Parallele Systeme

RRZE-Kolloquium 1997

Claus-Uwe Linster

IMMD IV, Lehrstuhl für Betriebssysteme
Universität Erlangen-Nürnberg

linster@informatik.uni-erlangen.de

© Claus-Uwe Linster – rrze.koll97: 19/2/97

Parallele Systeme

- Klassifikation Paralleler Systeme (... zur Erinnerung...)
- Marktgeschehen 95/96 (... der Markt ist übersichtlicher geworden ...)
- Der “Erlanger-Rechner-Park” (... kann sich sehen lassen!)
- Das Problem mit der Leistungsausbeute (... stolz sein auf 10%...)
- Benchmarks
  (...erst müssen wir die Benchmarks durchbringen, bevor wir Performanceverbesserungen realisieren können...)
- Programmiermodelle vs. Rechnerarchitekturen (... was geht gut, was geht weniger gut ...)
- Ausbildungsangebote (... auch für Nichtinformatiker...)

© Claus-Uwe Linster – rrze.koll97: 19/2/97

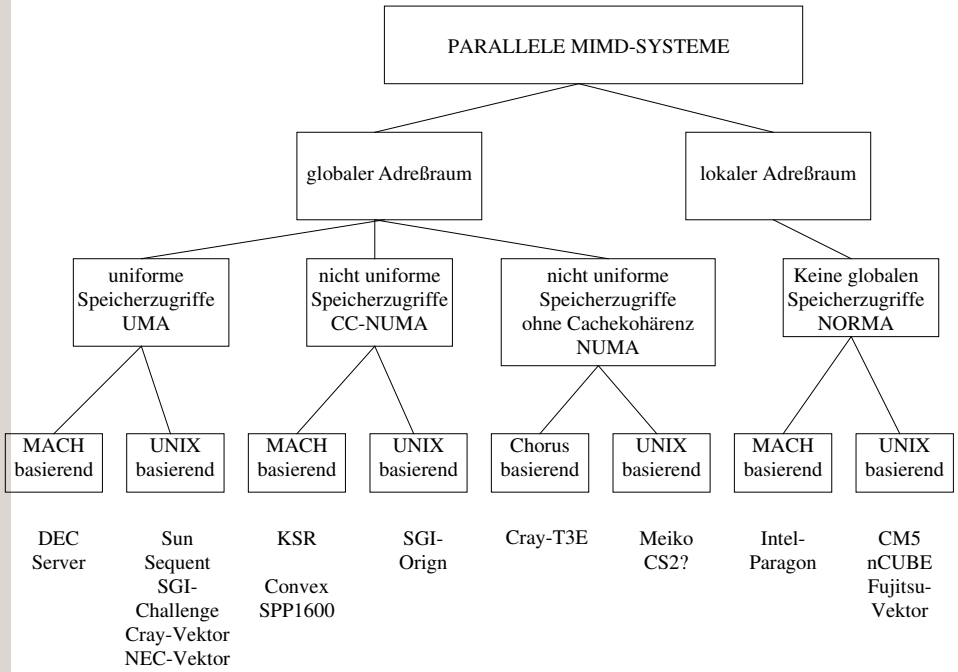
1 Klassifikation Paralleler Systeme

- Flynn's Klassifikation

Data - Stream / Instruction - Stream	SINGLE	MULTIPLE
SINGLE	**SISD** Klass. von Neumann-Architektur	**SIMD** Vektor-/Array - Prozessoren
MULTIPLE	**MISD** nicht sinnvoll	**MIMD** Parallelrechner; bestehend aus Vektor- oder SIMD - Prozessoren

© Claus-Uwe Linster – rrze.kol97: 19/2/97

1.1 Parallele MIMD - Systeme



© Claus-Uwe Linster – rrze.kol97: 19/2/97

2 Marktgeschehen 95/96

- Viele Parallelrechneranbieter haben den techn./wissenschaftlichen Markt weitgehend aufgegeben und engagieren sich vorwiegend auf dem Datenbank-Marktsektor: z. B. NCube, Parsytec, Sequent
- Einige haben den Markt als Rechnerhersteller ganz verlassen: z. B. KSR, TMC; (Meiko??; Intel: techn./wiss/. Einzelfertigung?)
- Einige hatten das “Glück”, von kapitalkräftigen Unternehmungen übernommen zu werden: Covex von HP; Cray von SGI
- Bei einem hat sich **nichts** geändert: IBM
- Die Japaner Fujitsu und NEC, (sowie Cray??) dominieren den zukünftigen Vektorrechnermarkt, der offensichtlich doch nicht so tot ist, wie viele glaubten. Das hat wohl zwei wesentliche Gründe:
  - Vektorrechner sind durch Einsatz von Standardtechnologien preiswerter geworden
  - Bei einigermaßen vektorisierbaren Anwendungen haben Parallelrechner auf Risc-Prozessorbasis im Vergleich keine Chance (Siehe auch: 4.)

© Claus-Uwe Linster – rrze.kol97: 19/2/97

Marktgeschehen 95/96

- Gründe der Veränderungen:
  - der techn./wiss. Markt hat einfach nicht das finanzielle Potential die Entwicklung der aufwendigen Technologien bei kleinen Stückzahlen zu tragen.
  - Durch die Leistungsentwicklung bei “Standardprodukten” bedingt, wird das “Grand-Challenge-Fenster” (d.h. Anwendungen, die auf “herkömmlichen” Rechnern nicht zu Ende zu bringen sind) immer kleiner und “wissenschaftlicher”. (Allerdings ist eine Gegentendenz durch Kopplung von Systemen erkennbar.)
  - Industrielle Unternehmungen stützen sich zunehmend auf “öffentliche” Rechensysteme, was den Markt weiter schrumpfen läßt; insbesondere aber auch die Vielfalt.
- Ein Überleben ist offensichtlich nur dann möglich, wenn eine breitere Basis bei den Anwendungen (z. B. Datenbanksysteme, Data-Mining, Fileserver) gefunden wird und die Einzelkomponenten ihre Basis im Massenmarkt haben oder deren Entwicklungsziel es im Rahmen von Technologie-weiterentwicklung ist, ein Massenprodukt zu werden.
- Zusammenfassend: Der Markt ist übersichtlicher geworden.

© Claus-Uwe Linster – rrze.kol97: 19/2/97

3 Der “Erlanger-Rechner-Park”

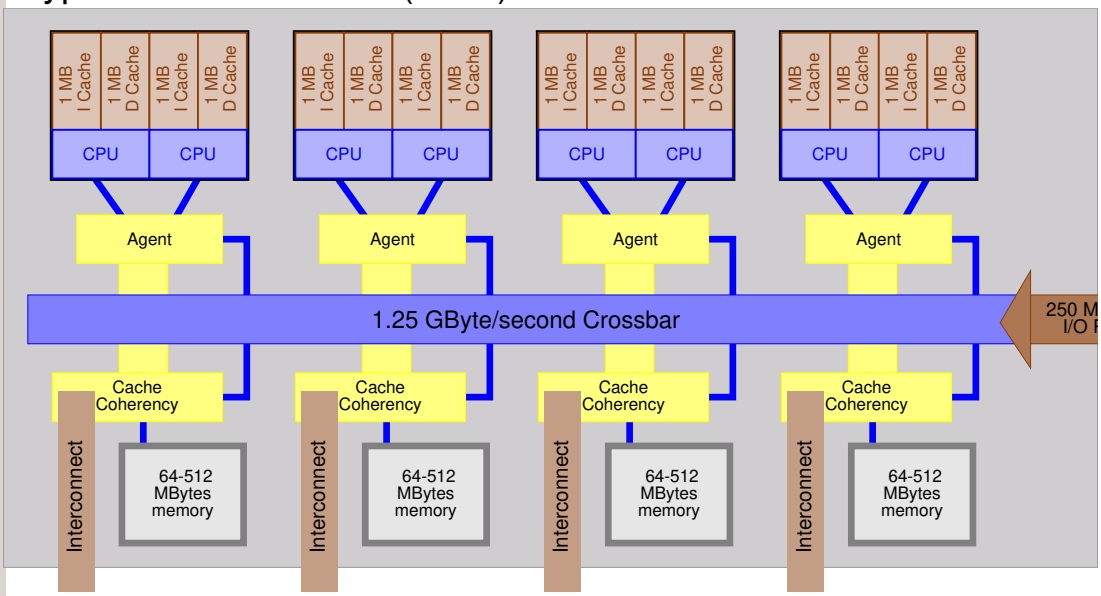
System	Anzahl der Proz.	Proz. Takt	Arbeits-speicher	Architektur	Netzwerk	Peak-GFlops	Standort
Convex-SPP1600	48 x HP 7200	120 MHz	6.0 GB	NUMA	--	11.520	RRZE
SUN E4000	8 x ULTRA1	250 MHz	2.0 GB	UMA	--	4.0	RRZE
CRAY-EL	4 x Cray-Vekt.	30 MHz	1.0 GB	Vektor-UMA	--	0.532	RRZE
Fujitsu - VPP300	6 x Fuj. - Vekt.	143 MHz	12 GB	Vektor-NORMA	Crossbar	26.4	RRZE-97
KSR1	32 x KSR-Proz	20 MHz	1.0 GB	NUMA	--	1.280	IMMD
CM5	32 x Vektor	20 MHz	1.0 GB	Vektor-NORMA	Fattree	4.096	IMMD
IBM-SP2[a]	70 x IBM-PC	66MHz	16 GB	NORMA	H-Crossbar	21	LRZ
Fujitsu VPP700[b]	52 x Fuj.-Vekt.	143 MHz	104 GB	Vektor-NORMA	Crossbar	114.4	LRZ-97
Cray-Triton	4 x Cray-Vekt.	450 MHz	1.0 GB	UMA	--	7.2	LRZ

a.Maschinenausbau nicht homogen (58 + 14 +5 Systeme, mit steigendem Speicher- und Plattenausbau)
b.Zunächst 34 Knoten; später 52

© Claus-Uwe Linster – rrze.koll97: 19/2/97

3.1 Convex SPP1600 (RRZE)

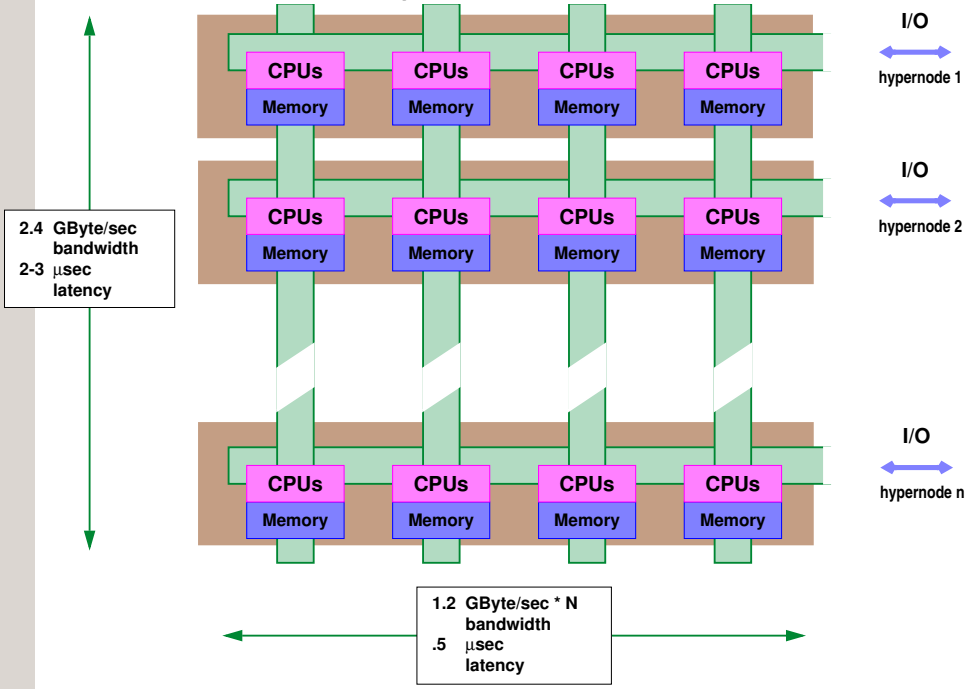
- Hypernode Architektur (UMA)



© Claus-Uwe Linster – rrze.koll97: 19/2/97

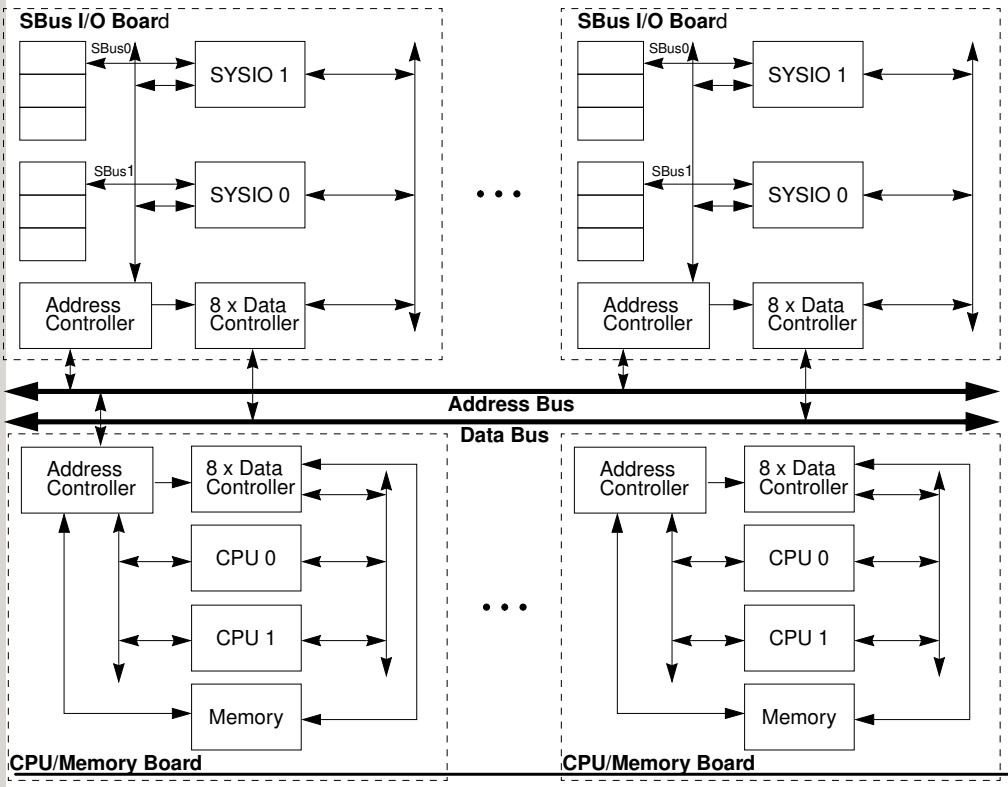
Convex SPP1600 (RRZE)

- Convex SPP1600 - Speicherhierachie (NUMA)



© Claus-Uwe Linster – rrze.koll97: 19/2/97

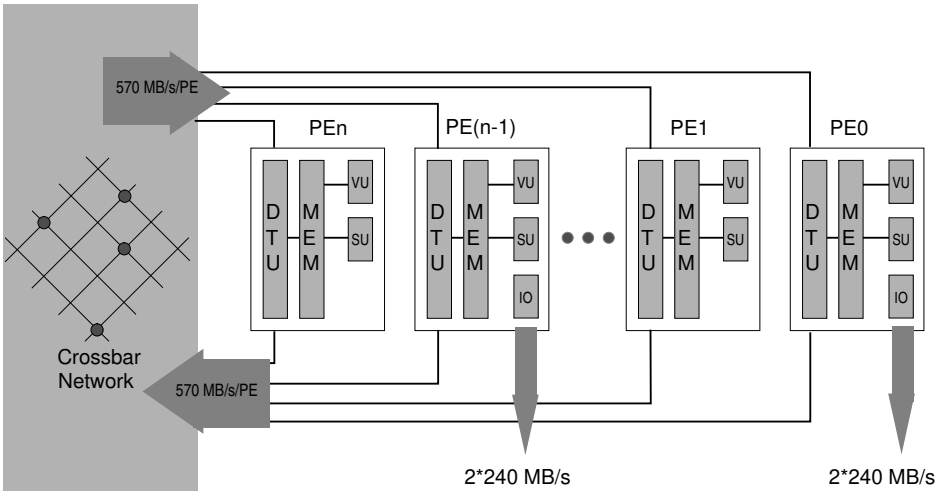
3.2 Sun-Enterprise-Server (RRZE, IMMD)



© Claus-Uwe Linster – rrze.koll97: 19/2/97

3.3 Fujitsu VPP300/700 (RRZE/LRZ)

Architektur

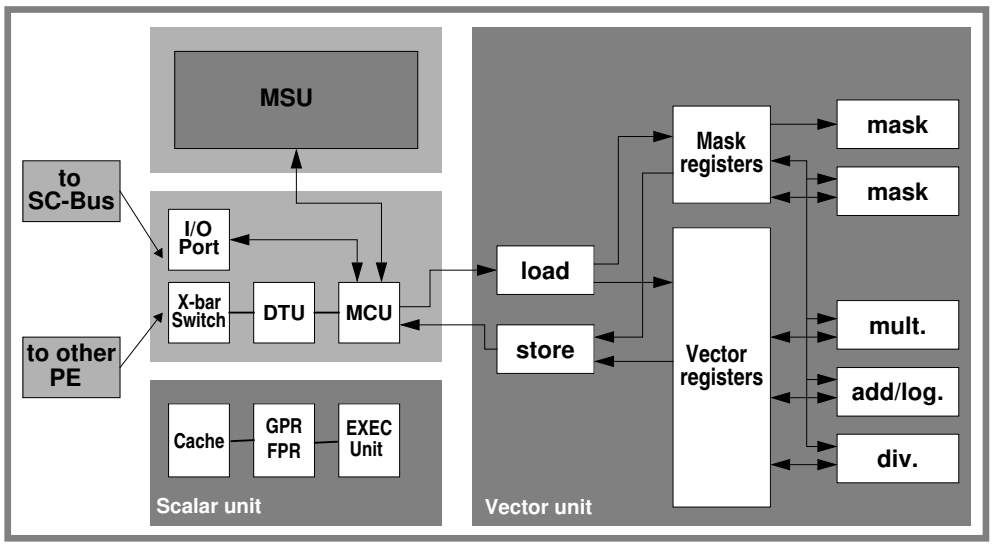


SU: Scalar Unit
VU: Vector Unit
IO: IO-Port
MEM: Memory
DTU: Data Transfer Unit

© Claus-Uwe Linster – rrze.kol97: 19/2/97

Fujitsu VPP300/700 (RRZE/LRZ)

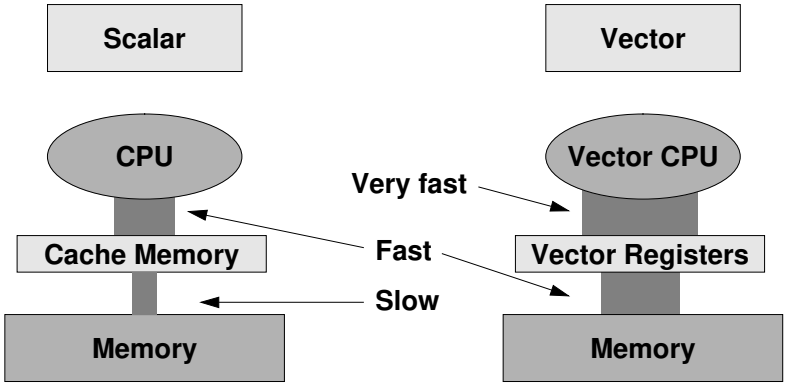
Prozessor



© Claus-Uwe Linster – rrze.kol97: 19/2/97

Fujitsu VPP300/700 (RRZE/LRZ)

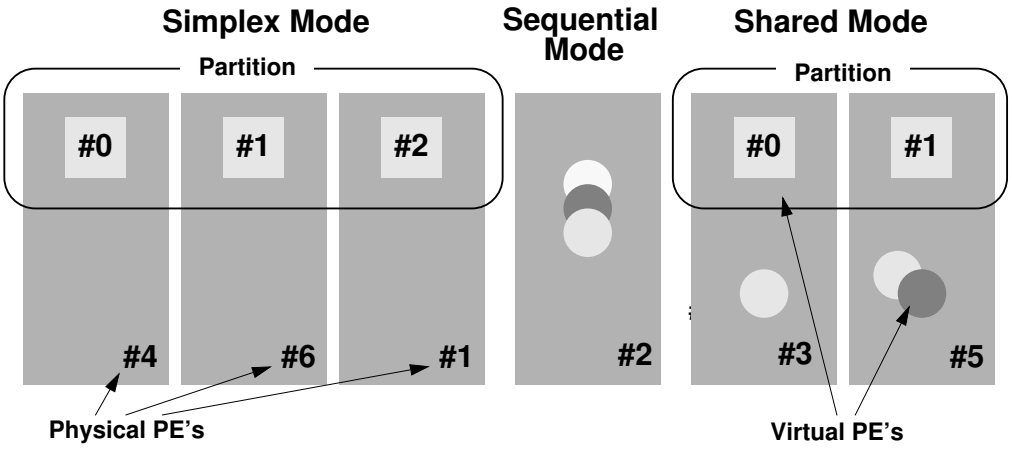
Scalar and Vector Processors



© Claus-Uwe Linster – rrze.koll97: 19/2/97

Fujitsu VPP300/700 (RRZE/LRZ)

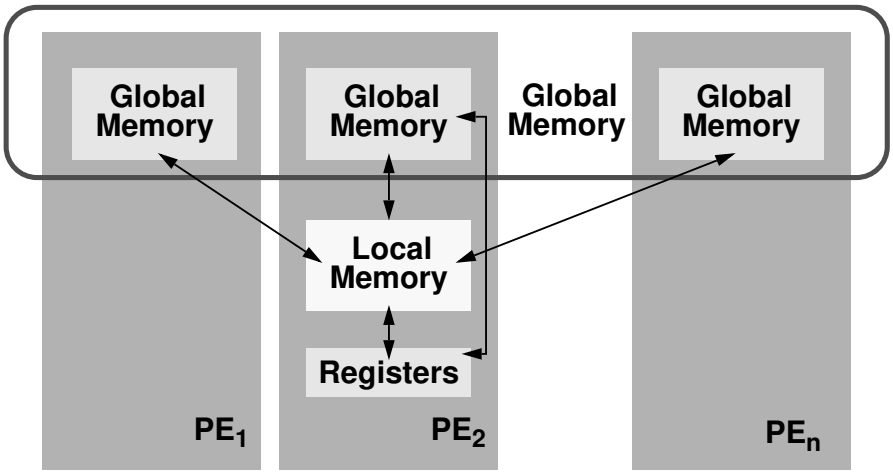
- Betriebsmodi



© Claus-Uwe Linster – rrze.koll97: 19/2/97

Fujitsu VPP300/700 (RRZE/LRZ)

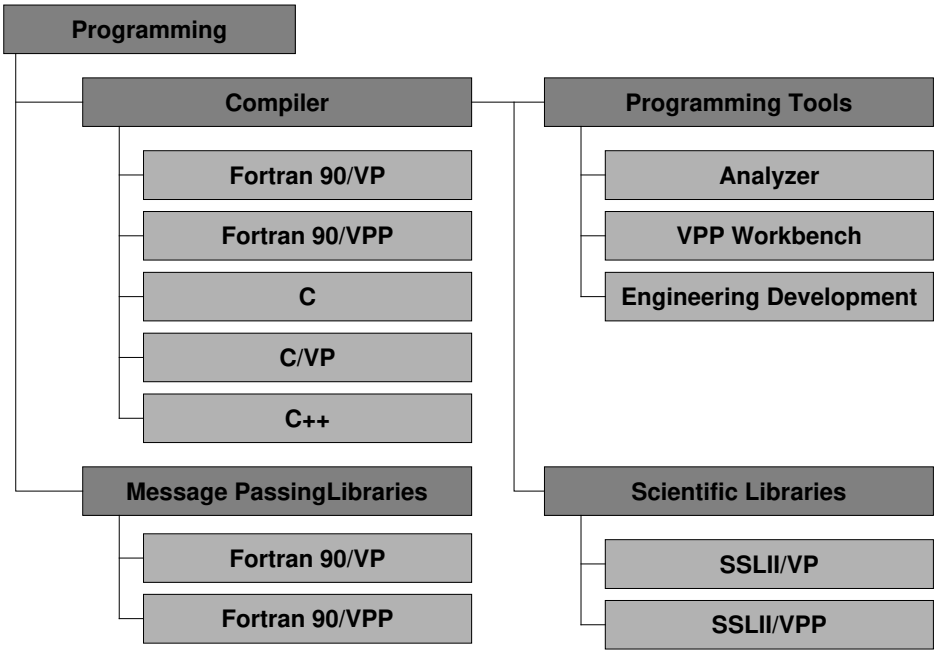
- Speicherzugriff



© Claus-Uwe Linster – rrze.kol97: 19/2/97

Fujitsu VPP300/700 (RRZE/LRZ)

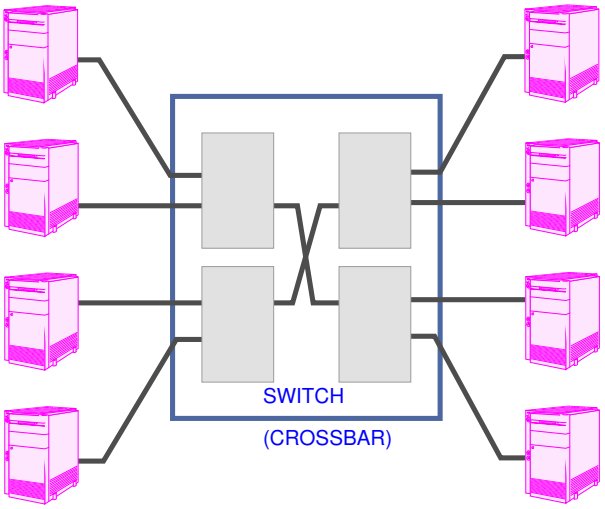
- Software-Umgebung



© Claus-Uwe Linster – rrze.kol97: 19/2/97

3.4 IBM-SP2 (LRZ)

- IBM SP-2 (max. 128 Prozessoren) am LRZ

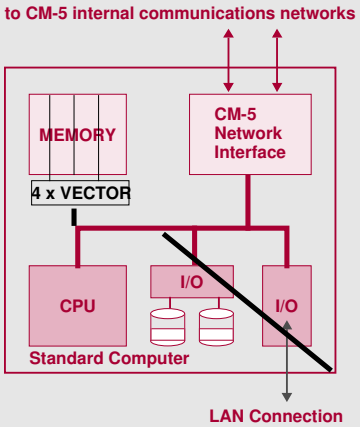


- 77 Knoten (9000/590)
- 14 dicke, 256MByte; 5 ganz dicke
- 58 dünne, 128MB, geringere Speicherbandbreite, kleinerer E/A-Ausbau, gleiche Peak-Leistung

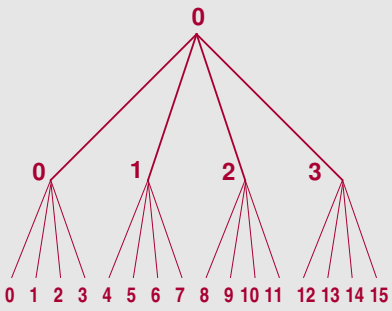
© Claus-Uwe Linster – rrze.kol97: 19/2/97

3.5 Thinking Machines CM-5 (IMMD)

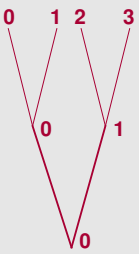
CM-5 - Processor



Data Net Implementation: 16 Leaves



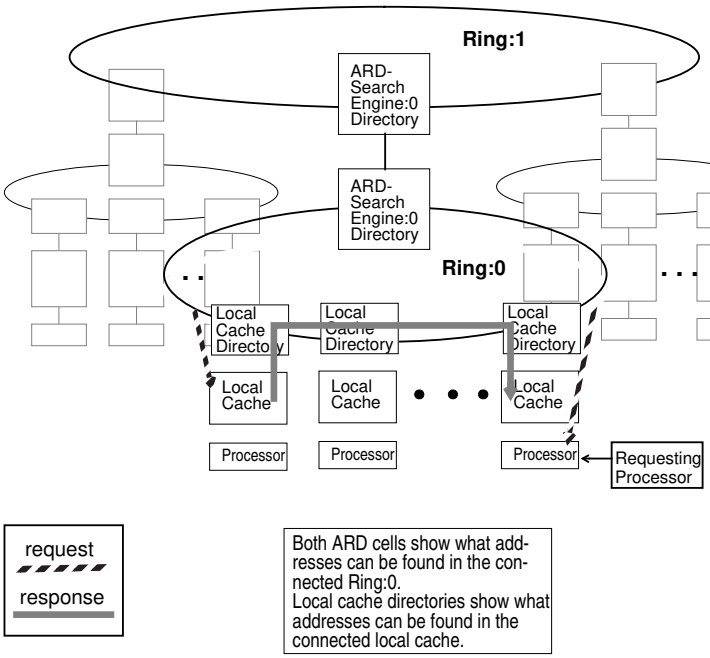
Front View



Side View

© Claus-Uwe Linster – rrze.kol97: 19/2/97

3.6 KSR 1 (IMMD)



© Claus-Uwe Linster – rrze.kol97: 19/2/97

4 Leistungsausbeute

- Warum ist eine Leistungsausbeute von 10% der Peak-Leistung - insbesondere bei Risc-Systemen - ein guter Wert?
- Der Verlust entsteht im wesentlichen aufgrund folgender Problemklassen

1 Mangelnde Nutzung der “inneren” Prozessorparallelität

2 Mangelnde Nutzung der Prozessor-/Speichermechanismen, Speicherzugriffslatenzzeiten zu verbergen, sowie mangelndes lokales Verhalten der Anwendung, die der intensiven Nutzung lokaler Hochleistungs-Ressourcen (Caches) entgegen steht.

3 Der mögliche Parallelitätsgrad der Anwendungen selbst, unter Berücksichtigung einer architekturadäquaten Granularität.

4 Die mangelnde Fähigkeit von Universalbetriebssystemen parallele numerische Programmsysteme optimal auf parallelen Architekturen zu unterstützen.

© Claus-Uwe Linster – rrze.kol97: 19/2/97

4.1 Mangelnde Nutzung der “inneren” Prozessorparallelität

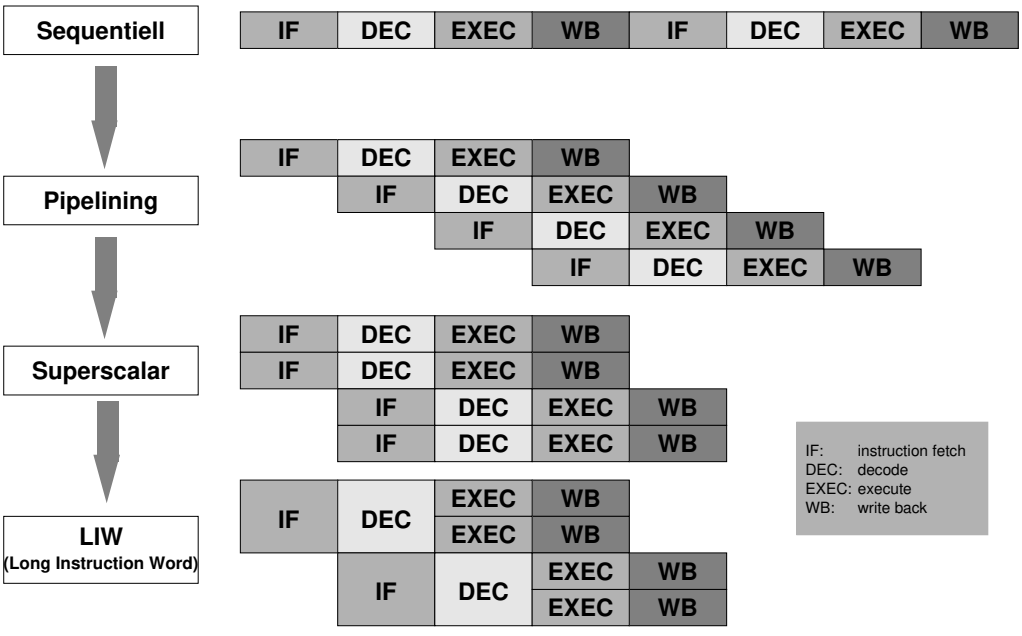
- Pipelining:
  verschiedene Phasen verschiedener Instruktionen werden parallel verarbeitet. Ein Instruktionsstrom; überlappende Verarbeitung
- Superscalar:
  die Möglichkeit mehrere Funktionen parallel zu verarbeiten
  z. B. Load / Store : Add / Multiply : Branch
  Paralleler Instruktionsstrom ausgewählter Instruktionstypen.
- VLIW:
  “Breite” Instruktions-Verarbeitung
  mehrere Befehle werden parallel bearbeitet.
  Paralleler Instruktionsstrom.

▲ Parallelisierung des Datenstroms, “breite” Daten-Verarbeitung

- Vektoreinheiten

© Claus-Uwe Linster – rrze.kol97: 19/2/97

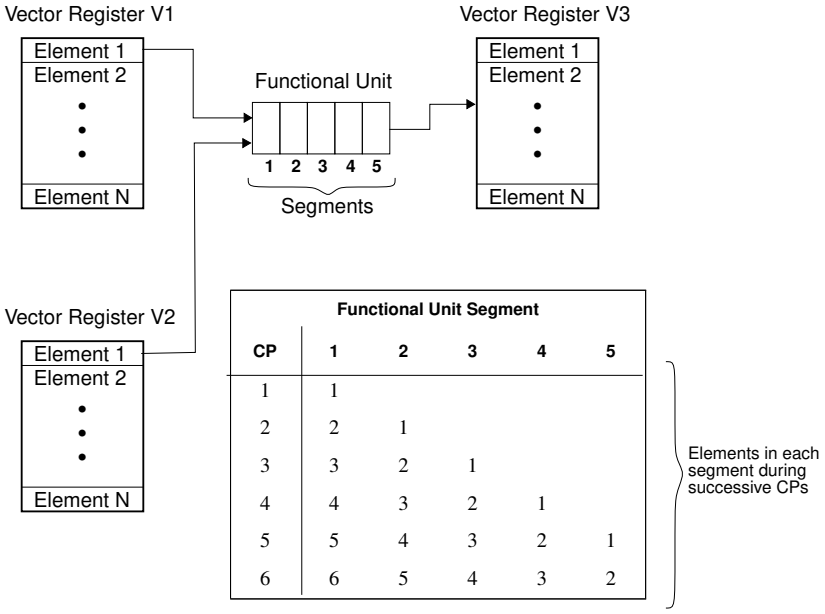
Mangelnde Nutzung der “inneren” Prozessorparallelität



© Claus-Uwe Linster – rrze.kol97: 19/2/97

Mangelnde Nutzung der “inneren” Prozessorparallelität

- “Breite” Daten-Verarbeitung (mit “breiten” Instruktionsströmen kombiniert) ein Vektor von Daten wird synchron in gleicher Weise bearbeitet



Functional Unit Segment

CP	1	2	3	4	5
1	1				
2	2	1			
3	3	2	1		
4	4	3	2	1	
5	5	4	3	2	1
6	6	5	4	3	2

Elements in each segment during successive CPs

© Claus-Uwe Linster – rrze.kol97: 19/2/97

Mangelnde Nutzung der “inneren” Prozessorparallelität

- die “innere” Parallelität von Prozessoren kann nur dann genutzt werden, wenn das Programm “zufällig” eine geeignete Befehlssequenz zuläßt:
  - beeinflußt von der gegenseitigen Abhängigkeit von Daten in sequentiellen Befehlsfolgen,
  - abhängig von der “Verschiedenheit” der Befehle in einem gewissen lokalen Umfeld, um die vorhandenen funktionalen Einheiten parallel zu nutzen
  - abhängig von der Qualität der Compiler die Möglichkeiten der lokalen Parallelisierung zu erkennen
  - abhängig von der Qualität des Programmcodes, Sachverhalte, für einen Compiler entscheidbar, dazustellen; ggf. Direktiven
  - und last not least auch programmiersprachenabhängig (z. B. Pointer)
  - Generell gilt:
    Je weiter Taktraten und Peakleistung auseinanderklaffen - also “innere” Parallelität vorhanden ist -, desto geringer ist die tatsächliche “Ausbeute” an Leistung.

© Claus-Uwe Linster – rrze.kol97: 19/2/97

4.2 Mangelnde Nutzung der Prozessor-/Speichermechanismen, Speicherzugriffslatenzzeiten zu verbergen, sowie ...

- Durch die techn. Entwicklung bedingt, stieg die Verarbeitungsgeschwindigkeit der Prozessoren stärker, als die der (zu einem vertretbaren Preis herzustellenden) Arbeitsspeicher.
- Seit Mitte der 70er Jahre: Prozessoren (Flops) ca. 1 : 10.000
  Speicherzyklus ca. 1 : 10

■ Abhilfe: Erweiterung der Speicherbandbreite durch

- Parallelisierung des Zugriffs
  - Speicherbänke
  - Interleaving (verschränkter Speicherzugriff)
  - Daten-Transfer in “Burst Mode” (einmal Adresse, viele Daten)
  - Verbreiterung der Datenpfade (mit Memory Banking)
  - Einführung von Schaltnetzwerken und damit potentiell vielen parallelen Zugriffspfaden

Diese technischen Verbesserungen sind weitgehend transparent und kommen jedem Programm zugute.

© Claus-Uwe Linster – rrze.kol97: 19/2/97

Mangelnde Nutzung der Prozessor-/Speichermechanismen, Speicherzugriffslatenzzeiten zu verbergen, sowie ...

■ Abhilfe: Erweiterung der Speicherbandbreite durch (cont)

- erhöhtem techn. Einsatz
  - Erhöhung der Übertragungsleistung durch höhere Taktraten auf dem Bus;
    Problem: Je schneller der Bus, desto “kürzer” der phys. Aufbau;
  - Einführung von Speicherhierachien
    Cache: on-node; on-board; on-chip
  - Verbesserte Cache-Kohärenz-Protokolle (write through; copy back)
  - Vergrößerung der Cache-Speicher (kann kontraproduktiv wirken).

Um diese Architekturspezifika optimal zu nutzen muß

- die Anwendungen ein den Cashgrößen und -techniken adäquates lokales Verhalten aufweisen und.....

© Claus-Uwe Linster – rrze.kol97: 19/2/97

Mangelnde Nutzung der Prozessor-/Speichermechanismen, Speicherzugriffslatenzzeiten zu verbergen, sowie ...

- Um diese Architekturspezifika optimal zu nutzen sollte(cont)
  - ....das Betriebssystem
    - durch seine Scheduling-Politik die Lokalität des Prozesses unterstützen (*hot-cache-reusing*).
    - Statistik über prozeßspezifisches Lokalitätsverhalten erheben und darauf abstützend
    - *“page-recolouring”* - Techniken einsetzen:

      Seiten, die die gleichen Cachebereiche benutzen, erhalten die gleiche “Farbe”; das Betriebssystem organisiert einen möglichst bunten *“workingset”*.

© Claus-Uwe Linster – rrze.kol97: 19/2/97

Mangelnde Nutzung der Prozessor-/Speichermechanismen, Speicherzugriffslatenzzeiten zu verbergen, sowie ...

- Der technologische Aufwand bedingt wiederum komplexere “Speicher-anschaltungen”, die sich aufgrund räumlicher Ausdehnung nicht beliebig hoch takten lassen (Einschwingverhalten) und deren Funktionalität sich nicht mit “einem” Takt realisieren läßt. Die Zeit, die zwischen dem Anstoß eines Zugriffs auf den Speicher und dem Eintreffen des “ersten” Datums vergeht, nennt man Latenzzeit.
  - Diese Latenzzeit ist in der Regel höher als die Zykluszeit des Speichers.
  - Heute typische Latenzzeiten auf den Zugriff lokaler Speicher beträgt etwa 50 Prozessorzyklen (etwa: 250 - 400 ns); das ist deutlich länger als der Zugriff eines Einzelprozessors auf seinen Speicher, bei vergleichbarer Technologie!

© Claus-Uwe Linster – rrze.kol97: 19/2/97

Mangelnde Nutzung der Prozessor-/Speichermechanismen, Speicherzugriffslatenzzeiten zu verbergen, sowie ...

- Maßnahmen zur Verdeckung der Latenzzeit *(Latencyhiding)*
  - ***Prefetching*;** d.h. rechtzeitiges Anstoßen eines Zugriffs; also nicht *“on demand”.*
  - Die Möglichkeit mehrere *Prefetchings* anstoßen zu können: ***oustanding loads***
  - Die Möglichkeit einer dynamischen Umordnung der Befehlsausführungs-reihenfolge: ***instructionreordering*;** d.h. es wird das ausgeführt, was gerade ausführbar ist.

Hier gelten teilweise die gleichen Argumente, die auch bei der Nutzung der inneren Parallelität eines Prozessors angeführt wurden, wie

- abhängig von der gegenseitigen Abhängigkeit von Daten in sequentiellen Befehlen
- abhängig von der Qualität der Compiler die Möglichkeiten der lokalen Parallelisierung - hier Prefetching - zu erkennen
- abhängig von der Qualität des Programmcodes, Sachverhalte, für einen Compiler entscheidbar, darzustellen; ggf. Direktiven; Programmierspr.

© Claus-Uwe Linster – rrze.kol97: 19/2/97

4.3 Verbindungsnetzwerkebene

- Aufgrund der technisch sehr aufwendigen (und kostenintensiven) Speichersysteme, insbesondere für Rechner mit sehr großen Prozessorzahlen, favorisierte man - insbesondere in den 80er Jahren - Lösungen mit verteilten lokalen Speichern; NORMA - Architekturen.
- Wunsch: Vollständiges Verbindungsnetzwerk; Komplexität steigt mit K!
  - Abhilfe:
    - Man beschränkt sich auf eine “gewisse” Nachbarschaft, die man direkt erreichen kann, wobei man sich auf Eigenschaften numerischer Lösungsverfahren stützt.
      Problem: globale Kommunikation / Synchronisation
    - globale Kommunikation wird mehrstufig mit - möglichst autarken, nicht die Rechenleistung einzelner Knoten belastenden - Routern realisiert.
- Heute vielfach Crossbar-Switches, bei größeren Systemen in hierachischer Form (IBM-SP2; Fujitsu VPP300/700)

© Claus-Uwe Linster – rrze.kol97: 19/2/97

Verbindungsnetzwerkebene

- Workstationcluster
  - gebildet mit Hochleistungs-Vernetzungskomponenten; spezielle Switches; z. B. IBM SP-2
  - gebildet auf der Basis von Standardvernetzungskomponenten; 100MBit Ethernet, ATM.
- Prognose für Systeme mit verteilten lokalen Speichern:
  - aufgrund der immer schneller werden Standard-Netzwerk-Komponenten, werden Spezialrechner aus Entwicklungskostengründen ihre Bedeutung verlieren; universell einsetzbare “Workstationcluster” werden sich hier durchsetzen.
  - Typische Vertreter: IBM-SP2; TMC mit Sun-Systemen
  - Problem: Geeignete Systemsoftware zur Administration und Betrieb solcher Systeme: Es gibt kaum mehr als NQS (Network-Queing-System).

© Claus-Uwe Linster – rrze.kol97: 19/2/97

4.4 Message Passing vs. Shared Memory

- Message Passing Architekturen
  - Programmiermodell/Kommunikationsmodell ist ’message passing’
  - Kein globaler Adreßraum; also NORMA-Architekturen NORMA = NO Random Memory Access
  - Das Verhältnis Befehlsausführungszeit/Latenzzeit zwischen Aktivitätsträgern(Threads, Prozesse) auf verschiedene Prozessoren liegt bestenfalls bei 1 : 10.000 (IBM SP-2); Hinzu kommt ein erheblicher Protokoll - Overhead
  - Eine ’shared memory’-Programmiermodell kann als ’shared distributed memory’ simuliert werden, ist aber extrem ineffizient. (’paging’ über ’message passing’); Beispiel: Intel Paragon

© Claus-Uwe Linster – rrze.kol97: 19/2/97

Message Passing vs. Shared Memory

- Shared Memory Architekturen
  - Programmiermodell/Kommunikationsmodell ist 'shared memory'
  - Globaler Adreßraum; also UMA/NUMA-Architekturen
    UMA = Uniform Memory Access
    NUMA = Non Uniform Memory Access
  - Das Verhältnis Befehlsausführungszeit/Latenzzeit zwischen Aktivitätsträgern(Threads, Prozesse) auf verschiedene Prozessoren reduziert sich auf Speicherzugriffe, bei NUMA-Architekturen hängt die Zeit von der "Entfernung" ab.
    Beispiel Convex SPP1600: 1 : 1 Cache
    1 : 40 Nodespeicher
    1 : 200 Internodespeicher
  - Ein 'message passing'-Programmiermodell läßt sich effizient auf einem 'shared memory'-System simulieren. ('message passing' über 'shared data'-Segmente)

© Claus-Uwe Linster – rrze.kol97: 19/2/97

Message Passing vs. Shared Memory

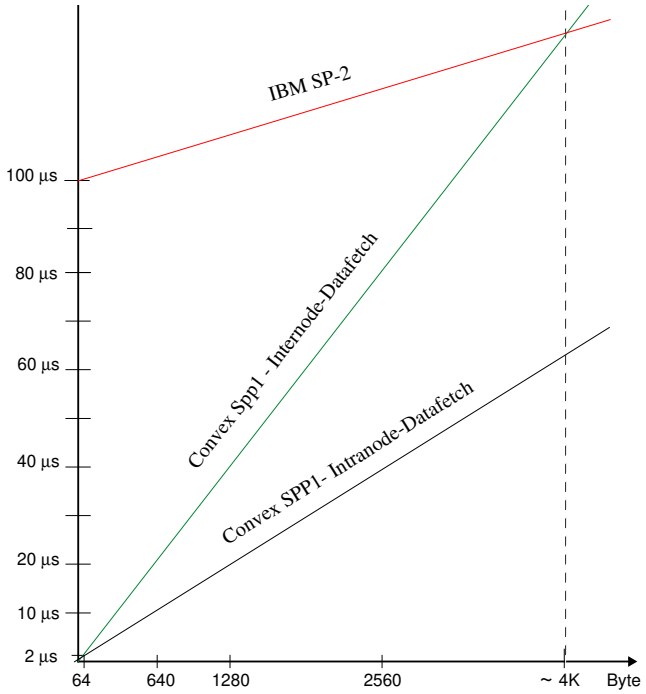
- Schlußfolgerung:
  - Abhängig von der Latenzzeit muß der Synchronisationsabstand (Datenaustausch oder Koordinierung) hinreichend groß gewählt werden.
  - Große Abstände, also grobgranularere Parallelität, führen in der Regel zu geringerer Parallelität der Anwendung; es sei denn die Anwendungs"größe" selbst skaliert ideal.
  - Grobgranulare Parallelität ist geeignet für beide Architekturklassen
  - Feingranulare Parallelität ist nur geeignet für 'shared memory'-Systeme
- Wann ist ein Programm grob-/feingranular?
  - Versuch einer Quantifizierung am Beispiel
    IBM SP-2/Convex SPP1600:

© Claus-Uwe Linster – rrze.kol97: 19/2/97

Message Passing vs. Shared Memory

- Qualitativer Vergleich

- IBM SP-2
  per Prozessor:
  40 µs Latenz +
  60 µs Protokoll-Overhead
  Transfereinheit “beliebig” groß
- Convex SPP1600
  per Node = 8 Prozessoren:
  2 µs Latenz +
  0 µs Protokoll-Overhead
  Transferblock 64 Byte



© Claus-Uwe Linster – rrze.kol97: 19/2/97

4.5 Der mangelnden Parallelitätsgrad der Anwendung selbst...

- *Speedup* (im Vergleich zum Ablauf des gleichen Programms auf einem Prozessor)

$$s(p) = \frac{t(1)}{t(p)}$$

- Effizienz (im Vergleich zum Ablauf des gleichen Programms auf einem Prozessor)

$$e(p) = \frac{t(1) \times 100}{p \times t(p)}[Prozent]$$

- Eine andere Variante der Betrachtung wäre den *Speedup* bzw. die Effizienz durch “Vergrößerung” des Problems zu bestimmen.

© Claus-Uwe Linster – rrze.kol97: 19/2/97

Der mangelnden Parallelitätsgrad der Anwendung selbst...

- Amdahls Gesetz

$$t(1) = t_s + (1 - t_s) = t_s + t_p$$

$$t(p) = t_s + \frac{1 - t_s}{p}$$

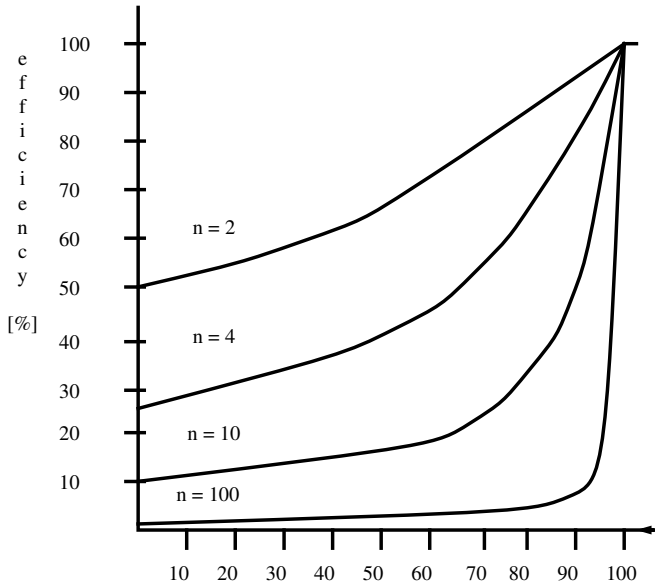
$$s(p) = \frac{1}{t_s + \frac{1 - t_s}{p}} \leq \frac{1}{t_s}$$

d.h. der maximale Speedup bzw. die max. Effiziens sind durch den sequentiellen Anteil (wesentlich) beschränkt.

- Der algorithmische Mehr-/Minderaufwand der durch die Parallelisierung des Problems selbst entsteht, wird in der Regel nicht berücksichtigt.

© Claus-Uwe Linster – rrze.kol97: 19/2/97

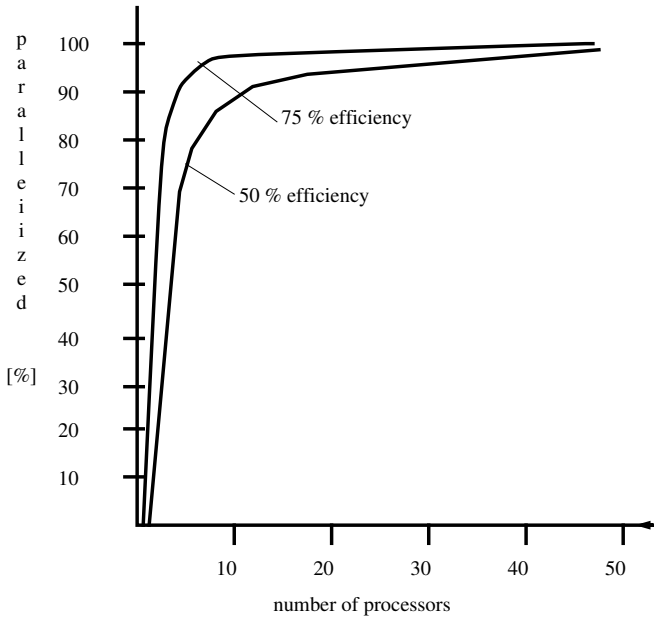
Der mangelnden Parallelitätsgrad der Anwendung selbst...



n = Anzahl der Prozessoren/HW-Komponenten

© Claus-Uwe Linster – rrze.kol97: 19/2/97

Der mangelnden Parallelitätsgrad der Anwendung selbst...



© Claus-Uwe Linster – rrze.kol97: 19/2/97

5 Leistungszahlen

- Weitere Leistungsmaße versuchen mehr eine Gesamteindruck zu vermitteln, sie geben jedoch immer nur spezifische Teilaspekte wieder. Beispiele:
  - LINPACK
  - LAPACK
  - Specmarks
  - LADDIS (NFS)
  - Xmarks
  - OPEN GL
  - TPC (Transaktionsleistung)
- Amerikaner lieben ’rankinglists’; beliebt ist die TOP 500 Liste, die auf der Grundlage des Linpack-Benchmarks aufgestellt wird; Erlangen befindet sich z Zt. auf Platz 233.
- Es soll Firmen geben, die Prozessoren (und Systeme) z. B. auf den Linpack-Test optimieren.

© Claus-Uwe Linster – rrze.kol97: 19/2/97

6 Programmiermodelle

- Programm- und datenparalleles Programmiermodell
  - Shared Memory Programmiermodell
  - Message Passing Programmiermodell
- Datenparalleles Programmiermodell oder
  Single Programm Multiple Data - SPMD
  - Shared Memory
  - Message Passing
- Parameterparalleles Programmiermodell
- ▲ Unabhängig vom Programmiermodell muß man beim Programmieren paralleler Systeme ein Optimum folgender Eigenschaften erreichen:
  - Minimierungen der Kommunikation
  - Maximierung des “lokalen” Verhaltens
  - Verteilung der Daten so, daß die Rechenlast auf allen Prozessoren möglichst gleich ist *(loadbalancing)*.

© Claus-Uwe Linster – rrze.kol97: 19/2/97

6.1 Shared Memory Programmiermodell

- Kommunikation über gemeinsame Speicherbereiche
- Explizite Koordinierung
- Aktivitätsträger sind Prozesse und/oder Threads
- Keine Unterstützung durch spezielle Sprachkonstrukte
- Keine “automatische” Parallelisierung
- Unterstützung durch Thread-Libraries
- Erzwingung von Prozessorzuordnungen nicht immer gegeben
- Scheduling des Betriebssystems unterstützt in der Regel Lokalitätsverhalten
- Geeignet insbesondere für UMA- und NUMA-Architekturen
- Feingranulare Parallelität möglich
- Programmiersprachen: C; C++
- Schwierige Fehlersuche
- Wenig Werkzeuge für paralleles Debuggen
- Geeignet, hoch effiziente (architekturspezifische) Programme zu schreiben

© Claus-Uwe Linster – rrze.kol97: 19/2/97

6.2 Message Passing Programmiermodell

- Kommunikation über *send/receive*-Mechanismen
- Explizite Koordinierung; implizit realisiert über *send/receive*-Mechanismen
- Aktivitätsträger sind Prozesse (lokal sind Threads möglich)
- Keine Unterstützung durch spezielle Sprachkonstrukte
- Keine “automatische” Parallelisierung
- Unterstützung durch Message Passing-Libraries: (PVM; MPI)
- Feste Zuordnung Prozeß/Prozessor, keine Migration von Prozessen möglich
- Geeignet für alle Rechnerarchitekturklassen
- Geeignet für grobgranulare Parallelität; Grad ist abhängig von verwendeter Rechnerarchitektur (*Latency* des Verbindungsnetzwerks).
- Programmiersprachen: C; C++; Fortran
- Schwierige Fehlersuche, wenig Werkzeuge für paralleles Debuggen
- Geeignet portable Programme für alle denkbaren Architekturen zu schreiben, wobei die Effizienz stark von der rechnerarchitektur-spezifischen, sinnvollen parallelen Granularität abhängt.

© Claus-Uwe Linster – rrze.kol97: 19/2/97

6.3 Datenparalleles Programmiermodell

- Shared Memory - Single Programm Multiple Data - SPMD

- Implizite Kommunikation, realisiert über gemeinsame Speicherbereiche
- Implizite Koordinierung
- Aktivitätsträger sind Prozesse mit identischer Kontrollflußstruktur; dynamisch verschiedener Kontrollfluß möglich.
- Unterstützung der parallelen Programmierung durch spezielle Sprachkonstrukte; typischer Weise “Feldverarbeitung”.
- Automatische Parallelisierung; Unterstützungsmöglichkeit durch Compilerdirektiven
- Erzwingung von Prozessorzuordnungen nicht immer gegeben
- Scheduling des Betriebssystems unterstützt in der Regel Lokalitätsverhalten.

© Claus-Uwe Linster – rrze.kol97: 19/2/97

Datenparalleles Programmiermodell

- Shared Memory - Single Programm Multiple Data - SPMD (cont)
  - Geeignet insbesondere für UMA- und NUMA-Architekturen
  - Feingranulare Parallelität möglich
  - Programmiersprachen: Fortran90/95
  - Einfachere Fehlersuche
  - Parallele Debugger einfacher realisierbar; da nur **ein** Kontrollfluß
  - Geeignet, portable Programme für UMA - und NUMA - Architekturen und potentiell auch für Vektorrechner zu schreiben, wobei die Effizienz vom maschinenspezifischen Granularitätsgrad und dem Lokalitätsverhalten; insbesondere bei NUMA-Maschinen, abhängt.

© Claus-Uwe Linster – rrze.kol97: 19/2/97

Datenparalleles Programmiermodell

- Message Passing - Single Programm Multiple Data - SPMD
  - Implizite Kommunikation, realisiert über *send/receive*-Mechanismen
  - Implizite Koordinierung; realisiert über *send/receive*-Mechanismen
  - Aktivitätsträger sind Prozesse mit identischer Kontrollflußstruktur; dynamisch verschiedener Kontrollfluß möglich
  - Unterstützung der parallelen Programmierung durch spezielle Sprachkonstrukte; typischer Weise “Feldverarbeitung”.
  - Automatische Parallelisierung; Unterstützungsmöglichkeit durch Compilerdirektiven
  - Feste Zuordnung Prozeß/Prozessor
  - Keine Migration von Prozessen möglich (und sinnvoll).

© Claus-Uwe Linster – rrze.kol97: 19/2/97

Datenparalleles Programmiermodell

- Message Passing - Single Programm Multiple Data - SPMD (cont)
  - Geeignet für alle Rechnerarchitekturklassen
  - Geeignet für grobgranulare Parallelität; Grad ist abhängig von verwendeter Rechnerarchitektur.
  - Programmiersprachen: High Performance Fortran - HPF
  - Einfachere Fehlersuche
  - Parallele Debugger einfacher realisierbar; da nur **ein** Kontrollfluß
  - Geeignet portable Programme für alle denkbaren Architekturen zu schreiben, wobei die Effizienz stark von der rechnerarchitektur-spezifischen, sinnvollen parallelen Granularität abhängt.

© Claus-Uwe Linster – rrze.kol97: 19/2/97

6.4 Parameterparalleles Programmiermodell

- Keine dynamische Kommunikation zwischen den Komponenten des parallelen Systems.
- Master/Slave - Konfiguration, wobei der Masterprozeß an die sonst unabhängigen Slaveprozesse Parametersätze verteilt und ggf. die Ergebnisse wieder einsammelt und auswertet.
- Im einfachsten Fall ist der Masterprozeß eine Shellprozedur.

© Claus-Uwe Linster – rrze.kol97: 19/2/97

6.5 Zusammenfassung

	nutzbare Parallelität	Programmier-sprachen	Bibliotheken	Compiler-unterstütz-ung	Synchroni-sation	Daten-austausch
Vektor-rechner	Datenparallel	C, (C++) Fortran90		automatische Vektorisierung	implizit	keiner
MIMD-NORMA	Datenparallel	HPF		Datenvertei-lung Feldverarbei-tung	implizit	implizit (send/receive)
MIMD-NORMA	Datenparallel und Programm-parallel	C, (C++) Fortran	PVM MPI	keine	explizit	explizit (send/receive)
MIMD-UMA/ NUMA	Datenparallel	HPF Fortran90/95		Datenvertei-lung (HPF) Feldverarbei-tung	implizit	implizit
MIMD-UMA/ NUMA	Datenparallel und Programm-parallel	C, C++	p-threads (PVM) (MPI)	keine	explizit	explizit

© Claus-Uwe Linster – rrze.kol97: 19/2/97

7 Ausbildungsangebot des IMMD IV/RRZE

Programmierung Paralleler Systeme

- als Gesamtveranstaltung für Informatiker gedacht, jedoch modular derart aufgebaut, daß man sich spezielle Kombinationen von Sprachen, Programmiermodell und Architekturen individuell zusammenstellen kann.
- Speziallkurse (=Untermengen) werden nach Bedarf und bei genügender Teilnehmerzahl angeboten.
- Näheres: http://www4.informatik.uni-erlangen.de/Lehre/SS97/V_PPS.shtml
- auf der gleichen Seite finden Sie den Eingang zu 2 WWW-Kursen der Universität Liverpool zum Selbststudium oder zur Vertiefung der Vorlesung:
  - Fortran90
  - High Performance Fortran (HPF)

Computational Engineering (Computer-Ingenieurwesen)

- wahrscheinlich schon ab Herbst 1997

© Claus-Uwe Linster – rrze.kol97: 19/2/97

Ausbildungsangebot des IMMD IV/RRZE

Programmierung Paralleler Systeme - Inhaltsübersicht

1. Einleitung und Überblick

1.1 Architekturprinzipien und Programmiermodelle

1.2 Rechnernetze: Infrastruktur für Paralleles Rechnen

2. Architektur der Convex SPP-1600

3. Programmiermodell “Message Passing”

3.1 PVM - “Parallel Virtuell Maschine”

3.2 PVM-Übungen

3.3 MPI - “Message Passing Interface”

3.4 MPI-Übungen

4.0 Datenparalleles Programmiermodell - SPMD

4.1 Fortran95

4.2 HPF - High Performance Fortran

4.3 Übungen: Shared Memory/Workstationcluster

4.4 Architektur der Connection Machine 5 (CM5)

© Claus-Uwe Linster – rrze.kol97: 19/2/97

Ausbildungsangebot des IMMD IV/RRZE

Programmierung Paralleler Systeme - Inhaltsübersicht (cont)

4.5 C-Star

4.6 Übungen C-Star/CM5

5.0 Programmiermodell “Shared Memory”

5.1 Sun Enterprise Architektur

5.2 Prozesse, Threads, Koordinierung

5.3 p-threads

5.4 Übungen “Shared Memory”

6.0 Programmierung von Vektorrechnern *)

6.1 Architektur Fujitsu 300/700

6.2 Besonderheiten bei der Programmierung von Vektorrechnern

6.3 Übungen “Vektorrechnen”

*) in Vorbereitung

© Claus-Uwe Linster – rrze.kol97: 19/2/97