Grundlagen: Ebenen der Parallelität

Bitebene/Wortbreite – Befehlsphasen (Pipelining) – Maschinenbefehle (Fes) – Anweisungen/Schleifen – Daten(strukturen) – Prozesse/Tasks/Threads – Jobs/Benutzerprogramme

Entwurf:

Partitionierung – Aufteilen des Problems in atomare Aufgaben/Tasks

Kommunikation – Informationsfluss und –struktur zwischen den Aufgaben

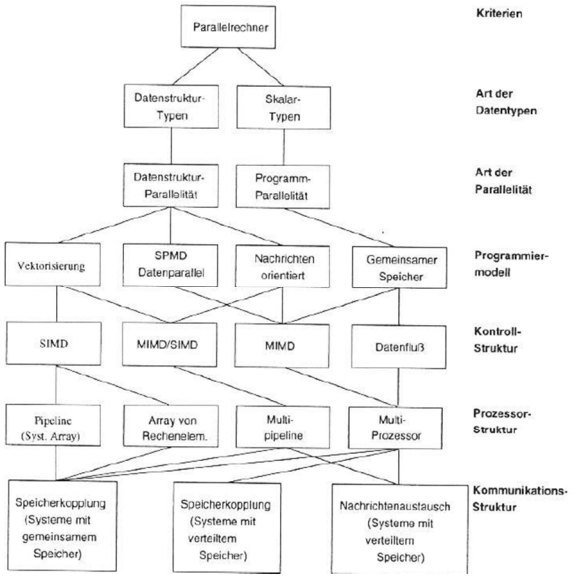
Agglomeration – Leistungsbewertung und evtl. Zusammenfassung von Aufgaben

Mapping – Abbilden der Aufgaben auf Prozessoren

Klassifikation:

	Single Instruction	Multiple Instruction
Single Data	SISD – Sequentiell	MISD – Praktisch nie umgesetzt
Multiple Data	SIMD – MP-1 von MassPar	MIMD – Intel Paragon

Giloi: Informations und Steuerstruktur



State of the Art:

TOP500 als Datenbasis, erster Exaflop 2019?

Mooresches: Transistorzahl auf Chip aller 1,5 Jahre verdoppelt

Cluster und MPP vorherrschend, Intel und Industry

Bell's Law: Alle 10 Jahre eine neue „Computing-Plattform"

Forschung -> Einführung/Reife -> Anwendung -> Ausklingen

Programmiermodelle:

Gemeinsamer Speicher -> Konkurrenzsituation der Prozesse

Parallelisierende Compiler -> Datenflussanalyse, FORTRAN/C

Open Multi-Processing (OpenMP) -> fork/join, handelt Multithreading, eng gekoppelte Systeme, Ind.-St.-API

Nutzt Pragmas zum Kennzeichnen von Parallelität

FORTRESS -> FORTRAN neu, mathematische Orientierung, implizite Schleifenparall. und explizite Parallelisierung mögl.

Verteilter Speicher:

MPI (Message Passing Interface): Nebenläufig nachrichtenbasiert, definiert API für Kommunikation, verschiedene Implementierungen, z.B. Cluster. Ab MPI-2 dynamische Erzeugung von Prozessen möglich, auch für verteilte Speichersysteme, Barrieren zur Synchronisation

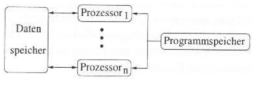
TCP/IP -> Kooperativ nachrichtenbasiert, Kooperation = Dienstleistungsverhältnis bsp. Server-Client. Lokalisierung des Partners notwendig, statisches o. dynamisches Binden. Nutzung von Sockets mit API-Funktionen zum synchronisierten Senden/Empfangen. De-Facto-Standard, bidirektional. Versch. Socket-Typen (Datagram, Stream, Raw, Packet)

(Kooperative Modelle mit entfernten Aufrufen -> scheinbar zentral, Remote Calls)

GPGPU: OpenCL, CUDA, C++ AMP

Architekturen: (Pipelining/superskalare Prozessoren, SIMD; Datenflussrechner, MIMD, GPU)

SIMD -> Verarbeitung geordneter Datenmengen

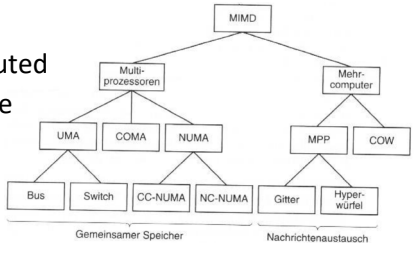


Feld/Array-Rechner vs. Vektorrechner, einfache Programmierung, dafür teure HW

Heute wenig verbreitet, vor allem in Spzialprozessoren (Grafik ‚Streaming SIMD')

Besteht aus Array Control Unit und Feld aus Processing Elements mit Speicher.

MIMD -> Multithreaded Architecture, Simultaneous Multithreading, Cluster, Grid, Multicore-CPUs, eng oder lose gekoppelte Multi-Prozessorsysteme)



Wichtig: physikalische Speicheranordnung – shared Memory vs. Distributed

UMA -> gleichförmige Zugriffslatenz, Netzwerk als Bottleneck, begrenzte

Anzahl von Prozessoren möglich. Einfache Programmierung. Multicore als Sonderfall: Hierarchisch, Pipeline oder Netzwerkbasiert. Homogen oder Heterogen.

	physikalischer Speicher gemeinsam	physikalischer Speicher verteilt
logischer Adressraum gemeinsam	Shared Memory Systeme UMA Multithreaded Architectures, eng gekoppelte Multi-prozessoren, SMP, Multicore-Prozessoren	Distributed *Shared* Mem. Systeme (Mischform) NUMA • NCC-NUMA, • CC-NUMA, • COMA lose gekoppelte Multi-prozessoren
logischer Adressraum verteilt	(leer)	Distributed Memory Syst. NORMA massiv parallele Systeme, Multicomputer, Cluster, Grid

SMT -> zusätzliche Pipelines und Registersätze, aber gemeinsamer Speicher, Bus und Cache. Mehrere Threads pro Prozessor => Parallelität auf Thread-Ebene. Kontextwechsel nach Befehl oder nach Block. Spezialhardware. Verbergen von Latenzen durch Speicherzugriffe

NUMA -> Non-Uniform Memory Access, keine gleichförmige Latenz private Speichermodule der Proz., virtueller Adressraum (DSM). Bessere Skalierung, Unterscheidung nach Caching. NCC-NUMA: Non Cache Coherent = lokaler Cache für lokale Daten, Speicherzugriffe auf anderen Prozessor immer über Netzwerk. CC-NUMA = HW-Unterstützung für Cache Kohärenz, Zustände der Cache-Zeilen in DB gespeichert. Lokales Cachen entfernter Daten. Schneller, aber Fehler bei zu großen Daten. COMA (Cache Only Memory Architecture) = Gesamter Speicher als Cache genutzt. Cachezeilen werden durch gesamtes System bewegt und Zugriffe werden entsprechend umgeleitet

NORMA -> No Remote Memory Access, kein direkter Zugriff auf entfernten Speicher. Kein DSM. Bspw. Mehrrechnersysteme. Keine Konkurrenz bei Zugriffen, Sehr hohe Skalierbarkeit, aber Alle Kommunikation via message passing über Netzwerk. MPP, Cluster und Grid als Beispiele.

Betriebssysteme:

Erweiterte BS mit: Scheduling, Speicherverwaltung, verteilte Dateisysteme, Lastverteilung.

Monolithisch oder Mikrokern-basiert, objektorientiert (modular, klare APIs).

Mehrstufig: Verteilte Umgebung, Netzwerk-BS, Verteiltes BS.

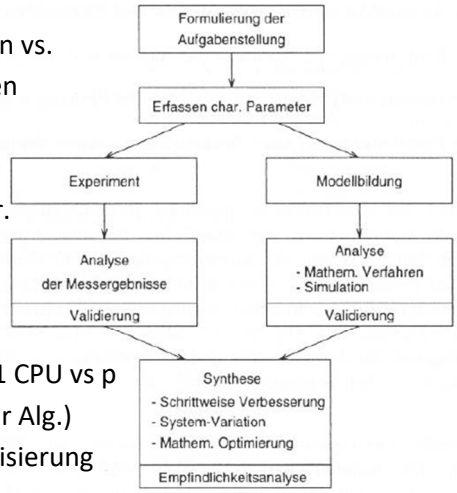
Betriebsarten: 1 BS je Prozessor/Rechner, 1 BS für alle Prozessoren, BS auf eigenem Frontend

Aktivitätsträger (Threads) zuordnen, user-level vs kernel threads

Koordinierung von Zugriffen, bspw. mit Sperrsynchronisation Ereignissynchronisation, mess. Passing

Speicherverwaltung, abhängig von DSM o. shared mamory o. distributed memory

Leistungsbewertung:



Globale u. lokale Leistungsmaße: Durchsatz, Antwortzeiten, Auslastungen vs. Zeitanteil für Verwaltung, Speicherzugriffe, Analyse von Unterbrechungen

$T_{CPU}(A) = n_c(A) * t_c$ Rechenzeit für Programm A

MIPS- (Million Instructions per Second) und FLOPS-Rate (Floating Point Operations per Second) als grobes Maß für Geschwindigkeit, meist theor.

Benchmarks (Synthetisch, Spielzeug, Programmkerne, komplette Progr.)

Es fehlen PR-Parameter wie Grad der Parallelität, Komm.-verhalten

$Tp(n) = T_{CPU} + T_{COM} + T_{WAIT} + T_{SYN} + T_{PLACE} + T_{START}$

Speedup: $S_P(n)=T_S/T_P$ Sequentielle Laufzeit durch parallele Laufzeit oder 1 CPU vs p

$S_P(n)>p$ = superlinearer Speedup (Caching-Effekte, implizite Änderung der Alg.)

Effizienz: $E_P(n)=S_P(n)/p$ Speedup je Prozessor, meist in %, Eff. der Parallelisierung

Amdahl: f=nicht-parallelisierbarer Anteil des Programms.

$T_P(n)\geq f*T_S(n) + (1-f)*T_P(n)/p$ (Zeit für sequentiellen Teil + Zeit für parallelen Teil)

$S_P(n)=p/(1+f*(p-1))$ Begrenzung des maximalen Speedups! f=10% -> $S_p(n)=10$!

Für $p\to\infty$ gilt näherungsweise: $S_P(n)\leq 1/f$ => Massive Parallelität nur für Probleme, die parallel sind…

Gustafons Gesetz: $S_P(n)=p*(1-f_1)+f_1$ (Mit mehr CPUs steigt n und sinkt f, $S_P(n)$ nähert sich p an.