Parallelrechner und Parallelprogrammierung am Karlsruher Institut für Technologie

Maximilian Heß

September 2017

Inhaltsverzeichnis

1 Parallelrechner und Parallelprogrammierung

Zusammenfassung der Vorlesung „Parallelrechner und Parallelprogrammierung“ aus dem Sommersemester 2017.[1]

1.1 Einführung

- **Klassifikation nach Flynn**

  Single Instruction Single Data (SISD): von-Neumann-Architektur (Einprozessorrechner)

  Single Instruction Multiple Data (SIMD): Vektorrechner

  Multiple Instruction Single Data (MISD): In der Praxis irrelevant. Ausnahme: Mehrere Geräte, die zur Berechnungsverifikation das selbe Datum mehrfach parallel berechnen

  Multiple Instruction Multiple Data (MIMD): Multiprozessorsystem

- **Multiprozessorsysteme**

  Speichergekoppelter: Gemeinsamer Adresseraum; Kommunikation über gemeinsame Variablen; skalieren mit > 1000 Prozessoren

  Uniform Memory Access Model (UMA): Alle Prozessoren greifen gleichermaßen mit gleicher Zugriffszeit auf einen gemeinsamen Speicher zu (symmetrische Multiprozessoren)

  Non-uniform Memory Access Modell (NUMA): Speicherzugriffszeiten variieren, da Speicher physikalisch auf verschiedene Prozessoren verteilt ist (Distributed Shared Memory System (DSM))

  Nachrichtengekoppelt: Lokale Adresseräume; Kommunikation über Nachrichten (No-remote Memory Access Model); theoretisch unbegrenzte Skalierung

  Uniform Communication Architecture Model (UCA): Einheitliche Nachrichtenübertragungszeit

  Nin-uniform Communication Architecture Model (NUCA): Unterschiedliche Nachrichtenübertragungszeiten in Abhängigkeit der beteiligten Prozessoren

1.2 Parallelrechner

- **Leistungsfähigkeit**
  - Maßzahl: Floating Point Operations per Second (FLOPS)
  - Nicht notwendigerweise protportional zur Taktgeschwindkeit: Eventuell mehrere Zyklen zur Berechnung notwendig; Vektorprozessoren verarbeiten viele Floating Point Operations gleichzeitig (Grafikkarten)
  - Messstandard: `LINPACK`-Benchmark

1.2.1 Shared Memory Multiprozessoren

- Programmierung durch effiziente automatische Parallelisierungswerkzeuge einfach und attraktiv
- **Cache-Kohärenz**
  - Problem: Replikate in Caches verschiedener Prozessoren müssen kohärent gehalten werden (Lesezugriffe müssen immer den Wert des zeitlich letzten Schreibzugriffs liefern)
  - Lösung: Verzicht auf strikte Konsistenz. Replikate müssen nicht zu jedem Zeitpunkt identisch sein

1.2.2 Distributed Memory Multiprozessoren

- Aufbau: Rechenknoten mit (mehreren) CPU(s), lokalem Speicher und ggf. lokaler Festplatte. Kopplung über Verbindungsnetzwerk
- Kriterien: Geschwindkeit, Parallelitätsgrad, Kosten, Latenz

IBM RS/6000 SP (1990-2000)

- Nachrichtengekoppelter Multiprozessor mit $2-512$ superskalaren, 64-bit `POWER`-Knoten (`POWER3-II` weist $1,5 GFLOPS$ Spitzenleistung auf)
- Verbindungsstruktur: High-Performance-Omega-Netzwerk (4×4) bidirektionale Kreuzschinen
- Skalierbares IO-System über FileServer-Knoten
- Betriebssystem: `IBM AIX`
- **Aufbau**
  - Frames: Jeweils 16 Knoten mit reduntanter Stromversorgung/Steuerungslogik sowie Hochleistungsnetzwerk
  - High-Performance Switch

[1] `https://www.scc.kit.edu/personen/11185.php`

    - Pro Frame ein 16×16 Schaltnetzwerk, besteht aus 4×4 Kreuzschinenschalter, zur Kommunikation mit anderen Frames
    - *Buffered Wormhole Routing*: Bei Kollisionen in den Verbindungselementen bleiben Nachrichten stehen und blockieren nachfolgende Datenpakete)

IBM Power 4 in der IBM p690 eSeries 1600 Cluster

- `POWER4`-**Prozessor**
  - Erster Dual-Core Prozessor mit gemeinsamen `L2`-Cache und $10,4$ GFLOPS
  - *Fabric Controller* zum Zusammenschalten von mehreren Chips → *Multi-Chip-Module* (MCM)
    - interne Kommunikation über unidirektionalen Ring mit 40 GBit/s
    - Gemeinsames Memory-Subsystem für $8 - 32$ GB
  - *Central Electronics Complex* (CEC) zum Zusammenschalten von bis zu 4 MCMs und bis zu 8 Speicherkarten
- **Logical Partitions (LPARs)**
  - Partitionierung von Ressourcen in physikalische Maschinen
  - Volle Flexibilität bei voller Isolation; jede Kopie führt `AIX/Linux` aus
- Speicherkonfiguration pro Rack: Bis zu 8 Speicherslots
- Beispiel: Juelich-Multi-Processor (JUMP) mit 41 `p690+` Schränken

1.2.3 Distributed Shared Memory Multiprozessoren

- **Überblick**
  - Gemeinsamer Adressraum, einzelne Speichermodule sind auf die einzelnen Prozessoren verteilt
  - Oft Cache-Kohärenz bei (lokalem) Speicherzugriffen
- **Zugriff**
  - Mikrobefehlsebene
    - Transparent für das Maschinenprogramm
    - Explizite Befehle für entfernten Speicherzugriff
  - Software DSM
    - Jeder Prozessor kann immer auf gemeinsame Daten zugreifen. Synchronisation mittels Schloss,- Semaphor- oder Bedigungsvariablen
    - DSM-System regelt Kommunikation selbständig über (zumeist) Message-Passing
    - Vorteile: Entlastung des Programmierers; leichte Partierbarkeit von/zu eng gekoppelten Multiprozessoren
    - Datenverwaltung

      Seitenbasiert: Nutzung der virtuellen Speicherverwaltung des Betriebssystems zur expliziten Platzierung der Daten (unterschiedliche Granularität der Seiten: 16 Byte bis 8 kByte)

      Objektbasiert:
    - Probleme bei seitenbasierter Datenverwaltung
      - Geringe Effizienz beim Nachladen über das Verbindungsnetz
      - Lineares, strukturloses Feld von Speicherwort
      - False Sharing und Flattern (Trashing)
        - False Sharing: Eine Speicherseite beinhaltet mehrere Datenwörter, die von verschiedenen Prozessoren benötigt werden (Kohärenz auf Seitenebene) → nach jedem Schreizugriff eines Datenwortes muss die komplette Seite neu zu den anderen Prozessoren übertragen werden
        - Flattern (Trashing): Bei mehrfachen Schreibzugriffen wird die Seite immer wieder übertragen
        - Gegenmaßnahmen
          - Verkleinerung der Seitengröße. Allerdings steigt damit der Seitenverwaltungsaufwand
          - Objektbasiertes Software SDM-System: Gemeinsame Variablenzugriffe werden vom Precompiler erkannt und durch Bibliotheksfunktionen für entfernte Zugriffe ersetzt → es werden nur Datenobjekte verschoben, die benötigt werden → *False Sharing* wird ausgeschlossen
    - Datenlokation und Datenzugriff
      - Jeder Knoten muss Daten finden können (Datenlokation) und darauf zugreifen (Datenzugriff)
      - Statische Verwaltung der Daten
        - Daten werden zentral auf einem oder mehreren Servern verwaltet (Verteilung wird nicht verändert)
        - Datenlokalisierung funktionsbasiert
        - Konsistenz durch Sequentialisierung auf dem zuständigen Server
      - Dynamische Verwaltung der Daten
        - Datum wird vor Zugriff auf zugreifenden Knoten verschoben → alle Zugriffe sind lokal (single-reader-single-writer-Konzept)

        - *False Sharing* bei seitenbasiertem System
        - Konsistenz durch Verschieben der Seiten
    - Replizierte Daten
        - Bisher: Knoten können nur sequentiell auf Daten zugreifen
        - Replikation ähnelt Caching

          Lesereplikation: Reine Lesekopie, kann nicht geändert und zurückgeschrieben werden

          Leseanfrage: Falls vorhanden, Verwerfen einer Schreibkopie; danach Anfordern einer neuen Lesekopie

          Schreibanfrage: Verwerfen aller Kopien; danach Anlegen der Schreibkopie so wie zurückschreiben
        - Volle Replikation: multiple readers/multiple writers
            - Jeder Knoten kann lokal Änderungen vornehmen → Konsistenz schwierig
            - Ansatz potentiell am effizientesten, da alle Zugriffe lokal sind

Beispielsystem: Cray T3E

- NCC-NUMA-Konzept: Puffern von Cache-Blöcken nur bei prozessorlokalen Speicherzugriffen (keine Kohärenz über alle Cache-Speicher)
- Erster Supercomputer, der > 1 TFLOPS bei einer wissenschaftlichen Anwendung erzielte (1998)
- Verwendung von $8 - 2176$ Alpha-Mikroprozessoren mit 3D-Torus-Nutzwerk
- **Adressierung**
  - Aufbau einer globalen Adresse: Verarbeitungselementnummer (PE-Nummer) und lokaler Offset
  - Adressumsetzung in Hardware (virtuelle → logische → physische Adresse)
- **Verbindugnsnetzwerk**
  - Dreidimensionaler Torus (dreidimensionales, ringförmig geschlossenes Gitter)[2]
  - Daten können gleichzeitig über separate Kommunikationspfade ohne Beteiligung der Verarbeitungselemente (unabhängige Hardware-Unterstützung für den Nachrichtenaustausch) in alle drei Richtungen transportiert werden → kurze Verbindungswege, schnelle Übertragung
- **Mechanismen zur Synchronisation**

  Barrier-Synchronisation: Barrier-Modus (`UND`-Verknüpfung) und Eureka-Modus (`ODER`-Verknüpfung)

  Fetch-and-Increment: Atomares Lesen eines Werts und Inkrementieren eines speziellen lokalen Registers

  Atomic-Swap: Atomares Tauschen von lokalem Registerinhalt mit dem Inhalt einer (entfernten) Speicherzelle
- Betriebssystem: Microkernel pro Verarbeitungselement

Beispielsystem: SGI Altix (LRZ)

- CC-NUMA auf Basis von Intel Itanium Dual-Cores mit insgesamt 9728 cores (Platz 10, 06/2007)
- Leistung: $56,5$ TFLOS (Linpack) bei ca. 1 MW Energiebedarf
- Nachfolgesystem `SuperMUC`: $3,2$ PFLOPS bei 2 MW Energiebedarf

1.2.4 Cluster Systeme

- Hintergrund:Zu Beginn häufig Spezialprozessoren in Supercomputern (Consumermarktentwicklung noch zu langsam). Später vermehrt Nutzung von `x86`-Standardprozessoren
- **Charakterisierung**
  - Jeder Knoten als eigenes System mit eigenem OS
  - Nachrichtenaustausch früher: Interprozesskommunikation über Netzwerk
  - Nachrichtenaustausch heute: Cluster sind durch Multicore Prozessoren hybride Systeme
    - Gemeinsamer Speicher in einem Knoten (OpenMP)
    - Verteilter Speicher zwischen den Knoten (MPI)
    - Somit zwei Parallelitätsebenen

[2] `https://en.wikipedia.org/wiki/Torus_interconnect`

HPC Cluster am KIT

- **HC3: HP XC3000 (seit Februar 2010)**
  - 356 Rechenknoten mit jeweils zwei `Intel Xeon E5540` Quad Core (27, 04 TFLOPS Spitzenleistung) und insgesamt ≈ 10 TB Hauptspeicher
  - OS: `HP XC Linux`
  - Infiniband Netzwerk
  - Nutzung durch KIT Institute
- **InstitutsCluster 2 (IC2)**
  - 6560 Xeon-Cores mit 135, 7 TFLOPS Spitzenleistung und ≈ 28 TB Hauptspeicher
  - OS: `SLES 11` mit `KITE` Managementsoftware
  - Infiniband Netzwerk
  - Nutzung durch KIT Institute
- **bwUniCluster**
  - Gemeinsames System aller Lindesuniversitäten Baden Württembergs
  - Systemarchitektur: `64bit x86` MultiCore-Prozessoren (sandy bridge); Infiniband FDR (56 GBit/s)
  - Kennzahlen Cluster: 8448 Cores mit 175 TFLOPS Spitzenleistung bei ≈ 200 KW Energieverbrauch
  - Kennzahlen Speichersystem: 900 TB Speicher bei ≈ 20 KW Energieverbauch
- **ForHLR Stufe 1**
  - Landeshöchstleistungsrechner: System der Ebene Tier-2; Rechenzeitvergabe nach wissenschaftlichem Begutachtungsprozess
  - 528 Xeon Prozessoren mit insgesamt 10752 Cores, ≈ 41 TB Hauptspeicher und 232 TFLOPS Spitzenleistung bei einem Energieverbauch von 226 KW
  - Infiniband 4X FDR
  - Benchmarks

    Top500: 243 (06/2014)

    Green500: 69 (06/2014)
- **ForHLR Stufe 2**
  - 1152 HPC-Knoten mit jeweils zwei DecaCore `Intel Xeon E5-2660v3`, 64 GB Hauptspeicher und 480 GB SSD
  - Spitzenleistung: 24.048 Cores mit 95 TB Hauptspeicher und 1, 1 PFLOPS Spitzenleistung
  - OS: `RHEL 7.x` mit `KITE` Managementsoftware
  - Heißwasserkühlung mit ≈ 40° Grad Vorlauftemperatur und > 45° Grad Rücklauftemperatur → freie Kühlung selbst bei tropischen Temperaturen
  - Visualisierungskomponenten
    - 21 Visualisierungsknoten mit jeweils 4 Dodeka-Core `Intel Xeon E7-4830v3`, 1 TB Hauptspeicher, 4 960 GB SSDs und 4 `NVIDIA GeForce GTX980 Ti`
    - Neues Visualisierungslabor mit zwei `4K` Projektoren
  - Dateisystem und Integration
    - 4, 8 PB bei 80 GB/s Durchsatz
    - Kopplung der Dateisysteme von `ForHLR I` (Campus Süd) und `ForHLR II` (Campus Nord) über redundante 320 GBit/s Infiniband-Leitung (35 km Laufstrecke)

Jülich Reasearch on Petaflop Architectures (JUROPA)

- **Aufbau**
  - `Sun Blade 6048 System` mit 2208 Rechenknoten, 17664 Cores und ≈ 52 TB Hauptspeicher
  - Pro Rechenknoten: Zwei `Intel Xeon X5570` QuadCore mit 24 GB Hauptspeicher
  - Gesamtleistung: 207 TFLOPS Spitzenleistung
  - Netzwerk: Infiniband QDR (non-blocking Fat Tree)
- **Schwestersystem**
  - 8640 Cores mit 101 FTLOPS Spitzenleistung
  - Exklusiv für europäische Fusions-Wissenschaften
- Besonderheiten der Systeme: Unterschiedliche Hersteller aber mit gleicher/kompatibler Hardware. Werden gemeinsam Entwickelt und können als ein Gesamtsystem betrieben werden

1.2.5 Vektorrechner

- Rechner mit pipelineartig aufgebautem Rechenwerk zur Verarbeitung von Vektoren von Gleitkommazahlen
- Skalarverarbeitung: Verknüpfung von Vektoren mit einzelnen Operanden
- **Beispiel: Vektor-Addition in vierstufiger Pipeline**

  Stufe 1: Laden von Gleitkommazahlen aus dem Vektorregister

  Stufe 2: Exponenten vergleichen und Mantisse verschieben

  Stufe 3: Verschieben der ausgerichteten Mantisse

  Stufe 4: Ergebnis normalisieren und zurückschreiben

- *Chaining*: Verketten von (spezialisierten) Pipelines. Ergebnisse der vorderen Pipeline werden direkt der nächsten Pipeline zur Verfügung gestellt
- **Möglichkeiten zur Parallelisierung**

  Vektor-Pipeline-Parallelität: Durch die Pipeline selbst gegeben

  Mehrere Pipelines pro Vektoreinheit: Parallelität durch *Chaining* von Pipelines

  Vervielfachen der Pipelines: Verwendung mehrerer parallel arbeitender, gleichartiger Pipelines

  Mehrere Vektoreinheiten: Mehrere unabhängige Vektoreinheiten

- Vektorrechner heutzutage: Keine Relevanz mehr, da Standardprozessoren `SIMD`-Operationen effizient ausführen können

NEC SX Serie

- **Beispielsystem am SCC (NEC SX-9)**
  - Ein `SX-9` Knoten enthält 16 Vektorprozessoren (mit jeweils 8 Vektorpipelines) mit 1 TB Hauptspeicher und $1,6$ TFLOPS Spitzenleistung
  - Sehr gut geeignet für Strömungsberechnungen
  - Größtmögliches System: 512 Knoten mit 970 TFLOPS Spitzenleistung
- **Weitere `SX-9` Systeme**
  - HLRS Stuttgart: 12 Knoten mit $19,2$ TFLOPS Spitzenleistung
  - Der Deutsche Wetterdienst besitze zwei unabhängige Systeme mit jeweils 14 Knoten und 23 TFLOPS Spitzenleistung

1.2.6 Ressourcenmanagement

- Typischerweise Knappheit von Parallelrechnerressourcen → Ressourcenmanagement/Job Scheduling notwendig
- **Job Scheduling**

Time-Sharing: Simultane Ausführung mehrerer Jobs auf der gleichen Ressource nach einem *Round-Robin*-Verfahren. Anwendung bei PCs

Space-Sharing: Jobs bekommen exklusiv Ressourcen zur Ausführung zugewiesen, müssen allerdings ggf. warten bis genügend Ressourcen frei sind (Batch-System). Laufzeiten müssen abgeschätzt werden, meist gibt es Obergrenzen. Anwendung bei Clustern/Supercomputern

- **Beispiel: LoadLeveler**
  - Job Scheduler von `IBM` zum Ausführen von Batch-Jobs
  - Grundlegende Befehle zum Hinzufügen/Anzeigen/Abbrechen/Status anzeigen von Jobs
  - Hinzufügen von Jobs per Job-Kommando-Datei (*cmdfile*)
- **Beispiel: Torque (Resource Manager) + Maui (Cluster Scheduler)**
  - Grundlegende Befehle zum Hinzufügen/Anzeigen/Abbrechen/Status anzeigen von Jobs
  - Hinzufügen neuer Jobs per Kommandozeile
  - Beispiel: `msub -l nodes=32:ppn=2,pmem=1800mb,walltime=3600 myscript`

1.3 Programmiermodelle und Grundlagen

1.3.1 Parallelitätsebenen/Parallelitätstechniken

- **Ebenen der Parallelität**
  1. Programmebene (Jobebene): Beispielsweise gleichzeitig, von einander unabhängige Programme auf einem Betriebssystem
  2. Prozessebene (Taskebene): Beispielsweise Linux-Prozesse eines Programms, die auf gemeinsamen Daten arbeiten
  3. Blockebene (Threadebene): Beispielsweise parallel ausgeführte Schleifeniteration
  4. Anweisungsebene: Beispielsweise einzelne Maschinenbefehle, die parallel ausgeführt werden (beispielsweise bei superskalaren Prozessoren oder VLIW)

5. Suboperationsebene: Beispielsweise Vektorbefehle in einer Vektropipeline; setzt vektorisierende Compiler und komplexe Datenstrukturen voraus

- **Körnigkeit/Granularität**
  - Verhältnis von Rechenaufwand zu Kommunikations-/Synchronisationsaufwand
  - Klassifizierung

    Grobkörnig: Programm-, Prozess- und Blockebene

    Mittelkörnig: Blockebene, jedoch selten verwendet

    Feinkörnig: Anweisungsebene

    Noch feinkörniger: Supoperationsebene

1.3.2 Maschinenmodelle

- **Random-Access Machine (RAM)**
  - Einprozessorrechner mit jeweils einmal: Recheneinheit, Programm, RW-Speicher, Eingabeband, Ausgabeband
  - Befehle mit Zielregister und zwei Operandenregistern

Parallel Random-Access Machine (PRAM)

- Modell eines idealen speichergekoppelten Parallelrechners. Keine Beachtung von Synchronisations-/Zugriffskosten
- `n`-Prozessor-PRAMs bestehen aus `n` identischen Prozessoren, mit gemeinsamen Speicher/Takt
- Praktische Bedeutung: PRAMs idealisieren Parallelrechner (s.o.) ohne Beachtung der Lokalität des global adressierbaren Speichers (Zugriff aller Prozessoren auf jede Speicherzelle immer in Einheitszeit)
- **Zugriffskonflikte**
  - Typen von Zugriffskonflikten

    Exclusive Read (ER): Pro Zyklus kann höchstens ein Prozessor die selbe Speicherzelle lesen

    Exclusive Write (EW): Pro Zyklus kann höchstens ein Prozessor die selbe Speicherzelle beschreiben

    Concurrent Read (CR): Pro Zyklus können mehrere Prozessoren die selbe Speicherzelle lesen

    Concurrent Write (CW): Pro Zyklus können mehrere Prozessoren die selbe Speicherzelle beschreiben → Schreibkonflikte müssen gelöst werden
  - Kombinationen von Zugriffskonflikten

    EREW-PRAM: Keine gemeinsamen Zugriffe

    CWER-PRAM: Gemeinsames Lesen erlaubt, gemeinsames Schreiben verboten

    ERCW-PRAM: Exklusiver Lesezugriff, gemeinsamer Schreibzugriff

    CRCW-PRAM: Gemeinsame Lese-/Schreibzugriffe erlaubt
  - Lösen von Schreibkonflikten

    Common (C-CRCW): Gemeinsamer Schreibzugriff nur erlaubt, wenn alle Prozessoren den selben Wert schreiben wollen

    Arbitrary (A-CRCW): Ein Prozessor gewinnt, alle anderen werden ignoriert

    Priority (P-CRCW): Der Prozessor mit dem kleinsten Index gewinnt
- Zusammenfassung: Speichergekoppelt; taktsynchron; gut zu programmieren

Bulk Synchronous Parallel (BSP)

- Nachrichtengekoppeltes Maschinenmodell
- Aufbau: Prozessoren (mit oder ohne lokalem Speicher); Kommunikationsnetz; Barrierensynchronisation
- Definiert über Anzahl der Prozessoren p, Kommunikationskostenfaktor g und die minimale Zeit für die Ausführung eines Superschrittes L
- *Super-Step* Betrieb
  - Sequentielle Ausführung von Superschritten. Pro Superschritt arbeiten die Prozessoren unabhängig von einander
  - Bestandteile pro Superstep
    * Berechnungsschritte auf lokalen Variablen, die zu Beginn des Superschritts zur Verfügung stehen
    * Senden/Empfangen von Nachrichten
    * Am Ende: Barrierensynchronisation aller Prozessoren vor dem nächsten Superschritt
- Zusammenfassung: Nahrichtengekoppelt, Superschritt-Synchronisation

LogP-Modell

- Prozessoren arbeiten unabhängig und tauschen Nachrichten aus
- **Definition einer Maschine**

  L: Maximale Latenz einer Übertragung einer kurzen Nachricht (weniger Wörter umfassend)

  o: Zeitbedarf zum Übertragen der Nachricht. Keine weitere Befehlsausführung während Senden oder Empfangen

  g: Untere Zeitschranke, die pro Prozessor zwischen zwei Übertragungen eingehalten werden muss

  P: Anzahl Prozessoren mit optionalem lokalen Speicher
- Endliche Netzwerkbandbreite: Es können immer nur $\frac{L}{g}$ Nachricht zeitgleich ausgetauscht werden
- L ist obere Schranke pro Nachrichtenübertragung aufzufassen
- Zusammenfassung: Nachrichtengekoppelt; explizite Kommunikation; geeignet für Laufzeitabschätzungen

1.3.3 Quantitative Maßzahlen für Parallelrechner

- Prozessorzustände während Programmausführung: *rechnend*, *kommunizierend*, *wartend* (auf Kommunikation)
- **Laufzeitmessung von $T(1)$**

  Relative Beschleunigung (Algorithmenabhängig): Paralleler Algorithmus wird auf Multiprozessorsystem so ausgeführt, als sei er sequentiell

  Absolute Beschleunigung (Algorithmenunabhängig): Verwenden des besten bekannten sequentiellen Algorithmus
- **Skalierbarkeit**
  - Definition: Das Hinzufügen von Verarbeitungselementen führt zur Verkürzung der Ausführungszeit, ohne dass das Programm geändert wird
  - Annahme: Konstante Problemgröße
  - Skalierung ist immer beschränkt: Sättigung ab einer bestimmten Prozessorzahl
  - Skaliert ein System, so tritt das Problem bei einer wachsenden Problemgröße nicht auf
- **Amdahls Gesetz**
  - Aufteilung eines Programms in einen parallelen und einen sequentiellen Anteil. Speedup-Berechnung unter der Annahme, das kein Programm komplett parallelisiert werden kann
  - Es gilt: $1 \leq S(p) \leq p$
  - Superlinearer Speedup: $S(p) > p$
    - Tritt nur selten auf, sorgt für Konfusion
    - Gründe: Cache-Effekte durch die Speicherhierarchie; mit steigender Prozessorzahl steigt wächst die Cache-Hierarchie → Reduzierung der Hauptspeicherzugriffe → kürzere Laufzeit bei paralleler Ausführung

1.3.4 Prozesse und Threads

- Parallelität ist eine spezielle Form von Nebenläufigkeit. Nebenläufige Anweisungen sind nur dann parallel, wenn zur Ausführung mehrere Prozessoren zur Verfügung stehen
- **Definitionen**

  Prozessumgebung: Geschützter Adressbereich eines Prozesses mit individuellem Code-/Datenbereich im Speicher → Prozesswechsel bedingt somit (i.d.R.) einen Umgebungswechsel

  Prozesskontext: Registerwerte des Prozessors während der Prozessausführung. Beeinhaltet u.a. Befehlszähler, Stackpointer, etc. → Prozesswechsel bedingt immer auch einen Kontextwechsel
- Laufzeitbetrachtung: Erheblicher Aufwand durch Prozesserzeugung; Zugriffsschutzmechanismen (wenn Prozesse mehrerer Benutzer ausgeführt werden); Prozesswechsel; Kommunikations-/Synchronisationsoperationen
- **Prozesstypen**

  Schwergewichtige Prozesse (Prozesse): Klassische Prozesse

  Leichtgewichtige Prozesse (Threads): Mehrere *Threads* teilen sich eine gemeinsame Prozessumgebung (Adressraum, Filepointer, geschützten Speicherbereich, etc.). Threadwechsel erzeugen keinen Umgebungswechsel, lediglich einen Kontextwechsel

1.3.5 Synchronisation und Kommunikation über gemeinsame Variablen

- **Synchronisation**
  - Koordination zwischen Prozessen/Threads
  - Einseitige Synchronisation: Abhängige Aktivitäten müssen in der richtigen Reichenfolge ausgeführt werden
  - Mehrseitige Synchronisation: Zeitliche Reihenfolge der Ausführung unerheblich, Ausführung darf allerdings nicht parallel erfolgen (gemeinsamer Ressourcenzugriff) → *Mutual Exclusion* (gegenseitiger Ausschluss)
  - *Critical Section*: Anweisungen, deren Ausführung einen gegenseitigen Ausschluss erfordern
- **Deadlocks (Verklemmungen)**
  - Deadlock: Prozesse warten auf Ereignisse, die nicht mehr eintreten können → Prozess wird undefinierbar verzögert (Aussperrung)
  - Voraussetzungen für Deadlocks
    1. Mutual Excklusion: Betriebsmittel sind nur exklusiv nutzbar
    2. No Preemption: Betriebsmittel können nicht entzogen werden
    3. Hold and Wait: Es werden Betriebsmittel gehalten während auf andere Betriebsmittel gewartet wird
    4. Circular Wait: Zirkuläre Abhängigkeit von Betriebsmittel von mindestens zwei Prozessen
  - Vermeidung von Deadlocks
    * Schlossvariable
      · Abstrakte Datenstruktur, auf die alle Prozesse zugreifen müssen, die einen kritischen Abschnitt betreten wollen
      · Operationen
      lock: Verschließt den kritischen Abschnitt von innen. Prozesse warten bis der kritische Abschnitt frei ist. Ist selbst ein kritischer Abschnitt → Implementierung über atomare Befehle (`TEST_AND_SET`, `RESET`) oder der spezielle Maschineninstruktion `swap` (tauscht den Inhalt eines Registers und eines Speicherplatzes ohne untebrochen werden zu können). *Spinlock* zum zyklischen Prüfen, ob ein Bereich wieder frei ist (ebenfalls per `swap`)
      unlock: Gibt den Bereich wieder frei
    * Semaphor
      · Aufbau: Nicht-negative Integer-Variable; atomare Operationen „passieren“ und „verlassen“
      · Initialisierung über maximale Anzahl Prozesse, die eine Ressource zeitgleich nutzen dürfen. Bei Betreten wird der interne Zähler dekrementiert, beim Verlassen inkrementiert. Ist der Zähler Null, so muss auf Freigabe durch einen anderen Prozess gewartet werden
      · Spezialfall: `new Semaphore(1)` wird als binäre Semaphore/Mutex Lock bezeichnet
      · Nachteil: Kann bei fehlerhafter Implementierung (Vergessen von *Verlassen*) das ganze System lahmlegen
      · Unterschied zu *Schlossvariable*: *Verlassen* kann von allen Prozessen/Threads aufgerufen werden
    * Kritischer Abschnitt
      · Zu jedem Zeitpunkt darf sich höchstens ein Prozess/Thread im kritischen Abschnitt befinden
      · Bedingter kritischer Abschnitt: Kritischer Abschnitt kann nur betreten werden, wenn die Bedingung wahr ist (Implementierung von Fairness/Prioritäten/etc.)
      · Wird der kritische Bereich freigegeben, bewerben sich alle wartenden Prozesse (bei denen die Bedingung wahr ist) um den kritischen Bereich
    * Monitor
      · Zugriff exklsuiv einem Prozess vorbehalten. Abstrahiert die Zugriffsfunktionen/Realisierung des gegenseitigen Ausschlusses
      · Im Monitor vereinbarte Variablen sind exklusiv und können nie gleichzeitig benutzt werden
      · Innerhalb des Monitors haben Funktionen ledigleich Zugriff auf Monitorvariablen
      · Auf monitorgebundene Variablen kann von außen nicht zugegriffen werden
      · *Verlassen* kann nur innerhalb des Monitors aufgerufen werden
      · Vorteil gegenüber Semaphor: Kann nicht durch weitere hinzugekommene Anwenderprozesse gestört werden, wenn einmal korrekt programmiert
      · Vorteil gegenüber bedingtem kritischen Abschnitt: Kann nicht nur Befehlsfolgen sondern auch Funktionen enthalten
    * Barriere
      · Synchronisationspunkt für mehrere Prozesse
      · Verwendung
        · Initialisierung von Barrier-Variable mit Anzahl der Prozesse, auf die gewartet werden muss
        · `wait-barrier` muss von jedem dieser Prozesse ausgeführt werden. Dabei wird der Zahler der Barriere dekrementiert bis sie Null ist
      · Verbindungsstruktur von Parallelrechner werden zur Implementierung sehr schneller Barriere-Sync-Operationen genutzt

1.3.6 Kommunikation über Nachrichten

- Verbrauchbarkeit: Nachrichten existieren nur für die Zeit des Übertragungsvorgangs
- Sequentialbeziehung (implizite Synchronisation): Nachrichten können erst nach dem Senden empfangen werden
- Bestandteile einer Nachticht: Ziel; Nummer; Speicherbereich, der verschickt werden soll; Anzahl und Datentyp der Datenelemente im Sendepuffer
- Sendeoptionen: Asynchron/Synchron; Gepuffert/Ungepuffert; Blockierend/Unblockierend
- Empfangsoptionen: Konsumierend (zerstörend)/konservierend; Asynchron/Synchron
- Adressierungsarten: Direkte Bennenung; Briefkasten; Port (global bekannt); Verbindung/Kanal (lokal in Prozessen vereinbart)

1.3.7 Parallele Programmstrukturen

- **Fork-Join Modell**
  - Prozesse werden zu Beginn eines parallelen Codeteils durch `fork`-Operationen erzeugt
  - Diese arbeiten unabhängig weiter und erreichen am Ende eine `join`-Operation zur Synchronisation mit Erzeugerthread (`T1`)
  - Nachteil: Aufwand zur Threaderzeugung
- **SPMD Modell**
  - Ziel: Vermeidung des Aufwands zur Thread-Ezeugung → Threads werden nur einmal beim Programmstart erzeugt und terminieren erst am Programmende
  - Sequentielle Codeteile werden von allen Prozessen ausgeführt; parallele individuell
- **Reusable-Thread-Pool Modell**
  - Idee: Kombinieren von Fork-Join Modell (Vermeiden der Mehrfachgenerierung der Prozesse) und SPMD Modell (Vermeidung der Mehrfachausführung der sequentiellen Codeteilen)
  - Threads werden nur einmal erzeugt (wenn sie benötigt werden) und „schlafen gelegt“, wenn sie gerade nicht gebraucht werden → sequentieller Code wird nur einmal ausgeführt

1.4 Appendix A: Formelsammung

1.4.1 Parallelrechner

Ende-zu-Ende Übertragungszeit

Overhead: Zeit, um Nachrichten in/aus dem Netzwerk zu bekommen

$$latency = t_{overhead} + t_{routing} + t_{transmission} + t_{contention} \tag{1.1}$$

1.4.2 Programmiermodelle und Grundlagen

Ausführungszeit

$$t = t_{computaton} + t_{communication} + t_{idle} \tag{1.2}$$

Übertragungszeit einer Nachricht

$$t_{msg} = t_{startup} + n \cdot \left(t_{transfer}\right) \tag{1.3}$$

Beschleunigung (Speedup)

$$S(p) = \frac{T(1)}{T(p)} \tag{1.4}$$

Effizienz

$$E(p) = \frac{S(p)}{p} \tag{1.5}$$

Amdahls Gesetz

$$T(p) = \underbrace{T(1) \cdot \frac{q}{p}}_{Paralleler\ Anteil} + \underbrace{T(1) \cdot \left(1 - q\right)}_{Sequentieller\ Anteil} \tag{1.6}$$

$$S(p) = \frac{T(1)}{T(p)} = \frac{T(1)}{T(1) \cdot \frac{q}{p} + T(1) \cdot \left(1 - q\right)} = \frac{1}{\frac{q}{p} + \left(1 - q\right)} \leq \frac{1}{1 - q} \tag{1.7}$$