereiche, da der PowerPC-Prozessor für die einstufige Adreßtransformation keine Zugriffe auf die Seitentabelle durchführen muß.

Kapitel 9

Prozesse

Dieses Kapitel beschäftigt sich mit der Realisierung der COSY-Kernobjekte vom Typ Prozeß. Während die Abarbeitung von Prozessen Aufgabe des Prozessors ist, muß deren Verwaltung vom Betriebssystem durchgeführt werden. Die Verwaltung von Prozessen beinhaltet in COSY unter anderem:

- Umschalten
- Blockieren und Deblockieren
- Erzeugen und Löschen

Abschnitt 9.1 beschreibt die auf Transputern implementierte Prozeßverwaltung. In Abschnitt 9.2 sind die Änderungen bei der Portierung auf PowerPC-Prozessoren dokumentiert.

9.1 Realisierung auf dem Transputer

Das Prinzip der Prozeßumschaltung in COSY auf Transputern wurde bereits in Abschnitt 8.1.1 angedeutet. Abbildung 9.1 auf Seite 78 zeigt den Vorgang im Detail. Der COSY-Prozeß, dem der Prozessor zugeteilt wurde, ist als Transputer-Prozeß im Transputer-Kern eingetragen. Beim Kerneintritt wird dieser Prozeß aus dem Transputer-Kern entfernt. Daraufhin führt der COSY-Kern eine Kernprozedur aus. Im Verlauf dieser Ausführung entscheidet sich, ob ein Prozeßwechsel durchgeführt werden muß (PW?). Ist dies der Fall, so erfolgt der Prozeßwechsel durch Änderungen an den Daten des COSY-Kerns. Beim Kernaustritt wird der COSY-Prozeß, dem der Prozessor zugeteilt wurde, wieder im Transputer-Kern eingetragen.

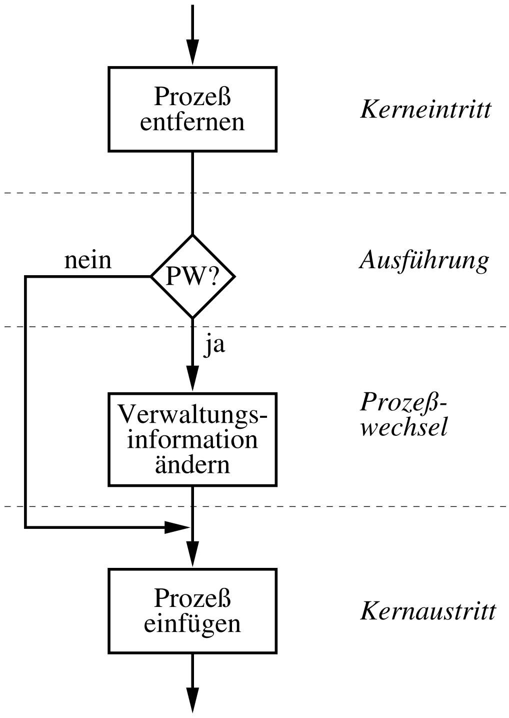

Abbildung 9.1: Prozeßumschaltung mit Transputer

Eine Ablaufumgebung, die beim Prozeßwechsel in den Prozessor geladen werden muß, existiert auf Transputern nicht. Die Verknüpfungsumgebung besteht aus dem Stapelzeiger, dem Befehlszeiger und zwei Speicherstellen, die von Werkzeugen zur Fehlersuche benutzt werden. Der Inhalt der zusätzlichen Register ist nach jedem Transputer-Befehl, der eine Prozeßumschaltung erlaubt, undefiniert und muß deshalb nicht gesichert werden.
Das Sichern und Laden des Befehlszeigers eines Prozesses wird vom Transputer-Kern übernommen. Das Sichern und Laden des Stapelzeigers und der Speicherstellen für Werkzeuge zur Fehlersuche ist Aufgabe des COSY-Kerns. Im Prozeßleitblock jedes COSY-Prozesses ist ein Bereich vorgesehen, in dem diese Werte abgelegt werden.
Der Stapelzeiger wird bereits beim Kerneintritt gesichert. Er dient als Argument für den Transputer-Befehl, mit dem der COSY-Prozeß beim Kernaustritt wieder in den Transputer-Kern eingefügt wird. Das Speichern und Laden der Speicherstellen für Werkzeuge zur Fehlersuche erfolgt bei jeder Prozeßumschaltung. Diese Speicherstellen sind den Daten des Kerns zuzuordnen. In Abbildung 9.1 ist das Laden und Speichern deshalb nicht explizit eingezeichnet, sondern im Block „Verwaltungsinformation ändern“ eingeschlossen.

Eine Sonderbehandlung wird für den *Leerlaufprozeß* durchgeführt. Der Leerlaufprozeß wird aktiv, wenn kein anderer Prozeß den Prozessor benötigt. Beim Kernaustritt nach einer Umschaltung zum Leerlaufprozeß wird kein COSY-Prozeß im Transputer-Kern eingetragen. In diesem Fall wird der Leerlaufprozeß des Transputer-Kerns aktiviert. Der Transputer führt dann keine Speicherzu-

Verwendung	Ganzzahlregister	Gleikommaregister
allgemein	32	32
Bedingungen	2	1
Befehlszeiger	1	0
sonstiges	3	0
gesamt	38	33
in Byte	152	264

Abbildung 9.2: Verknüpfungsumgebung auf PowerPC

griffe durch, was im Falle eines Cosy-Leerlaufprozesses zum Laden der auszuführenden Befehle notwendig wäre. Somit bleibt der Systembus frei für Zugriffe externer Hardware.
Beim Kerneintritt kann der Leerlaufprozeß nicht aus dem Transputer entfernt werden, was eine zusätzliche Abfrage erforderlich macht. Bei der Prozeßumschaltung erfolgt keine Sonderbehandlung, da auch der Leerlaufprozeß über einen Prozeßleitblock im Cosy-Kern verfügt.

Beim Blockieren eines Cosy-Prozesses werden Informationen über den Grund der Blockierung sowie die zur Fortsetzung benötigten Daten im Prozeßleitblock abgelegt. Der Prozeß wird aus der Menge der sich um den Prozessor bewerbenden Prozesse entnommen. Da das Blockieren bei der Ausführung einer Kernprozedur erfolgt, ist der zu blockierende Prozeß nicht im Transputer-Kern eingetragen. Eine zusätzliche Interaktion mit dem Transputer-Kern ist deshalb nicht notwendig.
Bei der Deblockierung werden die im Prozeßleitblock abgelegten Daten verwendet, um eventuelle Rückgabewerte, zum Beispiel die Kernobjektkennung eines erzeugten Objektes, im Adreßraum des Prozesses abzulegen. Danach wird der Prozeß wieder in die Menge der sich um den Prozessor bewerbenden Prozesse eingefügt.

Um einen Cosy-Prozeß zu erzeugen, muß zunächst ein Prozeßleitblock angefordert werden. In diesem werden neutrale Werte für die Speicherstellen für Werkzeuge zur Fehlersuche eingetragen. Auch der Stapelzeiger des neuen Prozesses, der ein Argument des Kernaufrufs ist, wird im Prozeßleitblock abgelegt. Die Adresse des ersten auszuführenden Befehlswortes, ebenfalls ein Argument des Kernaufrufs, muß im Stapel des Prozesses abgelegt werden, damit der Transputer-Kern den Prozeß korrekt abarbeitet.

9.2 Realisierung auf dem PowerPC

Die Tabelle in Abbildung 9.2 zeigt, daß die Verknüpfungsumgebung auf einem PowerPC-Prozessor aus 38 Registern für ganze Zahlen und Adressen sowie 33 Registern für Gleitkommazahlen besteht. Der in der Tabelle aufgeführte Befehlszeiger ist nicht direkt zugänglich, wird aber in bestimmten Situationen,

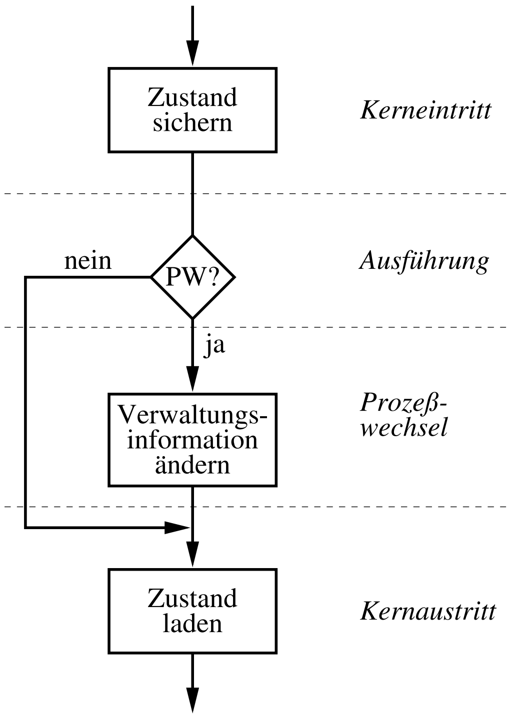

Abbildung 9.3: Triviale Prozeßumschaltung mit PowerPC

etwa beim Aufruf einer Prozedur oder zu Beginn einer Ausnahmebehandlung, in ein spezielles Register geladen. Register für ganze Zahlen sind 32 Bit breit, Register für Gleitkommazahlen enthalten 64 Bit. Die Verknüpfungsumgebung besteht also aus 152 Byte für ganze Zahlen oder Adressen und 264 Byte für Gleitkommazahlen. Insgesamt ist sie somit 416 Byte groß.

Die Ablaufumgebung eines Cosy-Prozesses auf PowerPC besteht aus dem Prozessorzustandsregister und den 16 Segmentregistern. Der im PowerXplorer verwendete PowerPC-Prozessor vom Typ MPC601 verfügt zusätzlich über 3 Register, die von Werkzeugen zur Fehlersuche eingesetzt werden können. Alle diese Register sind 32 Bit breit. Die Ablaufumgebung ist also 80 Byte groß.

Anmerkung. *Die Ablaufumgebung ist vom Betriebssystem abhängig. Wenn zum Beispiel ein Betriebssystem die zweistufige Adreßtransformation nicht einsetzt, dann gehören die 16 Segmentregister nicht zur Ablaufumgebung. Die Verknüpfungsumgebung dagegen ist durch den PowerPC-Prozessor vorgegeben. Sie enthält alle Register, die ein Prozeß verändern kann. Unter der Voraussetzung, daß Prozesse im Anwender-Modus ausgeführt werden, ist sie also eindeutig.*

In Anlehnung an den Prozeßwechsel auf Transputern könnte der Prozeßwechsel mit PowerPC-Prozessoren auf die in Abbildung 9.3 dargestellte Art erfolgen. Beim Kerneintritt wird die Verknüpfungsumgebung des Prozesses im Prozeßleitblock abgelegt. Daraufhin arbeitet der Kern die aufgerufene Kernprozedur

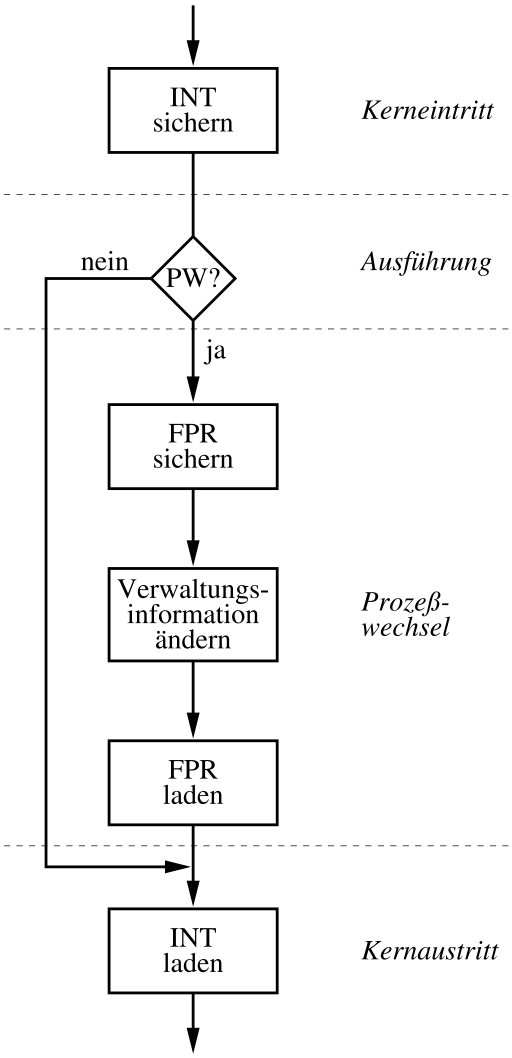

Abbildung 9.4: Verbesserte Prozeßumschaltung mit PowerPC

ab. Falls dabei ein Prozeßwechsel durchgeführt werden muß (PW?), so werden die Verwaltungsinformationen des Kerns angepaßt. Beim Kernaustritt lädt der Kern die Verknüpfungs- und Ablaufumgebung des dann aktiven Prozesses in den Prozessor. Aufgrund der Größe der Verknüpfungs- und Ablaufumgebungen ergibt sich jedoch bei diesem Verfahren ein hoher Aufwand.

Eine Verbesserung gegenüber dem einfachen Vorgehen kann durch die Aufteilung der Verknüpfungsumgebung erreicht werden. Der COSY-Kern verwendet keine Gleitkommaoperationen. Der Inhalt der Gleitkomma-Register muß also nicht gesichert werden, wenn kein Prozeßwechsel stattfindet. In Abbildung 9.4 ist die verbesserte Version der Prozeßumschaltung dargestellt.
Beim Kerneintritt wird zunächst nur der Teil der Verknüpfungsumgebung gespeichert, der die Register für ganze Zahlen und Adressen umfaßt (INT). Nur wenn die Ausführung der Kernprozedur zu einem Prozeßwechsel führt (PW?),

Zu sichernder Anteil der Verknüpfungs-umgebung eines PowerPC-Prozessors:

Umschaltung	Kerneintritt	
	freiwillig	erzwungen
nein	23,1 %	36,5 %
ja	59,6 %	100 %

Abbildung 9.5: Gewinn durch optimierte Prozeßumschaltung

werden die Register für Gleikommazahlen (FPR) gesichert. Nach dem Verändern der Kerndaten werden die Register für Gleitkommazahlen mit den Werten des neuen Prozesses initialisiert. Beim Kernaustritt müssen auf jeden Fall die Ganzzahl-Register der Verknüpfungsumgebung geladen werden.
Da die Register für Gleitkommazahlen 63,5% der Größe der Verknüpfungsumgebung ausmachen, ergibt sich durch diese Verbesserung ein erheblicher Zeitgewinn bei Kernaufrufen, die nicht zu einer Prozeßumschaltung führen.

Das Verfahren kann noch weiter verbessert werden. Für die Übersetzung von Prozeduraufrufen aus C in Assembler-Befehle sind Konventionen festgelegt, die vom Übersetzer eingehalten werden. Diese Konventionen legen unter anderem Register fest, die von der aufgerufenen Prozedur verändert werden dürfen. Wenn ein Kerneintritt *freiwillig* durch Aufruf einer Kernprozedur erfolgt, müssen diese Register nicht gesichert und somit bei der Rückkehr von diesem Kernaufruf auch nicht wieder geladen werden. Nur bei einem *erzwungenen* Kerneintritt, etwa nach Ablauf einer Zeitscheibe, muß der Prozessorinhalt, wie oben beschrieben, vollständig gesichert werden.
Abbildung 9.6 auf Seite 83 zeigt das Ergebnis. Bei einem freiwilligen Kerneintritt wird nur ein Teil der Ganzzahl-Register (INT) gesichert, ansonsten alle. Daraufhin kann eine Kernprozedur abgearbeitet werden. Erweist sich bei der Abarbeitung ein Prozeßwechsel als notwendig (PW?), so werden die Register für Gleitkommazahlen (FPR) gesichert. Auch hier erfolgt nach einem freiwilligen Kerneintritt nur eine teilweise Sicherung. Nach dem Ändern der Verwaltungsinformationen des Kerns ist das weitere Vorgehen davon abhängig, ob die Verknüpfungsumgebung des Prozesses, zu dem umgeschaltet wird, vollständig gespeichert wurde (VV?). Ist dies der Fall, so werden nun alle Gleitkommaregister geladen, anderenfalls nur ein Teil von ihnen. Beim Kernaustritt werden wie zuvor die Register für ganze Zahlen geladen, entweder vollständig oder teilweise.
Von den 38 Registern für ganze Zahlen müssen bei einem freiwilligen Kerneintritt nur 24 gerettet werden. Eine Prozeßumschaltung nach einem freiwilligen Kerneintritt erfordert zusätzlich das Sichern von 19 der insgesamt 33 Gleitkommaregister. In der Tabelle in Abbildung 9.5 ist aufgeführt, welcher Anteil der Verknüpfungsumgebung in den verschiedenen Situationen gesichert werden muß.

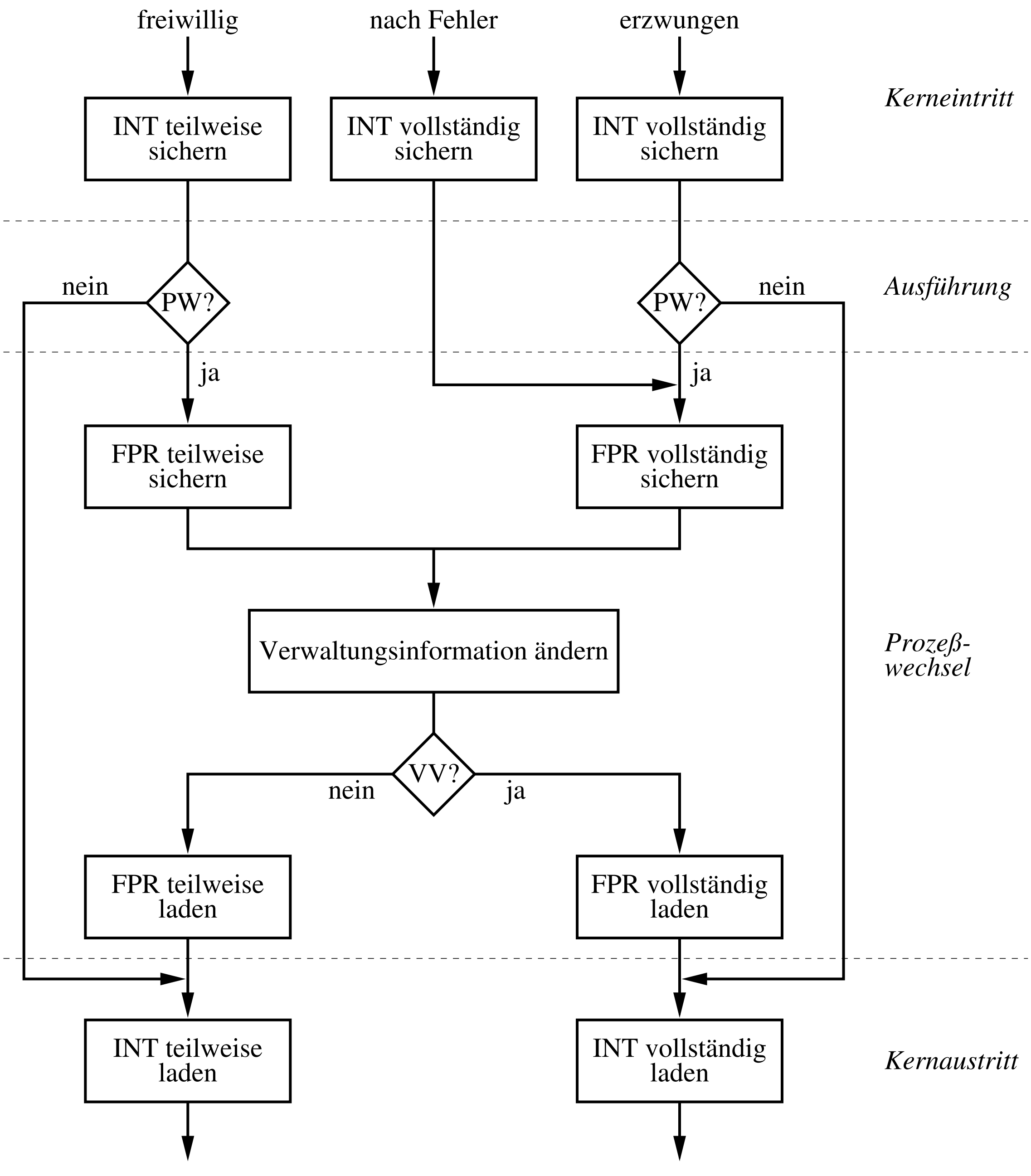

Abbildung 9.6: Optimierte Prozeßumschaltung mit PowerPC

Die Ablaufumgebung kann ebenfalls in zwei Bereiche aufgeteilt werden. Der Wert des Prozessorzustandsregisters und die Inhalte der 3 Register für Werkzeuge zur Fehlersuche sind für jeden Prozeß getrennt anzugeben. Der Inhalt der Segmentregister ist abhängig vom Adreßraum, in dem ein Prozeß existiert. Er kann für verschiedene Prozesse gleich sein.
Die einem Prozeß zugeordneten Register der Ablaufumgebung werden bei jedem Kernaustritt neu geladen. Die einem Adreßraum zugeordneten Segmentregister werden nur dann neu geladen, wenn bei einem Prozeßwechsel der neue Prozeß in einem anderen Adreßraum liegt als der alte. Die Abfrage, ob sich der Adreßraum geändert hat, erfolgt unmittelbar vor dem Kernaustritt.

Beim Blockieren und Deblockieren eines Prozesses ergeben sich keine Unterschiede zwischen PowerPC und Transputer. Bei der Erzeugung eines Prozesses sind geringe Änderungen notwendig. Ein Prozeßleitblock wird angefordert und mit dem Stapelzeiger initialisiert. Die Adresse des ersten abzuarbeitenden Befehls wird jedoch nicht im Stapel des Prozesses, sondern ebenfalls im Prozeßleitblock abgelegt. Dieser Unterschied ist darauf zurückzuführen, daß PowerPC-Prozessoren nicht über einen Hardware-Kern verfügen.

Bei der Abarbeitung eines Cosy-Prozesses durch einen PowerPC-Prozessor können Ausnahmesituationen entstehen, wenn zum Beispiel ein Zugriff auf eine Programmadresse durchgeführt wird, für die keine Adreßtransformation definiert ist. Das Resultat ist ein Kerneintritt *nach Fehler*, wie er in Abbildung 9.6 auf Seite 83 eingezeichnet ist. Die Feststellung und eventuelle Beseitigung der Fehlerursache ist zu aufwendig, um als Teil dieser Diplomarbeit realisiert werden zu können. Statt dessen wird die Verknüpfungsumgebung des fehlerhaft arbeitenden Prozesses vollständig gespeichert und der Prozeß blockiert.

Anmerkung. *In der aktuellen Implementierung wird noch keine Information über das Auftreten eines Fehlers in einem Prozeß weitergegeben. Der Cosy-Kern könnte dieses Ereignis zum Beispiel durch das Erhöhen einer zu diesem Zweck eingerichteten Zählsperre signalisieren, damit eine Systeminstanz darauf reagieren kann.*

Kapitel 10

Kommunikation

Nachdem die Schwerpunkte der Portierung des lokalen COSY-Kerns beschrieben wurden, beschäftigt sich dieses Kapitel mit der knotenübergreifenden Kommunikation. Diese beinhaltet sowohl das Transportsystem, über das der Kernverbund hergestellt wird, als auch den direkten Zugriff auf die Kommunikationsleitungen eines Knotens, der für die Inbetriebnahme und die Anbindung des Wirtsrechners benötigt wird.
Die knotenübergreifende Kommunikation in COSY beinhaltet Abläufe, die mit den bisher verwendeten Grafiken, also statischen Diagrammen und endlichen Automaten, nur unzureichend dargestellt werden können. In diesem Kapitel werden deshalb *Petri–Netze* verwendet, um Modelle zur Verdeutlichung der Mechanismen zu entwerfen.
Abschnitt 10.1 enthält eine Einführung in Petri–Netze, die zum Verständnis der erstellten Modelle ausreichen sollte. In Abschnitt 10.2 wird die auf dem MegaCluster implementierte, knotenübergreifende Kommunikation erläutert. Die bei der Portierung auf PowerXplorer vorgenommenen Änderungen sind in Abschnitt 10.3 beschrieben.

10.1 Petri–Netze

Die 1962 von Carl Adam Petri vorgestellten Petri–Netze [Pet62] erwiesen sich bald als ein vielseitiges Modellierungswerkzeug, das graphische Anschaulichkeit mit mathematischer Präzision kombiniert. Die gebräuchlichste Form von Petri–Netzen sind die allgemeinen, stochastischen Petri–Netze (engl.: Generalized Stochastic Petri Nets, GSPN), die auch hier zum Einsatz kommen.

Ein allgemeines, stochastisches Petri–Netz ist ein gerichteter, bipartiter Graph. Die Knoten dieses Graphen werden als *Stellen* beziehungsweise *Transitionen* bezeichnet. Eine Stelle kann *Marken* enthalten. Die *Markierung* eines Petri–Netzes gibt für jede Stelle an, wie viele Marken sie enthält. Die Markierung beschreibt den *Zustand* des Petri–Netzes.

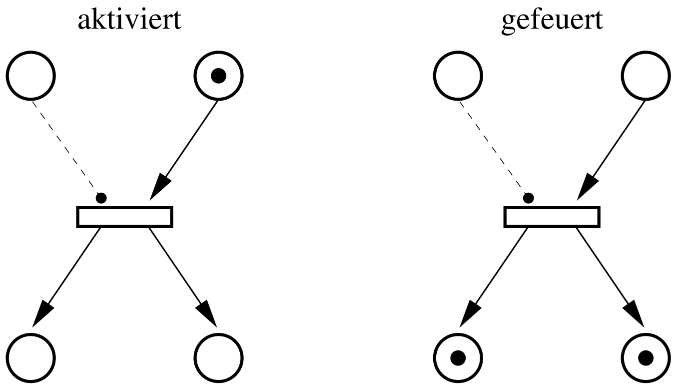

Abbildung 10.1: Feuern einer Transition

Mit Transitionen werden die möglichen *Zustandsübergänge* beschrieben. Transitionen sind mit Stellen durch drei Arten von Kanten verbunden: *Eingangskanten*, *Ausgangskanten* und *hemmende Kanten*. Die entsprechenden Stellen werden als Eingangs-, Ausgangs- und hemmende Stelle einer Transition bezeichnet.

Eine Transition ist *aktiviert*, wenn sich in jeder Eingangsstelle mindestens eine Marke befindet und keine hemmende Stelle eine Marke enthält. Eine aktivierte Transition kann *feuern*. Bei diesem Vorgang wird aus jeder Eingangsstelle eine Marke entnommen und in jeder Ausgangsstelle eine Marke abgelegt.
Abbildung 10.1 zeigt diesen Vorgang. Transitionen werden durch Rechtecke dargestellt, Stellen durch Kreise. Ein- und Ausgangskanten unterscheiden sich durch die Richtung der Pfeilspitze. Hemmende Kanten führen immer von einer Stelle zu einer Transition und werden durch gestrichelte Linien dargestellt, die in einem Punkt enden.
Im linken Teil der Abbildung ist eine aktivierte Transition eingezeichnet. Sie besitzt eine Eingangsstelle, in der sich eine Marke befindet, sowie eine hemmende Stelle, in der keine Marke enthalten ist. Beim Feuern wird die Marke aus der Eingangsstelle entfernt und in beiden Ausgangsstellen eine Marke abgelegt. Das Ergebnis ist im rechten Teil der Abbildung dargestellt.

In Abbildung 10.2 ist ein anderes Petri–Netz dargestellt. In diesem sind zwei Arten von Transitionen zu erkennen. Bei *zeitbehafteten* Transitionen, die durch umrandete Rechtecke dargestellt werden, vergeht Zeit zwischen Aktivierung und

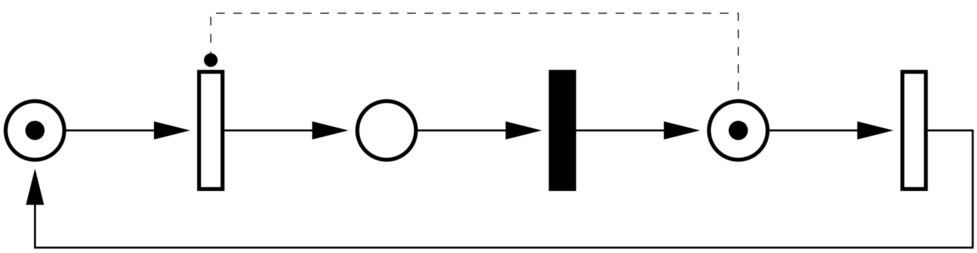

Abbildung 10.2: Ein allgemeines, stochastisches Petri–Netz

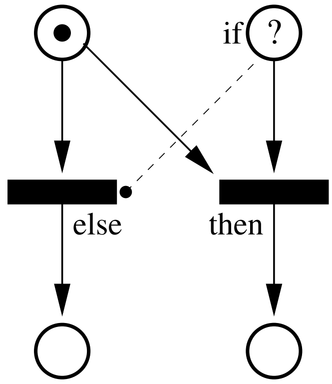

Abbildung 10.3: Deterministische Auflösung von Konflikten

Feuern. Die verstreichende Zeitspanne kann durch eine Zufallsvariable angegeben werden. Im Gegensatz dazu feuert eine *unmittelbare* Transition, durch ein gefülltes Rechteck dargestellt, zeitgleich mit der Aktivierung. Im folgenden werden Petri–Netze nur zur funktionalen Modellierung eingesetzt. Auf die Angabe von Zufallsvariablen für zeitbehaftete Transitionen wird deshalb verzichtet.

In einem Petri–Netz können mehrere Transitionen gleichzeitig aktiviert sein. Wenn verschiedene Transitionen gemeinsame Eingangsstellen besitzen, können dabei *Konflikte* auftreten. Ein Konflikt ist eine Situation, in der das Feuern einer Transition die Aktivierung einer anderen Transition aufhebt. Der Indeterminismus bei der Lösung von Konflikten trägt wesentlich zur Ausdrucksmächtigkeit von Petri–Netzen bei. Auf eine detaillierte Beschreibung von Konflikten wird an dieser Stelle dennoch verzichtet, da sie in den vorgestellten Petri–Netzen durch hemmende Kanten deterministisch aufgelöst werden.
Abbildung 10.3 zeigt ein Petri–Netz, in dem ein potentieller Konflikt durch eine hemmende Kante vermieden wird. Ohne die als **if** bezeichnete Stelle und die dazugehörigen Kanten wären beide Transitionen gleichzeitig aktiviert, das Feuern einer der Transitionen würde jedoch die Aktivierung der anderen aufheben. Die Stelle **if** ist eine Eingangstelle der Transition **then** und eine hemmende Stelle der Transition **else**. Der potentielle Konflikt wird somit deterministisch aufgelöst: *wenn* sich eine Marke in der Stelle **if** befindet, *dann* ist nur die Transition **then** aktiviert, *sonst* nur die Transition **else**.

10.2 Kommunikation auf dem MegaCluster

Das Transportsystem auf Transputer-Clustern ist als eine Gruppe von Transputer-Prozessen mit hoher Priorität realisiert. Auftragspakete für Cosy-Kerne auf anderen Knoten werden über einen Transputer-Kanal an das Transportsystem übergeben. Die Auslieferung von Auftragspaketen an den lokalen Kern erfolgt über einen zweiten Transputer-Kanal. Das Transportsystem leitet Auftragspakete, die einen Knoten nur passieren, ohne Mitwirkung des Cosy-Kerns weiter.

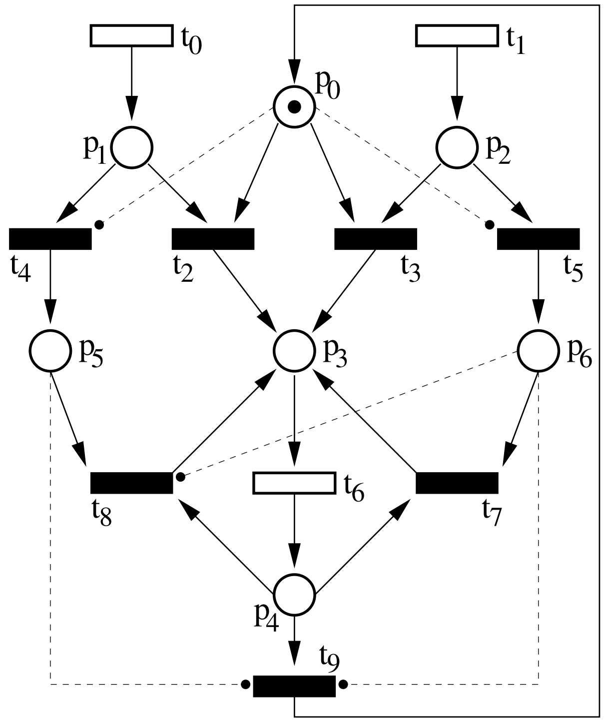

Abbildung 10.4: Zugang zum Transportsystem auf dem MegaCluster

Kommunikationsleitungen, die zu Nachbarknoten führen, werden nach der Inbetriebnahme nur vom Transportsystem benutzt. Falls weitere Leitungen vorhanden sind, können diese vom Kern genutzt und Cosy-Prozessen zur Verfügung gestellt werden, zum Beispiel zur Kommunikation mit einem Wirtsrechner. Der Zugriff des Kerns auf diese Leitungen kann über eine erweiterte Schnittstelle des Transportsystems, über ein vom Transportsystem abgesetztes System zur direkten Kommunikation oder durch direkten Zugriff auf die Kommunikationshardware erfolgen. In Cosy für Transputer wird ein abgesetztes System zum Zugriff auf die freien Kommunikationsleitungen verwendet, das wiederum als Gruppe von Transputer-Prozessen realisiert ist.

Der interne Aufbau der beiden Prozeßgruppen ist für die weiteren Überlegungen nicht relevant. Statt dessen zeigt Abbildung 10.4, wie der Cosy-Kern anfallende Aufträge in Pakete verpackt und an das Transportsystem übergibt.
Die zeitbehaftete Transition t_6 in der unteren Hälfte des Bildes modelliert das Transportsystem. Die Aktivierung dieser Transition entspricht der Übergabe eines Auftragspakets an das Transportsystem, das Feuern steht für das erfolgte Absenden. Das Transportsystem ist so realisiert, daß zu jedem Zeitpunkt maximal ein Auftragspaket abgeschickt werden kann. Im Modell bedeutet dies, daß in der Eingangsstelle p_3 des Transportsystems höchstens eine Marke enthalten sein darf. Dies muß vom Cosy-Kern garantiert werden.
Die Übergabe eines Auftragspakets an das Transportsystem erfolgt über das Feuern der Transitionen t_2 oder t_3. Beide Transitionen besitzen eine gemeinsame Eingangsstelle p_0, in der sich nur eine Marke befindet. Diese Marke zeigt

an, daß das Transportsystem aufnahmebereit ist. Da sie bei der Übergabe eines Auftragspakets an das Transportsystem entnommen wird, ist tatsächlich sichergestellt, daß sich in p_3 höchstens eine Marke befinden kann.

Aufträge zur Bearbeitung auf einem entfernten Knoten können auf zwei Arten entstehen. Der Aufruf einer Kernprozedur auf einem entfernten Knoten durch einen lokalen Prozeß wird durch das Feuern der Transition t_0 modelliert. Diese Transition hat keine Eingangsstelle und ist somit immer aktiviert. Dies entspricht der Tatsache, daß zu jedem Zeitpunkt ein entfernter Kernaufruf stattfinden kann.
Die Notwendigkeit, ein Paket an einen entfernten Knoten zu versenden, kann aber auch aus internen Abläufen des COSY-Kerns entstehen. So müssen zum Beispiel nach der Ausführung einer Kernprozedur, die von einem anderen Knoten aus aufgerufen wurde, Resultatsparameter zurückgeschickt werden.[1] Dies wird durch das Feuern der Transition t_1 modelliert.
Feuert die Transition t_0, während das Transportsystem beschäftigt ist, so ist durch die abgelegte Marke in p_1 nur die Transition t_4 aktiviert, da sich keine Marke in p_0 befindet. In diesem Fall feuert t_4 unmittelbar, die entstandene Marke wird in p_5 abgelegt. Die Stelle p_5 entspricht einer Folge wartender Prozesse. Die Informationen über den blockierten, entfernten Kernaufruf können im Prozeßleitblock abgelegt werden.
Ist das Transportsystem nicht beschäftigt, so ist statt der Transition t_4 die Transition t_2 aktiviert. In diesem Fall wird ein Auftragspaket zusammengesetzt und abgeschickt. Zu diesem Zweck ist ein Speicherbereich reserviert, solange sich eine Marke in p_0 befindet.
Ein Auftrag, der durch die Arbeit des COSY-Kerns entsteht, entspricht dem Feuern der Transition t_1. Wenn sich eine Marke in p_0 befindet, so feuert die Transition t_3, das heißt, ein Auftragspaket wird abgeschickt. Anderenfalls muß der Auftrag in eine Folge abzuarbeitender Aufträge eingefügt werden, die im Petri–Netz durch die Stelle p_6 modelliert ist.

Nachdem ein Auftragspaket abgeschickt wurde, also die Transition t_6 gefeuert hat, befindet sich eine Marke in der Stelle p_4. Gleichzeitig liefert das Transportsystem einen leeren Puffer zurück, in dem ein neues Auftragspaket abgelegt werden kann. Die Marke in p_4 aktiviert eine der Transitionen t_7, t_8 oder t_9. Der potentielle Konflikt wird durch hemmende Kanten aufgelöst.
Falls sich ein Auftrag des Kerns in der Folge abzuarbeitender Aufträge, also eine Marke in p_6, befindet, so wird dieser nun bearbeitet, was einem Feuern der Transition t_7 entspricht. Ist die Folge abzuarbeitender Aufträge leer, so kann der entfernte Kernaufruf eines Prozesses aus der Folge wartender Prozesse in p_5 ausgeführt werden. Ist auch diese Folge leer, so geht das Transportsystem durch das Feuern der Transition t_9 wieder in den Bereit-Zustand über. Der leere Puffer für ein Auftragspaket wird hinterlegt und steht zur Verfügung, sobald durch das Feuern von t_0 oder t_1 ein neuer Auftrag entsteht.

[1] Solche Antwortpakete können auch als „Auftrag“ zur Übergabe der Resultatsparameter interpretiert werden. Die Bezeichnung „Auftragspaket“ schließt also Antwortpakete ein.

10.3 Kommunikation auf dem PowerXplorer

Auf den Knoten eines PowerXplorers ist neben dem PowerPC-Prozessor auch ein Transputer-Prozessor vorhanden, der über lokalen Speicher verfügt. Durch diesen Aufbau ergibt sich ein Gestaltungsspielraum: Das Transportsystem kann auf den PowerPC-Prozessor migriert werden, so daß der Transputer-Prozessor nur direkte lesende und schreibende Zugriffe auf die Kommunikationsleitungen abwickeln muß. Es ist aber auch möglich, das Transportsystem auf den Transputer-Prozessoren arbeiten zu lassen und eine geeignete Schnittstelle anzubieten, die neben dem direkten Zugriff auf Kommunikationsleitungen auch das Abschicken und Entgegennehmen von Auftragspaketen erlaubt.

Das entscheidende Argument gegen ein Transportsystem auf dem PowerPC ist, daß der PowerPC-Prozessor eines Knotens auch durch das Weiterleiten von Nachrichten belastet würde, die diesen Knoten nur passieren. Die dazu notwendige Übertragung der Nachrichten in und aus dem gemeinsamen Speicher von PowerPC und Transputer würde zudem die Übertragungsgeschwindigkeit erheblich reduzieren, da Zugriffe des Transputers auf den gemeinsamen Speicher um den Faktor 3 langsamer erfolgen als Zugriffe auf den lokalen Speicher.
Auf dem Transputer ist das Transportsystem zudem ohne Änderungen lauffähig. Um es zu nutzen, ist nur eine Anbindung an den PowerPC-Kern erforderlich. Dieses Vorgehen entlastet den PowerPC-Prozessor und erlaubt es, den lokalen Speicher des Transputers als Pufferspeicher für Auftragspakete sinnvoll zu nutzen. Aus diesen Gründen wurde das Transportsystem nicht auf den PowerPC-Prozessor migriert.

Der folgende Abschnitt 10.3.1 enthält eine konzeptionelle Beschreibung der Anbindung des Transportsystems auf dem PowerXplorer. Abschnitt 10.3.2 beschäftigt sich mit wichtigen Details der Realisierung.

10.3.1 Konzept

Um der neuen Situation auf einem PowerXplorer Rechnung zu tragen, muß das in Abbildung 10.4 auf Seite 88 dargestellte Petri–Netz erweitert werden. Die Transition t_6, die das Transportsystem darstellt, wird weiterhin auf einem Transputer realisiert. Die restlichen Teile des Petri–Netzes sind Teile des Cosy-Kerns und werden folglich auf dem PowerPC-Prozessor eines Knotens ausgeführt.
Neben dem Abschicken von Datenpaketen muß es auch möglich sein, Pakete entgegenzunehmen und Kommunikationsleitungen direkt zu benutzen. Die Anbindung auf dem Transputer stellt entsprechende Dienste zur Verfügung.

Abbildung 10.5 auf Seite 91 zeigt ein Petri–Netz, das die Anbindung des Transportsystems modelliert. Im linken Teil der Abbildung ist der Teil der Anbindung dargestellt, der auf dem PowerPC-Prozessor ausgeführt wird. Der rechte Teil beschreibt die Anbindung auf dem Transputer. Der umrandete Bereich modelliert die Schnittstelle zwischen den Prozessoren.

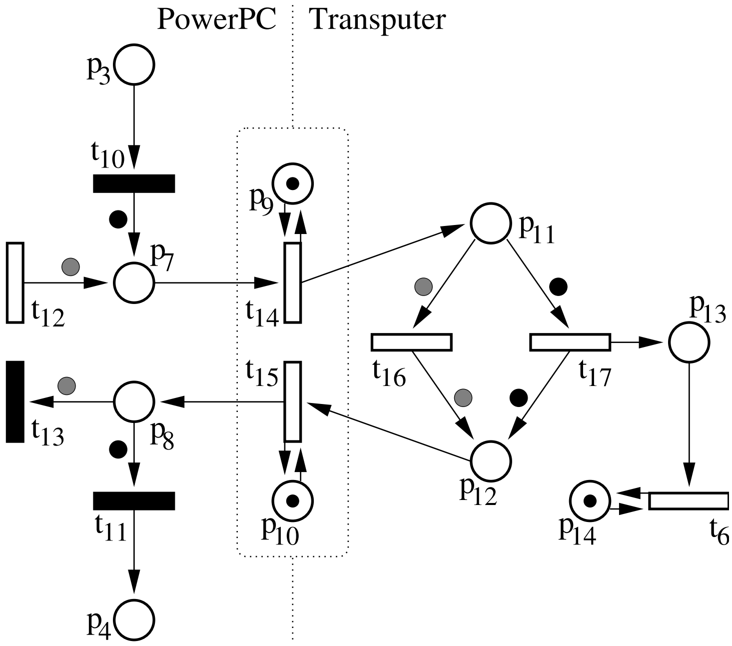

Abbildung 10.5: Zugang zum Transportsystem auf dem PowerXplorer

Das dargestellte Petri–Netz ist als Erweiterung des Petri–Netzes in Abbildung 10.4 auf Seite 88 zu verstehen. Im PowerPC-Bereich der Abbildung 10.5 finden sich die Stellen p_3 und p_4, die in Abbildung 10.4 noch Ein- und Ausgangsstelle der Transition t_6 waren. Der restliche Teil des neuen Petri–Netzes ersetzt die Transition t_6. Diese Transition des alten Modells befindet sich nun auf der rechten Seite im Transputer-Bereich.

Innerhalb der Anbindung werden die verschiedenen Arten von Aufträgen durch unterschiedlich *gefärbte* Marken gekennzeichnet. Schwarze Marken stehen für Aufträge zum Versenden von Datenpaketen. Graue Marken symbolisieren die Aufträge zur direkten Nutzung von Leitungen oder zum Entgegennehmen von ankommenden Paketen.
Einige Kanten sind mit solchen Marken versehen. Eine Marke an einer Ausgangskante bedeutet, daß die Transition beim Feuern Marken der entsprechenden Farbe ablegt. Eine Marke an einer Eingangskante deutet an, daß sich in der jeweiligen Stelle eine Marke der entsprechenden Farbe befinden muß, um die Transition zu aktivieren. Transitionen mit Kanten ohne Markierung (t_{14}, t_{15}) werden durch Marken beliebiger Farbe aktiviert und feuern die aktivierenden Marken, ohne deren Farbe zu verändern.

Der Zugang zum Transportsystem erfolgt zunächst wie in Abschnitt 10.2 beschrieben. Sobald eine Marke in p_3 eintrifft, wird diese durch die Transition t_{10} in die Stelle p_7 transportiert. Das Feuern der Transition t_{12} modelliert einen direkten Zugriff auf eine Leitung oder einen Auftrag zum Entgegennehmen eines Datenpakets. Dabei wird eine graue Marke in der Stelle p_7 abgelegt. Alle Aufträge für den Transputer sammeln sich also zunächst in p_7.

Das Feuern der Transition t_{14} entspricht der Übergabe eines Auftrags vom PowerPC zum Transputer. Die Stelle p_9 garantiert, daß zu jedem Zeitpunkt nur ein Auftrag übergeben wird. Diese Einschränkung vereinfacht die Schnittstelle zwischen den Prozessoren.
Die Marken in p_{11} aktivieren entsprechend ihrer Farbe entweder t_{16} oder t_{17}. Die Transition t_{16} bearbeitet Aufträge, die durch eine graue Marke dargestellt sind. Beim Verschicken eines Datenpakets, also einer schwarzen Marke, wird durch das Feuern von t_{17} eine Kopie dieser Marke in der Stelle p_{13} abgelegt. Dies entspricht einem Umkopieren des Pakets vom gemeinsamen Speicher in den lokalen Speicher des Transputers. In p_{13} sammeln sich Datenpakete, die nacheinander an das Transportsystem übergeben werden. Die Stelle p_{14} garantiert, daß das Transportsystem zu jedem Zeitpunkt nur einen Auftrag zu bearbeiten hat.
In der Stelle p_{12} sammeln sich Marken für bearbeitete Aufträge. Über die Transition t_{15} werden diese Marken wieder vom Transputer- an den PowerPC-Prozessor übergeben. Bearbeitete Aufträge zum direkten Zugriff auf Leitungen und entgegengenommene Datenpakete werden durch die Transition t_{13} aus der Stelle p_8 entnommen. Meldungen über kopierte Datenpakete werden von t_{11} in die Stelle p_4 transportiert. Der weitere Ablauf wurde bereits in Abschnitt 10.2 für das Petri–Netz in Abbildung 10.4 auf Seite 88 beschrieben.

10.3.2 Realisierung

In Abschnitt 8.2 wurde bereits erwähnt, daß im Kernadreßraum ein Speicherbereich für die Kommunikation zwischen PowerPC und Transputer reserviert ist. In diesem Bereich stimmen Programm- und Speicheradressen des PowerPC überein. Der Transputer verwendet ausschließlich Speicheradressen. Deshalb ist eine Adresse in diesem Bereich sowohl auf dem Transputer als auch im PowerPC-Kern gültig, was die Realisierung der Interaktion vereinfacht.
Der gemeinsam genutzte Speicherbereich ist in ein *Kontrollfeld* und ein *Datenfeld* aufgeteilt. Das Kontrollfeld dient zur Koordination der beiden Prozessoren. Im Datenfeld werden Auftrags- und Antwortpakete abgelegt.

Abbildung 10.6 zeigt eine Verfeinerung des Petri–Netzes in Abbildung 10.5. Die Schnittstelle zwischen den Prozessoren, die durch das Kontrollfeld im gemeinsamen Speicherbereich implementiert wird, ist in beiden Petri–Netzen umrandet. Abbildung 10.6 beschreibt diese Schnittstelle im Detail.
Die Stellen p_{15} und p_{16} entsprechen zwei Kanälen zwischen den Prozessoren. Dabei führt p_{15} vom PowerPC zum Transputer und p_{16} vom Transputer zum PowerPC. Die Transitionen t_{14} und t_{15} aus Abbildung 10.5 wurden in jeweils zwei Transitionen aufgeteilt, von denen eine (t_{xS}) die Sende- und eine (t_{xE}) die Empfangsoperation am jeweiligen Kanal modelliert.
Für die Stellen p_{15} und p_{16} ist im Kontrollfeld jeweils ein Speicherwort reserviert. In diesem Speicherwort wird bei der Sendeoperation, also beim Ablegen einer Marke, eine Adresse im Datenfeld des gemeinsamen Speicherbereichs eingetragen. An dieser Adresse ist ein Auftrags- oder Antwortpaket abgelegt, in dem auch die Art des Pakets, also die Farbe der Marke, vermerkt ist.

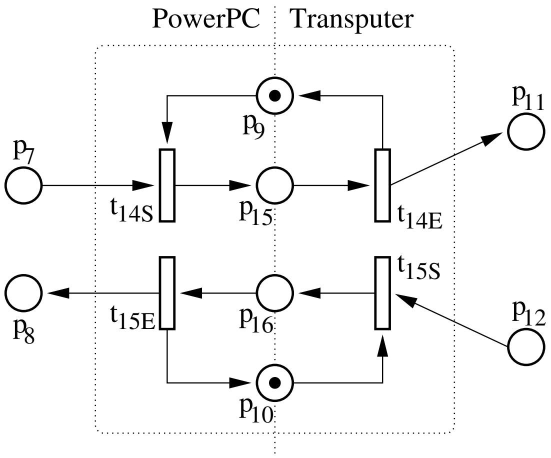

Abbildung 10.6: Schnittstelle zwischen Transputer und PowerPC

Für die Stellen p_9 und p_{10} ist kein Platz im Kontrollfeld vorgesehen. Statt dessen wird bei einer Empfangsoperation die abgelegte Adresse ausgelesen und durch den Wert 0 ersetzt. Der Wert 0 in der Speicherstelle für p_{15} (p_{16}) zeigt an, daß sich eine Marke in p_9 (p_{10}) befindet.

Damit nicht beide Prozessoren regelmäßig den aktuellen Zustand der Schnittstelle überprüfen müssen, werden Änderungen über ein Hardware-Signal an den jeweils anderen Prozessor gemeldet. Ein Prozessor, der ein solches Signal erhält, prüft daraufhin beide Kanäle. Durch dieses Vorgehen ist es nicht notwendig, zwischen Signalen für den Hin- und Rückkanal zu unterscheiden. Da sich die Überprüfung eines Kanals auf das Auslesen eines Speicherwortes beschränkt, kann der zusätzliche Aufwand vernachlässigt werden.
Das Signalisieren einer Änderung an der Schnittstelle kann auf beiden Prozessoren durch das Schreiben eines vorgegebenen Wertes in ein Hardware-Register erfolgen. Vor dem Löschen dieses Signals muß jedoch gesichert sein, daß es vom Partner-Prozessor erkannt wurde. Hier ergeben sich Probleme durch erhebliche Unterschiede zwischen dem PowerPC und dem Transputer.

Der PowerPC-Prozessor reagiert auf ein Hardware-Signal durch das Auslösen einer Ausnahmebehandlung.[2] Das Signal darf erst zurückgesetzt werden, nachdem diese Ausnahmebehandlung initiiert wurde. Die maximale Reaktionszeit des COSY-Kerns ist lang, da Ausnahmen durch ein externes Signal nicht erkannt werden, während der PowerPC-Prozessor eine COSY-Kernprozedur ausführt. Durch einfaches Abwarten der maximalen Reaktionszeit mit anschließendem Rücksetzen des Signals wird deshalb keine effiziente Kommunikation erreicht. Statt dessen muß der Transputer von der Ausnahmebehandlung des PowerPC darüber informiert werden, daß er das Signal löschen kann. Diese Information kann auf zwei Arten zum Transputer übertragen werden: durch ein Antwort-Signal oder über den gemeinsamen Speicher.

[2]Eine Ausnahmebehandlung aufgrund eines solchen Signals wird auch als Unterbrechung (engl.: Interrupt) bezeichnet.

Der PowerPC-Prozessor kann nur ein Hardware-Signal an den Transputer übertragen. Dieses Signal ist aber bereits mit einer Bedeutung belegt, da es Änderungen an der Schnittstelle zwischen den Prozessoren melden soll. Prinzipiell ist es möglich, ein solches Signal mit einer zusätzlichen Bedeutung zu überladen. Die resultierenden Protokolle sind jedoch komplex und unübersichtlich. Auch die Sicherheit und Verklemmungsfreiheit kann nur schwer überprüft werden. Da Verständlichkeit ein wichtiges Ziel beim Entwurf eines Systems ist, wurde in diesem Fall die zweite Alternative gewählt, also eine Bestätigung des Signals über den gemeinsamen Speicher.

Im Kontrollfeld des gemeinsam genutzten Speicherbereichs ist, neben den beiden Kanälen zur Kommunikation, ein weiteres Speicherwort reserviert. Bevor der Transputer ein Hardware-Signal an den PowerPC auslöst, initialisiert er diese Speicherstelle mit dem Wert 0. Der PowerPC-Prozessor verändert zu Beginn der Ausnahmebehandlung diesen Wert. Der Transputer überprüft nach dem Auslösen des Signals ständig den Wert dieser Speicherstelle, bis dieser sich ändert. Daraufhin wird das Signal gelöscht.
Bei Messungen wurde festgestellt, daß die Bestätigung für ein Signal vom Transputer zum PowerPC meist nach ein bis zwei Speicherzugriffen des Transputers vorliegt. Führt der PowerPC zum Zeitpunkt der Signalisierung jedoch eine aufwendige Kernprozedur aus, in der zum Beispiel ein Datenpaket für einen entfernten Kernaufruf erstellt wird, so kann die Zeit bis zur Bestätigung erheblich größer sein. In einem solchen Fall beeinträchtigen ständige Zugriffe des Transputers auf den gemeinsamen Speicher die Geschwindigkeit des PowerPC-Prozessors erheblich. Deshalb erfolgt die Abfrage der Speicherstelle in Intervallen von einigen Mikrosekunden, falls die Bestätigung nicht schon nach wenigen Zugriffen vorliegt.
Signale vom Transputer zum PowerPC werden von einem Dienstgeber auf dem Transputer ausgelöst. Dieser ist als Transputer-Prozeß niedriger Priorität implementiert und ersetzt den in Abschnitt 8.1.1 beschriebenen Cosy-Kern auf dem Transputer. Transputer-Prozesse mit hoher Priorität können den Dienstgeber-Prozeß verdrängen, so daß der Transputer weiterhin auf Ereignisse, zum Beispiel eintreffende Datenpakete oder Signale des PowerPC-Prozessors, reagieren kann.

Erhält der Transputer ein Signal über eine Hardware-Leitung, so deblockiert er einen Transputer-Prozeß, der am sogenannten Ereignis-Kanal blockiert ist. Daraufhin meldet der Transputer über eine Hardware-Rückleitung, daß das Signal erkannt wurde. Diese Bestätigung erfolgt innerhalb weniger Mikrosekunden. Der PowerPC-Prozessor kann die Rückleitung in einer Warteschleife abfragen und nach Eingang der Bestätigung das Signal löschen. Auf dem Transputer muß bei der Signalisierung ein Prozeß am Ereignis-Kanal blockiert sein, da sonst keine Bestätigung erfolgt.
Das Petri–Netz in Abbildung 10.7 zeigt das Protokoll zur Signalisierung vom PowerPC zum Transputer. Im linken Bereich der Abbildung wird durch die Stellen p_0 und p_1 sowie die Transitionen t_0 und t_6 die Aktivität des PowerPC-Prozessors modelliert. Eine Marke in p_0 zeigt an, daß der PowerPC aktiv ist, also zum Beispiel einen Cosy-Prozeß abarbeitet. Das Feuern von t_0 entspricht einer Signalisierung zum Transputer. Dabei geht der PowerPC-Prozessor in

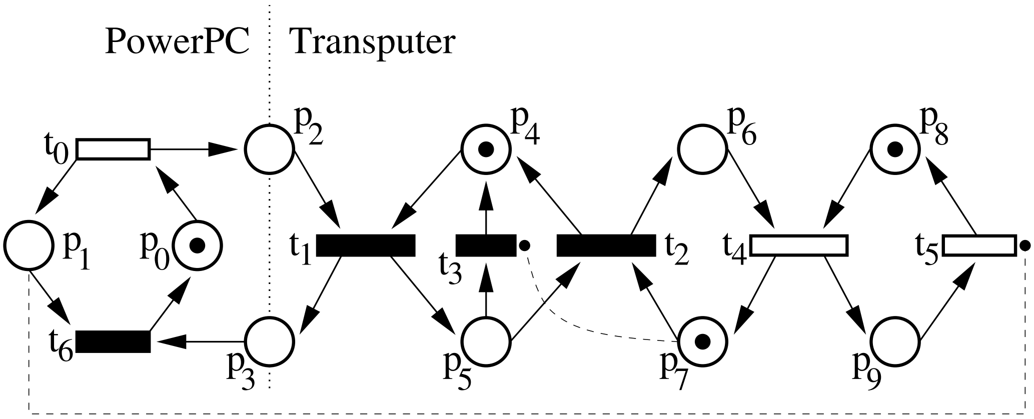

Abbildung 10.7: Signalisierung zum Transputer

einen Wartezustand über (p_1). Gleichzeitig wird eine Marke in p_2 abgelegt. Dies entspricht einem Signal auf der Hardware-Leitung zum Transputer. Der PowerPC-Prozessor wartet, bis über die Rückleitung eine Bestätigung für das Signal eintrifft, also eine Marke in p_3 abgelegt wird. Solange der PowerPC-Prozessor wartet, kann er Signale des Transputers nicht bestätigen.
Die Stellen p_8 und p_9 modellieren den Dienstgeber auf dem Transputer. Dieser muß Signale an den PowerPC senden (t_5) und blockiert, falls ein solches Signal nicht bestätigt wird. Dies ist durch die hemmende Kante von p_1 zur Transition t_5 angedeutet.
Eine Bestätigung des Transputers für ein Signal vom PowerPC erfolgt durch das Feuern der Transition t_1. Dazu muß sich eine Marke in p_4 befinden, also ein Prozeß am Ereignis-Kanal blockiert sein. Zu diesem Zweck existiert ein Transputer-Prozeß hoher Priorität, der auf Signale des PowerPC-Prozessors wartet. Nach der Deblockierung dieses Prozesses, also dem Feuern von t_1, befindet sich eine Marke in p_5. Der deblockierte Prozeß muß nun das Signal an den Dienstgeber auf dem Transputer melden. Blockiert der Prozeß beim Absetzen dieser Meldung, so würde ein weiteres Signal zu einer Blockierung des PowerPC-Prozessors führen. Sendet dann der Dienstgeber auf dem Transputer ein Signal an den PowerPC, so wird dieses nicht bestätigt und der Dienstgeber blockiert ebenfalls. Es entsteht also eine Verklemmung.

Damit keine Verklemmung entsteht, darf der Prozeß, der Signale des PowerPC-Prozessors entgegennimmt, beim Weiterleiten eines Signals nicht blockieren. Deshalb existiert ein *Zwischenprozeß*, der ebenfalls als Transputer-Prozeß hoher Priorität realisiert ist. In Abbildung 10.7 ist er durch die Stellen p_6 und p_7 modelliert. Seine Aufgabe beschränkt sich darauf, Nachrichten über Ereignisse entgegenzunehmen und weiterzuleiten. Da Transputer-Prozesse hoher Priorität nicht verdrängt werden, blockiert er entweder beim Empfangen (p_7) oder beim Weiterleiten (p_6) einer Nachricht. Im Petri–Netz modelliert die Transition t_2 den Empfangskanal und die Transition t_4 den Sendekanal dieses Prozesses.[3]

[3] Auf die Modellierung der in Abbildung 8.1 auf Seite 70 eingezeichneten Verwaltungsprozesse wurde zugunsten der Übersichtlichkeit des Petri–Netzes verzichtet.

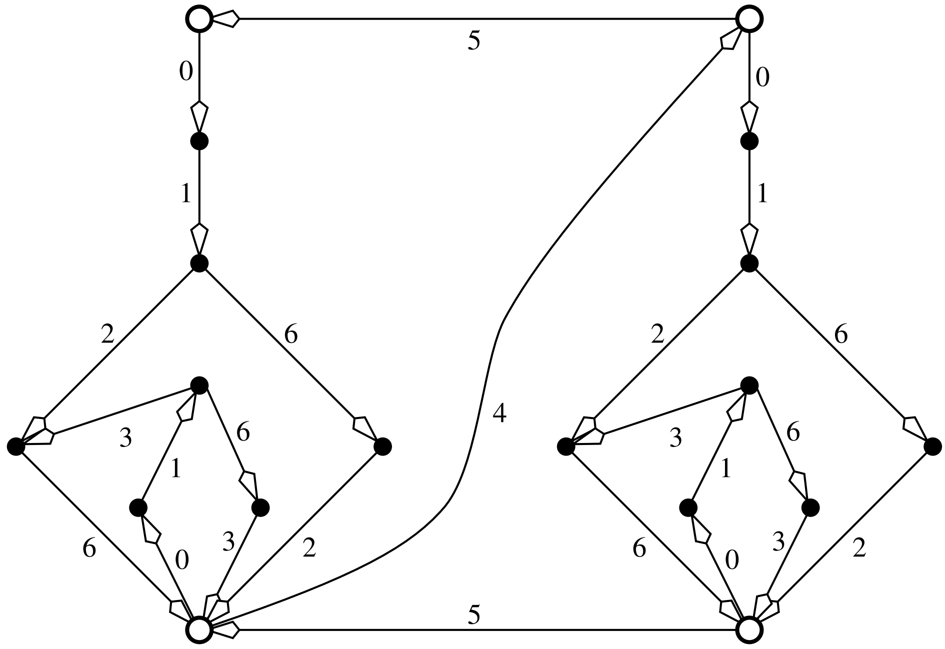

Abbildung 10.8: Zustandsübergänge bei der Signalisierung

Nachdem die Transition t_1 gefeuert hat, also ein Signal des PowerPC-Prozessors eingetroffen ist, muß die Marke in p_5 wieder nach p_4 transportiert werden. Das bedeutet, daß der Transputer-Prozeß wieder am Ereignis-Kanal blockieren muß, um das nächste Signal bestätigen zu können. Ist der Zwischenprozeß empfangsbereit, so befindet sich eine Marke in p_7. Es kann eine Nachricht über das eingetroffene Signal an den Zwischenprozeß gesendet werden, ohne daß der Sender dabei blockiert. Dies entspricht einem Feuern von t_2. Der Zwischenprozeß blockiert danach an seiner Sendeoperation, die Marke aus p_7 befindet sich in p_6. Der Dienstgeber auf dem Transputer kann die Nachricht empfangen (t_4) und bearbeiten (t_5).
Ist der Zwischenprozeß beim Eintreffen eines Signals bereits an seiner Sendeoperation blockiert, so feuert statt der Transition t_2 die Transition t_3. In diesem Fall wird also das empfangene Signal ignoriert, um eine Verklemmung zu vermeiden. Da der Zwischenprozeß bereits eine Nachricht über ein zuvor erkanntes Signal weiterleitet, führt das Ignorieren weiterer Signale nicht zu unerwünschten Folgen. Sobald der Dienstgeber die Nachricht des Zwischenprozesses empfängt, also t_4 feuert, überprüft er die beiden Kanäle der Schnittstelle zum PowerPC. Bei dieser Überprüfung werden auch die Änderungen erkannt, auf die ignorierte Signale aufmerksam machen sollten.

Um die Lebendigkeit des Petri–Netzes und damit die Verklemmungsfreiheit des Protokolls nachzuweisen, kann der Zustandsübergangsgraph des Petri–Netzes bestimmt werden. Dieser ist in Abbildung 10.8 dargestellt. Zustände sind durch Kreise symbolisiert, Zustandsübergänge durch Pfeile. Die Nummer eines Pfeils gibt an, welche Transition des Petri–Netzes in Bild 10.7 feuert, um den entsprechenden Zustandsübergang zu bewirken.

Vier der Zustände sind durch einen größeren, umrandeten Kreis dargestellt. Diese Zustände sind *beobachtbar*. Ein Zustand ist beobachtbar, wenn keine unmittelbare Transition aktiviert ist, durch die sofort ein Folgezustand erreicht würde. Nicht beobachtbare Zustände werden als *verschwindende* Zustände bezeichnet. Der beobachtbare Zustand im linken, oberen Bereich des Zustandsübergangsgraphen entspricht der in Abbildung 10.7 eingezeichneten Anfangsmarkierung des Petri–Netzes.

In den beobachtbaren Zuständen des Modells in Abbildung 10.7 kann die Transition t_0 feuern, also ein Signal vom PowerPC zum Transputer übertragen werden. Deshalb kann jeder beobachtbare Zustand aus Abbildung 10.8 über einen Pfeil verlassen werden, der mit der Nummer 0 beschriftet ist.
Die vier beobachtbaren Zustände entsprechen den möglichen Markierung der Stellen p_6 bis p_9 des Petri–Netzes. In den oberen beiden beobachtbaren Zuständen enthält die Stelle p_7 eine Marke, die Stelle p_6 nicht. Das bedeutet, daß der Zwischenprozeß an seiner Empfangsoperation blockiert, also auf eine Nachricht über ein Signal wartet. Löst der PowerPC in einem dieser Zustände ein Signal aus, so geht das Modell über einige Zwischenzustände in den jeweiligen unteren beobachtbaren Zustand über. Dabei feuert die Transition t_2 genau einmal, wodurch die Marke aus p_7 nach p_6 transportiert wird. Anschließend blockiert der Zwischenprozeß an seiner Sendeoperation.
In den linken Zuständen enthält die Stelle p_8 eine Marke, die Stelle p_9 nicht. Dies gilt sowohl für die beobachtbaren als auch für die verschwindenden Zustände und bedeutet, daß der Dienstgeber auf dem Transputer auf ein Signal des PowerPC wartet. Ein Übergang vom linken in den rechten Bereich findet statt, wenn die Transition t_4 feuert. Die entsprechende Kante verläuft diagonal vom linken unteren zum rechten oberen der beobachtbaren Zustände. Der Übergang vom rechten in den linken Teil des Zustandsübergangsgraphen erfolgt über eine der beiden waagerechten Kanten. Dabei feuert die Transition t_5, der Dienstgeber reagiert also auf ein empfangenes Signal.

Die Verklemmungsfreiheit des Protokolls ist daran zu erkennen, daß der Zustandsübergangsgraph zusammenhängend ist. Von jedem Zustand aus kann jeder andere Zustand erreicht werden. Für diese Eigenschaft ist das Ignorieren von Signalen, dargestellt durch die rautenförmigen Konstellationen über den unteren beobachtbaren Zuständen, entscheidend.
Durch das Feuern von t_0, also ein weiteres Signal des PowerPC-Prozessors, wird der erste Zustand in einer dieser Konstellationen erreicht. Über einige verschwindende Zwischenzustände kehrt das Modell wieder in den beobachtbaren Zustand zurück. In einem solchen Zyklus feuert die Transition t_3, was dem Ignorieren des Signals entspricht. Ohne diese Transition würde im rechten Teil des Zustandsübergangsgraphen ein neuer Zustand entstehen, aus dem kein Übergang mehr möglich ist. Ein solcher Zustand entspricht einer Verklemmungssituation.

Kapitel 11

Bewertung

Um das Resultat einer Portierung zu bewerten, muß ein Vergleich des zu portierenden Systems in der ursprünglichen Umgebung mit dem portierten System in der neuen Umgebung durchgeführt werden. Dieser Vergleich erfolgt nach zwei Kriterien: *Funktionsumfang* und *Leistung*.
Der Funktionsumfang darf nicht wesentlich eingeschränkt werden, damit Anwendungen mit minimalen Änderungen auf das neue System portiert werden können. Eine Erweiterung des Funktionsumfangs ist dagegen wünschenswert, um die Entwicklung neuer Anwendungen zu erleichtern. Die Geschwindigkeit sollte im Rahmen der durch die Plattform gegebenen Möglichkeiten steigen.

Bei der Portierung des Cosy-Kerns von Transputer auf PowerPC entfielen ausschließlich solche Aufrufe, die speziell auf Transputer zugeschnitten sind. Diese Kernaufrufe dienen zur Inbetriebnahme, Fehlersuche oder Leistungsmessung. Bei der Entwicklung von Anwendungen werden sie nicht verwendet. Deshalb können Anwendungen problemlos auf beiden Plattformen ausgeführt werden. Eine Erweiterung des Funktionsumfangs ergibt sich vor allem durch den implementierten Speicherschutz. Damit sind beide Forderungen bezüglich des Funktionsumfangs erfüllt.

Die Leistung eines Systems kann auf unterschiedlichste Arten bewertet werden, zum Beispiel nach dem Benutzungskomfort oder der Geschwindigkeit, mit der eine bestimmte Aufgabe bewältigt wird. Für die Einschätzung der Leistung eines Betriebssystems für Parallelrechner sind insbesondere zwei Kriterien relevant: die *lokale* Rechen- und Kommunikationsleistung auf einem Knoten und die *globale* Kommunikationsleistung, die für die Effizienz des Kernverbundes entscheidend ist.
Eine erste Einschätzung der zu erwartenden Leistungsunterschiede kann aus der Betrachtung der Architekturen der beiden Plattformen gewonnen werden. Lokale Operationen werden auf einem MegaCluster von einem Transputer-Prozessor ausgeführt, auf dem PowerXplorer dagegen von einem PowerPC-Prozessor. Der PowerPC-Prozessor wird mit einer höheren Taktfrequenz betrieben als der Transputer (80 MHz gegenüber 25 MHz). Speicherzugriffe des

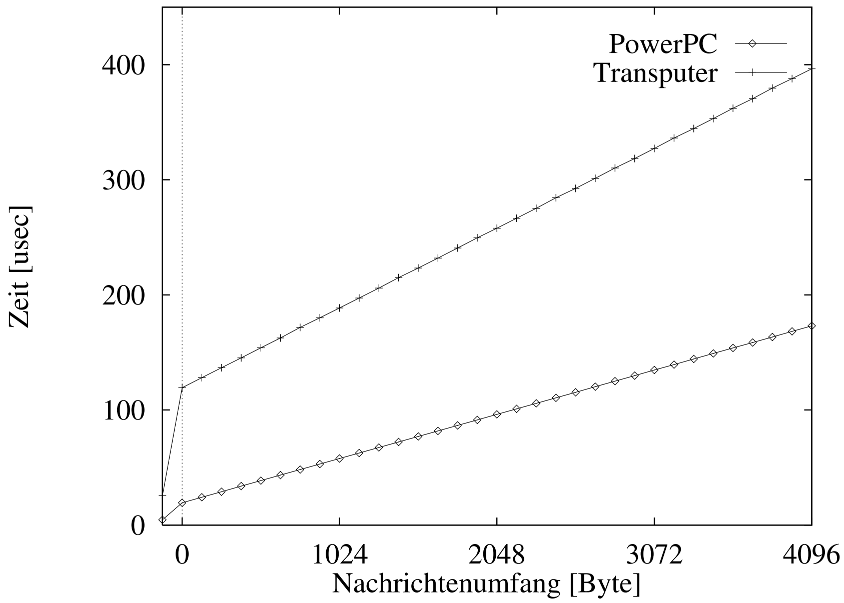

Abbildung 11.1: Lokale Kommunikation

PowerPC-Prozessors erfolgen mit doppelter Datenbreite (64 Bit gegenüber 32 Bit). Der größerer Registersatz des PowerPC-Prozessors und der integrierte Zwischenspeicher reduzieren zudem die Anzahl der notwendigen Speicherzugriffe. Deshalb sollten lokale Rechen- und Kommunikationsoperationen auf einem PowerXplorer mit einer wesentlich höheren Geschwindigkeit abgearbeitet werden als auf einem MegaCluster.
Die knotenübergreifende Kommunikation wird auf beiden Plattformen von Transputer-Prozessoren durchgeführt. Da Cosy für PowerPC die Adreßtransformation dieses Prozessors einsetzt, müssen auf einem PowerXplorer zusätzliche Kopiervorgänge zwischen Programmadreßräumen und dem von beiden Prozessoren genutzten Speicherbereich in Kauf genommen werden. Es ist deshalb zu erwarten, daß die globale Kommunikation auf einem PowerXplorer langsamer erfolgt als auf einem MegaCluster. Die Geschwindigkeitsdifferenz sollte unabhängig von der Entfernung zwischen zwei Knoten sein, da zusätzlicher Protokollaufwand nur auf dem Quell- und dem Zielknoten anfällt.

Abbildung 11.1 zeigt das Ergebnis einer lokalen Zeitmessung auf beiden Plattformen. Zu Vergleichszwecken sind auf der y-Achse die für einen leeren Kernaufruf benötigten Zeiten eingetragen. Der leere Kernaufruf dient speziell der Zeitmessung und erlaubt es, den Aufwand für eine Sequenz aus Kernein- und -austritt zu bestimmen. Die gemessenen Werte, $25\mu s$ auf dem Transputer und $4.6\mu s$ auf dem PowerPC, entsprechen den Erwartungen.
Die Meßreihe bestand aus einer Folge von lokalen Kommunikationsoperationen. Ein Meßwert setzt sich aus den Bearbeitungszeiten für zwei Kernein- und -austritte, ein asynchrones Senden, ein synchrones Empfangen, zwei Prozeßwechsel sowie ein einmaliges Umkopieren der gesendeten Nachricht zusammen. Die Nachrichtengröße ist auf der x-Achse abgetragen.

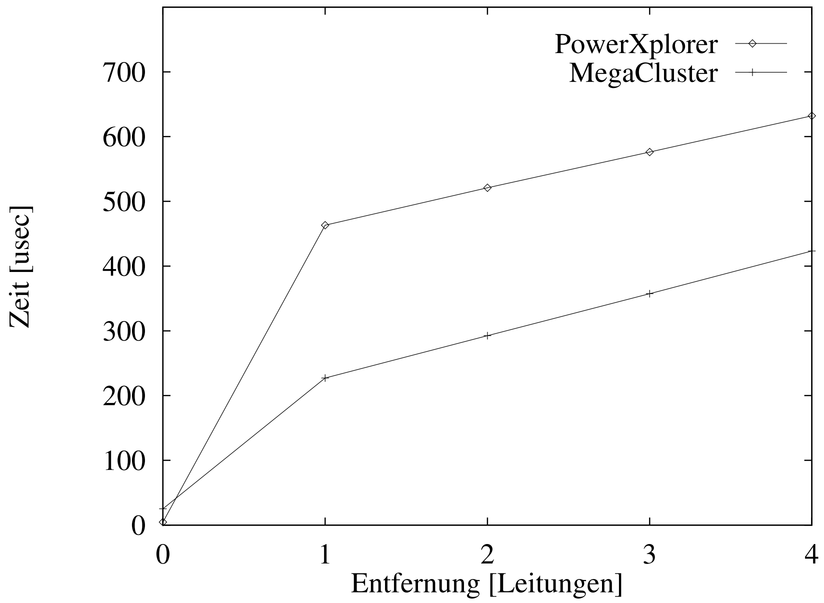

Abbildung 11.2: Entfernte Kernaufrufe

Bei den Latenzzeiten zum Senden einer Nachricht der Länge 0 zeigt sich ein deutlicher Unterschied zwischen den Prozessoren. Auf dem PowerPC-Prozessor ergibt sich eine Latenzzeit von etwa $19\mu s$, während sie auf dem Transputer bei $120\mu s$ liegt. Da die angegebenen Zeiten Mittelwerte aus je 1000 aufeinanderfolgenden Kommunikationsoperationen sind, ist dieser große Unterschied auf den Zwischenspeicher des PowerPC-Prozessors zurückzuführen, der nach der ersten Operation bereits alle Daten enthält, auf die in den folgenden Operationen zugegriffen wird.
Mit der Nachrichtengröße steigt auch die Übertragungszeit bei beiden Prozessoren linear an. Der Anstieg fällt jedoch auf dem PowerPC-Prozessor deutlich geringer aus. Hier macht sich die höhere Bandbreite bei Speicherzugriffen bemerkbar.
Das Kopieren der Nachricht könnte noch weiter beschleunigt werden. Bisher ist es durch eine einfache Schleife implementiert, die in Einheiten von einem Byte kopiert. Unter der Voraussetzung, daß Nachrichtenpuffer an einer durch 4 teilbaren Adresse[1] liegen und die Nachrichtengröße ebenfalls durch 4 teilbar ist, kann der Kopiervorgang mit Einheiten von 4 Byte durchgeführt werden.

Abbildung 11.2 zeigt die benötigten Zeiten für entfernte Kernaufrufe an. Gemessen wurde wiederum der oben beschriebene Kernaufruf, der keine Operation durchführt. Die Entfernung ist auf der x-Achse eingetragen. Als Plattform für die Messungen diente ein PowerXplorer mit 8 Knoten, die als 2x4-Gitter verschaltet sind. Die maximal erreichbare Entfernung zwischen zwei Knoten liegt in dieser Konfiguration bei 4 Leitungen, über die ein Datenpaket transportiert werden muß.

[1]Die englische Bezeichung für diese Eigenschaft lautet *word alignment*.

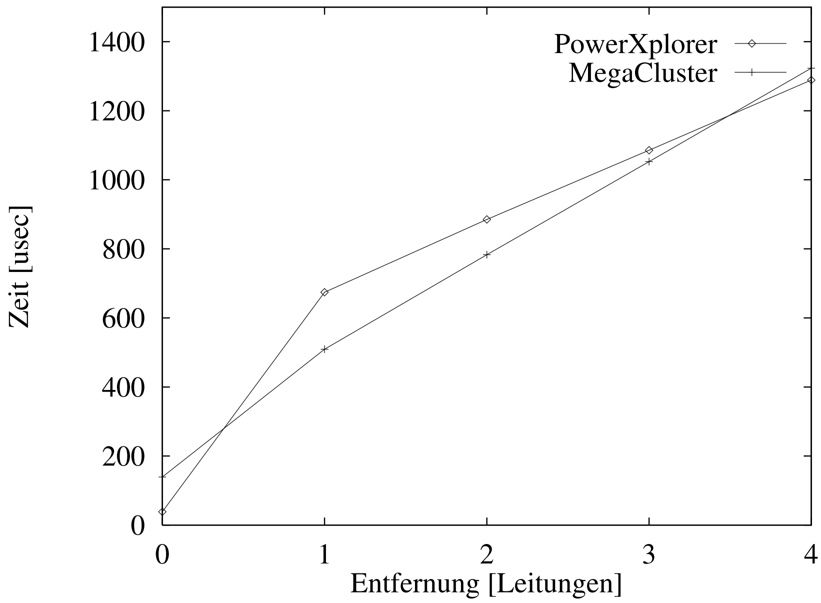

Abbildung 11.3: Entfernte Kommunikation mit kleinen Nachrichten

Das Ergebnis der Messung liegt im Rahmen der Erwartungen. Während lokale Operationen auf dem PowerPC schneller durchgeführt werden, macht sich bei entfernten Aufrufen der erhöhte Protokollaufwand deutlich bemerkbar. Auffallend ist jedoch, daß der PowerXplorer bei Aufrufen auf einem Nachbarknoten um $250\mu s$ langsamer ist, bei einer Entfernung von 5 Leitungen aber nur noch um $200\mu s$.
Die Transputer-Prozessoren eines PowerXplorer arbeiten mit der gleichen Taktfrequenz wie die in einem MegaCluster. Da Datenpakete auf dem PowerXplorer über den lokalen Speicher der Transputer weitergeleitet werden, ist auch die verfügbare Bandbreite bei Speicherzugriffen gleich. Deshalb sollten sich die Übertragungszeiten beider Plattformen um einen konstanten Wert unterscheiden.

Der in der vorhergehenden Messung aufgetretene Effekt zeigt sich noch deutlicher, wenn die Größe der zu übertragenden Datenpakete erhöht wird. In Abbildung 11.3 ist der Zeitaufwand für das asynchrone Senden einer Nachricht der Länge 256 Byte in Abhängigkeit von der Entfernung zwischen Sender und Empfänger eingezeichnet. Ab einer Entfernung von 4 Leitungen ist diese Operation auf einem PowerXplorer schneller als auf einem MegaCluster.
Offensichtlich ist die Übertragungsleistung der Transputer in einem PowerXplorer höher als in einem MegaCluster. Dies kann nicht auf Einflüsse der Hardware zurückgeführt werden. Der wesentliche Unterschied zwischen den Transputern in beiden Parallelrechnern besteht darin, daß auf dem Transputer in einem MegaCluster der vollständige COSY-Kern installiert ist, während auf dem Transputer in einem PowerXplorer nur der Kommunikationsdienstgeber abläuft. Das Transportsystem ist in beiden Fällen identisch.

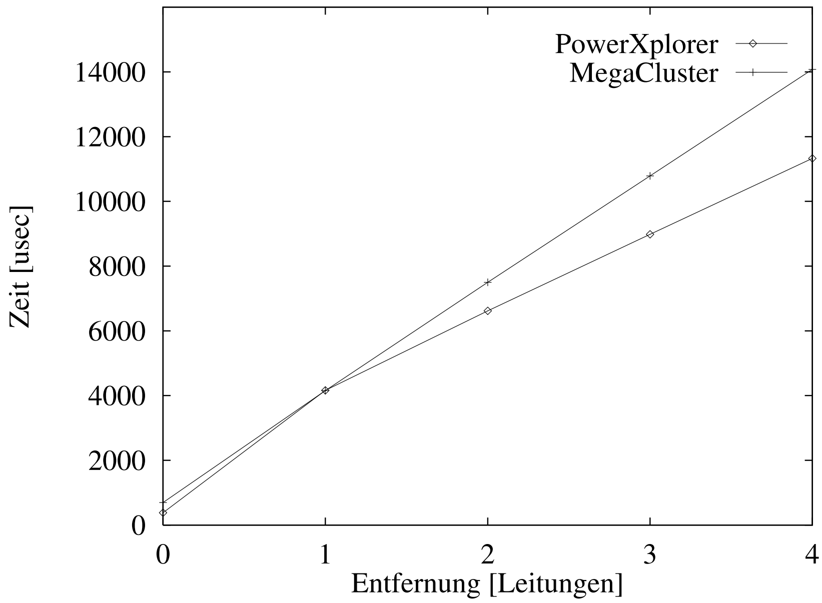

Abbildung 11.4: Entfernte Kommunikation mit großen Nachrichten

Während der Messungen war außer den Systemdienstgebern nur die Meßanwendung installiert, deren Einfluß nicht für diesen Effekt verantwortlich sein kann. Es kann sich somit nur um eine Eigenschaft des COSY-Kerns für Transputer handeln, die die Kommunikationsleistung des Transputers senkt. Als wahrscheinlichste Ursache ist die Implementierung der Systemuhr anzunehmen.
Der COSY-Kern für Transputer installiert einen Transputer-Prozeß hoher Priorität, der in Abständen von 10 Millisekunden eine Nachricht an den Kern sendet. Eine solche Nachricht wird wie ein Kernaufruf bearbeitet und bewirkt, daß der COSY-Kern die Systemuhr aktualisiert. Die Nachricht wird nicht direkt an den Kern gesendet, sondern über einen der in Abschnitt 8.1.1 beschriebenen Verwaltungsprozesse. Diese beiden Prozesse verbrauchen Rechenzeit, die für das Transportsystem nicht zur Verfügung steht.

Abbildung 11.4 zeigt eine Messung des asynchronen Sendens mit Nachrichten der Größe 4096 Byte, wiederum in Abhängigkeit von der Entfernung zwischen Sender und Empfänger. Die höhere verfügbare Übertragungsleistung der Transputer in einem PowerXplorer führt dazu, daß diese Operation für Nachbarknoten auf beiden Parallelrechnern gleich schnell ausgeführt wird. Bei größeren Entfernungen kommuniziert ein PowerXplorer schneller als ein MegaCluster.

Die Geschwindigkeit der globalen Kommunikation ist höher, als dies aufgrund der Architektur des PowerXplorer zu erwarten war. Auch in dieser Hinsicht kann die Portierung von COSY als erfolgreich bezeichnet werden. Ebenso wie bei der lokalen Kommunikation können aber auch hier Optimierungen durchgeführt werden, die die globale Kommunikationsleistung weiter steigern. Dies ist darauf zurückzuführen, daß im Rahmen dieser Diplomarbeit der Schwerpunkt auf die Portierung des Funktionsumfangs des Betriebssystems gelegt wurde.

Kapitel 12

Ausblick

Im Rahmen dieser Diplomarbeit wurde das COSY-Betriebssystem für Parallelrechner von einer Transputer- auf eine hybride Transputer/PowerPC-Architektur portiert. Zur Durchführung wurde ein systematischer Ansatz entwickelt, der unabhängig von Betriebssystem und Plattform eingesetzt werden kann.
Um die vielfältigen Möglichkeiten des PowerPC-Prozessors zu nutzen, mußten konzeptionelle Änderungen am COSY-Kern vorgenommen werden. Davon betroffen sind insbesondere die Speicherverwaltung und die Einbettung in Adreßräume. Die große Zahl der Register eines PowerPC-Prozessors wurde durch zusätzliche Änderungen beim Kerneintritt und an der Prozeßverwaltung berücksichtigt.
Zur Realisierung des COSY-Kernverbundes wurde eine Anbindung des COSY-Kerns auf dem PowerPC an einen Kommunikationsdienstgeber auf dem Transputer entworfen und implementiert. Die Anbindung wurde mit Hilfe von Petri–Netzen modelliert.
Alle Systemdienstgeber konnten mit wenigen Änderungen an den neuen COSY-Kern angepaßt werden. Als Resultat der Portierung steht COSY in vollem Umfang für PowerXplorer zur Verfügung. Messungen ergaben, daß durch die neue Architektur nicht nur eine höhere Rechenleistung zur Verfügung steht, sondern auch eine etwas höhere Kommunikationsleistung.
Dies soll nicht darüber hinwegtäuschen, daß die Möglichkeiten des PowerPC-Prozessors noch nicht in vollem Umfang genutzt werden. Eine Analyse des Kerns, etwa durch weitere Messung von Ausführungszeiten ausgewählter Kernprozeduren, kann Möglichkeiten zur Optimierung aufzeigen.

Die vorgestellte Methodik zur Durchführung von Portierungen ist, wie erwähnt, unvollständig. Im Rahmen weiterer COSY-Portierungen kann sie verfeinert, ergänzt und bei Bedarf auch korrigiert werden. Als Ziele für Portierungen könnten zum Beispiel die ebenfalls von Parsytec hergestellten Parallelrechner der Typen PowerPlus und CC dienen.
CC-Parallelrechner sind reine PowerPC-Rechner. Die Knoten verfügen über eine Kommunikationshardware, deren Unterstützung ein neues Transportsystem erfordert. Zudem ist jeder Knoten eines CC mit einer Festplatte ausgestattet,

was die Implementierung von virtuellem Speicher ermöglicht. Auch andere Peripherie, zum Beispiel ein Internet-Zugang, ist auf einigen Knoten vorhanden. Dies ermöglicht den Betrieb unabhängig von einem Wirtsrechner, erfordert aber auch die Implementierung von Peripherie-Treibern.
PowerPlus-Rechner sind Hybridrechner, bei denen jeder Knoten mit zwei PowerPC-Prozessoren ausgestattet ist. Zur Kommunikation verfügt jeder Knoten über vier Transputer-Prozessoren. Dieser Aufbau eröffnet eine Möglichkeit zur Portierung von COSY auf SMP-Architekturen.[1]

Zum Abschluß sei noch eines der bei der Entwicklung von COSY angestrebten Ziele erwähnt: der sichere *Mehrbenutzerbetrieb*. Mit der Unterstützung einer Adreßtransformationseinheit durch den PowerPC-Kern und dem damit realisierten Speicherschutz eröffnet sich nun eine konkrete Möglichkeit, dieses Ziel zu erreichen.

[1]SMP, englisch: Symmetric MultiProcessing

Literaturverzeichnis

[Baum96] Dietmar Baumstark
Experimentierumgebung zur Visualisierung und Bewertung von Zwischenspeichern
Studienarbeit an der Universität Karlsruhe (TH), 1996

[BuHe97] Roger Butenuth, Hans-Ulrich Heiß
Small, Scalable, and Efficient Microkernels for Highly Parallel Computers are possible: COSY as an Example
will appear in:
International Conference on Advances in Parallel and Distributed Computing
Shanghai, China, March 19-21, 1997

[Bute94] Roger Butenuth
The COSY-Kernel as an Example for Efficient Kernel Call Mechanisms on Transputers
Proceedings of the 6^{th} Transputer/Occam International Conference
Tokyo, June 1994

[Cam92] Rondo Cameron
Geschichte der Weltwirtschaft
Band 2: Von der Industrialisierung bis zur Gegenwart
Klett Verlag, 1992
ISBN 3-608-91430-7

[Duden] Duden: Rechtschreibung der deutschen Sprache
Bibliographisches Institut, Mannheim
Dudenverlag, 1961

[ELF93] Tools Interface Standard (TIS) Committee
Portable Formats Specification, Version 1.1
Executable and Linkable Format (ELF)
Intel Order Number 241597

[ELF95] Hongjiu Lu (hjl@nynexst.com)
ELF: From The Programmer's Perspective
1995

[GeMc88] Narain Gehani, Andrew McGettrick (Editors)
Concurrent Programming
Addison Wesley, 1988
ISBN 0-201-17435-9

[IMS88] INMOS Limited
Transputer Instruction Set - A compiler writer's guide
Prentice Hall, 1988
ISBN 0-13-929100-8

[IMS92] The Transputer Databook, Third Edition
INMOS Limited, 1992
INMOS document number: 72 TRN 203 02

[ISO90] Programming languages – C
International standard ISO/IEC 9899:1990

[Matt91] Bruno Mattes
Namensverwaltung und Objektkatalogisierung in Rechnerverbund-systemen
Dissertation an der Universität Karlsruhe (TH), 1991

[Pet62] Carl Adam Petri
Kommunikation mit Automaten
Bonn: Institut für Instrumentelle Mathematik
Schriften des IIM, Nr. 3, 1962
Englische Übersetzung:
Communication with Automata
New York: Griffiss Air Force Base
Technical Report RADC-TR-65-377
Vol 1, Suppl. 1, 1966

[PO94] PowerOpen Association, Inc.
PowerOpen Application Binary Interface
First Edition, 1994

[PPC93] PowerPC 601 RISC Microprocessor User's Manual
Motorola Inc, 1993
Part Number: MPC601UM/AD, REV 1

[PPC94] PowerPC Microprocessor Family: The Programming Environments
Motorola Inc, 1994
Part Number: MPCFPE/AD

[PXP93] Parsytec Xplorer-PowerEntry
Hardware User Guide, Rev. 1.0
PowerStone Gmbh, 1993

[Reis90] Wolfgang Reisig
Petrinetze: eine Einführung
Springer Verlag, 1990

[Schä95] Thomas Schäck
Anbindung von COSY an Unix-Wirtsrechner
Diplomarbeit an der Universität Karlsruhe (TH), 1995

[Sem93] Gesammelte Ausarbeitungen:
Spezifikation, Modellierung und Analyse mit Petrinetzen
Seminar an der Universität Karlsruhe (TH), 1992/93

[SGI96] Silicon Graphics Computer Systems
Power Challenge
Technical Report, 1996
Chapter 2

[Web96] Roland Weber
Entwurf und Implementierung eines Produktform–Algorithmus für GSPN
Studienarbeit an der Universität Karlsruhe (TH), 1996

[Wett93] Horst Wettstein
Systemarchitektur
Hanser Verlag, 1993
ISBN 3-446-17517-2

[Wett94] Horst Wettstein
Architektur von Rechnerverbundsystemen
Skriptum zur Vorlesung
Universität Karlsruhe (TH), 1994