Mehrprozessorsysteme und Parallelverarbeitung

Ein einführender Überblick

Weshalb mehrere Prozessoren?

- weil einer allein nicht ausreicht, um die nötige Verarbeitungsleistung zu erbringen,
- weil ein Verbund mehrerer kleinerer Prozessoren billiger ist als ein einziger Hochleistungsprozessor oder eine Spezialhardware,
- um die Organisations- und/oder Zeitprobleme des Multitasking durch Verteilung der Arbeit auf mehrere Verarbeitungseinrichtungen zu umgehen,
- um gewisse Eigentümlichkeiten gegebener Systemplattformen (Hard- und Software) zu umgehen,
- um die Peripherieausstattung zu erweitern,
- um die Betriebszuverlässigkeit zu erhöhen.

Ein Prozessor allein tut es nicht

Der klassische Fall der Parallelisierung. Wird hier nicht näher betrachtet.

Praxistip: Wenn es nur um die Systemleistung geht und wenn man die Wahl hat (= wenn hinreichend leistungsfähige – und kostengünstige – Einzelprozessoren am Markt sind), ist der Einzelprozessor typischerweise die bessere Wahl:

- wir müssen uns nicht um die Parallelisierung kümmern,
- die höhere Leistung kann allen Abläufen zugute kommen,
- der Overhead des Multitasking ist oftmals geringer als der der Parallelisierung(vernünftige Systemorganisation vorausgesetzt),
- die Programmentwicklung ist wesentlich einfacher.

Kostenoptimierung

Ist fallweise zu prüfen. Grundlage: Kosten über alles (TCO), also nicht nur Hardware, sondern auch Programmentwicklung, Programmpflege, Service und Weiterentwicklung (Upgrade).

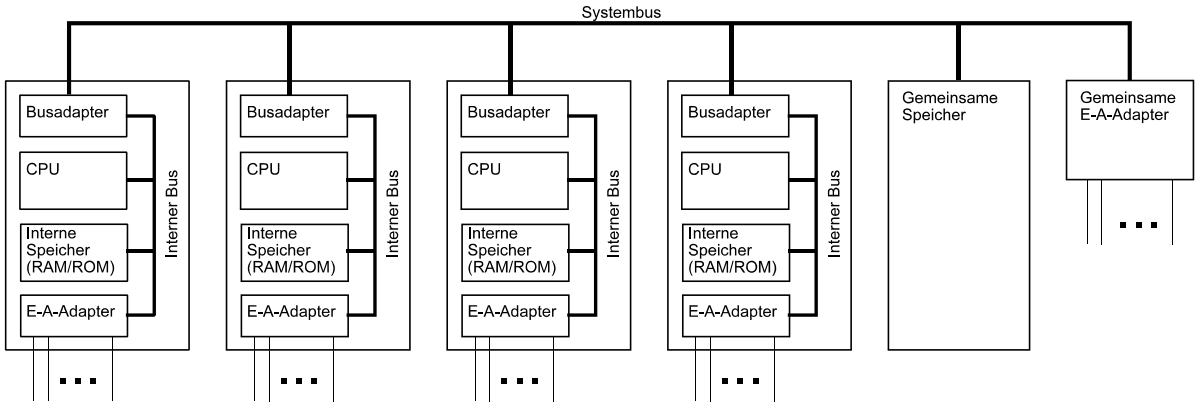
Mehrprozessorkonfiguration oder Multitasking?

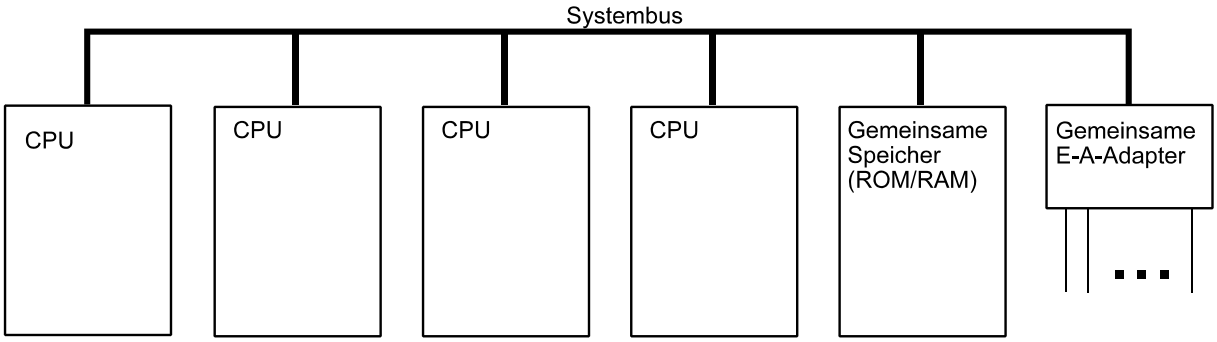
Ist fallweise zu prüfen. Was dauert länger, was kostet mehr Ressourcen, was ist funktionell komplizierter: Taskumschaltung oder Kommunikation zwischen den Prozessoren?

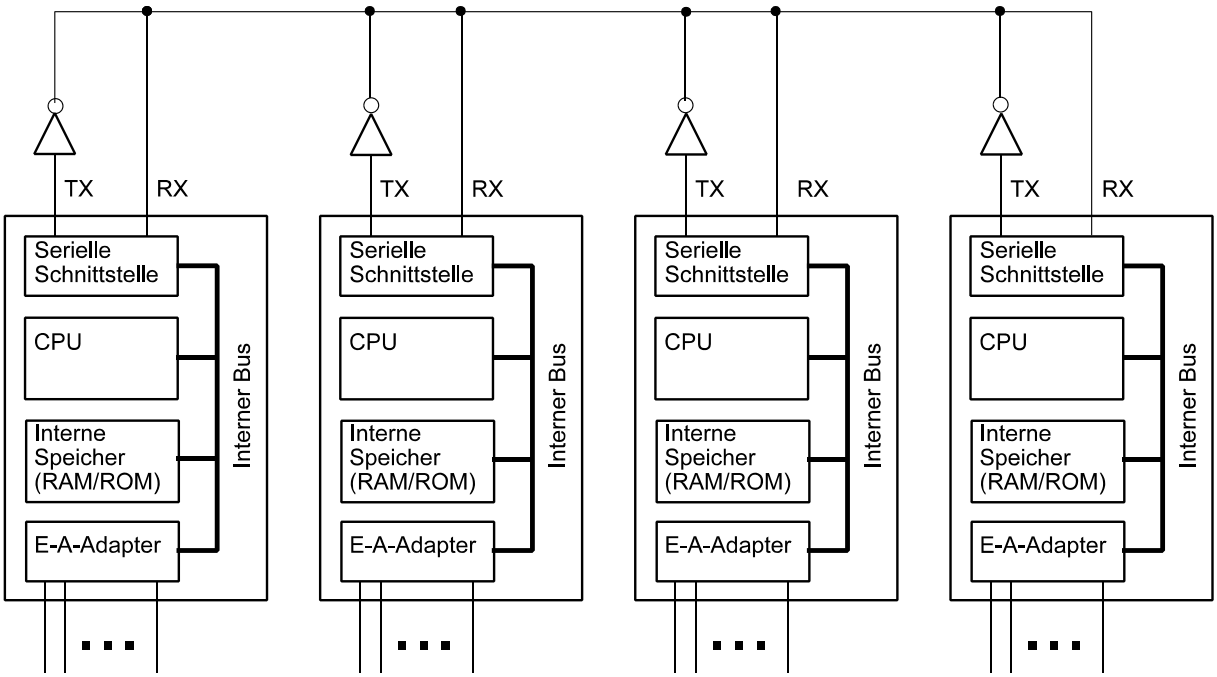
Auslagerung

Moderne Industriestandard-Plattformen (vor allem: PCs unter Windows, Linux usw.) sind nicht für Anwendungen in Embedded Systems vorgesehen – mag die einschlägige Werbung doch behaupten, was sie will... Auslagerung kann die Probleme entschärfen (in dem Sinne, daß keine Eingriffe in die Industriestandard-Plattform nötig sind), erfordert aber ihrerseits Aufwand (Partitionierung des Anwendungsproblems, besondere APIs, Protokolle usw.).

Praxistip: Wenn schon, dann harte Trennung bevorzugen (keine Mischlösungen). In Industriestandard-Plattform nicht eingreifen. Dort nur standardmäßige Schnittstellen (Interfaces, APIs) ausnutzen. Ist infolgedessen außen mehr Leistung erforderlich, dann entsprechende Hardware vorsehen – über alles gesehen (TCO) wird es sich rechnen... (Es kommt nicht nur auf die Lizenzkosten an, sondern auch darauf, ob man den kritischen Code ändern (oder womöglich überhaupt einsehen) darf...)


Systembus
Busadapter
CPU
Interne Speicher (RAM/ROM)
E-A-Adapter
Interner Bus
Gemeinsame Speicher
Gemeinsame E-A-Adapter


Systembus
CPU
Gemeinsame Speicher (ROM/RAM)
Gemeinsame E-A-Adapter


TX
RX
Serielle Schnittstelle
CPU
Interne Speicher (RAM/ROM)
E-A-Adapter
Interner Bus

Programmierte Peripherie
Mikrocontroller (und auch leistungsfähigere Prozessoren) sind oft eine kostengünstige Alternative zu speziellen E-A-Schaltkreisen und zu entsprechenden Eigenentwicklungen auf Grundlage von CPLDs und FPGAs.

Mikrocontroller oder CPLD?
Ein kleiner CPLD-Schaltkreis mit typischerweise 32...36 Makrozellen (= Flipflops) kostet ungefähr dasselbe wie ein PIC oder AVR oder 8051 usw. mit vergleichbarer Anzahl an Signalanschlüssen. Man hat also gelegentlich die Qual der Wahl. Ein naheliegendes Entscheidungskriterium ist die Reaktionszeit an den zu steuernden Schnittstellen:

- wenn es auf Mikrosekunden oder gar Nanosekunden ankommt: CPLD,
- wenn die Millisekunde kaum eine Rolle spielt: Mikrocontroller.

Hinweise:

1. Der Mikrocontroller bietet mehr fürs Geld, nämlich nahezu unbeschränkte Funktionsvielfalt (freie Programmierbarkeit), die CPLD hingegen nur das, was sich mit den wenigen Flipflops und einigen hundert Gattern realisieren läßt. Zudem haben viele Mikrocontroller eingebaute Schnittstellen (die man andernfalls selbst entwerfen müßte).
2. Die CPLD ist viel schneller. 100 MHz Takt heißt 10 ns Reaktionszeit vom Eingang zum Ausgang. Ein 100-MHz-Mikrocontroller hätte da bestenfalls einen einzigen Befehl ausgeführt – und der leistet nicht eben viel...
3. Extrem schnelle Mikrocontroller (von 25 MHz an aufwärts) werden angeboten. Sie sind deutlich teurer als die Massenfabrikate (mit typischerweise 5...20 MHz). Trotz der hohen Taktfrequenz sind die Reaktionszeiten in nichttrivialen Anwendungen enttäuschend, ja manchmal geradezu lausig.

Praxistip: Kleine Controller – auch ausgesprochen schnelle – sind nichts für komplizierte Programmierphilosophien. Besser: Beschränkung auf elementare Form der Programmorganisation (Abfrageschleifen, schnelle (lightweigth) Interrupts). 1 Controller = ein "weicher" (programmierter) Peripherieschaltkreis. Wenn nötig, einen mehr nehmen (keep it simple & stupid...).

Redundanz und Ausfallsicherung
Ein klassischer Anwendungsfall. Sind mehrere Prozessoren vorgesehen, können arbeitsfähige Prozessoren die Funktionen der ausgefallenen übernehmen. Hochinteressant, aber viel schwieriger, als es auf den ersten Blick aussieht. Wird hier nicht näher betrachtet.

Typische Varianten der Prozessorkopplung:

- Kopplung nur über Schnittstellen (Message Passing),
- Kopplung über gemeinsame Speicherbereiche (Shared Memory),
- Kopplung über gemeinsamen einheitlichen Speicheradreßraum (Unified Memory (UMA)).

1. Möglichkeiten und Grenzen der Parallelisierung

Ist das Leistungsvermögen eines Einzelprozessors für den gegebenen Anwendungsfall zu gering, so liegt es nahe, mehr als einen Prozessor zu verwenden. Schließlich ist es in vielen Gebieten der Technik üblich, mehrere Einrichtungen im Verbund einzusetzen, wenn eine allein nicht ausreicht, um eine bestimmte Leistung zu erbringen. Verbrennungsmotoren haben 4, 6, 8 oder mehr Zylinder; größere Flugzeuge haben 2, 3, 4 Triebwerke, manche sogar 6 oder 8; wenn die Fertigung eines Werkstückes n

Minuten dauert, aber jede Minute eines benötigt wird, sieht man n Fertigungseinrichtungen vor, auf denen gleichzeitig n Werkstücke bearbeitet werden usw. Kann man dieses Prinzip nicht "einfach" auf den Computer übertragen, um dessen Verarbeitungsleistung zu steigern? (Hier geht es nur um Leistungssteigerung, nicht um Mehrfachanordnungen im Sinne einer Ausfall-Reserve)

Die Antwort: Grundsätzlich ist so etwas wohl möglich, wenngleich nicht gerade "einfach". Schließlich muß es eine tiefere Ursache haben, daß Einzelprozessorsysteme immer noch in Millionenstückzahlen gefertigt werden und daß auch weiterhin viel Mühe - und Geld - aufgewendet wird, um Einzelprozessor-Architekturen zu vervollkommnen. Um sich das Problem deutlich vor Augen zu führen: der 8086-Schaltkreis umfaßt ungefähr 30 000 Transistoren, ein P6-Prozessor (Pentium II usw.) etwa 7,5 Millionen. Weshalb hat man (seinerzeit) nicht "einfach" einen Schaltkreis mit 7,5 Millionen : 30 000 = 250 8086's gebaut?

1.1 Grundlagen

Wenn wir die einfachste Leistungsangabe eines Prozessors betrachten, die Anzahl der Befehle, die er in der Sekunde ausführen kann (die MIPS also), so scheint der Fall klar: der 8086 braucht je Befehl im Durchschnitt, grob gerechnet, etwa 12 Taktzyklen, der P6 knapp 0,5 (Parallelausführung mehrerer Befehle); er kann also 24 mal so viele Befehle abarbeiten wie ein (in gleicher Technologie aufgebauter) einzelner 8086. Ein Aggregat aus 250 8086's wäre somit rund 10 mal schneller als der einzelne - gleich aufwendige - P6. Das soll kein Paradoxon sein oder gar ein Scherz. Seit mehr als zwei Jahrzehnten wird daran gearbeitet, Computersysteme aus sehr vielen, zumindest aus mehreren Prozessoren aufzubauen.

Man verfolgt damit zwei Ziele: (1) extreme Leistungssteigerung ("wenn ein Prozessor es allein nicht schafft, nehmen wir eben hinreichend viele"), (2) Ersatz des kostenaufwendigen Hochleistungsprozessors durch mehrere, notfalls viele preiswerte Mikroprozessoren (also ganz im Sinne des Vergleichs, den wir eben angestellt haben). Das Thema ist in der Fachpresse, auf Tagungen usw. recht beliebt. Viele der Leistungsangaben, die bei solchen Anlässen mitgeteilt werden, sind auf Grundlage der eben vorgeführten Milchmädchenrechnung ermittelt worden:

Gesamtleistung = Anzahl der Prozessoren ▯ Maximalleistung des Einzelprozessors.

Wenn hingegen der Anwender fragt, in welcher Zeit sein Problem gelöst ist, sieht die Sache oftmals ganz anders aus. Um es gleich am Extremfall zu verdeutlichen: Systeme mit -zig tausenden Prozessoren werden gelegentlich vorgeschlagen, solche mit einigen hundert bis wenigen tausend sind auch praktisch ausgeführt worden. Bezogen auf die theoretische Maximalleistung (Peak Performance) ist aber beim Anwender oft nur ein Anteil zwischen 10 und 15% wirksam geworden (das heißt beispielsweise: theoretisch Beschleunigung um das Hundertfache, praktisch Beschleunigung um das 10...15-fache).

Der Einzelprozessor hat Leistungsgrenzen (der typische Extremwert: 1 Befehl je Taktzyklus = 1 MIPS/MHz). Um sie zu überwinden, kann man die Tatsache ausnutzen, daß es im Ablauf üblicher Programme einen gewissen Grad an innewohnendem Parallelismus gibt, den man auf verschiedenartige Weise nutzen kann, bis hin zur gleichzeitigen Ausführung mehrerer Operationen (Superskalarprinzip). Im folgenden geht es hingegen um die *explizite* Parallelarbeit, um Möglichkeiten, mehrere oder gar viele für eine bestimmte Anwendung notwendige Informationswandlungen gleichzeitig zu erledigen. Wenn alle Möglichkeiten zur Leistungssteigerung des Einzelprozessors ausgeschöpft sind, ist dies in der Tat der einzige Weg, zu noch mehr Verarbeitungsleistung zu kommen. Hierzu müssen wir folgende Gesichtspunkte betrachten:

- die Natur der Anwendungsprobleme, das heißt der Grad des auf dieser Abstraktionsebene innewohnenden Parallelismus,

- die Art der einzelnen Verarbeitungseinrichtungen nach funktioneller Komplexität, Ausstattung und Leistungsvermögen, oder – aus programmseitiger Sicht gesehen –, die in der Architektur nutzbare Ebene des Parallelismus (im Englischen ist hierfür der bildhafte Ausdruck "Körnigkeit" – Granularity – üblich),
- die Gestaltung der Verbindungen zwischen den einzelnen Einrichtungen,
- die Prinzipien der Steuerung.

1.2 Die Parallelisierbarkeit von Anwendungsproblemen

Die Erfahrung zeigt, daß man Anwendungen in Hinblick auf die Parallelisierbarkeit grob in vier Klassen aufteilen kann:

1. Praktisch vollkommen parallelisierbar
Die gesamte Anwendung zerfällt in Teile, die voneinander weitgehend unabhängig sind. Wir müssen uns klarmachen, daß viele Aufgaben in einem Unternehmen, die heutzutage dem Computer übertragen werden, von Natur aus praktisch unabhängig voneinander sind. Die Gehaltsberechnung ist das eine, die Festigkeitsberechnung von Konstruktionsteilen das andere (weitere Aufgaben, wie das Gestalten von Werbeprospekten, das Programmieren von Werkzeugmaschinen, die Verwaltung der Lagerbestände usw. können das Beispiel ergänzen). Als es noch keine Computer gab, wurden derartige Aufgaben, ohne daß man besonders darüber nachgedacht hätte, von verschiedenen Bearbeitern in verschiedenen Abteilungen erledigt. Auch war es selbstverständlich, je nach Bedarf eine entsprechende Anzahl an Bearbeitern vorzusehen (für jeweils soundsoviele Arbeiter einen Lohnrechner, für einen bestimmten Umfang an Konstruktionsteilen einen Statiker usw.). Erst mit dem Aufkommen der Rechenzentren sind solche Aufgaben mehr und mehr zentralisiert worden; man hatte ein Programm für die Lohnrechnung, ein Programm für die Festigkeitsberechnung usw. Eine einzelne Aufgabe, z. B. das Drucken der Lohnzettel für mehrere tausend Arbeiter, beschäftigte den Computer stundenlang. Um zu ermöglichen, daß verschiedene Nutzer praktisch gleichzeitig Zugang zum Computer haben, wurden verschiedene Formen des Multitasking eingeführt. So kann man auf dem einen zentralen Computer Tasks für die Lohnrechnung, die Lagerverwaltung usw. aus der Sicht der einzelnen Nutzer gleichzeitig abarbeiten. Offensichtlich zerfallen solche Aufgaben von Natur aus in parallel ausführbare Teile. Dann liegt es nahe, für jede Aufgabe anstelle einer Task im zentralen Computer jeweils einen eigenen Computer vorzusehen. Das ist das Prinzip des *Masseneinsatzes von PCs* in Unternehmen.

Der Datenverbund
Ein ganz wesentlicher Vorteil des zentralen Computers: er hat Zugriff auf *alle* Daten aller Anwendungen. Man muß das Ganze "nur" richtig programmieren, dann wird (1) jede Stelle stets mit den jeweils zutreffenden, aktuellen Daten arbeiten, ohne daß dazu irgend etwas "von Hand" getan werden muß, und man kann (2) betriebliche Prozesse über Abteilungen, Bereiche usw. übergreifend gleichsam ganzheitlich optimieren.

Beispiele für (1): die Daten der Auftragssteuerung und der Produktionsdatenerfassung fließen unmittelbar in die Lohnrechnung ein; die Zugänge vom Wareneingang und die Abgänge an Fertigung bzw. Verkauf werden sofort verrechnet, um die Angaben zum Lagerbestand auf dem laufenden zu halten. Beispiele für (2): Konstruktionsdaten können unmittelbar für die Materialbestellung verwendet werden (wenn verschiedene Teile mit demselben Werkstoff auszuführen sind, kann man die Auswahl der Halbfabrikate optimieren); aus Aufträgen, Maschinenbelegungen, Fertigungszeiten, Lagerbeständen usw. kann man den Fertigungsdurchlauf planen.

Spricht dies gegen die Parallelisierung? Von Zuverlässigkeits- und Verläßlichkeitsfragen einmal abgesehen (die wir hier nicht weiter behandeln wollen), an sich nicht. Die Programme, die für die einzelnen Aufgaben vorgesehen sind, müssen nämlich vergleichsweise selten miteinander

kommunizieren. Folglich können sie auf verschiedenen, auch räumlich getrennt aufgestellten, Computern laufen, sofern zwischen diesen hinreichend leistungsfähige Verbindungen vorgesehen sind (diese sind aber technisch ohne weiteres beherrschbar): das ist das Prinzip des *Computer-Netzwerks*.

Wenn die Leistungsfähigkeit der Netzwerk-Verbindungen nicht ausreicht bzw. wenn die zentrale Datenhaltung und Verarbeitung aus Verläßlichkeitsgründen geboten erscheint, kann man mehrere Computer bzw. Prozessoren in einem einzigen Aggregat zusammenfassen und hochleistungsfähige Verbindungen zwischen ihnen vorsehen: das ist das Prinzip der "dicht gekoppelten" (closely coupled) *Multiprozessorsysteme*.

2. Hochgradig parallelisierbar
Viele "wissenschaftliche" Anwendungen, bei denen der Computer tatsächlich als "Rechenmaschine" eingesetzt wird, gehören hierher. Solche Programme arbeiten oft mit Matrizen, Vektoren oder anderen regulären Datenstrukturen, wobei die einzelnen Elemente solcher Strukturen jeweils gleichartigen Informationswandlungen unterzogen werden (Vektorverarbeitung). Auch in solchen Fällen ist die Parallelisierbarkeit unmittelbar einsichtig: wenn zwei Vektoren von je 1000 Elementen elementweise miteinander zu verknüpfen sind, kann man gleichzeitig in 1000 Verarbeitungseinrichtungen dieselben Operationen auf jeweils zwei Elemente anwenden.

Auch manche Aufgaben, die mit dem numerischen Rechnen nichts zu tun haben, sind derart parallelisierbar. Beispiele: (1) Rechtschreibprüfung, Seitenumbruch, Silbentrennung usw. bei umfangreichen Texten (um einen Text von 1000 Seiten zu bearbeiten, könnte man 1000 Prozessoren einsetzen, wobei jeder für eine Seite zuständig ist), (2) das Durchmustern von umfangreichen Datenbeständen (ein beliebiger Textbestand, der auf bestimmte Begriffe hin zu durchsuchen ist (Volltextrecherche), kann, vergleichbar zum Beispiel (1), auf viele Prozessoren aufgeteilt werden).

Es ist naheliegend, für solche Aufgaben Anordnungen vorzusehen, die aus einer sehr großen Anzahl von Verarbeitungseinrichtungen bestehen (*massiv parallele Systeme*).

Datenabhängigkeiten (Data Dependencies)
Massiv parallele Systeme erbringen sehr gute Leistungen, wenn alle Einrichtungen mit Daten versorgt sind und unabhängig voneinander arbeiten können. Schwierig wird es dann, wenn eine Einrichtung Daten von einer anderen braucht. Hierzu sind die Einrichtungen untereinander zu verbinden, und die Parallelarbeit ist durch besondere Steuerungsmaßnahmen zu koordinieren. Abhängig von den jeweiligen (anwendungsspezifischen) Besonderheiten der Datenkommunikation sind bestimmte Verbindungsprinzipien zu bevorzugen (ein Grund dafür, weshalb sich, trotz langjähriger Forschung, noch kein "Industriestandard" herausgebildet hat).

Auslastung
Solche Systeme arbeiten dann wirtschaftlich, wenn alle Einrichtungen stets mit nützlichen Aufgaben ausgelastet sind. Was passiert aber, wenn man beispielsweise 1000 Prozessoren hat, aber nur Vektoren mit 300 Elementen zu verarbeiten sind? - Es würden 700 Prozessoren ungenutzt bleiben, trotzdem wäre die Arbeit auch nicht eher fertig. (Der herkömmliche Vektorrechner wäre demgegenüber stets ausgelastet. Sind die Vektoren kürzer, ist die Arbeit eher beendet, und die Maschine ist für neue Aufgaben verfügbar.)

3. In mittlerem Grade parallelisierbar
In solchen Aufgaben gibt es stärkere Datenabhängigkeiten. Einige Informationswandlungen können aber parallel ausgeführt werden, beispielsweise das Berechnen arithmetischer Ausdrücke, Adreßrechnungen usw. Wegen der Datenabhängigkeiten braucht man hochleistungsfähige und flexible Verbindungen zwischen den einzelnen Einrichtungen. Prozessoren, die zwar dicht gekoppelt sind, aber unabhängig voneinander Befehle abarbeiten, sind hierfür nicht immer zweckmäßig: die Zeitverluste infolge der

erforderlichen Kommunikation und Synchronisation (wenn z. B. der Prozessor 1 auf ein Ergebnis von Prozessor 2 warten muß) wären einfach zu groß. Deshalb wird diese Art des Parallelismus vorwiegend in Superskalarmaschinen genutzt.

4. Kaum parallelisierbar
Manche Aufgabenstellungen führen auf Häufungen von Programmabläufen, die ihrer Natur nach praktisch nicht parallelisierbar sind (man nennt sie "inhärent sequentiell"). Zu solchen Abläufen gehören das Aufrufen von Unterprogrammen, die Listenverarbeitung (wobei, im Gegensatz zur Vektorverarbeitung, oft die gerade in Bearbeitung befindlichen Listenelemente darüber bestimmen, welche folgenden Listenelemente auf welche Weise zu verarbeiten sind) sowie bedingte Entscheidungen bzw. Verzweigungen. Hat eine Anwendung ihrer Natur nach einen hohen Anteil an solchen Aktivitäten, lohnt sich eine Parallelisierung der verbleibenden Abläufe praktisch kaum.

1.3 Die "Körnigkeit" des nutzbaren Parallelismus

Dieser bildhafte Begriff (engl. Granularity) kennzeichnet, auf welcher Betrachtungsebene – bezogen auf das gesamte Anwendungsproblem – die Parallelisierung angesetzt wird. Damit hängen unmittelbar die Anzahl, Art und Komplexität der parallel angeordneten Verarbeitungseinrichtungen zusammen. Tabelle 1.1 gibt einen entsprechenden Überblick.

Körnigkeit		**Was wird parallelisiert**	**Anzahl der Einrichtungen**	**Art der Hardware**	**Praktische Bedeutung**
Coarse Grain	sehr grob	Ganze Programme und Programmkomplexe: Task-Parallelismus	2 .. 256	Vernetzte Computer, Multiprozessor-Syteme	sehr groß; weiterhin zunehmend
	grob	Unterprogramme, "aufgerollte" äußere Schleifen	32 .. 1024	Massiv parallele Systeme, zumeist mit Hochleistungs-prozessoren	Im Höchstleistungsbereich mehr und mehr zunehmend
Medium Grain	mittel	Vektoroperationen, "aufgerollte" innere Schleifen	32 .. 4096		
Fine Grain	fein	einzelne Befehle: Superskalarprinzip	2 .. 32	Superskalar-prozessoren	vor allem auch in Prozessoren für PCs und Workstations
	sehr fein	elementare Informationswandlungen innerhalb von Befehlen (Schritte der Multiplikation, der Bildung des Skalarprodukts usw.)	mehrere Tausend	zelluläre Systeme, "systolische Arrays"	sehr gering, möglicherweise für Sonderzwecke oder neuartige Funktionen

Tabelle 1.1 Abstufungen der Körnigkeit (Granularity) des nutzbaren Parallelismus

1.4 Verbindungsprinzipien

Man unterscheidet zwischen lose und dicht gekoppelten Parallelverarbeitungssystemen (Loosely bzw. Tightly Coupled Systems). In ersteren arbeiten die einzelnen Einrichtungen stets weitgehend autonom. Hingegen können dicht gekoppelte Systeme mit unabhängigen Computern, mit autonom arbeitenden Prozessoren oder auch mit zentral gesteuerten Verarbeitungseinrichtungen aufgebaut sein. (Die bekannten Computernetzwerke kann man durchaus als lose gekoppelte Systeme auffassen.)

1.5 Steuerungsprinzipien

Parallelverarbeitungssysteme ann man aus unabhängig arbeitenden, also autonom gesteuerten, Einrichtungen aufbauen. Man kann aber auch eine zentrale Steuerung aller Einrichtungen vorsehen. Jede Auslegung hat für bestimmte Anwendungsfälle jeweils ihre Berechtigung (auch in dieser Hinsicht gibt es noch keinen "Industriestandard"). Eine Systematik der verschiedenen Steuerprinzipien wurde bereits Anfang der 60er Jahre aufgestellt. Da diese Begriffsbildungen auch heutzutage in der Fachliteratur recht häufig gebraucht werden, geben wir eine entsprechende Übersicht (Tabelle 1.2).

Befehls-ströme	**Daten-ströme**	**Bezeichnung**		**Beispiele**
einer	einer	SISD	Single Instruction Single Data	jeder "klassische" Einzelprozessor
einer	mehrere	SIMD	Single Instruction Multiple Data	Parallelverarbeitungssysteme mit mehreren gleichartigen Verarbeitungseinrichtungen, die zentral - jeweils über einen (für alle geltenden) Befehl - gesteuert werden
mehrere	einer	MISD	Multiple Instruction Single Data	Mehrere Befehle wirken parallel auf dieselben Daten ein; in "reiner" Form praktisch nicht ausgeführt. Superskalar- und Datenflußmaschinen könnte man hier einordnen
mehrere	mehrere	MIMD	Multiple Instruction Multiple Data	Parallelverarbeitungssysteme mit mehreren unabhängig arbeitenden Prozessoren

Tabelle 1.2 Klassifizierung von Computersystemen nach Flynn

Unter einem Befehls- bzw. Datenstrom kann man sich einen einzelnen Zugriffsweg (Adresse + Daten) zum Speicher vorstellen. In einer SISD-Maschine bearbeitet jeder Befehl nur "seine" Daten, die üblicherweise über nur einen Zugriffsweg geholt werden. In einer SIMD-Maschine werden viele gleichartige Verarbeitungswerke mit den zugehörigen Datenwegen von einem einzigen Befehl gesteuert, wobei alle Werke das gleiche tun (beispielsweise "ihre" Daten miteinander multiplizieren). In einer MIMD-Maschine arbeiten mehrere (viele) unabhängige Prozessoren, jeder an seinem eigenen Programm und mit eigenen Daten.

1.6 Leistungsgewinn durch Parallelverarbeitung

Um wieviel schneller läuft ein Programm bei Nutzung der Parallelverarbeitung im Vergleich zur seriellen Ausführung auf einem üblichen Einzelprozessor? Wir bezeichnen die Ausführungszeit bei serieller Verarbeitung mit t_{ser}, bei paralleler mit t_{par}. Damit ergibt sich der Leistungsgewinn (Speedup) s folgendermaßen:

$$s = \frac{t_{ser}}{t_{par}}$$

Ist unser Programm vollständig parallelisierbar, können wir dessen Ausführung also derart auf n Prozessoren verteilen, daß (1) alle sinnvoll beschäftigt sind und zum gewünschten Endergebnis beitragen und daß es (2) keine Zeitanteile für Transporte, für das Warten auf Zwischenwerte (Datenabhängigkeiten) usw. gibt, so brauchen wir statt t_{ser} nur noch eine Verarbeitungszeit von

$$\frac{t_{ser}}{n}$$

Damit ergibt sich ein Leistungsgewinn s = n (linearer Speedup).

Jetzt nehmen wir an, unser Programm sei nicht vollständig parallelisierbar: dann wird es Abläufe geben, die zur Parallelverarbeitung geeignet sind, und solche, die unbedingt seriell abgearbeitet werden müssen. Die gesamte Laufzeit t_{ser} auf dem Einzelprozessor können wir uns demnach zusammengesetzt denken aus dem Zeitanteil, den die parallelisierbaren und aus dem, den die nicht parallelisierbaren Abläufe benötigen (diese Anteile heißen t_p und t_s):

$$t_{ser} = t_p + t_s$$

Nun stehen uns n Einrichtungen zur Verfügung, um die parallelisierbaren Abläufe tatsächlich parallel auszuführen. Dann sinkt der entsprechende Anteil auf

$$\frac{t_p}{n}$$

Die seriellen Abläufe können nicht weiter beschleunigt werden. Setzen wir nun diese Zeiten in unsere Formel für den Speedup ein:

$$s = \frac{t_p + t_s}{t_s + \frac{t_p}{n}}$$

Dieser Ausdruck ist seit vielen Jahren als *Amdahls Gesetz* bekannt. Oft setzt man - $t_{ser} = t_p + t_s = 1$ (normiert also die "serielle" Ausführungszeit auf Eins). Damit ergibt sich:

$$s = \frac{1}{t_s + \frac{t_p}{n}}$$

Sehen wir uns diese Funktion als Kurve an (Abb. 1.1).

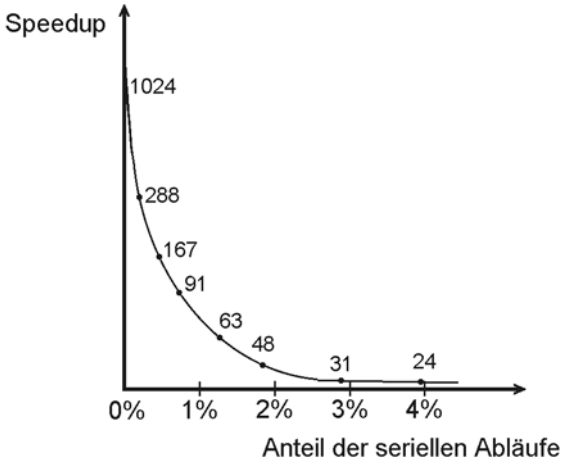


Abb. 1.1 Amdahls Gesetz: Leistungsgewinn (Speedup) bei Parallelverarbeitung (Beispiel mit n = 1024)

Wir erkennen, daß bereits recht mäßige Anteile an nicht parallelisierbaren Abläufen den Leistungsgewinn deutlich verringern. Auch wenn die Anwendung an sich in hohem Grade parallelisierbar ist, aber einen nicht vernachlässigbar kleinen seriellen Anteil hat, bietet Amdahls Gesetz recht pessimistische Aussichten. (Gemäß Abb. 1.1 würde eine Anordnung aus immerhin 1024 Prozessoren bei einem Anteil der seriellen Abläufe von nur 4% gerade mal eine 24fache Beschleunigung ermöglichen...)

In der Praxis sieht es aber nicht immer so ungünstig aus: tatsächlich läßt sich in manchen Fällen – wenngleich mit beträchtlichem Forschungs- und Programmieraufwand – ein sehr hoher Leistungsgewinn nachweisen (nahezu linearer Speedup). In den betreffenden Anwendungsgebieten kann man es sich leisten, das Problem zu "vergrößern", wenn man mehr Verarbeitungseinrichtungen hat (das "Vergrößern" betrifft den Datenumfang, die Komplexität der einzelnen Rechenschritte usw.), das heißt, die Aufgabe in einem gegebenen Zeitrahmen - einfach gesagt - "besser" zu erledigen (zu solchen Problemen gehören viele Anwendungen der numerischen Mathematik, bei denen das mathematische Modell um so bessere Ergebnisse liefern kann, je mehr Rechenzeit bzw. Verarbeitungsleistung nutzbar ist). Dann kann man die Verarbeitungszeit auf einem System aus n Prozessoren zur Grundlage nehmen. t_{pn} ist dann die Zeit, die insgesamt zur Bearbeitung des parallelisierten Anteils benötigt wird. Der herkömmliche Einzelprozessor würde dafür einen Zeitanteil von $n \cdot t_{pn}$ erfordern (weil er das, was die n Prozessoren gleichzeitig tun, nacheinander erledigen muß). Somit ergibt sich ein - auf die Problemgröße bezogener (skalierter) - Speedup S_s zu:

$$S_s = \frac{t_s + (n \cdot t_{pn})}{t_s + t_{pn}}$$

Das ist ein linearer Leistungsgewinn, wie er auch in der Praxis näherungsweise auftritt.

2. Mehrprozessorsysteme

Mehrprozessorsysteme bestehen aus vergleichsweise wenigen Prozessoren (es sind mindestens zwei; im Extremfall, z. B in einem größeren Netzwerk, einige hundert). Solche Systeme kann man aus technischer und aus architekturseitiger Sicht bewerten. Technische Gesichtspunkte betreffen den Aufbau (verteilt aufgestellte oder kompakt – in einem gemeinsamen Gehäuse – angeordnete Einrichtungen), die Auslegung und Struktur der Verbindungen (Einzelleitungen, Netzwerke, Lichtleiter-Kabel, Bussysteme, Punkt-zu-Punkt-Verbindungen) usw. Wir wollen uns im folgenden an architekturseitigen Merkmalen orientieren und dabei zwischen lose und dicht gekoppelten Systemen unterscheiden.

2.1 Lose gekoppelte Mehrprozessorsysteme

Andere Fachbegriffe: (1) Massively Parallel Processing (MPP), (2) Shared Nothing Approach.

In lose gekoppelten Systemen hat jeder Prozessor ausschließlichen Zugriff auf seinen eigenen Adreßraum. Die Komponenten (Moduln) eines solchen Systems sind vollständige, für sich arbeitsfähige Computer. Die Kommunikation zwischen ihnen wird durch softwareseitige Protokolle abgewickelt (Message Passing). Die Übertragungswege selbst können serielle Verbindungen sein, aber auch parallele Bussysteme, die von der Software entsprechend genutzt werden. Auch Computer-Netzwerke kann man als lose gekoppelte Mehrprozessorsysteme ansehen. Solche Konfigurationen werden gelegentlich als verteilte Systeme (Distributed Systems) bezeichnet. Sie sind im besonderen für Anwendungen geeignet, die ihrer Natur nach praktisch vollkommen parallelisierbar sind (Task-Parallelismus).

Der sofort erkennbare Vorteil: man kann ein Mehrprozessorsystem aus Komponenten der Massenfertigung ohne weiteres zusammenstecken. Der oben genannte Fachbegriff (2) bezeichnet dies bildhaft: die einzelnen Computer teilen sich nichts untereinander - jeder hat seinen eigenen Adreßraum, seine eigene Software usw.; die gesamte Kommunikation beschränkt sich auf das Versenden und Empfangen von Nachrichten (Messages).

Das Prinzip ist deshalb im akademischen Bereich sehr beliebt - man denkt hier auch daran, extrem viele Prozessoren im Verbund einzusetzen (256...4k und mehr).

Im kommerziellen Bereich finden wir vor allem Verbundsysteme aus mehreren Serven (Server Clusters, Server Farms; Abb. 2.1).

Topologien
Der Übergang zwischen einem (lokalen) Netzwerk und einem lose gekoppelten Mehrprozessorsystem ist fließend – viele solcher Systeme sind tatsächlich über LANs zusammengeschaltet. Deshalb finden wir hier auch die bekannten Netzwerktopologien. Hinzu kommen Verbindungsstrukturen, die eigens für solche Systeme entwickelt wurden:

- mehrere Prozessoren an jeweils einem Bus (Cluster-Bus); Verbindung der "Cluster" über weitere Bussysteme oder Hochleistungs-Netzwerke (Abb. 2.1),
- Flächen- bzw. Raumgitter (Abb. 2.2),
- Hyperwürfel. Die Anzahl n der Prozessoren in einem Hyperwürfel ist eine Zweierpotenz. Jeder Prozessor hat ld n Nachbarn, mit denen er direkt verbunden ist (Abb. 2.3). In einem Hyperwürfel mit 4096 Prozessoren hat jeder Prozessor somit nur Verbindungen zu ld 4096 = ld 2^{12} = 12 Nachbarn. Der Wert ld n repräsentiert die *Dimension* des Hyperwürfels.
- Binärbaum (Abb. 2.4). Der offensichtliche Vorteil des Binärbaums besteht in technischer Hinsicht darin, daß jeder Prozessor nur mit drei Verbindungspfaden auskommt, gleichgültig wie viele Prozessoren vorhanden sind.

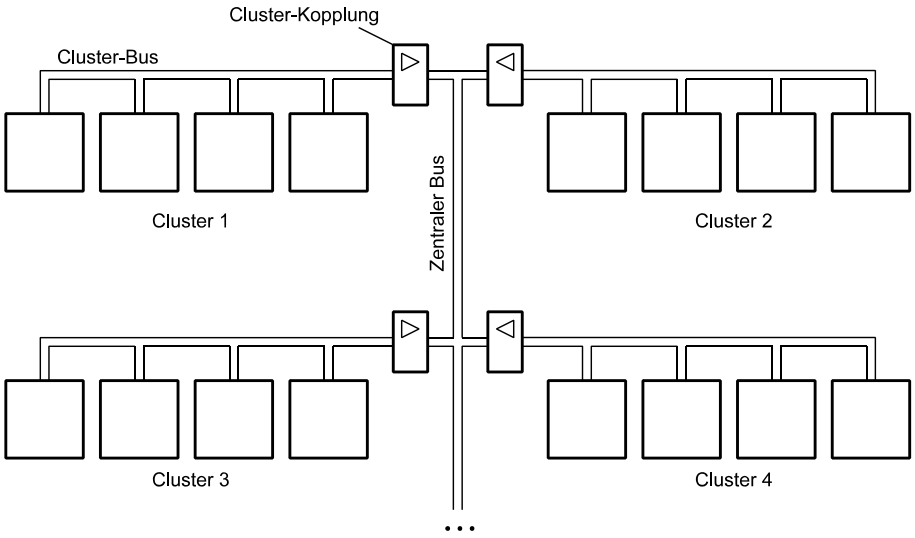


Abb. 2.1 Mehrprozessorsystem aus Prozessor-Clustern

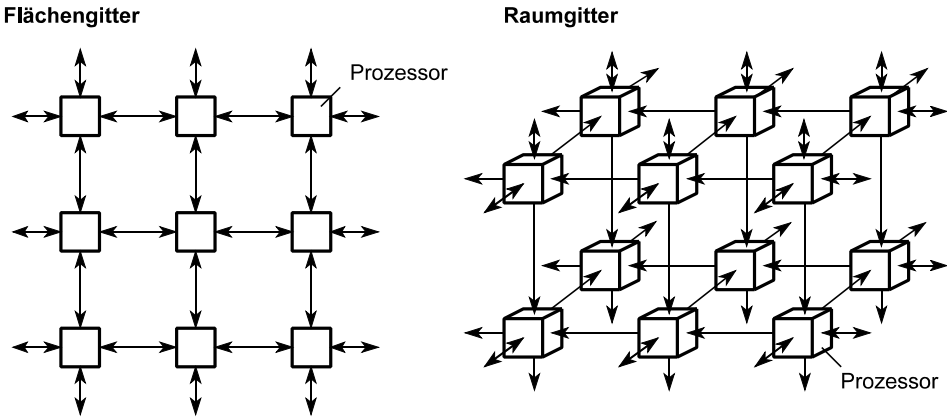


Abb. 2.2 Mehrprozessorsysteme in Gitterstruktur. Links: Flächen-, rechts: Raumgitter

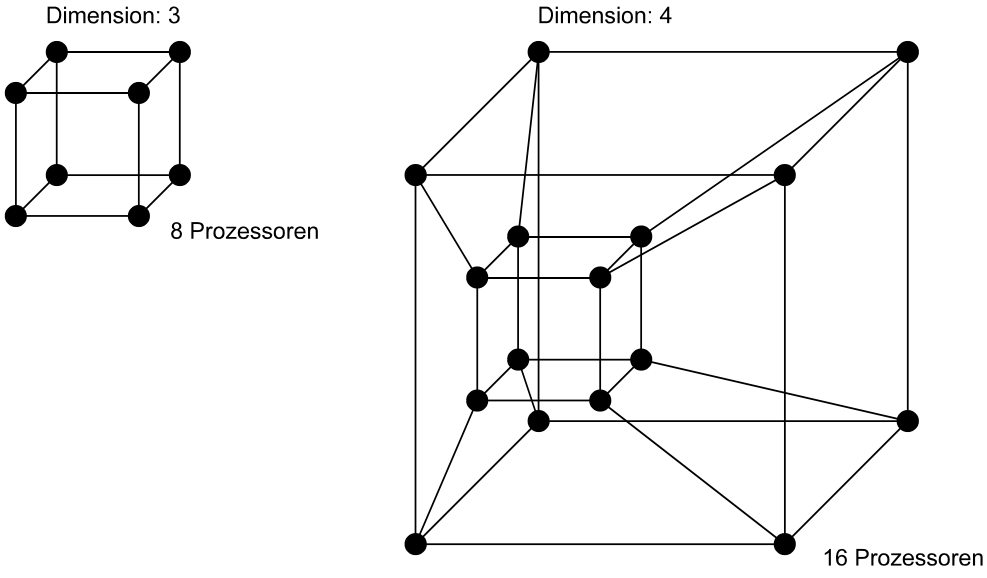


Abb. 2.3 Mehrprozessorsysteme als Hyperwürfel

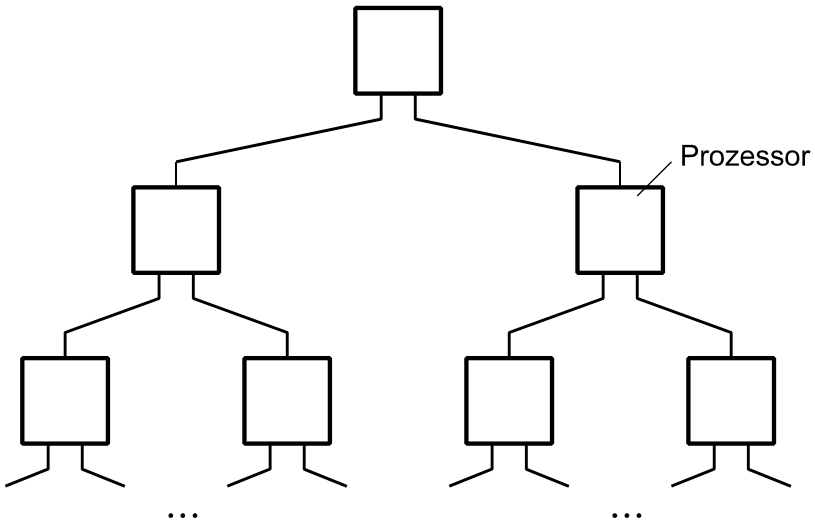


Abb. 2.4 Mehrprozessorsystem als Binärbaum

2.2 Dicht gekoppelte Mehrprozessorsysteme

Lose gekoppelte Systeme leisten dann Beachtliches, wenn sich die Anwendungsaufgaben soweit parallelisieren lassen, daß die einzelnen Prozessoren weitgehend selbständig arbeiten können, wobei nur vergleichsweise wenige Nachrichten untereinander auszutauschen sind. Die offensichtlichen Probleme:

- die Anwendungsaufgaben müssen an sich parallelisierbar sein,
- die Parallelisierung ist oft mühevoll,
- die einzelnen parallelisierten Programmabläufe müssen weitgehend mit “lokalen” (im jeweiligen Computer gespeicherten) Daten auskommen (wenn sich mehrere Computer gemeinsame Datenbestände teilen müssen (Data Sharing), wird es uneffektiv).

Vor allem kommerzielle Anwendungen, in denen eine Vielzahl von Nutzern auf eine große gemeinsame Datenbasis zugreift, entsprechen nicht diesem Schema. Deshalb werden hier dicht gekoppelte Systeme bevorzugt.

Andere Fachbegriffe: (1) Symmetrical Multiprocessing (SMP), (2) Shared Memory Multiprocessing, (3) Shared All Approach.

In dicht gekoppelten Systemen haben alle Prozessoren Zugang zu einem gemeinsamen Adreßraum, der dem lokalen Adreßraum des einzelnen Prozessors voll oder teilweise entspricht. Der oben genannte Fachbegriff (3) bezeichnet dies bildhaft: die einzelnen Computer teilen sich praktisch alles untereinander: den Speicheradreßraum, die E-A-Ausstattung usw. Anstatt untereinander Nachrichten zu versenden (zeitaufwendig, hoher Overhead auf Grund der erforderlichen Übertragungsprotokolle), greifen die einzelnen Prozessoren einfach auf den gemeinsamen Speicheradreßraum zu (hierzu genügen die üblichen Maschinenbefehle, es gibt also gar keine Kommunikationsprotokolle und damit keinen Overhead). Mehrere Prozessoren können somit gleichzeitig an einem gemeinsamen Datenbestand arbeiten.

Was bedeutet hier “symmetrisch” (das “S” in SMP)?
Der Begriff bezeichnet eine Systemorganisation, in der alle Prozessoren vollkommen gleichberechtigt sind:

- alle Prozessoren greifen auf denselben Speicheradreßraum zu und finden unter gleichen Adressen dieselben Daten vor,
- alle Prozessoren nutzen dasselbe E-A-Subsystem,
- jeder Prozessor kann Interrupts von jeder beliebigen Quelle empfangen.

Die wesentlichen Vorteile der “symmetrischen” Organisation:

- alle Prozessoren können mit einer einzigen Kopie des Betriebssytems arbeiten,
- die Software ist uneingeschränkt lauffähig (ohne Neucompilierung o. dergl.), gleichgültig wieviele Prozessoren jeweils installiert sind.

Es versteht sich geradezu von selbst, daß dies mit Schwierigkeiten verbunden ist:

1. mehrere Prozessoren wollen gleichzeitig auf einen bestimmten Speicherbereich zugreifen. Offensichtlich klappt das nicht so ohne weiteres: wir brauchen irgendeine “Schiedsrichter”-Funktion (Speichervermittlung, Arbitrierung – ähnlich einem Multi-Master-Bussystem). Zudem entstehen Wartezeiten, weil die einzelnen Prozessoren nur nacheinander an die Reihe kommen können.
2. mehrere Prozessoren wollen gleichzeitig auf eine bestimmte Speicheradresse schreiben. Es ist klar, daß daraus Probleme entstehen können, auch wenn die Zugriffe untereinander vermittelt werden (sieht man nichts Besonderes vor, so behält gleichsam jener recht, der den letzten Zugriff zugesprochen bekommen hat).
3. mehrere Prozessoren wollen gleichzeitig ein bestimmtes Betriebsmittel nutzen, z. B. ein Modem oder einen Drucker. Auch hier müssen offensichtlich Vermittlungsfunktionen wirksam werden.

Grundsatzlösungen:

Zu Punkt 1 - Mehrfachzugriffe:
Es gibt zwei Ansätze (die in der Praxis typischerweise im Verbund angewendet werden):

- das Speichersubsystem wird so leistungsfähig gemacht, daß Wartezeiten als Folge der Speichervermittlung kaum noch eine Rolle spielen,
- den einzelnen Prozessoren werden Caches zugeordnet.

Zu den Punkten 2 und 3
Beide können auf ein einziges Grundsatzproblem zurückgeführt werden: auf den konkurrierenden Zugriff zu gemeinsam genutzten Ressourcen (Speicherpositionen, Geräten usw.). Dies muß auf sozusagen höherer Ebene gelöst werden (in der Praxis: im Rahmen der Systemsoftware). Die Software braucht hierzu aber bestimmte Vorkehrungen in der Hardware. Einzelheiten in Kapitel 3.

Zum Stand der Technik

Kopplung zwischen Prozessoren und Speichersubsystem
Herkömmlicherweise gibt es zwei Prinzipien:

- leistungsfähige Bussysteme mit Multi-Master-Fähigkeit (Abb. 2.5),
- Crossbar-Netzwerke, über die alle Prozessoren mit gemeinsamen Speicheranordnungen verbunden sind (Abb. 2.6).

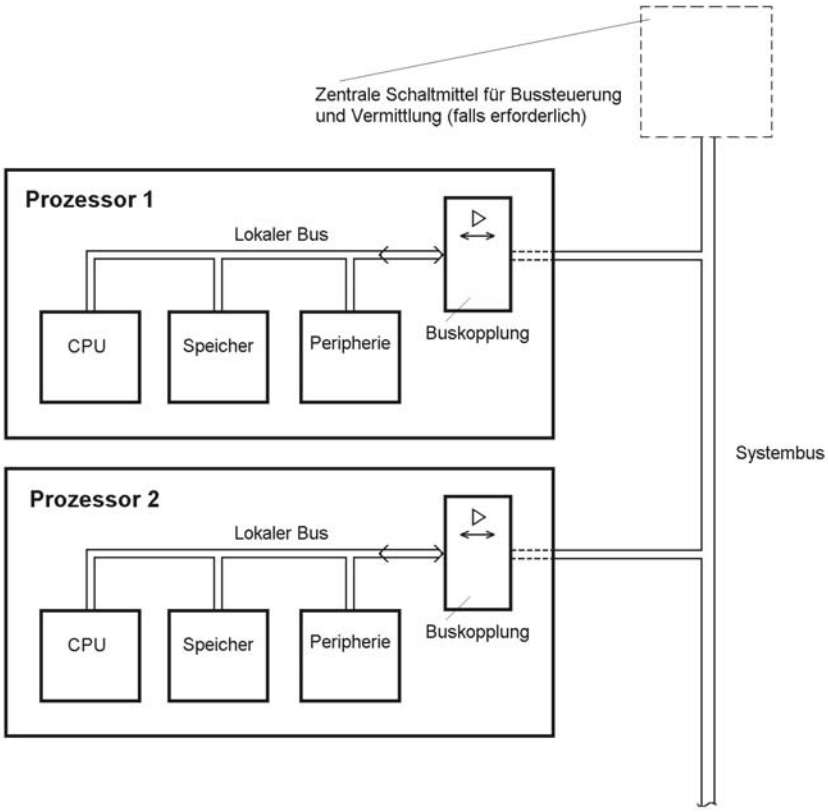


Abb. 2.5 Multiprozessorsystem mit Systembus

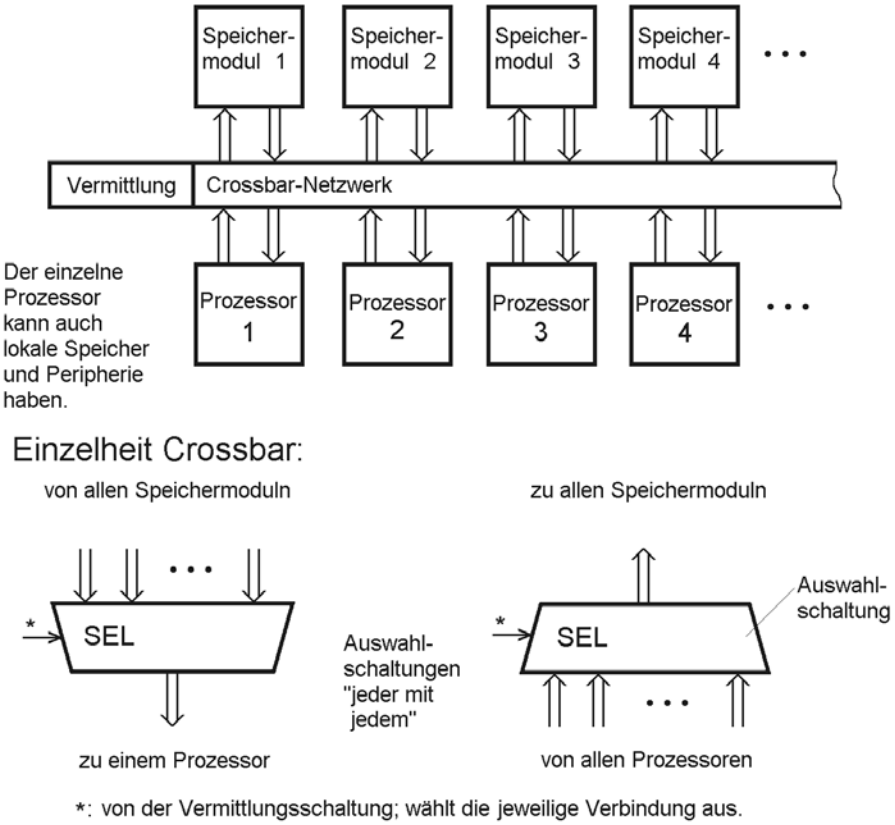


Abb. 2.6 Multiprozessorsystem mit Crossbar-Netzwerk

Derartige Multiprozessorsysteme enthalten typischerweise 2...64 Prozessoren. Der Systembus hat den Vorteil der problemlosen Erweiterbarkeit und des vergleichsweise geringen Aufwandes. Zu einer Zeit ist aber nur eine Master-Slave-Kommunikation möglich. Crossbar-Netzwerke sind wesentlich aufwendiger, es können aber mehrere unabhängige Kommunikationsvorgänge gleichzeitig ablaufen. (Erklärung: Ein Crossbar-Netzwerk enthält Schaltmittel, die es ermöglichen, zwischen zwei beliebigen angeschlossenen Einrichtungen wahlweise Punkt-zu- Punkt-Verbindungen durchzuschalten.)

Anordnung von Caches
Caches ermöglichen – entsprechend ihrer Trefferrate – weitgehend "lokale" Zugriffe des jeweiligen Prozessors, so daß die Signalwege der Multiprozessorkopplung (Bus, Crossbar) entlastet werden. Wirklich hochleistungsfähige Systeme kann man praktisch nicht ohne Caches bauen. Aus dem Einsatz der Caches ergibt sich aber das wohl schwierigste Problem der Auslegung von SMP-Systemen: die Gewährleistung der Cache-Kohärenz. Dieser Fachbegriff bezeichnet die Forderung, daß alle Einrichtungen im System beim Zugriff auf eine bestimmte Speicheradresse dort dieselben (und aktuellen) Daten vorfinden, gleichgültig, ob Caches zwischengeschaltet sind oder nicht. Die herkömmliche Lösung: Zugriffsprüfung bzw. -beobachtung (Snooping).

Hinweis: Es liegt nahe, das Problem der Cache-Kohärenz zu umgehen, indem man die betreffenden Zugriffsfälle ausschließt, also säuberlich zwischen lokalen und gemeinsam genutzten Speicherbereichen trennt, wobei letztere gar nicht in das "Caching" einbezogen werden. Diese einfache Lösung hat sich aber als ungeeignet erwiesen. Beispielsweise kann ein Compiler gar nicht immer von vornherein wissen, welche Daten als lokal und welche als gemeinsam genutzt anzusehen sind.

2.3 SMP-Systeme im PC-Bereich

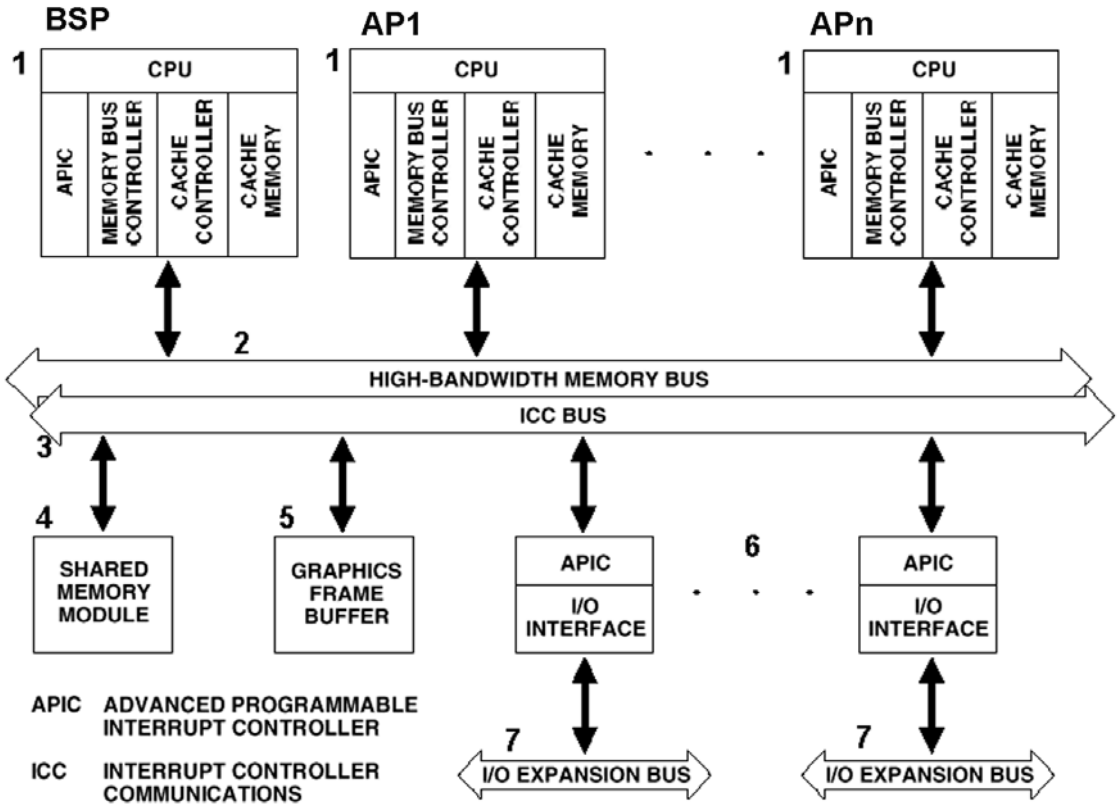
Die Multiprozessorspezifikation von Intel
Abb. 2.7 zeigt ein typisches, gemäß dieser Spezifikation aufgebautes SMP-System im Blockschaltbild.

Herkömmliche Grundlagen
Das erklärte Ziel von Intel: die herkömmlichen PC-Prinzipien soweit wie möglich beizubehalten (bis hin zur Adreßraumaufteilung, die auf den klassischen PC/AT zurückgeht). Die Prozessoren hat man – ohne Grundsätzliches zu ändern (Abwärtskompatibilität) – um Funktionen zur Unterstützung von SMP-Konfigurationen erweitert. Hierzu gehören u. a.:

- höher entwickelte Interruptcontroller (APICs) ,
- die bereichsweise wirkende Beeinflussung von Speicherzugriffen (MTRRs, PAT),
- die Verlängerung der physischen Adresse auf 36 Bits,
- Prozessor-Bussysteme, die es ermöglichen, 2 oder 4 Prozessoren zusammenzuschalten,
- Cache-Steuerungen, die die Cache-Kohärenzin solchen Konfigurationen gewährleisten können.

Kleine Konfigurationen
Es können zwei oder vier Prozessoren an einem Bussystem betrieben werden. Herkömmliche Beispiele: Zweiprozessorsysteme mit Pentium II oder Pentium III, Vierprozessorsysteme mit Pentium Pro oder Pentium Xeon. Solche Konfigurationen (vgl. Abb. 2.7) sind typischerweise auf einem einzigen Motherboard angeordnet. Das Problem der Cache-Kohärenz wird dadurch gelöst, daß alle Cache-Subsysteme die Buszugriffe beobachten (Snooping).

Hinweis: Nicht alle IA-32-Prozessoren unterstützen SMP (das betrifft vor allem jene Typen, die ausdrücklich für den Massen-Markt entwickelt wurden).



1 - Prozessoren; 2 - Hochleistungs-Speicherbus; 3 - APIC-Bus; 4 - eines der Speichermoduln; 5 - Bildspeicher; 6 - E-A-Steuereinrichtungen mit eigenem APIC; 7 - Bussysteme zum Betreiben von E-A-Einrichtungen (PCI, ISA, LPC usw.). Jeder der Prozessoren 1 enthält einen Interruptcontroller (APIC), eine Speicherbussteuerung sowie Caches nebst zugehöriger Cache-Steuerung.

Abb. 2.7 SMP-System (Intel)

BSPs und APs
Hierin unterscheiden sich die Prozessoren:

- BSP = Bootstrap Processor. Dieser Prozessor dient zum Initialisieren des Systems, zum Laden des Betriebssystems und zum Herunterfahren (Shutdown).
- AP = Application Processor. Diese Prozessoren werden erst nach dem Laden des Betriebssystems aktiv.

In jedem System gibt es nur einen BSP (welcher Prozessor diese Rolle spielt, wird hardwareseitig festgelegt (teils im Zusammenwirken mit dem BIOS)). Während des normalen Betriebs sind alle Prozessoren (BSP und APs) völlig gleichberechtigt.

Weiterentwicklungen
Platinen mit 2...4 Prozessoren können über eine Koppelplatine zu einem System mit beispielsweise 8 Prozessoren verbunden werden.

Größere Konfigurationen
Typisch sind Maximalbestückungen von 30...64 Prozessoren. Diese Systeme beruhen auf herstellerspezifischen Lösungen. Solche Maschinen wurden bereits Anfang der 90er Jahre angeboten (Beispiel: Sequent Symmetry S2000 mit maximal 30 486-Prozessoren).

CC-NUMA und SCA

CC-NUMA = Cache Coherent Non-Uniform Memory Access, SCA = Scalable Coherent Interface. Diese Fachbegriffe bezeichnen Entwicklungen, die das Ziel haben, leistungsfähigere und trotzdem kostengünstige SMP-Systeme zu schaffen (es soll möglich sein, wesentlich mehr Prozessoren zusammenzuschalten, wobei Teilsysteme auch räumlich weiter voneinander abgesetzt sein können).

SCA

Es handelt sich um ein Hochgeschwindigkeits-Interface, das vorzugsweise zum Zusammenschalten von Mehrprozessorsystemen bestimmt ist. Ausgewählte Merkmale im Überblick:

- Verbindungsprinzip: Punkt-zu-Punkt-Verbindungen, die typischerweise zu Ringstrukturen zusammengeschaltet werden (Abb. 2.8),
- Paketorientierung,
- eine SCA-Konfiguration kann bis zu 64k Knoten (Nodes) haben (16-Bit-Adressierung),
- getrennte Übertragung (Split Transactions = Trennung zwischen Auftragserteilung (Adresse + Art des Zugriffs) und eigentlicher Datenübertragung),
- verschiedene technische Ausführungen: (1) bidirektionaler Bus aus 18 Signalen, über den alle 2 ns 16 Bits übertragen werden, (2) bitserielles Interface mit einer Datenrate von 1 GBits/s, (3) Interfaces gemäß der FC-Spezifikation (Fibre Channel),
- die Übertragungsprotokolle "erledigen" u. a. die Cache-Kohärenz.

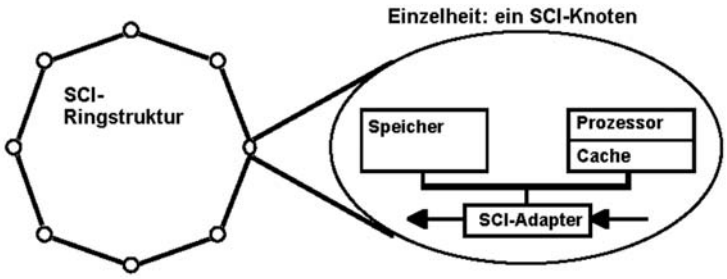


Abb. 2.8 Eine SCA-Konfiguration im Überblick

Wie wird die Cache-Kohärenz gewährleistet?

Es ist offensichtlich undurchführbar, über ein solches Interface Buszugriffe zu beobachten. Vielmehr muß man eine Lösung finden, die auf dem Prinzip der Nachrichtenübertragung (Message Passing) beruht. Der Ansatz: die einzelnen Cache-Einträge und auch die Speicherpositionen, die den Cache-Einträgen entsprechen, sind um Angaben ergänzt, in denen vermerkt ist, wo sich überall Kopien des betreffenden Eintrags befinden (man spricht hier von einem verzeichnisgestützten – Directory Based – Protokoll).

SCA unterstützt Cache-Einträge (Cache Lines) von 64 Bytes Länge. Diese 64-Byte-Blöcke sind sowohl im Cache als auch im Arbeitsspeicher um Verzeichnisangaben ergänzt (Abb. 2.9). Die Verzeichnisangaben bilden jeweils verkettete Listen – in jeder Einrichtung "kennt" jeder Eintrag nur die jeweils nächsten Knoten, die eine Kopie des Eintrags enthalten.

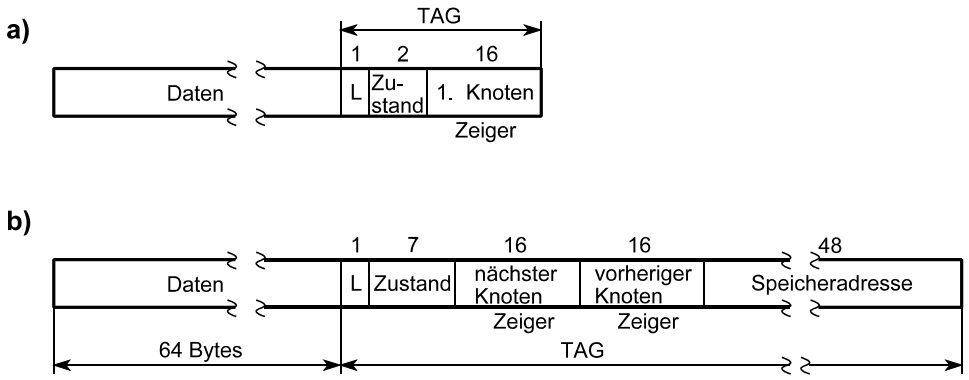


Abb. 2.9 SCA: Verzeichnisangaben. a) im Speicher, b) im Cache

Jeder Eintrag enthält 64 Datenbytes und ist um ein Kennzeichnungsfeld (TAG-Feld) erweitert (Speicher; 19 Bits, Cache: 88 Bits). Die Abbildung zeigt die Länge der einzelnen Felder in Bits.

- L = Lock. Kennzeichnet, daß der Eintrag gerade aktualisiert wird.
- Zustand. Ein Eintrag im Speicher kann 3 verschiedene Zustände haben (Tabelle 2.1 oben), ein Cache-Eintrag 7 (Tabelle 2.1 unten).
- Zeiger zu benachbarten Knoten. Der Eintrag im Speicher verweist auf den ersten Knoten, der eine Kopie des Eintrags hat. Ein Cache-Eintrag enthält Verweise auf die beiden benachbarten Knoten (auf den nächsten und auf den vorhergehenden), in denen Kopien des Eintrags gespeichert sind.
- Speicheradresse. In den Cache-Einträgen wird die ursprüngliche Speicheradresse mitgeführt.

Bezeichnung	Bedeutung
MS_HOME	Speicherinhalt gültig, keine Kopie in den Caches
MS_FRESH	Speicherinhalt stimmt mit den Kopien in den Caches überein
MS_GONE	es kann sein, daß Kopien in den Caches nicht mit dem Speicherinhalt übereinstimmen
CS_INVALID	Cache-Eintrag ist frei (darf belegt werden)
CS_ONLY_FRESH	es ist der einzige Cache-Eintrag, und der stimmt mit dem Speicherinhalt überein
CS_ONLY_DIRTY	es ist der einzige Cache-Eintrag, und der stimmt mit dem Speicherinhalt nicht überein
CS_HEAD_FRESH	Anfang einer Liste von Cache-Einträgen, die noch mit dem Speicherinhalt übereinstimmen
CS_HEAD_DIRTY	Anfang einer Liste von Cache-Einträgen, die nicht mehr mit dem Speicherinhalt übereinstimmen
CS_MID_VALID	ein mittleres Element einer Liste von Cache-Einträgen
CS_TAIL_VALID	das letzte Element einer Liste von Cache-Einträgen

Tabelle 2.1 SCA: Zustände von Cache-Einträgen (MS: im Speicher; CS: im Cache)

Auf Einzelheiten wollen wir hier nicht weiter eingehen. Womöglich kann man sich schon anhand von Abb. 2.9 und der doch recht suggestiven Zustandsbezeichnungen in Tabelle 2.1 eine ungefähre

Vorstellung davon bilden, wie die Cache-Kohärenz gewährleistet wird (ganz volkstümlich: beim Zugriff auf den jeweiligen Eintrag ist anhand der Verzeichnisangaben sofort zu erkennen, ob etwas zu tun ist oder nicht – es ist deshalb nicht mehr erforderlich, *alle* Zugriffe zu beobachten).

Betrachten wir abschließend nur ein Szenarium: ein Cache Miss führt zum Nachsehen im Arbeitsspeicher, um den Eintrag heranzuschaffen. Die Reaktion hängt vom jeweiligen Speicherzustand ab:

- bisheriger Zustand MS_HOME: eine Kopie wird in den Cache geholt. Es ist der einzige Cache, der eine Kopie hat. Das Zeigerfeld im Speicher verweist auf den betreffenden Knoten. Beide Zeigerfelder im Cache verweisen auf den Speicher. Übergang zu MS_FRESH.
- bisheriger Zustand MS_FRESH: eine Kopie wird in den Cache geholt. In anderen Caches stehen schon Kopien, die aber alle gültig sind. Der aktuelle Cache wird in die verkettete Liste aufgenommen, die zu diesem Eintrag gehört. (Hierzu werden die Zeiger in den betreffenden Cache-Einträgen entsprechend geändert.) Speicherzustand bleibt.
- bisheriger Zustand MS_GONE: der Speicherinhalt ist womöglich nicht mehr aktuell. Anhand der Zeiger in den TAG-Feldern werden jene Knoten angesprochen, die Kopien des Eintrags enthalten. Derjenige Knoten, der die veränderte Kopie enthält, liefert diese an den Arbeitsspeicher zurück. Zudem werden die Einträge in allen betroffenen Caches auf “ungültig” (CS_INVALID) gestellt (Prinzip der Cache Line Invalidation).

NUMA
“Uniform Memory Access” steht in der englischen Fachsprache dafür, daß alle Prozessoren in gleicher Weise auf den Speicheradreßraum zugreifen (z. B. über einen gemeinsamen Bus oder über das SCA-Interface). “Non-Uniform” bedeutet dann, daß es verschiedene Formen des Speicherzugriffs gibt (wobei trotzdem die Cache-Kohärenz gewährleistet bleibt).

Ein Ausführungsbeispiel: NUMA-Q (IBM)
Die einzelnen Knoten des Systems beruhen auf Motherboards, die mit je vier Xeon-Prozessoren bestückt sind. Eine solche Anordnung wird als Quad bezeichnet. Innerhalb des Quads wird die Cache-Kohärenz auf herkömmliche Weise gewährleistet (Stichwort: MESI-Protokoll). Diese Quads wiederum werden über SCI zu einem größeren System zusammengeschaltet (maximale Ausbaustufe: 64 Quads bzw. 256 Prozessoren). Hierzu ist in jedem Quad ein Adapter vorgesehen, der einen externen Cache (Beispiel: 128 MBytes, 4-fach blockassoziativ) gemäß den SCI-Prinzipien enthält (ansonsten entsprechen aber die Quads den üblichen Industriestandards).

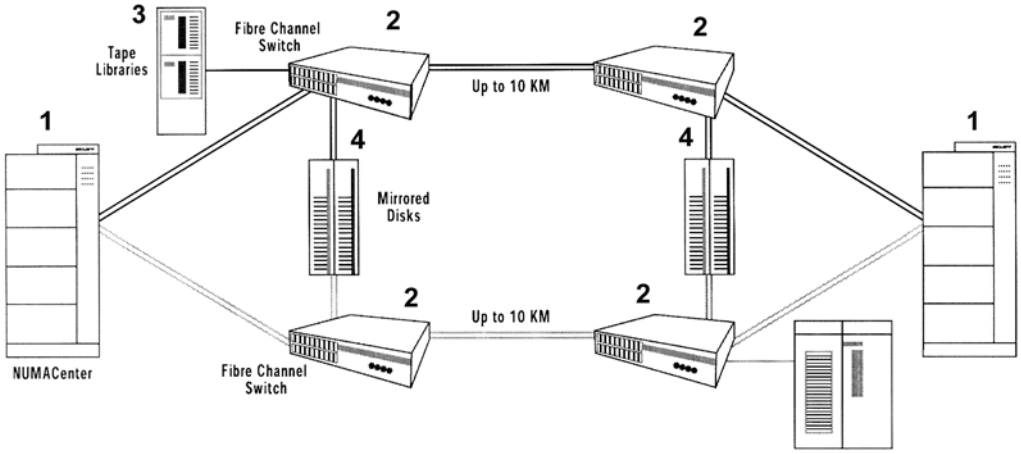
Multiprozessorsysteme im PC-Bereich – ein zusammenfassender Überblick
Wir können mehrere Auslegungen und Leistungsklassen voneinander unterscheiden. Nachstehend nehmen Leistungsvermögen und Kosten in der Reihenfolge der Aufzählung zu:

- Systeme auf Grundlage vernetzter PCs. Grundlage: Hardware des Massenmarktes. Fachbegriff: SHV (Standard High Volume).
- SMP-Systeme mit gemeinsamem Bus (vgl. Abb. 2.7). Stand der Technik: 2...8 Prozessoren.
- SMP-Systeme mit spezifischen Verbindungsstrukturen (Hochleistungs-Bussysteme, Crossbar-Strukturen). Stand der Technik: bis zu ca. 64 Prozessoren.
- Vernetzte Systeme auf Grundlage von Hochleistungs-Hardware (z. B. NUMA-Q).

Hinweise:

1. Mit Ausnahme der SHV-Lösungen gehen diese Systeme hinsichtlich der Kosten – und auch der technischen Auslegung – weit über den gewohnten PC (wie er auf jedermanns Schreibtisch steht) hinaus.

2. Was sich im obersten Leistungsbereich durchsetzen wird, ist noch keineswegs entschieden: dicht gekoppelte Systeme auf Grundlagen von Crossbar-Strukturen oder Vernetzungen ähnlich NUMA-Q (Abb. 2.10). Womöglich haben beide Auslegungen über längere Zeit ihre Existenzberechtigung.
3.



1 - NUMA-Q-Systeme; 2 - Fibre-Channel-Schaltverteiler; 3 - Magnetbandsubsysteme (Wechselautomaten); 4 - "gespiegelte" Festplattensubsysteme (redundante Datenhaltung). Fibre Channel (FC) ermöglicht hier zwischen den Schaltverteilern Leitungslängen von 10 km.

Abb. 2.10 Ein verteiltes Hochleistungssystem nach dem Prinzip des Storage Area Network (Sequent)

3. Betriebsmittelverwaltung: architekturseitige Voraussetzungen

Das Szenarium
Im System sind Betriebsmittel vorgesehen, die von mehreren Prozessoren gemeinsam genutzt werden (z. B. periphere Geräte). Solche Betriebsmittel sind jeweils einem nutzenden Prozessor zuzuteilen (es ist einleuchtend, daß nicht mehrere Prozessoren gleichzeitig beispielsweise Zeichen zu einem Drucker übertragen können).

Ein Multiprozessorsystem, das an sich für *unabhängige Zugriffe* mehrerer Prozessoren ausgelegt ist, erfordert also Vorkehrungen, um diese Unabhängigkeit gelegentlich aufheben zu können. Dafür kommen zwei Prinzipien in Frage: (1) die hardwareseitige *Blockierung*, (2) die softwareseitige *Verwaltung*.

Blockierung
Das betreffende Betriebsmittel wird für anderweitige Zugriffe gesperrt, wenn es erst einmal vermittelt worden ist. Andere Einrichtungen werden von der Vermittlung ausgeschlossen, bis die Blockierung beendet wird (eine andere Einrichtung, die während der Blockierung zugreifen will, muß auf deren Ende warten). Aus technischer Sicht ist es dazu notwendig, bei jedem Zugriffswunsch anzuzeigen, ob ein Blockieren von Fremdzugriffen erforderlich ist oder nicht. Sobald ein Zugriff mit Blockierungswunsch vermittelt wurde, darf an einem Bussystem kein neuer Busmaster ausgewählt werden bzw. ein Verteilernetzwerk darf für den betreffenden Adreßbereich keine anderen Zugriffe durchschalten.

Verwaltung
Jeder Prozessor, der ein Betriebsmittel nutzen möchte, muß vor dem ersten Zugriff eine Reservierungsroutine und nach dem letzten Zugriff eine Freigaberoutine rufen. Findet die Reservierungsroutine das Betriebsmittel als frei vor, so deklariert sie es als besetzt und stellt es dem betreffenden Prozessor zur Verfügung. Ist das Betriebsmittel bereits besetzt, so muß der Prozessor bis

zur Freigabe warten (wobei zwischenzeitlich andere Tasks Laufzeit erhalten könnten). Wird ein besetztes Betriebsmittel freigegeben, so kann es an den nächsten Prozessor vermittelt werden, der darauf wartet.

Diskussion
Das Blockierungsverfahren hat den Nachteil, daß es Prozessoren, die von der Vermittlung ausgeschlossen werden, in einen Wartezustand überführt, so daß sie in der Wartezeit keine anderen Aufgaben erledigen können.

Das Verwaltungsverfahren hat den Vorteil, daß die Arbeit in wartenden Prozessoren zwischenzeitlich umverteilt werden kann und daß sich vielfältige Prioritätsschemata verwirklichen lassen. Nachteilig ist, daß die Reservierungs- und Freigaberoutinen Laufzeit benötigen.

Verwaltung hat Blockierung zur Voraussetzung
Es kann vorkommen, daß mehrere Prozessoren gleichzeitig die Reservierung desselben Betriebsmittels wünschen. Die Reservierung- und Freigabeabläufe verschiedener Prozessoren können somit gleichsam aufeinandertreffen. Will man Verwaltungsroutinen implementieren, so hat dies folglich hardwareseitige Blockierungsmaßnahmen zur Voraussetzung. In vielen Mikroprozessoren ist dafür eine *Busverriegelung* (Bus Locking) vorgesehen.

Busverriegelung (Locking)
Damit sind Zugriffe ausführbar, die ein Bit im Speicher abfragen und es setzen, sofern es nicht gesetzt ist ("Test & Set"-Abläufe). Solche Abläufe bilden die elementare Voraussetzung für die programmseitige Verwaltung gemeinsam genutzter Betriebsmittel (Abb. 3.1).

1. Beispiel
Ein Prozessor will einen Drucker ansprechen. Er fragt dazu ein entsprechendes Besetzt-Bit ab. Ist es gelöscht, so ist der Drucker frei und kann benutzt werden. Um die Benutzung anzuzeigen, muß das Besetzt-Bit eingeschaltet werden.

2. Beispiel
Zwei Prozessoren suchen nach ausführbaren Tasks. Sie stoßen dabei beide auf ein geeignetes Taskzustandssegment (TSS). Um zu dieser Task umzuschalten, muß das BUSY-Bit im TSS-Deskriptor gesetzt werden. Es ist offensichtlich, daß nur ein Prozessor die Task ausführen kann. Ohne besondere Vorkehrungen könnten beide Prozessoren gleichzeitig BUSY abfragen, es als gelöscht erkennen, es setzen und die Task abarbeiten.

In derartigen Abläufen darf zwischen dem Lesen und dem Schreiben keine andere Einrichtung zum selben Speicherplatz zugreifen. Hierzu haben die Hochleistungs-Mikroprozessoren entsprechende Vorkehrungen (besondere Maschinenbefehle, Signale (Beispiel: das LOCK-Signal der IA-32-Prozessoren), Buszustände usw.).

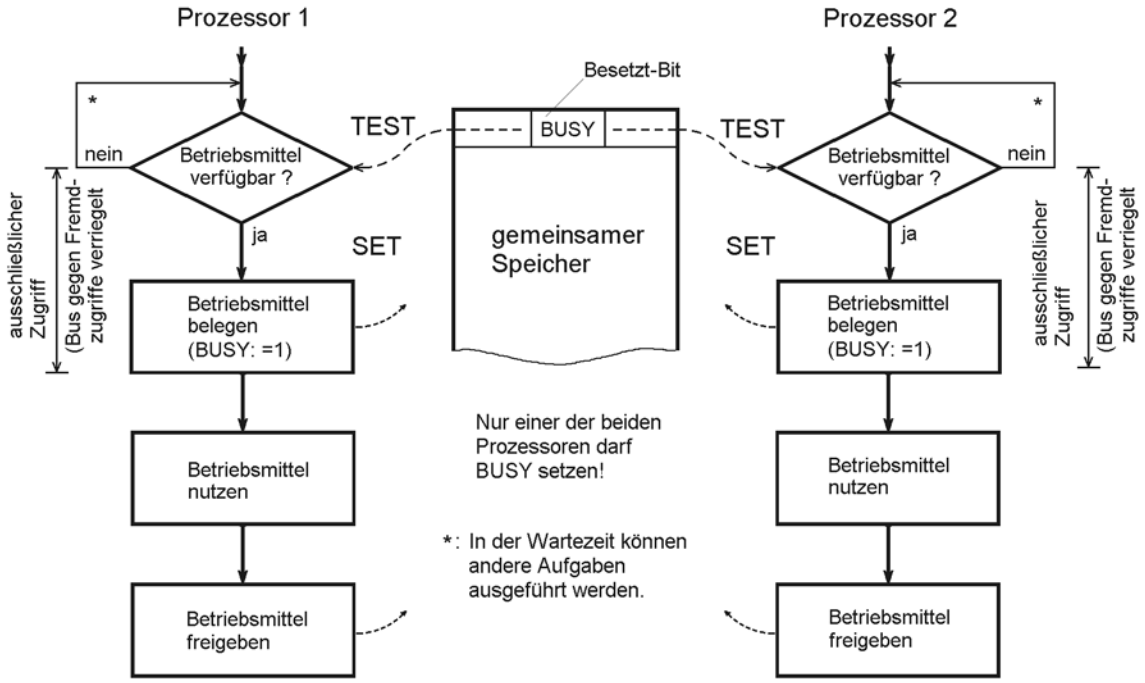


Abb. 3.1 Ablauf "Test & Set" als Grundlage der Betriebsmittelverwaltung

Beispiel: die Busverriegelung der IA-32-Prozessoren
Das LOCK-Signal kann sowohl befehlsseitig als auch - in bestimmten Situationen - automatisch aktiviert werden.

Automatische Aktivierung
Diese wird in Unterbrechungsbestätigungszyklen sowie bei bestimmten Speicherzugriffen wirksam, nämlich bei solchen, die Informationsstrukturen, die zur Systemsteuerung dienen, lesen, auswerten und wieder speichern (Read-Modify-Write-Zugriffe). Ein Beispiel ist das bereits angesprochene Setzen des Besetzt- (BUSY-) Bits in einem TSS.

Befehlsseitige Aktivierung
In bestimmten Befehlen kann LOCK durch ein entsprechendes Vorsatzbyte (Lock Prefix) aktiviert werden, das dem Befehl vorangestellt wird. Das betrifft die Befehle, die in programmseitigen Vermittlungsabläufen von besonderer Bedeutung sind, z. B: "Bit abfragen und setzen", "Vergleichen und Austauschen", "Austauschen und Addieren".

Beispiel einer alternativen Lösung: Mips
Konkurrierende Zugriffe werden nicht blockiert, sondern es wird nur überwacht, ob solche Zugriffe stattfinden oder nicht. Hierzu enthält jeder Prozessor ein Hilfsregister (1 Bit). Bei entsprechenden Zugriffen wird dieses Register von der Hardware gestellt (beim eigenen Zugriff wird es gesetzt, bei fremden gelöscht) und von der Software abgefragt. Die Nutzung: wir führen die Zugriffsfolge Lesen - Schreiben (Test & Set) mit entsprechenden Befehlen aus und fragen anschließend das Hilfsregister ab. Finden wir es als gesetzt vor, so hat der Ablauf geklappt, ist es gelöscht, so ist uns ein anderer Prozessor dazwischengekommen (unser Programm müßte dann ggf. so lange in einer Schleife kreiseln, bis es gelungen ist, die Test&Set-Zugriffsfolge ohne fremde Einmischung zu erledigen).