Rechnerstrukturen im SS2006

Parallelrechner: Leistung, Verbindungs-techniken und Parallelisierung

Dr. Jie Tao

tao@ira.uka.de
http://itec.uka.de/~tao

05. 07.2006

Leistungsfähigkeit von Multiprozessorsystemen

Aufgabe 1:

Betrachten Sie ein Multiprozessorsystem mit 16Prozessoren. Die Leistungssteigerung gegenüber einem Einprozessorsystem sei S(16)=8. Die Ausführungszeit auf dem Einprozessorsystem sei T(1)=80 und die Anzahl der auszuführenden Einheitsoperationen auf dem Multiprozessorsystem sei P(16)=16.

- Berechnen Sie die Effizienz E(16), Parallelindex I(16) und parallele Ausführungszeit T(16).
- Ermitteln Sie anhand von Amdahls Gesetz den Bruchteil der Programme, die nur sequentiell ausführbar ist.

Leistung von Parallelrechnersystemen

- Speedup

$$S(n) = \frac{\text{Sequentielle Ausführungszeit } T(1)}{\text{Parallele Ausführungszeit } T(n)}$$

- Effizienz

$$E(n) = \frac{S(n)}{n}$$

- Parallelindex

$$I(n) = \frac{P(n)}{T(n)}$$

Lösung

- $E(16) = \frac{S(16)}{16} = \frac{8}{16} = 0{,}5$
- $T(16) = \frac{T(1)}{S(16)} = \frac{80}{8} = 10$
- $I(16) = \frac{P(16)}{T(16)} = \frac{16}{10} = 1{,}6$

- Amdahls Gesetz: $T(n) = T(1) * ((1-a)/n + a)$

→ $10=80*((1-a)/16+a)=80*((1+15a)/16)=5*(1+15a)$

→ $15a=1$ → $a=1/15$ =6,7% des Programmcodes sind nur sequentiell ausführbar

Aufgabe2:

Die Ausführungszeit einer sequentiellen Anwendung betrage T Sekunden. Von dieser Zeit lassen sich 20% nicht parallelisieren und der Rest 80% werden zwischen den Prozessoren fair verteilt. ``Fair'' bedeutet, dass jeder Prozessor ungefähr den gleichen Anteil der zu parallelisierenden Aufgabe bearbeitet und jeder benötigt gleich viel Zeit. Beispielsweise betrage die Ausführungszeit der parallelen Anteile der Anwendung 20% der sequentiellen Ausführungszeit, wenn vier Prozessoren sie ausführen.

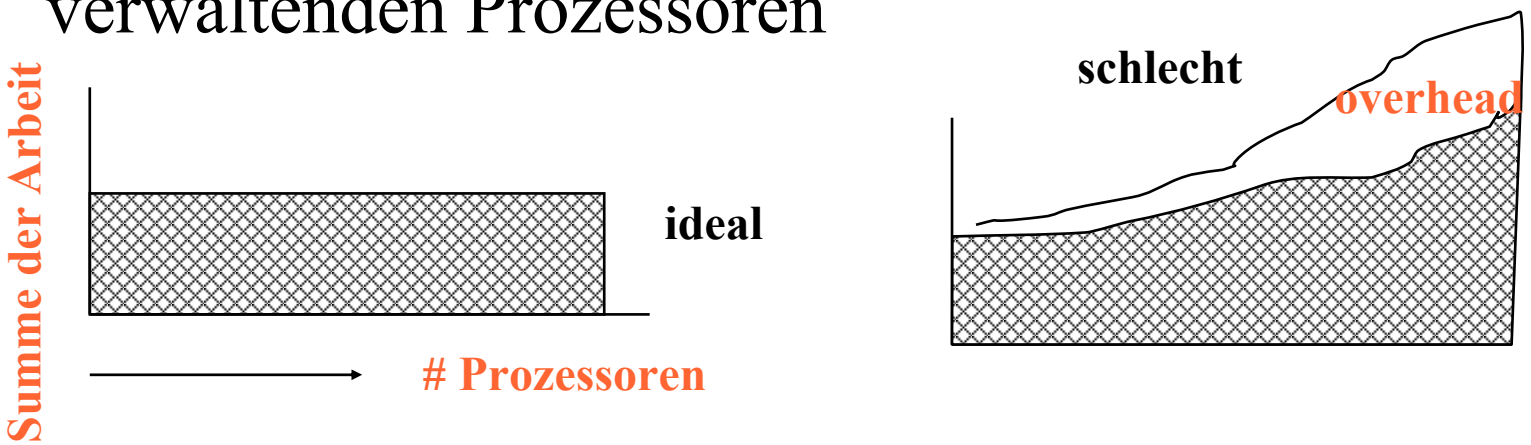
- Unter der Annahme, dass die Parallelisierung keinen Aufwand verursacht. Berechnen Sie Speedup und Effizienz bei unterschiedlicher Anzahl von Prozessoren (2, 4, 8, 16, 32). Evaluieren Sie die Skalierbarkeit und berechnen Sie den Geschwindigkeitszuwachs
- Durch jeden zusätzlichen Prozessor wird ein Aufwand von 1% der sequentiellen Ausführungszeit eingefügt. Berechnen Sie Speedup und Effizienz für 64 Prozessoren

Beeinflussfaktoren

- Parallelität
  - Sequentiellteil vs. Parallelteil
- Overhead
  - Synchronisation
    - Lock
    - Barrier
- Last-Balancierung
- Speicherleistung
  - Cache
  - Datenlokalität auf NUMA

Probleme in MP-Systemen

- Verwaltungsaufwand / Overhead steigt mit Zahl der zu verwaltenden Prozessoren



- Sättigungserscheinungen durch Auslastung oder Engpässe (bottlenecks)
- Skalierbarkeit
- Möglichkeit von Systemverklemmungen (deadlocks)

Lösung

- Parallele Ausführungszeit von P Prozessoren =

  (20% + 80%/P)*T

	P=2	P=4	P=8	P=16	P=32
Speedup	1,67	2,5	3,33	4	4,44
Effizienz	0,83	0,625	0,42	0,25	0,14

- Skalierbarkeit: sehr schlecht
  - Ein großer Teil vom Code nicht parallelisierbar
  - Zunehmende Anteil der gesamten Ausführungszeit mit aufsteigender Anzahl von Prozessoren

$$\text{Speedup} = \frac{T}{(20\%+80\%/P)*T} = \frac{1}{20\%} \quad \text{(Wenn P sehr groß ist)} = 5$$

- Mit 1% Overhead je Prozessor:

$$\text{Speedup} = \frac{T}{(20\% + 80\% / 64 + 1\% * 64) * T} = 1{,}17$$

Effizienz = 1,17 / 64 = 0,018

Aufgabe 3:

- Einsetzen eines mathematischen Koprozessors. Dieser bietet eine zweifach schnellere Ausführung der Gleitkommaarithmetik als der vorhandene Systemprozessor. Es ist allerdings keine parallele Verarbeitung, d.h. der gleichzeitige Einsatz von Haupt- und Koprozessor, möglich.
- Ausbau zu einem 2-SMP-System, d.h. die Installation eines zweiten zum vorhandenen identischen Hauptprozessor.

Das zu bearbeitende Problem ist zu 25% parallelisierbar. Der Anteil der Gleitkomma-arithmetik am Gesamtprogramm beträgt 30%. Bestimmen Sie, welche der beiden Möglichkeiten unter dem Gesichtspunkt der Ausführungszeit zu bevorzugen ist.

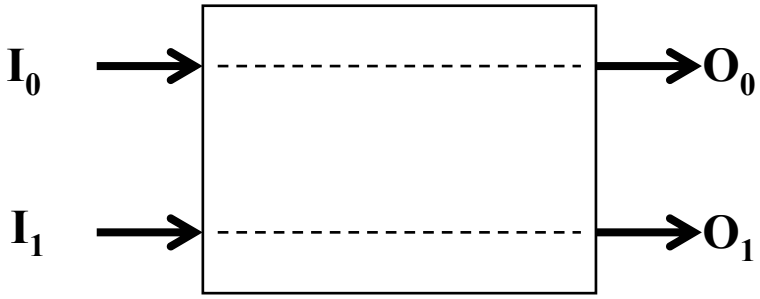
- Der einfachere Fall: 2-Fach SMP
  - Aufgrund der Angaben n=2, a=1-0,25=0,75
  - T(n)=T(1)* ((1-a)/n + a); T(2)=T(1)*0,875
  - S(2)=1/0,875=1,14
- Der andere Fall: Einsetzen eines mathematischen Koprozessors
  - Alte Ausführungszeit (ein Prozessor): T(old)
  - Neue Zeit: T (new) = (1-30%)*T(old) + 30% /2 * T(old)
  - S(2)=T(old)/T(new)=1/0,85=1,18 > 1,14

Verbindungsnetze (I)

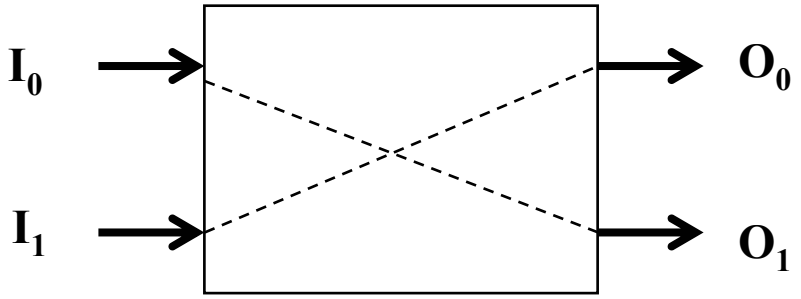
Aufgabe:

Eine Möglichkeit, mehrere Prozessoren eines Multiprozessorsystems miteinander blockierungsfrei zu verbinden, ist das Benes-Netzwerk.

Grundschaltung

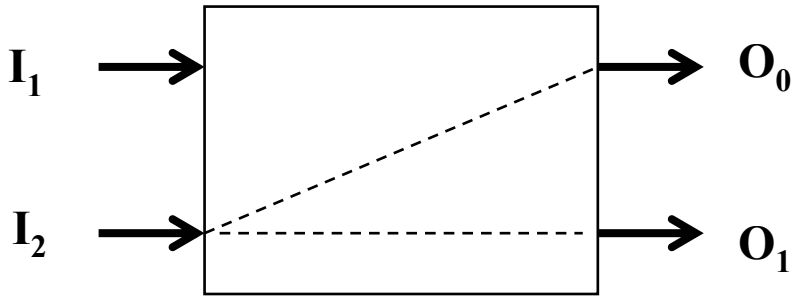


Durchschaltung/forward

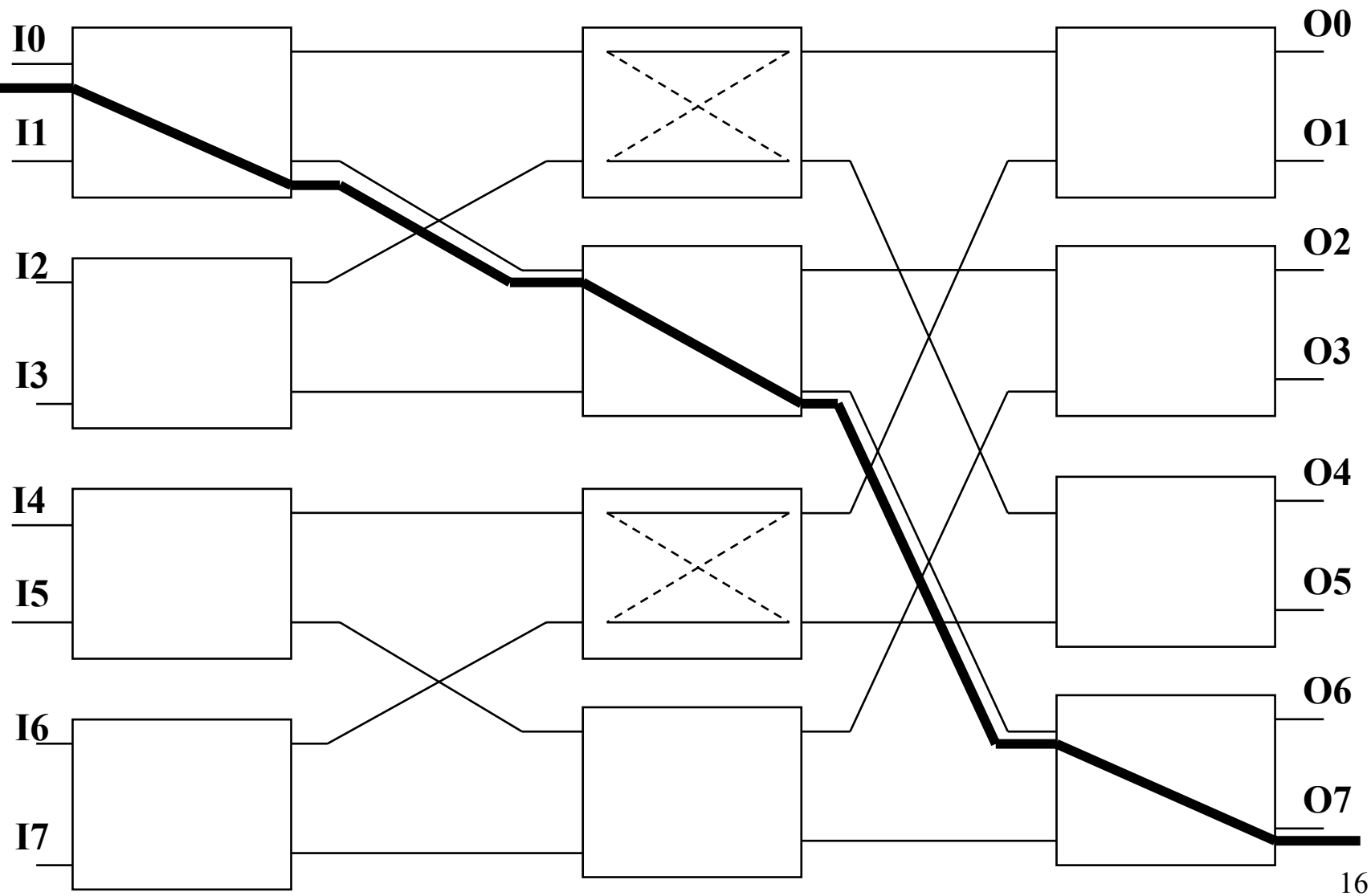


Vertauschung/exchange

Erweiterung (Broadcast)



Banyan-Netzwerk

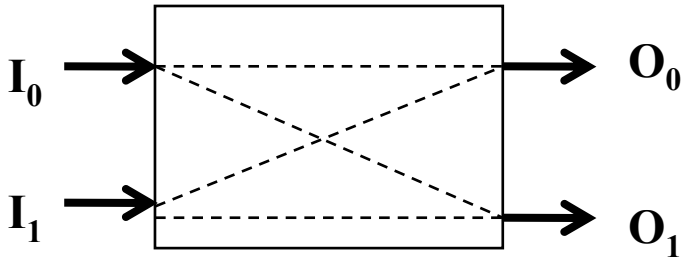


Banyan-Netzwerk

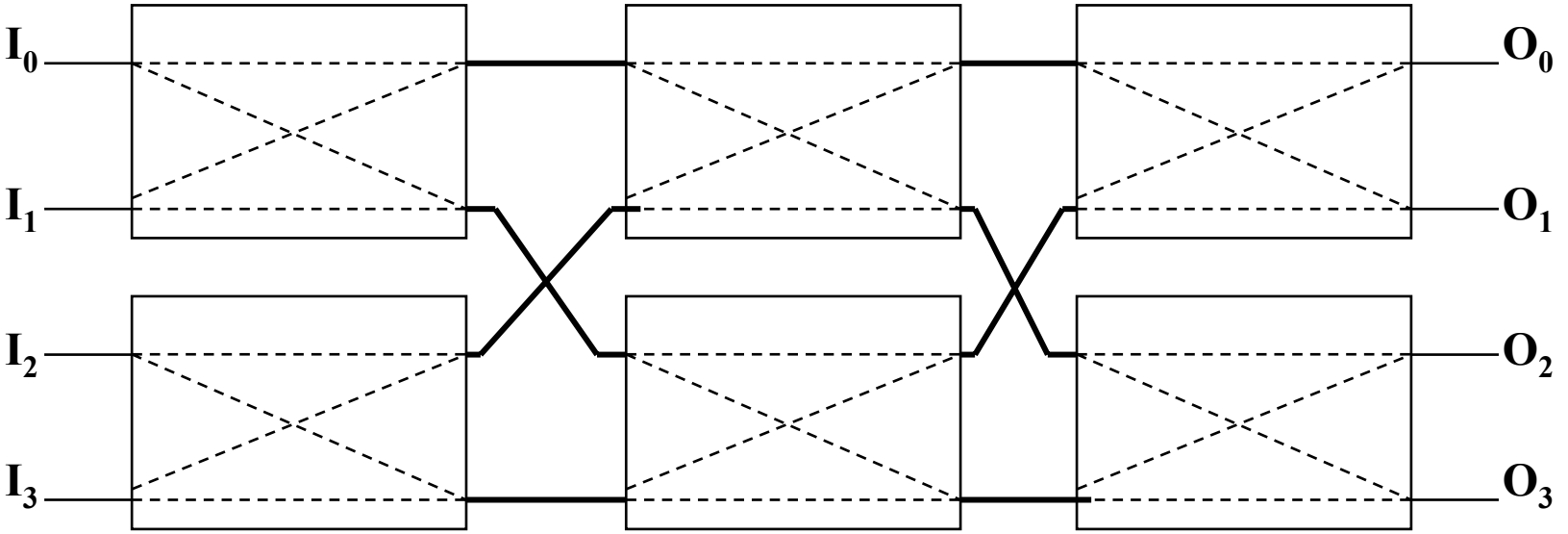
- Bei N Ein- und Ausgängen
  - Ebene (Spalte): $log_2 N$
  - Zeile: $N/2$
  - Schaltungselemente: $N/2*(log_2 N)$
- Nicht blockierungsfrei
  - Ein Verbindungsweg lässt viele andere nicht zu
- Verbesserung
  - Zusätzliche Ebene

Benes-Netzwerk

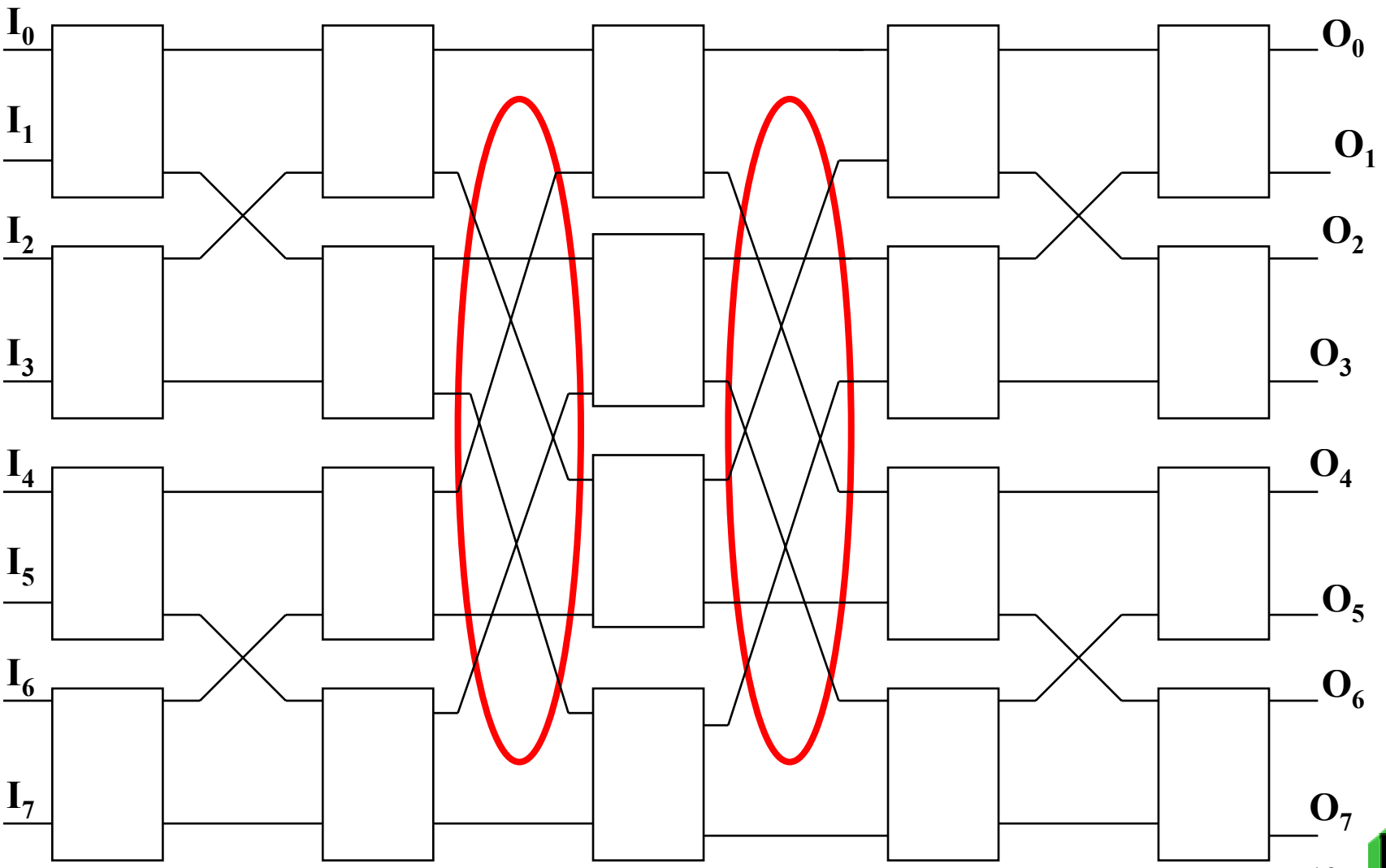
1-D:



2-D:



Benes-Netzwerk: 3-D



Benes-Netzwerk

- Bei N Ein- und Ausgängen
  - Ebene (Spalte): $2 * log_2 N - 1$
  - Zeile: $N/2$
  - Schaltungselemente: $N/2*(2 * log_2 N - 1)$
- Blockierungsfrei
  - Erzeugung jeder gewünschten Ein/Ausgangs-verbindung möglich

Aufgabe:

- Zeichnen Sie die folgenen blockierungsfreie parallele Verbindungswege

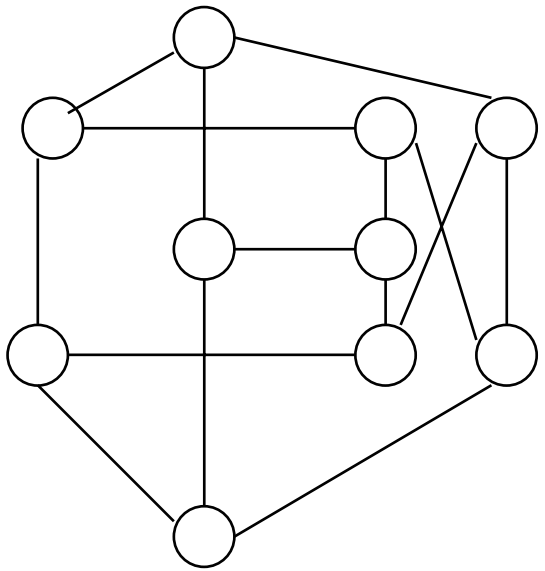
0→3, 1→5, 2→4, 3→7, 4→6, 5→0, 6→1,7→2

- Begründen Sie, dass dieses Benes-Netzwerk blockierungsfrei ist

Verbindungsnetze (II)

Aufgabe:

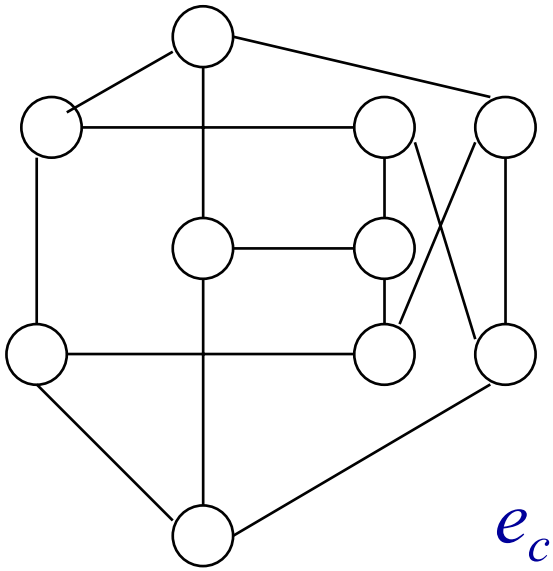
Betrachten Sie folgendes Verbindungsnetzwerk und bestimmen Sie die Kosteneffektivität. Berechnen Sie weiterhin Diameter, Diskonnektivität und minimale Bisektionsbreite



Charakterisierung von Verbindungsnetzen

- Übertragungszeit (latency)
  - Software-Overhead
  - Kanalverzögerung (channel delay)
  - Schalt- und Routingverzögerung (switching/routing delay)
  - Blockierungszeit (contention)
- Durchsatz
  - Übertragungsbandbreite (bandwidth, *b*) (MBit/s oder MB/s)
  - Bisektionsbandbreite (bisection bandwidth, *bb*)
    - Bisektionsbreite (# der Bisektionlinien, *e*)
- Durchmesser oder maximale Pfadlänge (diameter, *r*)
- Verbindungsgrad eines Knotens (node degree connectivity *P*)
- Diskonnektivität $d=n/e$ (n: # Knoten, e: minimale Bisektionsbreite)
- Kosteneffektivität $e_c = P * max\ (r,d)$

Beispiel



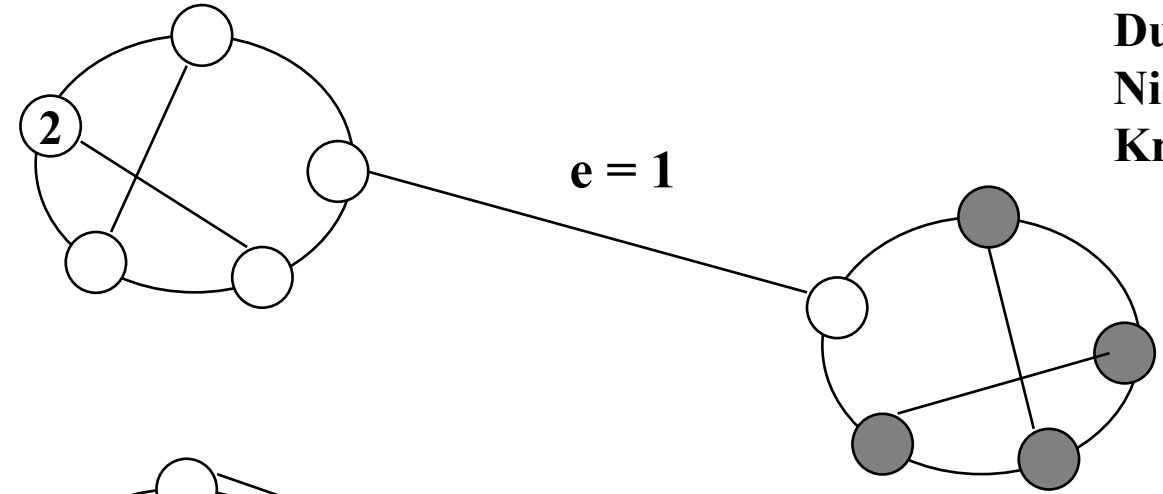
Aufgabe: berechne der Kosten-effektivität und Diskonnektivität

$e_c = P * max\ (r,d)$ wobei

- P = 3 (# direkter Verbindungen)
- r = 2 (maximale Pfadlänge)
- $d = n/e;\ n = 10$

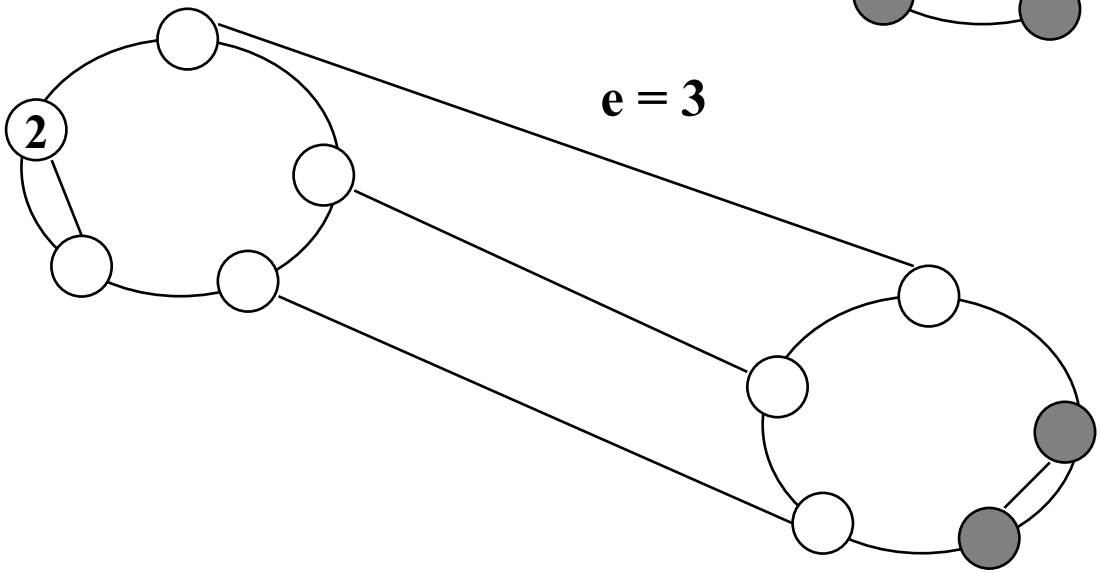
Zur Bestimmung der minimalen Bisektionsbreite wird Der Graph in zwei disjunkte Teilgraph mit je 5 Knoten Zerlegt, die einen Verbindungsgrad von 3 haben.
Es sind nur Bisektionsbreiten von 1, 3 und 5 möglich

P = 3



Dunkel gefärbten Knoten Nicht in 2 Schritten von Knoten 2 erreichbar:

Widerspruch zum Diameter von 2



Somit ergibt sich e=5

$d = n/e = 2$

$e_c = P * max\ (r,d)$

$= 3 * max\ (2,2)=6$

Parallelisierung

Aufgabe:

Zu bearbeiten ist ein 2D n*n Gitterfeld. Die Berechnung ist in zwei Phasen geteilt. In der ersten Phase wird eine Einzeloperation unabhängig auf allen Gitterpunkten durchgeführt. In der zweiten Phase wird die Summe von allen n^2 Werten der Gitterpunkte berechnet

- Erste Zerlegung und Festlegung:
  - Jeder Prozessor wird in der ersten Phase n^2/P Punkte zugeteilt. Die Berechnung benötigt n^2/P Zeitschritte.
  - In der zweiten Phase addiert jeder Prozessor die zugeteilten n^2/P Werte auf eine globale Summen-Variable.
  - Berechnen Sie den Speedup und dessen Tendenz.

Zweite Zerlegung und Festlegung:

- Jeder Prozessor wird in der ersten Phase mit n^2/P Punkten zugeteilt. Die Berechnung benötigt n^2/P Zeitschritte.
- In der zweiten Phase addiert jeder Prozessor zunächst die n^2/P Werte in eine lokale Summen-Variable und anschließend wird
- Eine globale Akkumulation auf die P lokalen Summen durchgeführt.
- Berechnen Sie den Speedup und dessen Tendenz

Parallelisierungsprozeß

- Schritt 1: Aufteilung/Dekomposition
  - Zerlegt die Berechnung in Tasks
  - Ziel: Aufdecken der potentiellen Nebenläufigkeit
- Schritt 2: Zuweisung
  - Ordnet die Tasks den Prozessen zu
  - Ziel: Lastbalancierung, geringe Kommunikation
- Schritt 3: Festlegung
  - Legt die notwendigen Datenzugriffe, Kommunikationen und Synchronisationen zwischen Prozessen fest
  - Ziel: Lokalität der Datenzugriffe
- Schritt 4: Abbildung
  - Bindet Prozessen an Prozessoren

Beispiel: Aufteilung

- Erste Zerlegung
  - Erste Phase: jeder Prozessor zugeteilt mit n^2/P Punkten und fertig in Zeit n^2/P
  - Zweite Phase: Prozessor addiert jede der zugeteilten n^2/P Werte in eine globale Summevariable
  - Leistung
    - Sequentielle Zeit: $2\ n^2$
    - Parallele Zeit: $n^2/P + n^2$
    - Speedup: $$\frac{2\ n^2}{n^2/P + n^2} = \frac{2P}{P+1} < 2$$

Beispiel: Aufteilung

- Zweite Zerlegung
  - Erste Phase: jeder Prozessor zugeteilt mit n^2/P Punkten und fertig in Zeit n^2/P
  - Zweite Phase: Prozessor addiert jede der zugeteilten n^2/P Werte in eine lokale Summevariable
  - Dritte Phase: globale Akkumulation von P lokalen Summen
    - Seriell aber nur P Operationen, anstatt n^2
  - Leistung
    - Sequentielle Zeit: $2\ n^2$
    - Parallele Zeit: $n^2/P + n^2/P + P = 2\ n^2/P + P$
    - Speedup: $$\frac{2\ n^2}{2n^2/P + P} \approx \frac{P \bullet 2\ n^2}{2\ n^2 + P^2}$$

Wenn n >> P, linear von Anzahl der Prozessoren

Problemlösung und Zielsetzung

- Zielsetzung: möglichst optimale Implementation, d.h.
  - Ausgewogene Lastverteilung (Balanzierung)
  - Minimale Interprozeßkommunikation
  - Maximale Datenlokalität

    → Minimaler Aufwand zur Laufzeit
  - Ideal: Umsetzung skaliert mit Anzahl der Knoten
- Muss bereits bei der Partitionierung berücksichtigt werden
- Unterstützt durch Werkzeuge (Leistungsanalyse, Visualisierungswerkzeuge)

Programmiermodelle

- Multiprogramming
  - Sequentielle Modelle
- Gemeinsamer Speicher
  - Shared-Memory Modelle
  - OpenMP (Standard)
- Nachrichtengekoppelt
  - Message-Passing Modelle
  - MPI (Standard)
- Datenparallelismus
  - HPF (High Performance Fortran) (Standard)

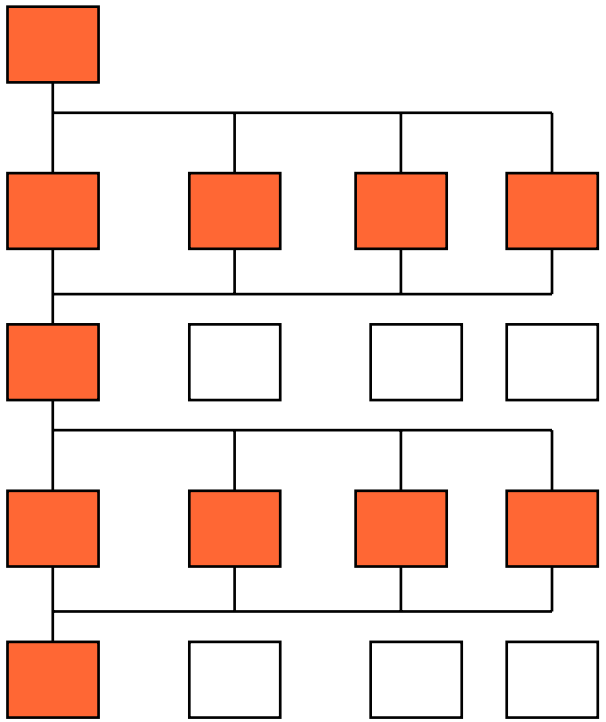
Message Passing Modell

- Benutzerspezifizierte Datenverteilung
- Mehrere Prozesse für Ausführung einer Anwendung
  - Jeder Prozess hat einen separaten Adressraum
- Nachrichtaustausch zwischen Prozessen durch explizite Send/Receive Operationen
- Vorteil
  - Steuerung der Kommunikation
- Nachteil
  - komplizierte Programmierung
  - Explizite Kommunikation

Shared Memory Modell

- Eigenschaften
  - Eine Anwendung besteht aus parallelen Threads, die auf gemeinsame Daten zugreifen
  - Implizite Datendekomposition
  - Implizite Kommunikation
- Thread-basierte SM Modelle
  - Explizite Programmierung kooperativer Threads
  - Vorteil: Portabilität
  - Nachteil: komplizierte Programmierung
- Directive-basierete SM Modelle
  - High-level parallele Operationen, z.B. parallele Schleifen mit impliziten Synchronisationen
  - Vorteil: einfache Programmierung
  - Nachteil: schwierige Steuerung von Leistung

Shared Memory Modell: Fork-Join



Datenparallelismus

- Synchrone Ausführung paralleler Operationen auf große verteilte Datenstrukturen
- Benutzerspezifizierte Datenverteilung
- Implizite Kommunikation
- Vorteil: High-level Programmierung
- Nachteil: Leistungsproblem