Karlsruher Institut für Technologie

Übung 5 – Algorithmen II

Hans-Peter Lehmann, Daniel Seemaier – *{hans-peter.lehmann, daniel.seemaier}@kit.edu*

`http://algo2.iti.kit.edu/AlgorithmenII_WS21.php`

Institut für Theoretische Informatik - Algorithmik II

Themenübersicht

- Parallelverarbeitung
  - Maschinenmodelle
  - Hyperwürfel Präfixsumme
  - Paralleler Quicksort
- Flüsse
  - Ford Fulkerson
  - Dinitz

Parallelverarbeitung

Modelle

PRAM (Shared Memory)

- synchrone Prozessoren
- gemeinsamer Speicher
- Speicherkonflikte

(symmetrisch) gemeinsamer Speicher

Verteilter Speicher (Distributed Memory)

Bulk**S**ynchronous**P**arallel

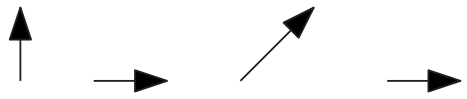
- kollektiver Nachrichtenaustausch aller Rechner
- BSP* berücksichtigt Nachrichtenlänge

Verbindungsnetzwerke

Struktur

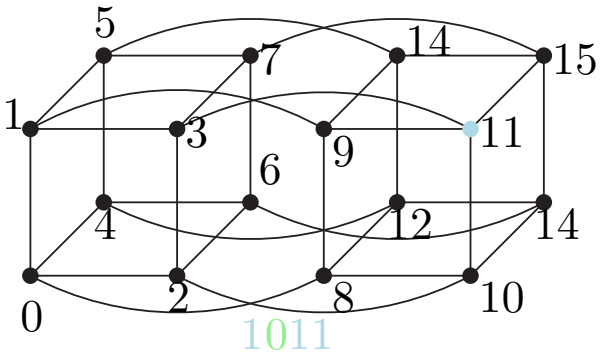
Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p\cdot(p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex $i \to j$
  - Telefon $i \leftrightarrow j$
  - Duplex $i \to j, k \to i$



Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $***1** \leftrightarrow ***0**$

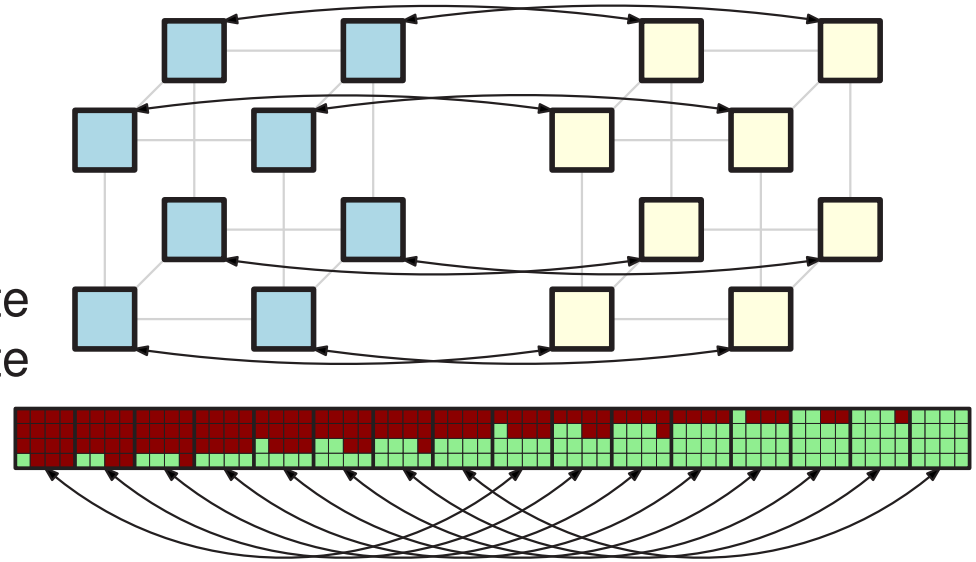


Kosten

- Kostenmaß Kommunikation $T_{comm} = T_{start} + l \cdot T_{byte}$

Anwendungen

Präfixsumme - Hypercube



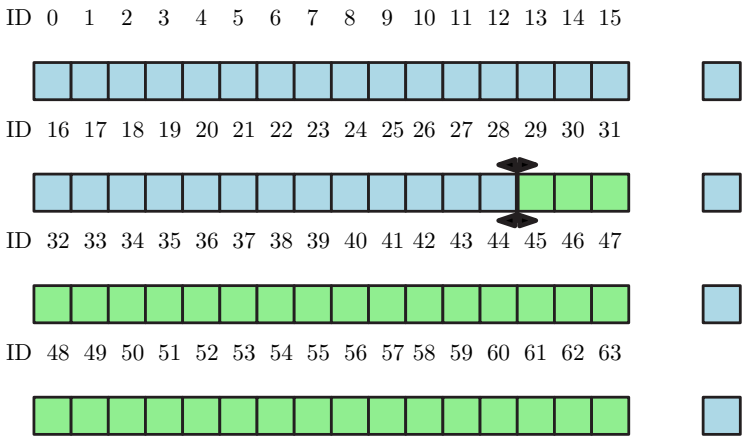
- Jede CPU speichert zwei Werte
  1. Summe aller bekannten Elemente
  2. Summe aller bekannten Elemente von CPUs mit kleinerer ID
- In Schritt k tauschen CPUs entlang der k-ten Dimension ihre Summen aus.
  - $\mathrm{ID}[k] = 1$: Summe aller Elemente beim Kommunikationspartner kommt von CPUs mit kleinerer ID $\rightarrow$ gehört zur Präfixsumme
  - $\mathrm{ID}[k] = 0$: Erhaltene Daten gehören zu CPUs größerer ID $\rightarrow$ gehört nicht zur Präfixsumme
- $T(n,p) = (T_{start} + l \cdot T_{byte}) \cdot \log p$

Anwendungen

Paralleler Quicksort

Rekursives Verfahren

1. ein PE stellt Pivot zufällig
2. Broadcast
3. lokaler Vergleich
4. kleine Elemente durchnummerieren
   $\rightarrow$ Präfixsumme
5. umverteilen
   - Präfixsumme für große Elemente folgt direkt aus ID und Präfixsumme kleiner Elemente
   - Position kleine Elemente ist Präfixsummenwert
   - Position großer Elemente ist Anzahl kleiner Elemente plus Wert der Präfixsumme für große Elemente
6. Prozessoren aufspalten Problematisch bei unbalancierter Verteilung
7. parallele Rekursion



Parallele Programmierung

Ein Einstieg

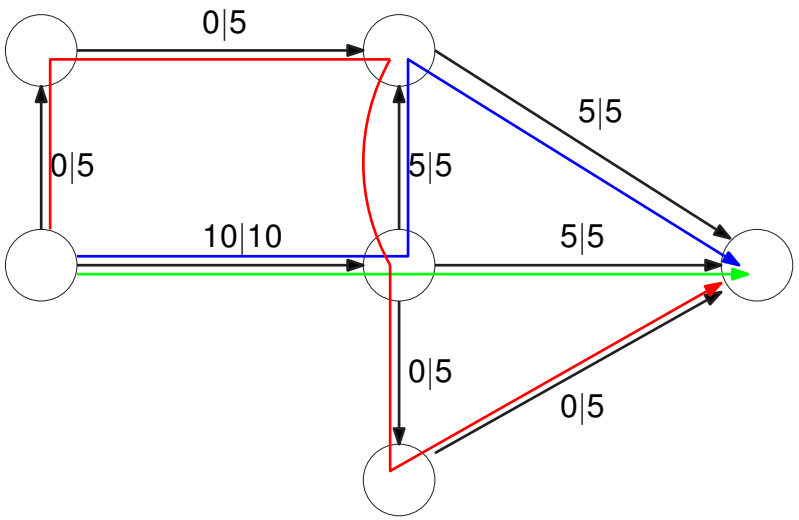
Karlsruher Institut für Technologie

Einstieg in parallele Programmierung?

- OpenMP
  - `www.openmp.org`
  - enthalten im GCC Compiler
  - Parallelität über Preprozessorflags #pragma omp parallel
- Intel Thread Building Block Library (TBB)
  - `https://software.intel.com/en-us/tbb`
  - mehr objektorientiert als *OpenMP*
  - enthält konkurrente Datenstrukturen und viele parallele Primitive
    - Konkurrente *Queues*, *Arrays*, ...
    - Parallel *Sort*, *For*, *While*, ...
    - Komplexe *Dataflow*-Graphen
    - Geschachtelter und rekursiver Parallelismus
- Message Passing Interface (MPI)
  - `https://www.mcs.anl.gov/research/projects/mpi/`
  - Standard für verteilte Programmierung
  - implementiert die gängisten Kommunikationsprimitiven (`C`-Style Interface)

Ford Fulkerson

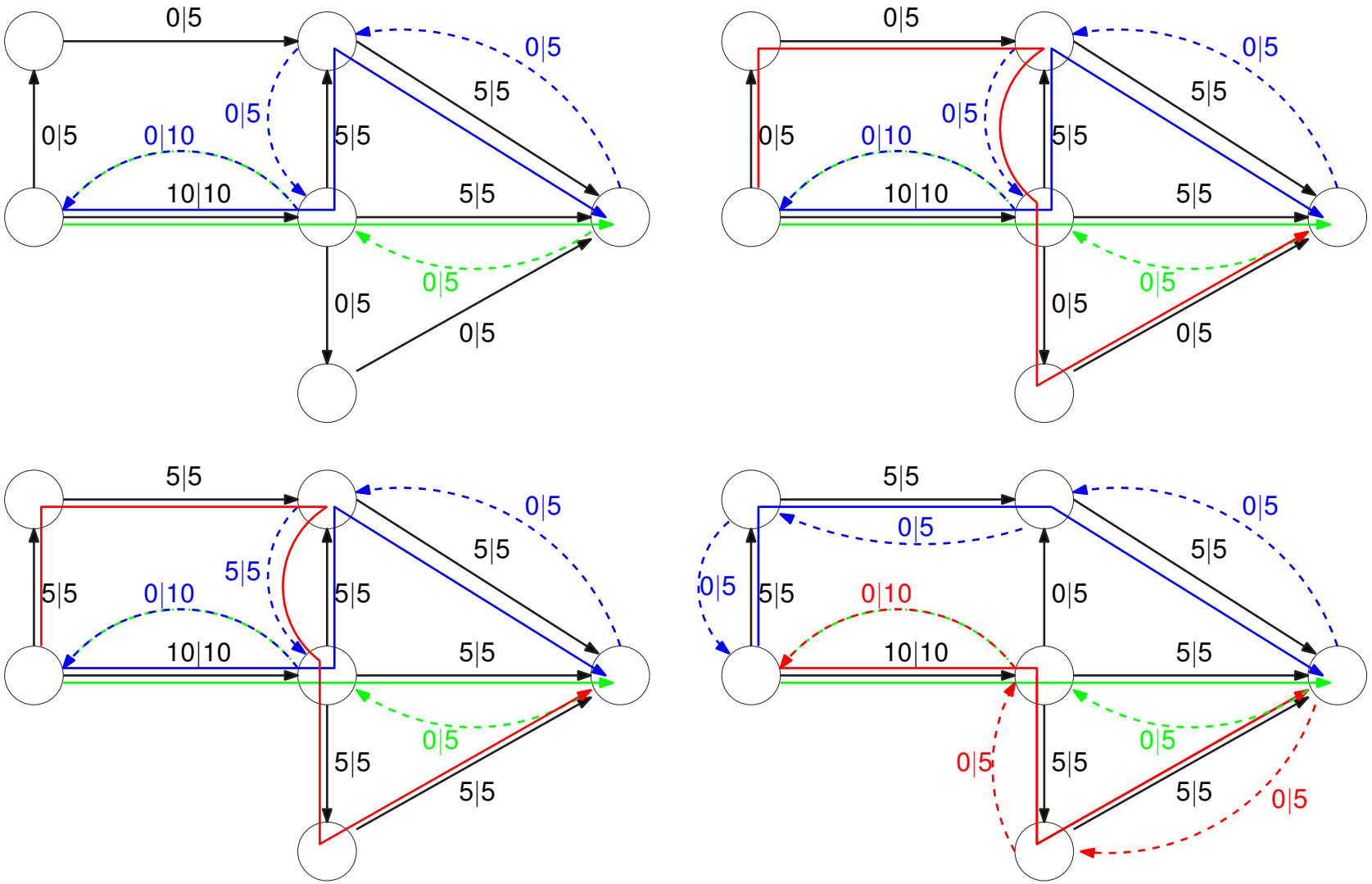
Flüsse und Ford Fulkerson



Residualgraph

- Verwalten von Restkapazitäten
- Modellierung und Erkennung von “Gegenflüssen”
- $c^f(e) = c(e) - f(e)$: Restkapazität
  Hier: Fluss $f(e)$ und Gesamtkapazität $c(e)$
- $c^f(e^{\text{rev}}) = f(e)$: Fluss über Kante e
  Hier: Restkapazität und Gesamtkapazität von e
- Keine 0-Gewicht Kanten
- Flüsse über Kanten e und e^{rev} erlaubt
  - Fluss über Kante $\rightarrow$ Update beider Kanten

Flüsse und Ford Fulkerson



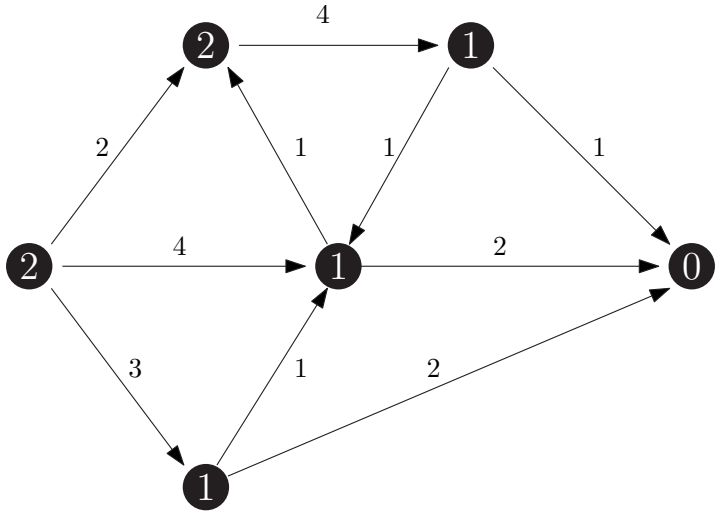
Dinitz Algorithmus

Dinitz

Distanz Label

- Geben Distanz im Residualgraphen (hop-based) zur Senke t an
- Rückwärtsgerichtete Breitensuche
  - Start bei Knoten t
  - Layer: Knoten mit gleicher Distanz zu s im BFS-Baum
  - Knoten in einem Layer: gleiches Label
- *Layered graph*

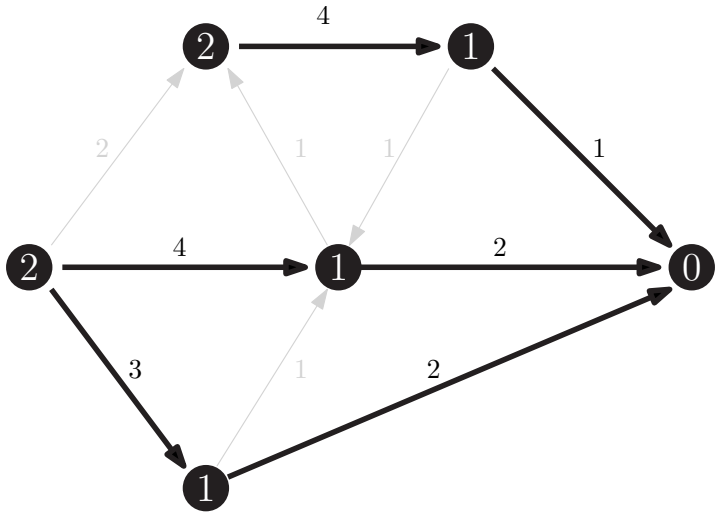
  auch: kürzeste Wege Netzwerk, Schichtgraph



Dinitz

Schichtgraph für Graph $G = (V, E)$

- Knotenmenge $V_S = V$
- $E' = \{e = (u, v) \in E | f(e) < c(e),$
  $d(u) = d(v) + 1\}$
  $E'^{rev} = \{e^{rev} = (v, u)^{rev} | f(v, u) > 0,$
  $d(u) = d(v) + 1\}$
  Kantenmenge $E_S = E' \cup E'^{rev}$
- betrachte Analogie zu Edmonds-Karp (kürzeste erhöhende Wege)

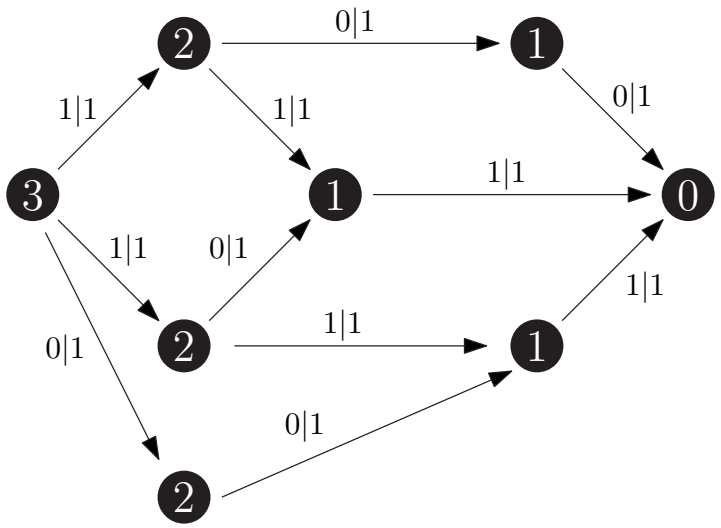


Dinitz

Blockierender Fluss

- Kein weiterer Fluss möglich

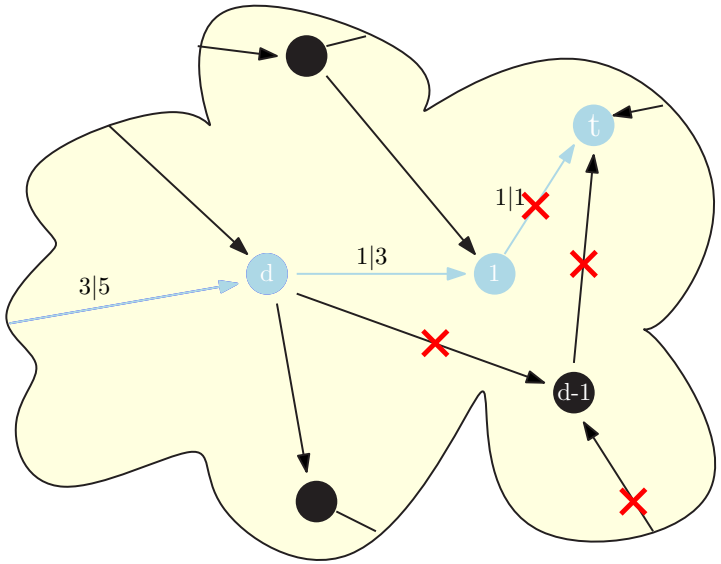
  "Auf jedem Weg durch den Graphen mindestens eine Kante bis zur maximalen Kapazität ausgelastet ist"
- *Blocking flow als atomare Operation*
  - Berechnung auf Schichtgraph
  - Kein Residualgraph
  - Kein Rückfluss möglich
- i.A. nicht maximal auf Schichtgraph und Redisualgraph



Dinitz

Blockierender Fluss: Operationen

- *Blocking flow*: Berechnung basiert auf Tiefensuche von Knoten s
- Drei Operationen
  - **extend** - gehe einen Knoten näher ans Ziel (Schichtgraph)
  - **retreat** - Sackgasse gefunden, gehe zurück, lösche Kante
  - **breakthrough** - Tiefensuche hat Senke erreicht, lösche saturierte Kanten



Dinitz

Kosten pro Blockierendem Fluss

- #breakthrough $\leq m$
  - Jedes breakthrough saturiert mindestens eine Kante
    $\rightarrow$ Kein breakthrough über saturierte Kante mehr möglich
  - **Laufzeit** $O(n)$
- #retreat $\leq m$
  - Jedes retreat löscht eine Kante
  - **Laufzeit** $O(1)$
- #extends $\leq$ #retreats $+ n \cdot$ #breakthrough
  - **Retreat:** Vorher **ein extend**
    Ohne breakthrough nur retreats
  - **Breakthrough:** Vorher $\leq n$ **erfolgreiche extends**
    Schichtgraph schließt Kreise aus
  - **Laufzeit** $O(1)$
- Blockierender Fluss in $O(nm)$

Dinitz

Laufzeit

- Pro Phase
  - Rückwärtsgerichtete Breitensuche: **Laufzeit** $O(n+m)$
  - Blockierender Fluss: **Laufzeit** $O(nm)$
- Phase in $O(nm)$
- #Phasen $\leq n$ (Beweis, wie in Vorlesung, nicht hier)

  (Lemma 1: Jede Phase erhöht Label von s um mindestens 1)

  $\rightarrow$ Gesamtlaufzeit $O(n^2 m)$

Ende!

Feierabend!