Proseminar KVBK

Mehrprozessorarchitekturen (SMP, Cluster, UMA/NUMA)

Arian Bär

12.07.2004

1. Einleitung:

Multiprozessor Systeme sind aus der Computerwelt heutzutage nicht mehr weg zu denken. Suchmaschinen wie z. B. Google benötigen Hochleistungs-Multiprozessor-Systeme um die Flut von Anfragen, die pro Sekunde ankommt, abzuarbeiten. Aber auch in der Klimaforschung und der Wettervorhersage kann ein einziger Prozessor nicht die erforderliche Leistung erbringen. Aus diesem Grund entstand schon früh der Gedanke mehrere Prozessoren zu einem einzigen Gesamtsystem zu verbinden. Die daraus entstandenen Systeme werden im Folgenden näher erläutert.

2. Symmetrische Multiprozessoren (SMP):

2.1. Allgemeines

Seit der Einführung des Multi-User Modes unterstützen alle modernen Betriebsysteme Aktivitätsträger die auch Threads genannt werden. Durch diese Threads wird es möglich, mehrere Aufgaben gleichzeitig zu bearbeiten, d.h. man kann gleichzeitig Musik hören und Texte schreiben. Da der Prozessor nur einen Thread bearbeiten kann wird bei einem Monoprozessorsystem zwischen den einzelnen Threads hin und hergeschaltet. So werden die parallelen Threads sequentiell abgearbeitet. Da diese Voraussetzung bereits erfüllt ist, arbeiten moderne Betriebssysteme sehr gut mit SMPs zusammen.
Bei SMPs können mehrere Threads gleichzeitig bearbeitet werden. Dadurch wird die Zeit, die normalerweise zum hin und herschalten zwischen den Threads verwendet wird, verringert. Da das Betriebssystem nun aber die Verteilung der Aufgaben übernehmen muss und auch der zugriff auf gemeinsam genutzte Variablen komplizierter wird (siehe später), steigt die Leistung des Gesamtsystems nicht linear mit der Anzahl der Prozessoren an.
Achtet man bereits bei der Implementierung darauf, Aufgaben, die parallel abgearbeitet werden können in einzelne Threads zu verpacken, können sich durch den Einsatz von SMPs enorme Leistungssteigerungen ergeben.
Ein Vorteil von SMPs gegenüber asymmetrischen Multiprozessoren ist, dass jeder Prozessor jede Aufgabe im System übernehmen kann. Stellt ein Prozessor eine bestimmte Ressource zur Verfügung, kann es dazu kommen, dass mehrere andere Prozessoren gleichzeitig auf diese zugreifen und so warten müssen.

2.2. Architektur

Eine SMP Architektur besteht aus mehreren identischen Prozessoren, die jeweils einen eigenen Cache haben, und über einen gemeinsamen Speicher miteinander kommunizieren. Jeder dieser Prozessoren ist gleichwertig, d. h. es gibt kein *master/slave* Verhalten in der Bearbeitung der Befehle. Verschiedene Prozessoren können die gleichen Kopien von Daten in ihrem Cache haben und es kann passieren, dass zwei oder mehr Prozessoren den Versuch starten, diese Daten zum gleichen Zeitpunkt zu ändern. Damit Programme zuverlässig arbeiten, muss sichergestellt sein, dass jeder Prozessor immer die neueste Version jeglichen Datums für die Verarbeitung verwendet. Methoden, die dies sicherstellen, heißen Cache kohärent.

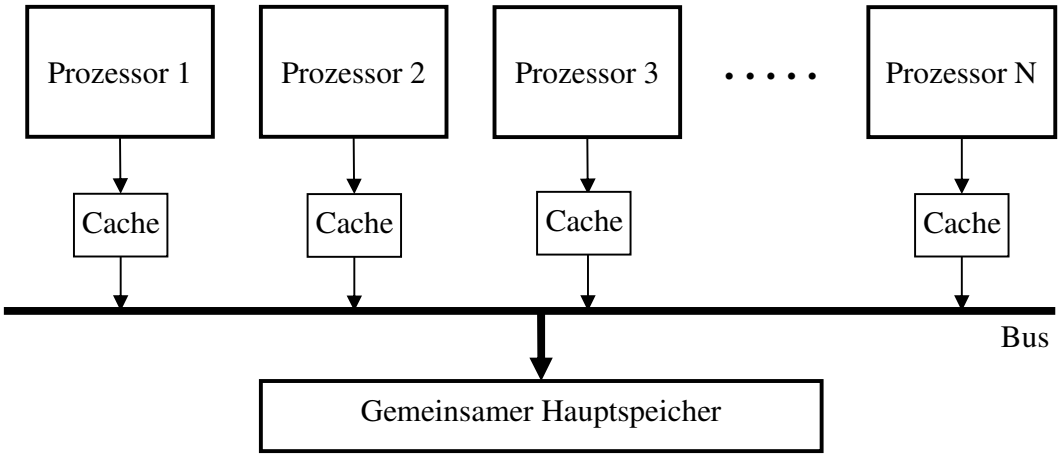

Abb. 1: Aufbau eines SMP-Systems

Die einfachste Möglichkeit inkonsistente Cachespeicher zu vermeiden, besteht darin, das Halten gemeinsam genutzter Variablen, die gelesen und geschrieben werden sollen zu unterbinden und diese als ‚noncacheable' zu markieren. Da dies aber zu einer komplizierteren Programmierung und einer geringeren Trefferrate bei Cache Zugriffen führt, sinkt die Leistungsfähigkeit des Systems.
Eine andere Möglichkeit konsistente Daten sicherzustellen, ist der so genannte „*snoop bus*" . Bei diesem Verfahren sendet der Prozessor, der eine bestimmte Speicheradresse benötigt, die nicht in seinem Cache ist, an alle anderen Prozessoren eine Anfrage nach dieser Speicheradresse auf den Bus. Jeder Prozessor kontrolliert diesen Bus ständig nach Adressanfragen(snooping). Empfängt er eine solche Anfrage, überprüft er ob diese Adresse in seinem Cache vorhanden ist und sendet eine Antwort an den anfragenden Prozessor zurück aus der hervorgeht, ob dies der Fall ist oder nicht. Ist nun die Adresse im Cache eines anderen Prozessors, wird sie von dort aus in den Cache des anfragenden Prozessors übertragen. Wenn nicht kann der anfragende Prozessor sie aus dem gemeinsamen Speicher laden und es wird sichergestellt, dass die neueste Version der Daten zur Verarbeitung verwendet wird.

3. Speicherarchitekturen

3.1. Gemeinsamer Speicher (shared memory - SM):

In Systemen mit gemeinsamem Speicher - auch *shared memory architectures* genannt - wird ein von allen Prozessoren zugänglicher Speicher verwendet. Ein Verbindungs-Netzwerk stellt die Mittel zur Verfügung, die nötig sind um Daten aus dem gemeinsamen Speicher zu den Prozessoren zu übertragen.

Grundsätzlich lassen sich Systeme mit gemeinsamem Speicher in drei Klassen einteilen:

- gleichförmiger Speicherzugriff (*uniform memory access* - UMA)
- ungleichförmiger Speicherzugriff (*non-uniform memory access* - NUMA)
- nur Cachespeicherzugriffe (*cache-only memory architecture* - COMA)

Bei der UMA-Architektur ist die Zugriffsweise aller Prozessoren auf den Speicher identisch. Prozessoren und Speichermodule stehen sich gegenüber und sind durch ein schnelles Verbindungsnetzwerk miteinander verknüpft. Die Latenz der Speicherzugriffe ist für alle Prozessoren gleich. Da die Bandbreite des Verbindungsnetzwerkes sehr hoch sein muss, lassen sich UMA-Architekturen nur mit wenigen Prozessoren realisieren.
Mainboards die mehr als einen Prozessor unterstützen, wie z. B. das Intel® Server Board SE7525GP2, verwenden meist eine UMA-Architektur.

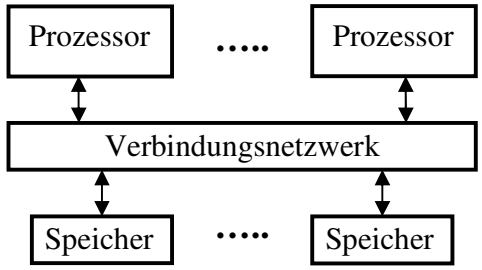

Abb. 2: UMA-Architektur

Um die hohen Anforderungen an das Verbindungsnetzwerk zu verringern und somit auch mehrere hundert Prozessoren zu einem Parallelrechner zusammen zu schalten, kann man die Speichermodule den einzelnen Prozessoren lokal zuordnen und nur globale Zugriffe über das Verbindungsnetzwerk realisieren. Die Einheit, die aus der Verbindung von Prozessor, Speicher und Ein-/Ausgabe Geräten entsteht, nennt man Knoten (s. Abb. 3). Durch die unterschiedlichen Latenz-Zeiten der Speicherzugriffe erhält man eine NUMA-Architektur. Da bei NUMA-Architekturen, wie auch bei Einprozessor-Systemen oder UMA-Architekturen, nur ein großer Adressraum existiert, merkt der einzelne Prozessor nur an der längeren Latenz-Zeit, ob er auf seinen eigenen, lokalen Speicher zugreift oder nicht. Dies hat zur Folge, dass die Effizienz solcher Systeme stark von der Qualität und der Lokalität des Programmcodes abhängig ist.
Bei Servern die aus mehreren AMD® Opteron Prozessoren aufgebaut sind kommt die NUMA-Architektur zum Einsatz. Bei den Opterons werden Hypertransport-Kanäle, die eine Bandbreite von bis zu 12.8 GB/s zur Verfügung stellen, zur Hauptspeicher Anbindung und als Verbindungsnetzwerk für den Mehrprozessor einsatz verwendet. Da

jeder Prozessor direkt über einen Hypertransportchannel mit seinem lokalen Speicher verbunden ist und wiederum jeder Prozessor mit ein bis zwei anderen Prozessoren sind Speicherzugriffe auf entfernte Speichermodule langsamer wie auf den lokalen Speicher.

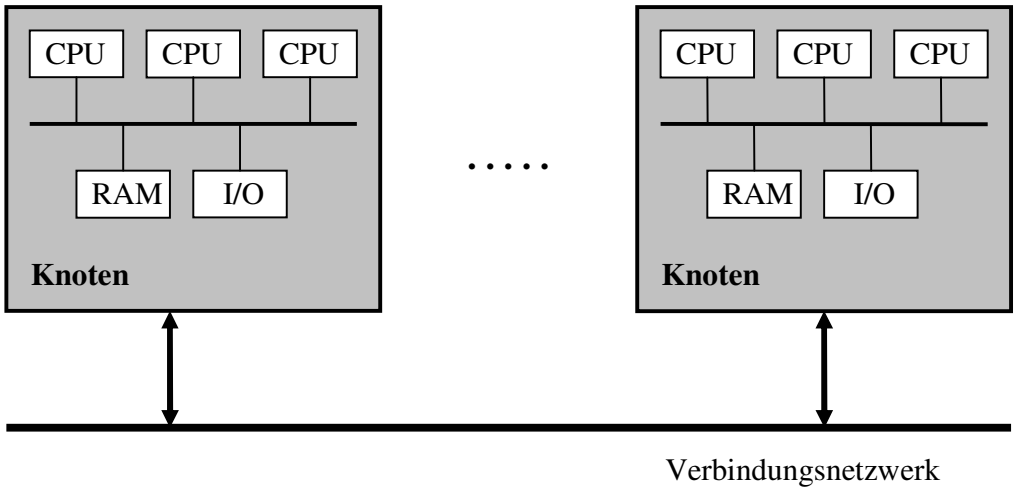

Abb. 3: Aufbau eines NUMA-Systems

Bei COMA-Architekturen werden die Speichermodule ausschließlich als Pufferspeicher verwendet. Alle Cachespeicher befinden sich in einem globalen Adressraum. Aus der Sicht eines Prozessors wird der eigene Speicher als Pufferspeicher gesehen, der restliche Speicher des Systems als Hauptspeicher. Der Zugriff auf entfernt liegende Caches wird durch ein *cachedirectory* unterstützt. Die Datenobjekte werden durch den „Cache-Mechanismus“ zu dem Prozessor transportiert, von dem sie angefordert werden.
Im Moment gibt es keine kommerziellen Multiprozessorsysteme, die die COMA-Architektur einsetzen.

3.2 Verteilter Speicher

In Systemen mit verteiltem Speicher (*distributed memory architecture*) sind die einzelnen Speichermodule über das System verteilt und typischerweise einem Prozessor physikalisch zugeordnet. Hat jeder Prozessor nur Zugriff auf seinen lokalen Speicher, so spricht man von verteiltem Speicher mit lokalem Adressraum (distributed local memory). In solche Architekturen kann jeder Prozessor nur seinen eigenen Speicher direkt ansprechen (no-remote memory access - NORMA). Diese werden deshalb auch als NORMA-Architekturen bezeichnet.
Die Verbindung der Prozessoren untereinander erfolgt über ein Verbindungsnetzwerk. Das Verbindungsnetzwerkinterface (VNI), den Prozessor und seinen lokaler Speicher bezeichnet man hier als Knoten.
Die Prozessoren besitzen eine sehr schnelle Verbindung zu ihren lokalen Speichermodulen. Die Verbindung der Knoten untereinander erfolgt nur über das VNI und ist im Vergleich zu SM-Systemen eher langsam. Die Kommunikation erfolgt über den Austausch von Nachrichten.

4. Cluster Computer

Ein Cluster ist ein paralleles oder verteiltes Verarbeitungssystem, welches aus einer Sammlung von allein stehenden Computern besteht, die über ein Netzwerk miteinander verbundenen sind.
Ein Computer (Knoten) kann ein Ein- oder Mehrprozessorsystem (z. B. SMP) sein, mit Speicher, Ein-/ Ausgabegeräten und einem eigenen Betriebssystem. Typischerweise besteht ein Cluster aus zwei oder mehreren verbundenen Knoten. Die Knoten befinden sich entweder in einem einzigen Gehäuse oder sind physikalisch getrennt und über ein LAN miteinander verbunden. Ein Cluster kann sich für die Anwendungen und Benutzer wie ein einzelner Computer verhalten. Typischerweise ist ein Cluster wie in folgendem Bild (Abb. 4) aufgebaut.

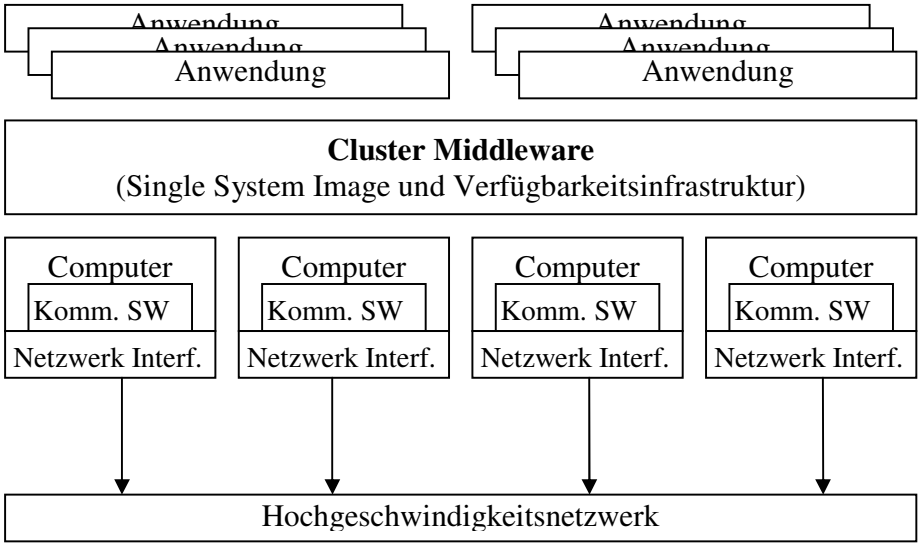

Abb. 4: Cluster Computerarchitektur

Die Netzwerkschnittstelle dient dem Cluster als Kommunikations-Prozessor und ist für den Transport der Datenpakete zwischen den Knoten und dem Netzwerk (switch) verantwortlich.
Die Kommunikations-Software stellt sicher, dass die Pakete schnell und zuverlässig übertragen werden. Manche Implementierungen umgehen das Betriebssystem, um den Kommunikations-*overhead* zu umgehen. Die Middleware kann dadurch direkt auf das Netzwerk Interface zugreifen. Meist verwenden Cluster Hochgeschwindigkeits-Netzwerke wie Gigabit Ethernet oder Myrinet zur Datenübertragung.
Die Cluster Middleware sorgt dafür, dass alle beteiligten Computer sich für den Benutzer wie ein einziges System darstellen (*single system image* - SSI). Sie teilt die Arbeit unter den einzelnen Knoten gerecht auf (*load balancing*). Load balancing wird häufig bei großen Webauftritten eingesetzt. Der Server der die eingehenden Anfragen empfängt ist einem Netz aus mehreren Webservern vorangestellt. Er überwacht diese und weiß wie

stark jeder einzelne Server ausgelastet ist. Kommt nun eine Anfrage an so leitet er diese an den am wenigsten ausgelasteten Webserver weiter und sorgt so dafür, dass sich die Last gleichmäßig verteilt. Außerdem ist die Cluster Middleware dafür verantwortlich, die Verfügbarkeit des Clusters zu überwachen und an ausgefallene Computer keine Nachrichten mehr zu senden.
Oft sind Cluster kostengünstiger als SMPs oder Systeme mit verteiltem Speicher da sie aus Unterschiedlichsten Computern bestehen können so das es möglich ist viele Arbeitplatzrechner Nachts zu einem Cluster zusammen zuschalten und so zeitaufwendige Berechnungen durchzuführen.

Literaturverzeichnis:

I. Klaus Waldschmidt (Hrsg.): Parallelrechner Architekturen – Systeme – Werkzeuge, Stuttgart: Teubner, 1995

II. Rajkumar Buyya: High Performance Cluster Computing Architectures and Systems, Volume 1, Prentice Hall PTR Upper Saddle River, New Jersey 1999

III. George Coulouris, Jean Dollimore, Tim Kindberg: Verteilte Systeme Konzepte und Design, 3., überarbeitete Auflage, Pearson Education Limited, 2002

IV. Sequent Computer Systems, Ltd., http://parallel.ru/ftp/computers/sequent/NUMA_SMP_REV.pdf , 16.05.2004

V. Jörg Wirtgen/c't: AMD will NUMA-Architektur forcieren, http://www.heise.de/newsticker/meldung/14776, 16.05.2004

VI. Stefan Schumacher: Parallelrechner: Multiprozessorsysteme, http://www.uni-magdeburg.de/steschum/multiprozessorsysteme.pdf 16.05.2004

VII. Intel® E7525 Memory Controller Hub (MCH) Chipset Datasheet, ftp://download.intel.com/design/chipsets/datashts/30240501.pdf, 07.07.2004

VIII. AMD: Porting to AMD64 FAQ, http://www.amd.com/ us-en/assets/content_type/Additional/AMD64_Porting_FAQ.pdf 07.07.2004

IX. Aad van der Steen: ccNUMA machines, http://www.top500.org/ORSC/2003/ccNUMA.html, 07.07.2004