**Übungszettel 1**

**Aufgabe 1:**

1. Nennen Sie drei unterschiedliche Einsatzbereiche für Parallelisierung.
2. Was sind die Voraussetzungen für „Parallel Processing“?
3. Erklären Sie den Begriff „Speedup“.

**Aufgabe 2:**

Erläutern Sie zwei Gemeinsamkeiten und Unterschiede der TM, RAM und des v. Neumann Konzepts.

**Aufgabe 3:**

Betrachte das folgende Programm 1.

Programm 1

Read 0

JGTZ gr\_0

Write =1

Halt

gr\_0 Store 1

Sub =1

Store 2

JGTZ loop

Write =1

Halt

loop Load 1

Mult 2

Store 1

Load 2

Sub =1

Store 2

JGTZ loop

Write 1

Halt

1. Welche Berechnung wird mit dem Programm durchgeführt und welches Konzept steckt dahinter?
2. Wie sieht der Verlauf der Register aus, wenn das Programm aufgerufen wird?

**Lösungen zum Übungszettel 1**

**Aufgabe 1:**

1. Meteorologie (Wettervorhersage), Medizin (MRT), Biologie (Gene Mapping)
2. Das Problem muss auf Daten- und/oder Prozess-Ebene aufteilbar sein.
3. „Speedup“ ist der Quotient aus serieller und paralleler Ausführungszeit. Er gibt den Geschwindigkeitszuwachs an, der an der seriellen Ausführung gemessen wird.

**Aufgabe 2:**

Unterschiede:

* V. Neumann Rechner einziger real umgesetzter Rechner
* TM besitzt Lese- und Schreibkopf auf einem I/O-Band, RAM besitzt einen Lesekopf für ein I-Band und einen Schreibkopf für das O-Band.

Gemeinsamkeiten:

* Grundmodelle zur Berechenbarkeit
* Alle besitzen eine serielle Ausführung
* Ein Programm wird benötigt

**Aufgabe 3:**

1. Das Programm berechnet für eine vorgegebene Zahl die Fakultät.  
   n! = n \* (n – 1) \* (n – 2) \* … \* 1  
   Das Konzept entspricht der RM, Registermaschine.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **R0** | 3 | 3 | 2 | 2 | 3 | 6 | 6 | 2 | 1 | 1 | 6 | 6 | 6 | 1 | 0 | 0 | ***1*** |
| **R1** | 0 | 3 | 3 | 3 | 3 | 3 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | ***6*** |
| **R2** | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | ***1*** |

**Übungszettel 2**

**Aufgabe 1:**

1. Welche Thesen zum Speedup wurden in der Vorlesung vorgestellt?
2. Wie sind die Thesen bzgl. des Speedup Faktors einzuordnen?
3. Geben Sie Möglichkeiten an, die Thesen zu erweitern.

**Aufgabe 2:**

Das Gesetz von *Amdahl* beschäftigt sich mit Aussagen über die parallelen und sequenziellen Anteile eines Programms. Hieraus lässt sich der bereits besprochene Speedup herleiten, den ein Programm durch eine mögliche Parallelisierung maximal erzielen kann.

1. Warum kann die Ausführungszeit eines Programms durch Hinzufügen von *N* Prozessoren nicht beliebig verringert werden?
2. Welcher ist momentan der wesentlichste Faktor, der die lineare Skalierung bezüglich der Ausführungszeit verhindert?
3. Nennen Sie drei unterschiedliche Interconnect-Varianten bei Parallelrechnern. (Die Antworten können sich auf die Topologie als auch die verwendete Technik beziehen.)

**Aufgabe 3:**

Die Taxonomie von *Flynn* teilt Rechnerarchitekturen in vier Klassen ein:

**SISD / MISD / SIMD / MIMD**

1. Was bedeuten die vier Akronyme?
2. Falls möglich, finden Sie zu jeder Klasse zwei real existierende Beispiele.
3. Gibt es noch andere Taxonomien für Parallelrechner? Wenn ja, welche?

**Lösungen zum Übungszettel 2**

**Aufgabe 1:**

1. Wilke’sche paradoxon: Superlinearer Speedup  
   Minsky:   
   Amdahl:
2. Amdahl: Kommunikationsaufwand und Speichergeschwindigkeit mit berechnen

**Aufgabe 2:**

1. Der Speedup wird durch den Anteil des seriellen Codes im Programm irgendwann gehemmt.
2. Speichergeschwindigkeit und Zugriffszeit ist bisher noch das Bottleneck.
3. Tightly coupled (eng gekoppelt)  
   Loosely coupled (lose gekoppelt)  
   Hybrid

**Aufgabe 3:**

1. **SISD**: Single Instruction, Single Data   
   **SIMD**: Single Instruction, Multiple Data  
   **MISD**: Multiple Instruction, Single Data   
   **MIMD**: Multiple Instruction, Multiple Data
2. **SISD**: PC (v. Neumann Rechner)  
   **SIMD**: Vektorrechner, PRAM  
   **MISD**: Schachcomputer, Shuttlecomputer (Redundante Rechnung)  
   **MIMD**: Cray-1 (80 Mflops), Cray XK7 (1 Petaflops, CPU+GPU), usw.
3. Hardware level Support (Klassen):  
   Klassifizierung aufgrund der Methode, die durch die Hardware technisch unterstützt wird (Multicore, Symmetric multiprocessing, Distributed computing, Cluster, Massive parallel, Grid, GPGPU, Circuits, Vector processor, usw.)

**Übungszettel 3**

In dieser Übung sollen einige Grundlagen zur parallelen Programmierung in der Programmiersprache Java kennengelernt und ausprobiert werden.

**Aufgabe 1:**

1. Moderne Betriebssysteme unterstützen die Nebenläufigkeit in Prozessen und Threads. Beschreiben Sie den Unterschied, Vorteil und Nachteil von Prozessen und Threads im Kontext der Programmiersprache Java.
2. Programmieren Sie eine kleine Java-Anwendung mit folgenden Schritten:  
   - 10 Threads werden in einer „for“-Schleife mit der „Thread“-Klasse gestartet  
   - jeder Thread gibt zu Beginn der Ausführung den Text „Start“ + Threadname aus  
   - jeder Thread wartet anschließend für 5 Sekunden  
   - jeder Thread gibt zum Schluss den Text „End“ + Threadname aus
3. Starten Sie das Programm und beschreiben Sie die Ausgabe. Was fällt Ihnen auf?

**Aufgabe 2:**

1. Neben den einfachen Threads bietet die Java Concurrency-API weitere Möglichkeiten zur parallelen Programmierung. Beschreiben Sie was ein „Thread-Pool“ ist und welche Vorteile er gegenüber normalen Threads bietet.
2. Machen Sie sich mich dem „ExecutorService“ in Java vertraut und programmieren Sie das Beispiel aus Aufgabe 1 mit einem ExecutorService Thread-Pool der Größe 10.
3. Welchen Unterschied bemerken Sie, wenn der Thread-Pool mit einer Größe von 2 erstellt wird und warum?

**Aufgabe 3:**

Die bisherigen Methoden führen den Code lediglich einmal im Thread aus. Manchmal ist es allerdings sinnvoll den gleichen Code in periodischen Abständen auszuführen.

1. Geben Sie Beispiele bei denen diese periodische Ausführung sinnvoll wäre.
2. Auch hierfür gibt es in Java eine Implementierung: „ScheduledExecutorService“.
3. Programmieren Sie eine kleine Java-Anwendung, welche mit Hilfe des ScheduledExecutorService alle 5 Sekunden die aktuelle Zeit in der Form „hh-mm-ss“ ausgibt.

**Lösungen zum Übungszettel 3**

**Aufgabe 1:**

1. Prozesse sind vom Betriebssystem ausführbare Programme, welche getrennte Speicherbereiche nutzen. Threads hingegen werden von einem Prozess erstellt und greifen auf den gleichen Speicherbereich des Prozesses zu.  
   Threads haben den Vorteil, dass die Kommunikation untereinander leichter zu implementieren ist und sie schnell erstellt werden können. Einen weiteren Prozess zu starten, dauert länger und kostet z.B. bei Java die Ressourcen einer kompletten JVM-Instanz. Nachteil der Threads ist die nötige Synchronisation von Datenzugriffen.
3. Die Ausgaben sind durch die parallele Ausführung nicht zwangsläufig in Reihenfolge.

**Aufgabe 2:**

1. Ein Thread-Pool ist eine Ansammlung von Threads, welche für Aufgaben erstellt und wiederverwendet werden können. Die Aufgaben werden hierbei in einer Warteschlange abgearbeitet. Bei einem begrenzten Thread-Pool wird entsprechend auf den nächsten freien Thread gewartet.
3. Die Ausführung mit einem Thread-Pool der Größe 2 zeigt deutlich die in a) beschriebene Funktion der Warteschlange mit einem begrenzten Thread-Pool. Die Ausgaben sind in 2er Blöcke gruppiert.

**Aufgabe 3:**

1. Periodische Pings auf eine Ressource als Verfügbarkeitscheck, periodischer Check auf neue Emails, usw.

**Übungszettel 4**

**Aufgabe 1:**

1. Beschreiben Sie das grundlegende Konzept der Parallelen Registermaschine (PRAM).
2. Welche Erweiterungen beinhaltet die PRAM gegenüber der RAM?
3. Wie ist der Ablauf bei einem FORK-Aufruf der PRAM?
4. Durch die entstehende Nebenläufigkeit kann es zur Laufzeit von PRAM-Programmen zu Konflikten kommen. Welche Konflikte können auftreten und wie wurden diese theoretisch gelöst?

**Aufgabe 2:**

Erstellen Sie mittels der Schaltsymbole von Halb- und Volladdierern einen Schaltkreis, welcher zwei Binärzahlen der Länge 3 Bit addieren kann.

**Aufgabe 3:**

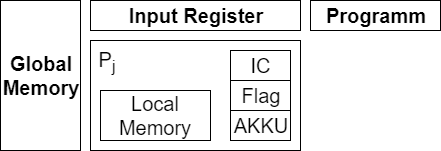
Erstellen Sie die Schaltkreise C3 (n=3) und C5 (n=5), welche die Sprache der Palindrome über B erkennen.

**Aufgabe 4:**

Wieso werden parallele Berechnungen oft mit Schaltkreisen simuliert?

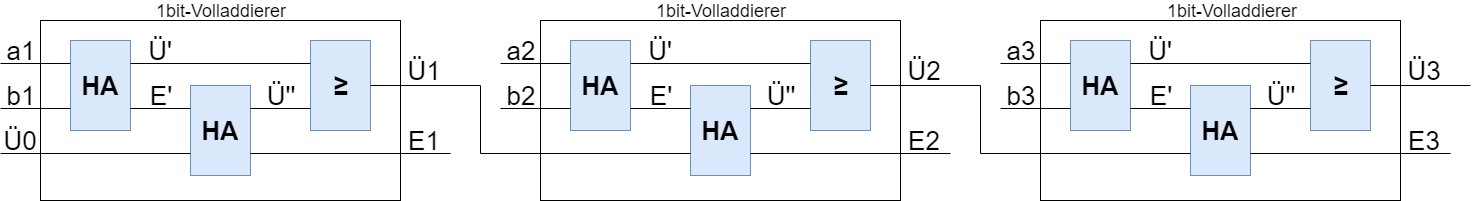
**Lösungen zum Übungszettel 4**

**Aufgabe 1:**

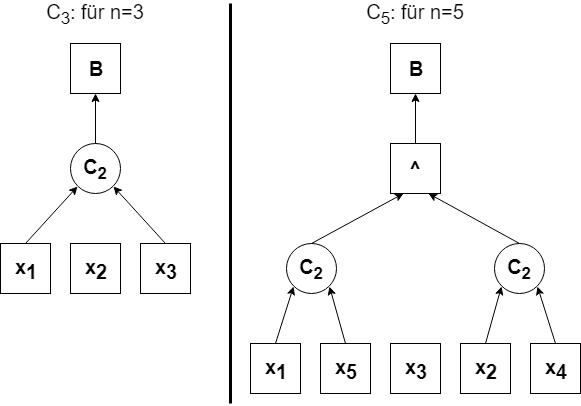


1. Der Instruktionsbereich wird durch den FORK-Befehl erweitert, der neue Prozessoren generieren kann.
2. Programmablauf trifft auf den Befehl „**FORK *label***“:  
   1. Schritt: Freier Prozessor wird gesucht - Pj  
   2. Schritt: Speicher von Pj löschen  
   3. Schritt: AKKU-Inhalt von P0 in Pj übertragen  
   4. Schritt: Pj bei ***label*** starten  
   **P0 und Pj laufen nebenläufig.**
3. Konflikte:  
   - Zwei Prozessoren wollen gleichzeitig aus einer Zelle lesen/in eine Zelle schreiben.  
   - Prozessor will aus einer Zelle lesen, in die ein anderer schreiben will.  
   Lösungen sind geeignete PRAM-Modelle: EREW-PRAM, CREW-PRAM, CRCW-PRAM

**Aufgabe 2:**



**Aufgabe 3:**



**Aufgabe 4:**

* Einfaches paralleles Modell, dass sofort in Hardware gegossen werden kann
* Komplexitätsmaße leicht definierbar (Tiefe = Worst Case Laufzeit, Größe = erforderliche Hardware). Platz und Zeit sind realistische Größen.
* Schaltkreise sind leicht übersetzbar in andere parallele Modelle (TM, RAM, PRAM) oder Graphen.

**Übungszettel 5**

**Aufgabe 1:**

1. Erklären Sie das Modell des Zellulären Automaten (ZA).
2. Welche Vor- und Nachteile beinhaltet das Modell?

**Aufgabe 2:**

*Simulation eines ZA (Conway’s Game of Life):* <https://bitstorm.org/gameoflife/>

Die Welt ist in diesem Fall zweidimensional und besteht aus einem quadratischen Gitter. Die Zellen auf diesem Gitter werden geboren, bleiben am Leben oder sterben ab in Abhängigkeit der Bevölkerungsdichte in ihrer Moore-Nachbarschaft. Die Zellen haben zwei mögliche Zustände: 0 = tot und 1 = lebend. Die Regeln sind wie folgt:

|  |  |  |
| --- | --- | --- |
| **Zustand** | **Nachbarzahl** | **Folgezustand** |
| 0 | 3 | 1 |
| 1 | 3 | 1 |
| 0 | < 2 oder > 3 | 0 |
| 1 | < 2 oder > 3 | 0 |
| 0 | 2 | 0 |
| 1 | 2 | 1 |

Folgende Startkonfiguration mit der Anzahl lebender Nachbarn (grün) ist gegeben:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **A** | **B** | **C** | **D** | **E** | **F** | **G** | **H** | **I** | **J** | **K** |
| **1** |  |  |  |  |  |  |  |  |  |  |  |
| **2** |  |  |  |  |  |  |  |  |  |  |  |
| **3** |  |  |  |  |  |  |  |  |  |  |  |
| **4** |  | 0 | 1 | 1 | 1 | 0 |  |  |  |  |  |
| **5** |  | 0 | 2 | 2 | 3 | 1 |  |  |  |  |  |
| **6** |  | 1 | 3 | 4 | 3 | 2 |  |  |  |  |  |
| **7** |  | 1 | 1 | 4 | 2 | 2 |  |  |  |  |  |
| **8** |  | 1 | 1 | 2 | 1 | 1 |  |  |  |  |  |
| **9** |  |  |  |  |  |  |  |  |  |  |  |

1. Simulieren Sie zwei Folgegenerationen anhand der beschriebenen Regeln.
2. Welche Auffälligkeit ist bei der Simulation erkennbar?

**Aufgabe 3:**

1. Erstellen Sie einen Datenflußgraph nach Aufrollen der folgenden Schleife:  
   DO I = 1,5  
    C(I) = A(I) \* B(I)  
    E(I) = C(I) + D(I)  
    F(I) = C(I) – D(I)  
   ENDDO
2. Welche Möglichkeiten gibt es, diese Rechenschritte zu parallelisieren?  
   (Tipp: horizontal und vertikal)
3. Welche Art der Parallelisierung wird von Vektormaschinen angewandt?

**Lösungen zum Übungszettel 5**

**Aufgabe 1:**

1. Unter einem zellulären Automaten (ZA) versteht man eine (meist) zweidimensionale, gitterförmige Anordnung quadratischer Zellen nebst zugehöriger Regeln, die beschreiben, in welcher Weise die Zustände der Nachbarzellen den Zustand einer Zelle beeinflussen (Nüchel 1995). Ein ZA ist demnach ein Verbund endlicher Automaten, die einer eingeschränkten TM entsprechen. Die Grundcharakteristika eines ZA sind wie folgt:  
   - Seine Entwicklung findet in Raum und Zeit statt  
   - Sein Raum ist eine diskrete Menge zahlreicher Zellen  
   - Simulation verteilter Systeme mit ZAs  
   - Jede dieser Zellen hat nur eine endliche Anzahl möglicher Zustände  
   - Die Zustände verändern sich in diskreten Zeitschritten  
   - Alle Zellen sind identisch und verhalten sich nach den gleichen Regeln  
   - Die Entwicklung einer Zelle hängt nur ab von ihrem Zustand und dem ihrer sie lokal umgebenden Nachbarzellen
2. Vorteile:  
   - Mit sehr einfachen Regeln kann ein sehr komplexes Verhalten modelliert werden  
   - Die Dynamik ist „exakt“, d.h. da nur mit diskreten Werten gearbeitet wird, treten keine Rundungsfehler auf, die sich akkumulieren können und so Einfluss auf die Dynamik nehmen  
   - Sehr einfache Implementierung und Kontrolle der Software  
   - Es besteht häufig ein Geschwindigkeits- und Speicherplatzvorteil bei der Simulation  
   - Erfahrungswissen lässt sich häufig leicht in Regeln überführt werden  
   Nachteile:  
   - Der ZA berücksichtigt lediglich direkte Zell-Nachbarschaften  
   - Sequentielle und daher keine parallele Verarbeitung möglich  
   - Relativ hoher Hardwareaufwand erforderlich

**Aufgabe 2:**

a)

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **A** | **B** | **C** | **D** | **E** | **F** | **G** | **H** | **I** | **J** | **K** |
| **1** |  |  |  |  |  |  |  |  |  |  |  |
| **2** |  |  |  |  |  |  |  |  |  |  |  |
| **3** |  |  |  |  |  |  |  |  |  |  |  |
| **4** |  | 0 | 1 | 1 | 1 | 0 |  |  |  |  |  |
| **5** |  | 0 | 2 | 2 | 3 | 1 |  |  |  |  |  |
| **6** |  | 1 | 3 | 4 | 3 | 2 |  |  |  |  |  |
| **7** |  | 1 | 1 | 4 | 2 | 2 |  |  |  |  |  |
| **8** |  | 1 | 1 | 2 | 1 | 1 |  |  |  |  |  |
| **9** |  |  |  |  |  |  |  |  |  |  |  |

D5, E7, E6 bleiben am Leben; D6, C7 sterben; C6, E5 werden lebendig

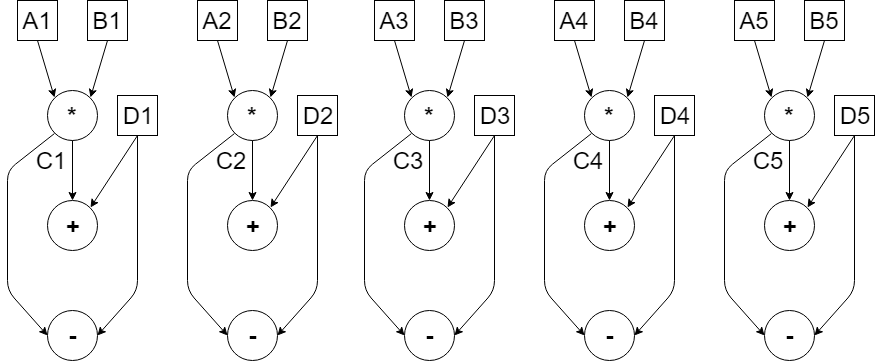
|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **A** | **B** | **C** | **D** | **E** | **F** | **G** | **H** | **I** | **J** | **K** |
| **1** |  |  |  |  |  |  |  |  |  |  |  |
| **2** |  |  |  |  |  |  |  |  |  |  |  |
| **3** |  |  |  |  |  |  |  |  |  |  |  |
| **4** |  |  | 1 | 2 | 2 | 1 |  |  |  |  |  |
| **5** |  | 1 | 2 | 3 | 2 | 2 |  |  |  |  |  |
| **6** |  | 1 | 1 | 4 | 3 | 3 |  |  |  |  |  |
| **7** |  | 1 | 1 | 3 | 1 | 2 |  |  |  |  |  |
| **8** |  |  |  | 1 | 1 | 1 |  |  |  |  |  |
| **9** |  |  |  |  |  |  |  |  |  |  |  |

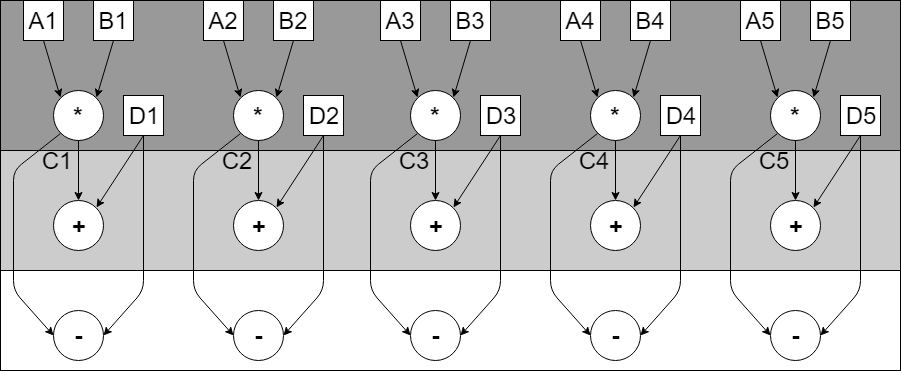
D5, E5, E6 bleiben am Leben; C6, E7 sterben; D7, F6 werden lebendig

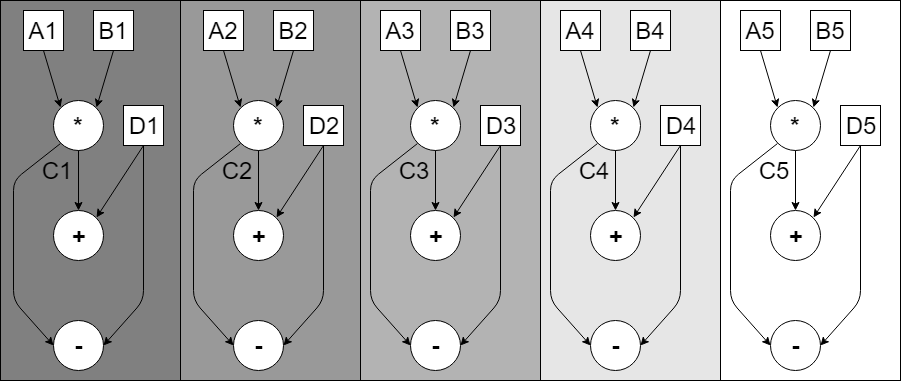
|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **A** | **B** | **C** | **D** | **E** | **F** | **G** | **H** | **I** | **J** | **K** |
| **1** |  |  |  |  |  |  |  |  |  |  |  |
| **2** |  |  |  |  |  |  |  |  |  |  |  |
| **3** |  |  |  |  |  |  |  |  |  |  |  |
| **4** |  |  | 1 | 2 | 2 | 1 |  |  |  |  |  |
| **5** |  |  | 1 | 2 | 3 | 3 | 1 |  |  |  |  |
| **6** |  |  | 2 | 4 | 4 | 2 | 1 |  |  |  |  |
| **7** |  |  | 1 | 1 | 3 | 2 | 1 |  |  |  |  |
| **8** |  |  | 1 | 1 | 1 |  |  |  |  |  |  |
| **9** |  |  |  |  |  |  |  |  |  |  |  |

b) Das Ergebnis ist die Anfangsfigur horizontal gespiegelt und um 90° nach rechts gedreht.

**Aufgabe 3:**



1. Horizontale Parallelisierung - Vektorisierung  
     
   Vertikale Parallelisierung – Iteration



1. Die horizontale Parallelisierung wird von den Compilern der Vektormaschinen angewandt, aber auch von manchen Compilern für superskalare Prozessoren. Auf Vektorrechnern sinnvollerweise eingesetzte Programmiersprachen kennen den Datentyp Vektor. Der Programmierer formuliert den Algorithmus demnach nicht als Schleife, sondern als drei Vektoroperationen. Der Compiler braucht dann diese Transformation nicht mehr leisten.

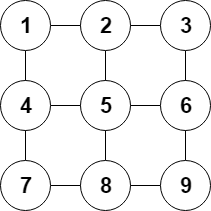
**Übungszettel 6**

**Aufgabe 1:**

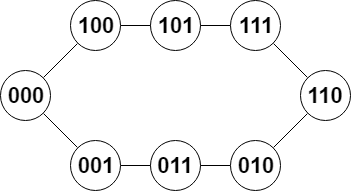
1. Erklären Sie den Begriff „Verbindungsnetzwerk“ und seine Aufgaben.
2. Welche Arten von Verbindungsnetzwerken gibt es? Erläutern Sie die Unterschiede und nennen Sie jeweils zwei typische Vertreter dieser Netzwerke.

**Aufgabe 2:**

1. Bestimmen Sie Grad, Durchmesser und Kantenkonnektivität des nachfolgenden Verbindungsnetzwerkes. Welche Topologie wird hier dargestellt?



1. Überführen Sie den nachfolgenden Ring mit 8 Knoten in ein 2x4-Gitter und anschließend in einen 3-dimensionalen Würfel.



**Aufgabe 3:**

1. Was ist eine „Connection Machine (CM)“ und wofür wurde sie konzipiert?
2. Welches Verbindungsnetzwerk wurde bei der CM-1 verwendet? Wie wurde im Netzwerk kommuniziert?

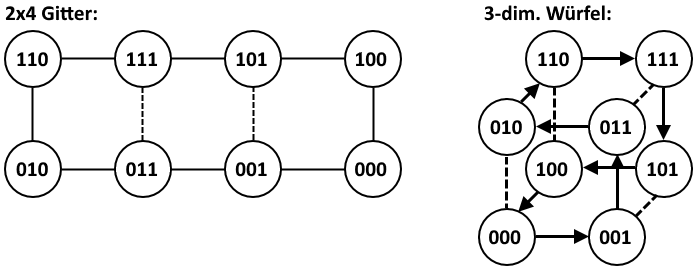
**Lösungen zum Übungszettel 6**

**Aufgabe 1:**

1. Wichtiger Aspekt: Fehlerfreie und möglichst schnelle Übertragung der Nachrichten/Daten durch das Verbindungsnetzwerk
   * Verbindet einzelne Verarbeitungseinheiten (Knoten) des Netzwerks miteinander
   * Dient der Koordination von Rechenaufgaben
   * Hauptaufgabe: Übertragung von Nachrichten zwischen einzelnen Verarbeitungseinheiten
2. Statische Verbindungsnetzwerke: Jeder Kommunikationsknoten im Netz ist fest mit Nachbarn verbunden. (Ring, Gitter bzw. Torus, Hyperwürfel, Binärbaum)  
   Dynamische Verbindungsnetzwerke: Mindestens ein Schaltelement, an dessen Ein- und Ausgänge die Kommunikationsknoten angeschlossen sind. (Beta-Zellen (Kreuzschalter, Crossbar), Clos und Benes. Permutationsnetze wie z.B. Perfect-Shuffle-Permutation, Butterfly-Permutation (2x Butterfly = Benes)  
   *Unterschiede*:
   * Im Gegensatz zu statischen Netzwerken keine feste Punkt-zu-Punkt-Verbindung
   * Aufgebaut aus physikalischen Leitern und dazwischenliegenden Schaltern
   * Verbindung einzelner Knoten bei Bedarf
   * Deshalb auch Bezeichnung: indirekte Verbindungsnetzwerke
   * Verwendung meist in Systemen mit gemeinsam genutzten Speicher
   * Zur Einordnung: heranzieren topologischer Merkmale
   * Je komplexer ein Netzwerk ist, desto höher sind die Hardwarekosten aber auch die Leistung des Netzwerks

**Aufgabe 2:**

1. **Grad** = 4  
   Knoten Nr. 5 hat mit vier Verbindungen den größten Grad dieses Netzwerks.  
   **Durchmesser** = 4  
   Distanz von Knoten Nr. 1 zu Knoten Nr. 9 maximal 4.  
   **Kantenkonnektivität** = 2  
   Min. 2 Kanten müssen aus dem Netzwerk entfernt werden, um einen Knoten vollständig zu trennen und das Netz zu unterbrechen.  
   **Topologie** = Gitter-Netz



**Aufgabe 3:**

1. Die erste Connection Machine war ein massiv paralleles System mit bis zu 65536 1-Bit Prozessoren (Processor/Memory-Zellen, P/M-Zellen). Die CM-1 war vorwiegend zur Lösung von Problemen aus dem Bereich der KI konzipiert. Das Ziel war es, einen Rechner zu entwerfen, der das alltägliche, logische Denken simulieren sollte, um grundlegende Funktionen des menschlichen Gehirns zu verstehen.
2. 16 Prozessoren eines Chips sind über ein Gitter miteinander verdrahtet. 4096 Chips kommunizieren über ein 12-dim. Hypercube-Verbindungsnetzwerk. Die Kommunikation lief über die Funktionen „injection, routing, buffering, referring und delivering“ ab.

**Übungszettel 7**

**Aufgabe 1:**

1. Nennen Sie die zentralen Verwendungszwecke eines Computerclusters.
2. Erläutern Sie die zugrunde liegende Architektur eines *Beowulf-Clusters*.
3. Wie werden die Aufgabenverteilung, die Kommunikation und der Datenaustausch auf einem *Beowulf-Cluster* realisiert?

**Aufgabe 2:**

1. Welche Arten von Deadlocks gibt es?
2. Wie lassen sich Deadlocks verhindern?

**Aufgabe 3:**

In dieser Aufgabe soll das Message-Passing-Interface (MPI) Konzept praktisch nähergebracht werden. Hierfür müssen einige Vorbereitungen getroffen werden:

1. Zur Verwendung von MPI wird eine Bibliothek benötigt. Da diese Aufgabe in Java umgesetzt wird, verwenden wir in diesem Fall „MPJ Express“ (http://mpj-express.org) welches hier in der aktuellen Version heruntergeladen werden kann: <https://sourceforge.net/projects/mpjexpress/files/latest/download>
2. Der Pfad der Bibliothek muss als Umgebungsvariable gesetzt werden, z.B.:  
   **Linux**: *export MPJ\_HOME="/path/to/mpj-v0\_44"*  
   **Windows**: *setx MPJ\_HOME "C:\\path\\to\\mpj-v0\_44"*
3. Der Java Compiler benötigt ebenfalls den Pfad zur Bibliothek für unser Programm:  
   **Linux**: *javac -cp $MPJ\_HOME/lib/mpj.jar Program.java* **Windows**: *javac -cp %MPJ\_HOME%/lib/mpj.jar Program.java*

MPI ermöglicht es, Aufgaben in kleinere Teile zu zerlegen und über die Prozesse zu verteilen. Hier eignet es sich oft die Prozesse in einen Master- und die restlichen in Slave-Prozesse zu unterteilen. Beginnend mit dem Java-Code in Programm 2 schreiben Sie ein MPI Programm, welches die Summe der Zahlen von 1 bis 10.000 berechnet. Die Prozesse berechnen hierbei Teilsummen aus gleichmäßigen Stücken des Bereichs [1, 10.000] und der Master akkumuliert die Teilergebnisse zur Gesamtsumme. Die Slaves schicken die Teilergebnisse über „MPI.COMM\_WORLD.Send“ an den Master, welcher diese mit „MPI.COMM\_WORLD.Recv“ empfängt.

Starten Sie das Programm mit 4 Prozessen wie folgt:  
**Linux**: *$MPJ\_HOME/bin/mpjrun.bat -np 4 Program***Windows**: *%MPJ\_HOME%/bin/mpjrun.bat -np 4 Program*

Programm 2

import mpi.\*;

public class Program {

public static void main(String[] args) throws Exception {

MPI.Init(args);

int id = MPI.COMM\_WORLD.Rank();

int size = MPI.COMM\_WORLD.Size();

System.out.println("Process " + id + " of " + size);

int[] sum = new int[1];

// TODO

if (id == 0) { // Master

System.out.println("The sum from 1 to 10000 is: " + sum[0]);

}

MPI.Finalize();

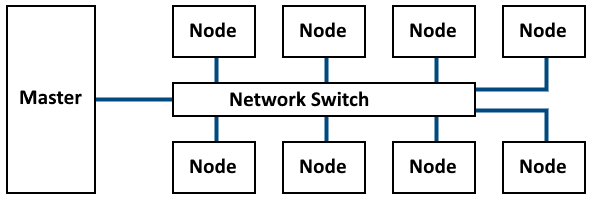
}

}

**Lösungen zum Übungszettel 7**

**Aufgabe 1:**

1. Hochverfügbarkeitscluster:  
   Hochverfügbarkeitscluster (engl. High-Availability-Cluster – HA-Cluster) werden zur Steigerung der Verfügbarkeit bzw. für bessere Ausfallsicherheit eingesetzt. Tritt auf einem Knoten des Clusters ein Fehler auf, werden die auf diesem Knoten laufenden Dienste auf einen anderen Knoten migriert. Die meisten HA-Cluster besitzen 2 Knoten.  
     
   Load-Balancing-Cluster:  
   Load-Balancing-Cluster werden zum Zweck der Lastverteilung auf mehrere Maschinen aufgebaut. Die Lastverteilung erfolgt in der Regel über eine redundant ausgelegte, zentrale Instanz. Viele Webseiten nutzen dies zum Beispiel um die große Menge an individuellen Seitenaufrufen bewältigen zu können.  
     
   High-Performance-Computing-Cluster:  
   High-Performance-Computing-Cluster (HPC-Cluster) dienen zur Abarbeitung von Rechenaufgaben. Diese Rechenaufgaben werden auf mehrere Knoten aufgeteilt. Entweder werden die Aufgaben in verschiedene Pakete aufgeteilt und parallel auf mehreren Knoten ausgeführt oder die Rechenaufgaben (Jobs genannt) werden auf die einzelnen Knoten verteilt. Die Aufteilung der Jobs übernimmt dabei meistens ein Job Management System. HPC-Cluster finden sich oft im wissenschaftlichen Bereich. In der Regel sind die einzelnen Elemente eines Clusters untereinander über ein schnelles Netzwerk verbunden.
2. Ein Beowulf-Cluster besteht aus einer gewissen Anzahl von Rechen-Knoten (compute nodes), einem oder mehreren Server-Knoten (server nodes) und in der Regel aus einem (oder mehreren) Zugangs-Knoten (front end), auf dem bzw. denen sich die Nutzer einloggen können. Von dort aus können sie sich die benötigte Menge von Rechen-Knoten für ihre Arbeit reservieren und diese benutzen.



1. Die Bedienung des Clusters erfolgt über einen Master-Knoten, der auch die zu erledigende Aufgabe „Job“ in kleine Teile zerlegt (Decomposition-Programm) und dann mittels eines Job-Scheduling-Programms auf die Knoten verteilt. Die einzelnen Knoten kommunizieren über IP. Der Datenaustausch/die Kommunikation zwischen den auf verschiedenen Koten laufenden Job-Teilen geschieht in der Regel über standardisierte Bibliotheken, die eine abstrakte Kommunikationsschnittstelle zur Verfügung stellen. Die bekanntesten Vertreter solcher Bibliotheken sind Message Passing Interface (MPI) und Parallel Virtual Machine (PVM).

**Aufgabe 2:**

Deadlocks (Kommunikation):

Verklemmungen können bei paketvermittelnden Netzen in allen Netztopologien auftreten.  
*Verhinderung:* Die Zweier-Verklemmung lässt sich durch getrennte Sende- und Empfangspuffer vermeiden, weil durch diese Maßnahme auch dann noch Daten von Nachbarknoten entgegengenommen werden können, wenn die Sendepuffer voll sind. Oft muss man virtuelle Kanäle etc. einführen.

Deadlocks (Prozessebene):

Ein Deadlock tritt immer dann auf, wenn zwei (oder mehr) Prozesse ein Objekt beanspruchen, das jeweils der andere Prozess gerade in Besitz hat und damit blockiert. Deadlocks können in verschiedensten Situationen in der parallelen Programmierung auftreten.  
*Verhinderung:* Ein Ansatz liegt darin, alle für einen Prozess erforderlichen Objekte vor der Ausführung zu sperren und dabei die Sperrzeit so gering wie möglich zu halten. Verschiedene Programmiersprachen bieten Timeouts an, die dann Deadlocks verhindern sollen.

Erkennung von Deadlocks:

Dazu ist der Wartegraph zu bestimmen. Jeder Zyklus im Wartegraph repräsentiert einen Deadlock.

**Übungszettel 8**

**Aufgabe 1:**

1. Beschreiben Sie das grundlegende Konzept des Datenflussrechners (DFR).
2. Welche Architekturen von DFR gibt es? Erläutern Sie die Unterschiede.

**Aufgabe 2:**

Gegeben ist der folgende Ausdruck:

z = (x \* y) – (x + y);

1. Erstellen Sie einen Berechnungsgraphen für den obigen Ausdruck. Wann beginnt eine Operation?
2. Wie sieht die Aktivitätstabelle in einem statischen DFR aus?
3. Wie lassen sich die Befehle in der Maschine (Datenflussgraph) darstellen?

**Aufgabe 3:**

Erstellen Sie einen Datenflussgraphen mit Verzweigungen für folgenden Code:

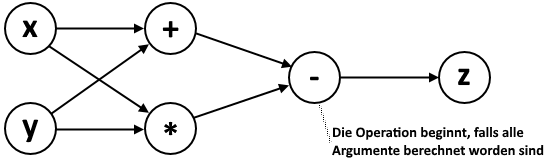
Input x  
if x < 5 and x >= 0  
{  
 y = (x + 10) \* 4  
}  
else if x < 10 and x >= 5  
{  
 y = (x – 3) \* 2  
}  
else  
{  
 y = (x / 2) \* 2  
}

**Lösungen zum Übungszettel 8**

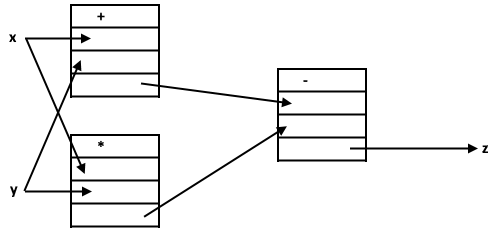
**Aufgabe 1:**

1. Datenflussrechner versuchen, die Möglichkeiten der Parallelverarbeitung ihrer Rechenaufträge durch das nebenläufige Ausführen einer Vielzahl von Threads auszunutzen. Der Datenflussrechner basiert auf dem Datenflussprinzip, welches die Ablaufsteuerung gänzlich anders organisiert. Einzig die Verfügbarkeit der Operanden löst die Ausführung einer Maschinenoperation auf diesen Operanden aus. Die Resultate können dann wieder zur Ausführung anderer Operationen führen. Deshalb wird dieses Operationsprinzip auch als datengesteuert oder datengetrieben (Data-Driven) bezeichnet. Nach dem Operationsprinzip können mehrere Threads von Maschinenbefehlen, die einen sogenannten Datenflussgraphen beschrieben, verarbeitet werden und einen hohen Grad an Nebenläufigkeit besitzen. Daher sind keine Befehlszähler und globale Variablen oder zentraler Speicher verfügbar im Vergleich zu der von-Neumann-Architektur. Zudem können Befehlsfolgen bei parallelen Kontrollflussprinzipien, die nach dem von-Neumann-Prinzip aufgeführt werden, durch explizite parallele Kontrollstrukturen (z.B. fork, …, join) verknüpft sein und parallel zueinander aufgeführt werden.
2. Zur Klassifikation von Datenflussrechnern existieren unterschiedliche Ansätze. Die am häufigsten verwendete Klassifikation basiert auf dem in Datenflussrechnern verwendeten Prinzip der Synchronisation von Operationen. Dieser Ansatz unterscheidet statische und dynamische Datenflussrechner. In der Klasse der statischen Datenflussrechner findet nur Parallelismus feiner Granularität Anwendung, während die Klasse der dynamischen Datenflussrechner aufgrund der Granularität des unterstützten Parallelismus weiter differenziert werden kann. Dabei werden Datenflussarchitekturen, die Parallelismus gröberer Granularität unterstützten unterschieden. Unter feiner Granularität wird hierbei Parallelismus auf der Instruktionsebene verstanden. Architekturen, die Parallelität auf der Block- oder Prozessebene unterstützen, werden unter dem Sammelbegriff Datenfluss- von-Neumann-Hybridarchitekturen zusammengefasst.

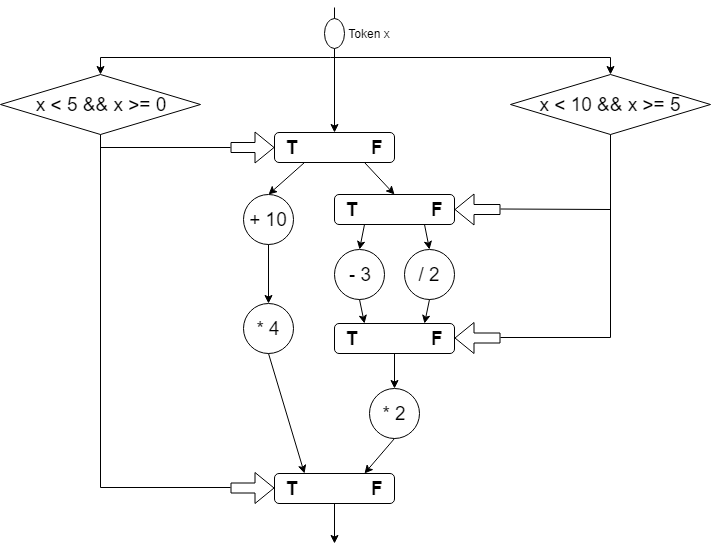
**Aufgabe 2:**



1. T1: (\*, x, y; T3/l)  
   T2: (+, x, y; T3/r)  
   T3: (-, (), (); z)
2. Verkettete Listen, Immediate-Argumente, Opcodes und Ziel-Listen vom Compiler angelegt



**Aufgabe 3:**



**Übungszettel 9**

**Aufgabe 1:**

1. Was unterscheidet einen Feldrechner von einem Vektorrechner? Erläutern Sie die unterschiedliche Ausführung anhand der Vektoroperation *D = (A – B) \* C*
2. Was wird unter dem Prinzip des Pipelining verstanden?

**Aufgabe 2:**

1. Bestimmen Sie die DLXV-Maschinenbefehlsfolge für die nachstehende Schleife:  
   for (i = 0; i < 32; i++)  
   {  
    Y[i] = a / (X[i] – Y[i]);  
   }
2. Was unterscheidet eine Vektorisierung von einer Iteration der Schleife?
3. Was ist bei der Vektorisierung zu beachten?

**Aufgabe 3:**

Gegeben ist die nachstehende Schleife mit den Anweisungen S1 bis S4:

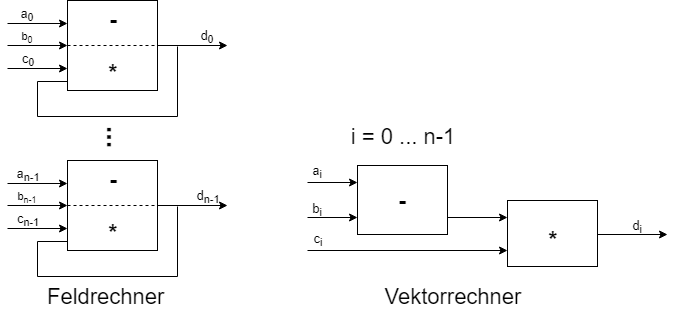
DO I = 1,N  
S1: A(I) = B(I)  
S2: C(I) = 2.0 \* A(I)  
S3: D(I) = C(I + 1) + B(I)  
S4: A(I) = C(I) + D(I)

1. Welche Abhängigkeiten schränken die Vektorisierung ein?
2. Wie lässt sich die obige Schleife vektorisieren? Ist sie auch parallelisierbar?

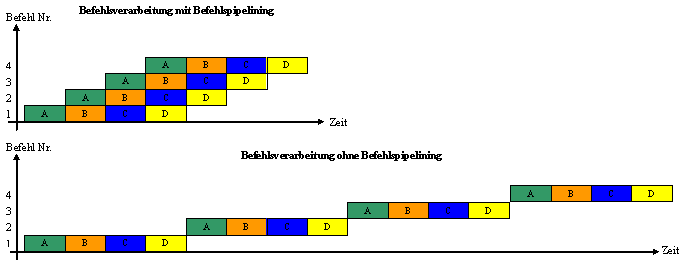
**Lösungen zum Übungszettel 9**

**Aufgabe 1:**

1. Der grundlegende Unterschied dieser beiden Ansätze ist, dass Feldrechner aus n Rechenelementen (oder Verarbeitungseinheiten) bestehen, die alle auf dem gleichen Instruktionsstrom arbeiten, aber natürlich auf verschiedenen Daten.  
   Im Gegensatz dazu berechnet der klassische Vektorrechner Probleme nach dem Pipeline-Prinzip. Dieses Prinzip der Fließbandverarbeitung ist immer dann anwendbar, wenn komplexe Operationen aus mehreren aufteilbaren Rechenschritten bestehen. Zusätzlich können verschiedene Funktionseinheiten (Addierer, Multiplizierer, …) aneinandergereiht („chaining“) werden, um auf einem Datenstrom zu arbeiten.



1. Die Verarbeitung eines Vektors erfolgt nach dem Pipelining-Verfahren. Dafür wird die Operation in möglichst gleichlange Teiloperationen zerlegt, die dann wie in einer Pipeline zeitsequentiell hintereinander in den verschiedenen Stufen bearbeitet werden. Die Ausführung der verschiedenen Teiloperationen überlappen sich dabei für die einzelnen Vektorelemente. Der Gewinn einer Pipelineverarbeitung gegenüber der sequentiellen Verarbeitung ist für lange Vektoren gleich der Stufenzahl der Pipeline [Hockney,Jesshop].



*[Von Frank Jacobsen - http://de.wikipedia.org/wiki/Datei:Befehlspipeline.PNG, CC BY 3.0, https://commons.wikimedia.org/w/index.php?curid=20551519]*

**Aufgabe 2:**

1. LV V1, RX; Vektor X laden  
   LV V2, RY; Vektor Y laden  
   SUBV V3, V1, V2; Vektor Subtraktion  
   LD F0, a(R0); Skalar a laden  
   DIVSV V4, F0, V3; Skalar-Vektor Division  
   SV RY, V4; Resultat in Y speichern
2. Die Vektorisierung stellt eine horizontale Parallelisierung dar, während die Iteration eine vertikale Parallelisierung darstellt.
3. Datenunabhängigkeit: Datenflussabhängigkeiten, Antiabhängigkeiten, Ausgabeabhängigkeiten  
   Prozedurale Abhängigkeiten: Fallunterscheidungen, Iterationen)  
   Operationale Abhängigkeiten: Abhängigkeit von der Anzahl der zur Verfügung stehenden Funktionseinheiten  
     
   *Daher*: einfache Schleifen, Felder und invariante Ausdrücke auf der rechten Seite und nur Zuweisungen benutzen.

**Aufgabe 3:**

1. Es besteht eine *echte Datenabhängigkeit bzw. Datenflussabhängigkeit* zwischen S1 und S2, eine *Output-Abhängigkeit* zwischen S1 und S4 und eine *Antiabhängigkeit* zwischen aufeinanderfolgenden Iterationen der Anweisungen S2 und S3.
2. Eine Schleife ist vektorisierbar, wenn nur vorwärts gerichtete Abhängigkeiten bestehen. Nach Vertauschen der Anweisungen S2 und S3 und der Umbenennung der Ausgabevariable von S4 lässt sich die obige Schleife vektorisieren.  
     
   Eine Schleife ist parallelisierbar, wenn aufeinanderfolgende Iterationen (Schleifendurchläufe) voneinander unabhängig sind. Zwischen den Anweisungen des Schleifenkörpers, der einen Iterationsschritt darstellt, dürfen Abhängigkeiten bestehen. Das obige Beispiel ist damit nicht parallelisierbar.

**Übungszettel 10**

**Aufgabe 1:**

1. Erläutern Sie die Entwurfsphasen paralleler Software nach Foster.   
   *Hinweis: Die Originalliteratur ist online verfügbar und umfasst viele weitere Bereiche der parallelen Entwicklung (https://www.mcs.anl.gov/~itf/dbpp/text/node15.html#SECTION02310000000000000000)*
2. Wenden Sie die Phase der Partitionierung auf den Ausdruck  
   z = (x \* y) – (x + y);  
   an.

**Aufgabe 2:**

Gegeben ist der folgende Ausdruck:

A = a + b \* c \* d + e + f + g

1. Welche Methoden zur Transformation von Ausdrücken gibt es?
2. Wenden Sie die direkte Methode auf den obigen Ausdruck an. Welche Ergebnisse liefern die einzelnen Durchläufe? Wie verändert sich die Ausführungszeit?

**Aufgabe 3:**

1. Erläutern Sie den Ansatz bei der automatischen Parallelisierung mittels Compiler.
2. Gegeben sind folgende Schleifen. Welche lassen sich nicht automatisch parallelisieren und warum?  
   1) for (i=0; i<N-2; i++) {  
    a[i] = b[i] \* c;  
    b[i] = a[i+2] \* d;  
    }  
   2) for (i=0; i<N; i++)  
    for (j=0; j<M; j++)  
    a[i][j] = b[i][j] \* c[i][j];  
   3) for (i=0; i<N; i++)  
    if (b[i] = = 0) break;  
    a[i] = a[i] / b[i];  
    }
3. Nennen Sie drei Probleme, die bei der automatischen Parallelisierung von Schleifen auftreten können.
4. Machen Sie drei Lösungsvorschläge zur Behebung der Probleme.

**Lösungen zum Übungszettel 10**

**Aufgabe 1:**

1. PCAM Design-Methode:
   1. (Spezifikation)
   2. Partitionierung  
      Problemabhängige Zerlegung der Berechnung und der Daten in Teile (Tasks). Ohne Berücksichtigung der Zielarchitektur. Fokus auf Such nach möglichen parallelisierbaren Stellen.
   3. Kommunikation  
      Analyse der Datenabhängigkeit. Festlegung der Kommunikationsanforderungen.
   4. Bündelung  
      Zusammenfassung stark zusammenhängender Teile zu größeren Tasks. Ziel: Effizienzsteigerung durch Kostenminimierung (Kommunikation).
   5. Abbildung  
      Abbildung der resultierenden Struktur auf konkrete Zielarchitektur. Aufgaben werden den Prozessoren zugeordnet. Ziel: Maximierung der Prozessorauslastung (statische oder dynamische Lastenverteilung)
2. Mögliche Gebietszerlegung:  
   a) Eingabematrizen/-vektoren zu Tasks  
   b) Ausgabevektoren zu Tasks  
   => Datenparallelität  
     
   Funktionszerlegung:  
   a) Multiplikationen  
   b) Additionen  
   => Kontrollparallelität

**Aufgabe 2:**

1. Vergleich der Prioritäten, polish-notation, Syntax-Tabelle, Distributivmethode
2. *1. Durchlauf:*  
   T1 \* d + T2 + T3  
    T1 = (b \* c)  
    T2 = (a + e)  
    T3 = (f + g)  
     
     
   *2. Durchlauf:*  
   T4 + T5  
    T4 = (T1 \* d)  
    T5 = (T2 + T3)  
     
   *3. Durchlauf:*  
   T0 = T4 + T5  
     
   **Ausgabe:** (((b \* c) \* d) + ((a + e) + (f + g))  
   Die Ausführungszeit wird von TA=6 auf TA=3 reduziert!

**Aufgabe 3:**

1. Idee: Compiler extrahiert automatisch die Parallelität aus einem sequentiellem Programm und generiert eine MPI-basierte oder Thread-basierte parallele Implementierung.  
   Ansatz: In den meisten HPC-Anwendungen wird die meiste Rechenzeit für (geschachtelte) Schleifen benötigt.  
   Analyse von Datenabhängigkeiten in Schleifen zur Erkennung von sicher parallelisierbaren Schleifen. Generierung von parallelem Code für einfache Schleifen mit Barrieren-Synchronisation am Schleifenende. Einige einfache Code-Transformationen werden zur Erhöhung der Anzahl parallelisierbarer Schleifen durchgeführt.
2. 1) Schleifen mit Index-Abhängigkeiten zwischen Iterationen nicht automatisch parallelisierbar.  
   2) Einfache Schleife (d.h. ohne Index-Abhängigkeiten zwischen Iterationen), gut automatisch parallelisierbar.  
   3) Schleifen mit mehreren Ausgängen nicht automatisch parallelisierbar.
3. Probleme:
   1. Loop Carrier Dependencies
   2. Schleifen mit Index-Abhängigkeiten zwischen Iterationen
   3. Schleifen mit Funktionsaufrufen
   4. Zeigerbasierte Schleifen
   5. Schleifen mit (evtl. versteckten) Reduktionen
4. Code-Transformationen können die Probleme lösen:
   1. Skalare Variablen durch Feldvariablen ersetzen
   2. Variablen in Schleifen umbenennen
   3. Anweisungen oder Funktionen einsetzen
   4. Reduktionen von Vektoren automatisch erkennen und durch Aufrufe von parallelen Reduktionsfunktionen ersetzen

**Übungszettel 11**

**Aufgabe 1:**

1. Was sind „Evolutionäre Algorithmen (EA)“? Erläutern Sie den Ablauf eines evolutionären Algorithmus.
2. Welche Verfahren haben sich im Bereich der EA entwickelt?
3. Wo werden EA angewendet?

**Aufgabe 2:**

1. Was wird unter „Metabolic Computing“ verstanden?
2. Was beschreibt das metabolische System? Wie wird die biochemische Ausdrucksfähigkeit gewährleistet?

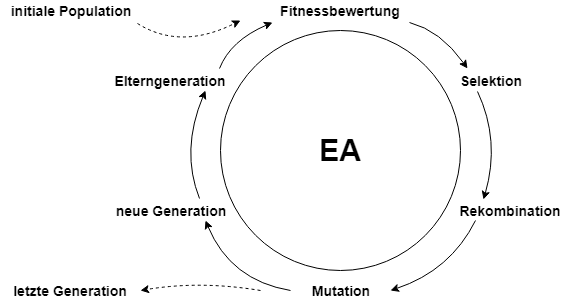
**Aufgabe 3:**

1. Was ist ein „Classifier System (CS)“? Welches Prinzip steckt dahinter?
2. Erklären Sie die Funktionsweise eines CS.
3. Was unterscheidet ein CS von einem „Learning Classifier System (LCS)“?

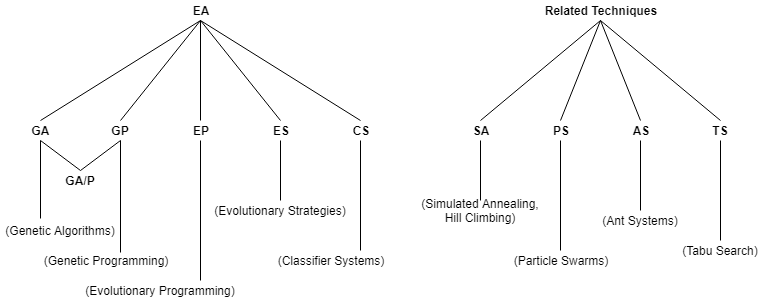
**Lösungen zum Übungszettel 11**

**Aufgabe 1:**

1. EA sind implementierte stochastische (heuristische) Such- und Optimierungsverfahren, welche von der klassischen Evolutionstheorie abgeleitet wurden. Die Grundidee ist, dass sich Populationen von Individuen an ihre Umwelt anpassen, um langfristig überlebensfähig zu sein. Dazu werden einige Prinzipien der darwinistischen Evolutionstheorie, deren einfachste Formulierung in der Regel „survival oft he fittest“ zu finden ist, in Computerprogrammen umgesetzt.  
   EA sind populationsbasierte Metaheuristiken. Ein EA basiert auf einer Population P von möglichen Lösungen ei ∈ P für das zu lösende Optimierungsproblem bzw. der Heuristik. Eine Lösung wird im Rahmen der populationsbasierten Metaheuristiken auch als Individuum bezeichnet. Grundsätzlich durchläuft ein EA die folgenden Schritte:  
     
     
   Initialisierung: Zunächst wird eine Population bestehend aus zufälligen Individuen initialisiert.  
   Evaluation: Die Fitness f(ei) wird für alle ei ∈ P evaluiert. Die Fitnessfunktion bezeichnet den zu optimierenden Wert (die Bewertungsfunktion) des Optimierungsproblems bzw. der Heuristik.  
   Selektion: Basierend auf dem Fitnesswert der Individuen werden (zufällige) Individuen, die als Eltern für die nächste Generation dienen, bestimmt. Dabei werden Individuen mit einer hohen Fitness bevorzugt (je nach Verfahren).  
   Reproduktion: Aus den selektierten Eltern werden die Nachkommen erzeugt. Dies kann durch einfaches Kopieren oder auch durch *Rekombination* von je 2 oder mehr Elternteilen erfolgen.  
   Rekombination und Mutation: Die Nachkommen werden untereinander rekombiniert (binäre Operation) und Individuen werden mutiert (unäre Operation). Dieser Schritt schafft evolutionäre Variation.  
   Ersetzung: Die Nachkommen ersetzen Teile der aktuellen Population oder die gesamte Population.  
     
   Dieser Schritte (ausgenommen der Initialisierung) werden solange wiederholt, bis ein Abbruchkriterium erreicht ist. Ein solches Kriterium kann zum Beispiel eine maximal erreichte Anzahl an Iterationen oder eine erreichte maximale Fitness darstellen.







1. EAs finden Anwendung beim Lösen von kombinatorischen Such- und Optimierungsproblemen, beim maschinellen Lernen, in den Ingenieurswissenschaften und vielem mehr. EA werden hauptsächlich bei sehr komplexen und bei größeren Datenmengen angewendet. Also genau da, wo herkömmliche Algorithmen keine brauchbaren Ergebnisse erzeugen oder diese Ergebnisse eine viel zu lange Zeit beanspruchen würden.  
   Typische Beispiele:
   1. Berechnung eines Stundenplans einer sehr großen Schule
   2. Platzierung von Container auf einem Schiff (Stabilität, schnelles Entladen)
   3. Entwurf von Kommunikationsnetzwerken (Leitungen, Knotenpunkte)
   4. Entwurf von Mantelstrom-Düsentriebwerken

**Aufgabe 2:**

1. MC beschreibt die Theorie der regelbasierten Modellierung des Zellstoffwechsels. Dazu ist die Zelle als chemische Maschine zu interpretieren.
2. Das metabolische System ist ein regelbasiertes System, das die Formalisierung von Biosynthesen, Proteinsyntheseprozessen und metabolischen pathways sowie Zelllkommunikationsprozessen mit Hilfe einer universellen Regel ermöglicht. Um die biochemische Ausdrucksfähigkeit dieser Regel zu gewährleisten, setzt sie sich aus fünf Komponenten zusammen. Die erste erlaubt die Spezifikation der Regelwahrscheinlichkeit (p), die unter der biologischen Interpretation als Reaktionsgeschwindigkeit vom Modellierer definierbar ist. Die vier weiteren Komponenten sind spezifische Mengen von Metaboliten, die eine detaillierte Modellierung einer biochemischen Reaktion ermöglichen.

**Aufgabe 3:**

1. CS sind massiv parallele, nachrichtenübermittelnde, regelbasierte Systeme. Ein CS besteht aus einer Menge von Classifiern, einer Message-Menge, einer Eingabe- und einer Ausgabe-Einheit. CS sind so ausgelegt, dass sie neue Informationen kontinuierlich aus ihren Umgebungen aufnehmen, indem sie konkurrierende Hypothesen (ausgedrückt als Regeln) konstruieren, ohne die bereits erworbenen Fähigkeiten zu stören.  
   Es steckt das Datenflussprinzip dahinter.
2. Funktionsweise:
   1. Alle Messages der Eingabe-Einheit werden zur aktuellen Message-Menge hinzugenommen
   2. Vergleiche alle Messages mit allen Bedingungen (Aktivierung)
   3. Alle aktivierten Classifier übertragen ihren Aktionsteil in die neue Message-Menge
   4. Ersetze die alte Message-Menge durch die neue Message-Menge
   5. Die Message-Menge wirkt auf die Ausgabe-Einheit
   6. Gehe zu a
3. Ein LCS ist ein Verfahrendes maschinellen Lernens, bei dem evolutionäre Algorithmen mit klassischen Lernalgorithmen kombiniert werden, um adaptive Systeme zu erzeugen. Diese Systeme basieren auf Regeln, die traditionell die Form der bedingten Anweisung (Wenn-Dann) aufweisen und das bestmögliche Verhalten aufgrund einer bestimmten Eingabe (Input) ausführen. Sie werden dazu mit einem EA angepasst.

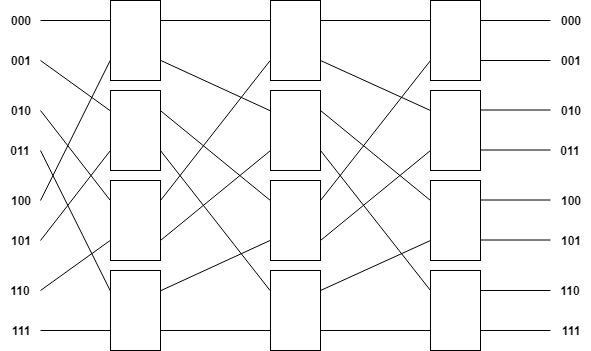
**Übungszettel 12**

**Aufgabe 1:**

1. Was wird unter „Load Balancing (LB)“ verstanden? Was unterscheidet LB von „Load Sharing (LS“)?
2. Welche Typen von LB-Algorithmen gibt es?
3. Welche Load Balancing-Verfahren werden zur Steigerung der Webserver-Performance eingesetzt? Erläutern Sie kurz diese Verfahren.

**Aufgabe 2:**

1. Was ist das Ziel von Routing?
2. Was unterscheidet ein deterministisches von einem adaptiven Routing-Verfahren?
3. Zeichnen Sie in das nachfolgende Omega-Netz das Routing von 001 nach 101 ein.



**Aufgabe 3:**

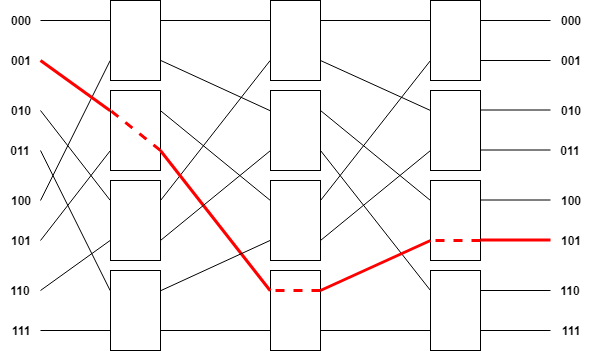
1. Welche Ziele verfolgt das „Parallel I/O“?
2. Wieso spielen der virtuelle Speicher und die Datenabhängigkeitsanalyse eine wichtige Rolle?

**Lösungen zum Übungszettel 12**

**Aufgabe 1:**

1. Load Balancing fordert, dass die Last zwischen den Knoten zu jedem Zeitpunkt bezüglich eines quantitativen Maßes ausgeglichen ist. D.h., LB beschäftigt sich mit der Fragestellung: Wie verteile ich die Last (Prozesse/Aufgaben) in verteilten Computer-Systemen so um, dass die Prozesse von den schwer geladenen Systemen auf die leicht geladenen Systeme verschoben werden.  
   Der Unterschied zwischen LB und LS ist, dass LB impliziert, dass alle Prozessoren nahezu dieselbe Last (load) haben. LS erlaubt auch Prozessoren mit höherer und niedrigerer Last ihre Last zu teilen.
2. LB Algorithmen können je nach Informationen, die vom System den Aufgaben und Benutzer bereitgestellt werden, eine statische, dynamische oder adaptive Lastverwaltung aufweisen.
   1. Senderinitiierte Lastverteilung
   2. Empfängerinitiierte Lastverteilung
   3. Hybrides System (symmetrisch initiierte Algorithmen)
3. Es gibt verschiedene Verfahren:
   1. DNS-Variante: Das klassische Load Balancing ist das DNS selber. Dazu wird dem DNS-Server der eigenen Domain mehrere IP-Adressen eintragen, unter der der Host-Name zu erreichen ist. Hinter jeder IP-Adresse befindet sich ein separater und eigenständiger Server. Die Anfragen der Clients bedient der DNS-Server der Reihe nach mit den eingetragenen IP-Adressen.
   2. Round-Robin-Verfahren: Das Round-Robin-Verfahren kommt mit einer einzigen IP-Adresse aus. Anstatt des DNS-Servers übernimmt ein NAT-Proxy die Lastverteilung. Anstatt einer Liste mit den verfügbaren Servern leitet der Proxy alle Anfragen an die ihm bekannten Zielsysteme weiter. Dabei merkt er sich welche IP-Adresse mit welchem Server eine Verbindung hatte und leitet eine erneute Anfrage an diesen Server weiter.
   3. NAT mit Feedback
   4. URL-basiertes Verfahren
   5. Dienst-basiertes Verfahren: Meist laufen mehrere Dienste, wie http, FTP und E-Mail auf ein und demselben Server. Unter Last erweist sich der Parallelbetrieb als Problem, wenn ein Dienst allen anderen Diensten Rechenleistung klaut. Alle Dienste benutzen unter TCP einen eigenen Port, anhand dem ein Datenpaket einer Anwendung oder einem Dienst zugewiesen wird. Wenn man die Dienste auf unterschiedlichen und eigenständigen Servern betreibt, lässt sich die Last verteilen.

**Aufgabe 2:**

1. Vermeiden von Wartezeiten, Staus und Deadlocks
2. Deterministischer Ansatz: Wahl des Pfades nur abhängig von Start- und Zielknoten der Nachricht  
   Adaptiver Ansatz: Mehrere Pfade werden unter Berücksichtigung der Auslastung des Netzwerks berechnet.  
     
   Bei beiden Ansätzen gibt es zwei Varianten:  
   - minimale Algorithmen: wählen immer den kürzesten Weg  
   - nicht-minimale Algorithmen: erlauben auch Umwege, um Staus zu vermeiden
3. 

**Aufgabe 3:**

1. Serielle I/O verursacht ein Bottleneck, da die Berechnung stets auf die I/O warten muss. Die Leistung der CPU-Kerne wird verschwendet und wenig Arbeit verrichtet.  
   Parallele I/O führt mehrere I/O Operationen gleichzeitig aus. Dabei greift parallele I/O gleichzeitig auf Daten auf der Festplatte zu, anstatt die I/O Anfrage seriell auszuführen. Dies ermöglicht dem System höhere Schreibgeschwindigkeiten und Bandbreite.  
   Bei paralleler I/O ist ein Teil der logischen Kerne auf dem Multicore-Chip für die Verarbeitung von I/O der virtuellen Maschinen vorgesehen und die restlichen Kerne bedienen die anderen Anwendungen. Dies ermöglicht es dem Prozessor, mehrere Lese- und Schreiboperationen gleichzeitig zu verarbeiten. Parallele I/O hilft, I/O-Bottlenecks zu vermeiden, die den Datenfluss stoppen oder beeinträchtigen können.
2. Üblicherweise verwendet man Multicore CPUs in virtuellen Servern. Die umfangreiche parallele Berechnung wird meist auf mehreren virtuellen Maschinen ausgeführt, die wiederum virtuelle Speicher für parallele I/O benötigen. Bei paralleler I/O sind die Datenabhängigkeiten zu prüfen, um Wartezeiten zu reduzieren.