Algorithmen für Routenplanung

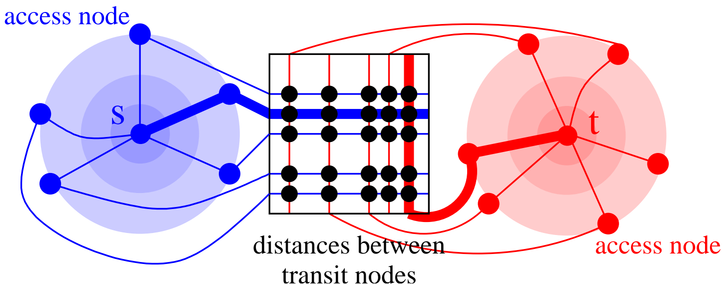
9. Sitzung, Sommersemester 2013

Julian Dibbelt | 27. Mai 2013

INSTITUT FÜR THEORETISCHE INFORMATIK · ALGORITHMIK · PROF. DR. DOROTHEA WAGNER

Letztes Mal

Transit-Node Routing



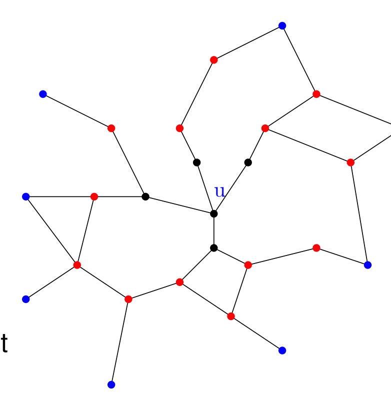
ML: “Runter”propagieren von ANs

Idee: Access-Nodes können propagiert werden

- Berechne $\overrightarrow{A}_\ell(v)$ für alle $v \in T_{\ell-1}$
- Berechne $\overrightarrow{A}_{\ell-1}$ für alle $u \in V$
- Schließe von $\ell - 1$ auf ℓ:

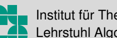
$$\overrightarrow{A}_\ell(u) := \bigcup_{v \in \overrightarrow{A}_{\ell-1}(u)} \overrightarrow{A}_\ell(v)$$

- kann auf mehrere Level verallgemeinert werden
  Berechne ANs für TNs “runter”,
  berechne ANs für “normale” Knoten “hoch”
- können mit Distanztabellen wieder ausgedünnt werden



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Kombinationen



Fünf Bausteine

Landmarken

L2
t
s
L1

Bidirektionale Suche

s
t

Kontraktion

Arc-Flags

Table-Lookups

access node
t
distances between transit nodes
s
access node

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

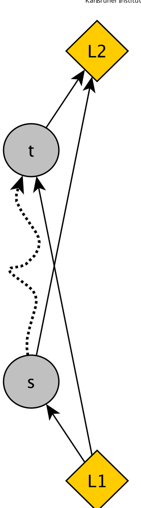
Landmarken [Goldberg, Harrelson 05]

Vorberechnung:

- wähle eine Hand voll ($\approx$ 16) Knoten als Landmarken
- berechne Abstände von und zu allen Landmarken

Anfrage:

- benutze Landmarken und Dreiecksungleichung um eine untere Schranke für den Abstand zum Ziel zu bestimmen
- verändert Reihenfolge der besuchten Knoten
- bevorzugt Knoten zwischen Start- und Zielknoten
- bekannt als ALT: A*, Landmarken, Triangle Inequality



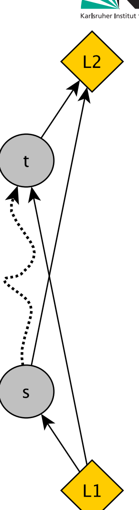
Landmarken

Vorteile:

+ Anpassung einfach
+ einfache und schnelle Vorberechnung
+ sehr robust gegenüber der Eingabe
+ zusätzliche Staus können berücksichtigt werden

Nachteile:

– geringe Beschleunigung ("nur" Faktor von ca. 100)
– dies auch nur mit bidirektionaler Suche erreichbar
– Vorberechnung benötigt viel Speicher (2 Distanzen pro Knoten und Landmarke)

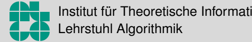


Bidirektionale Suche

s

t

- starte zweite Suche von t
- relaxiere rückwärts nur eingehende Kanten
- stoppe die Suche, wenn beide Suchräume sich treffen



Kontraktion [Sanders,Schultes 05]

Beobachtung:

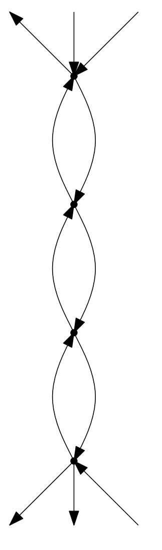
- Knoten mit niedrigem Grad sind unwichtig

Knoten-Reduktion:

- entferne diese Knoten iterativ
- füge neue Kanten (Abkürzungen) hinzu, um die Abstände zwischen verbleibenden Knoten zu erhalten

Kanten-Reduktion:

- behalte nur relevante Shortcuts
- lokale Suche während oder nach Knoten-reduktion



Kontraktion [Sanders,Schultes 05]

Beobachtung:

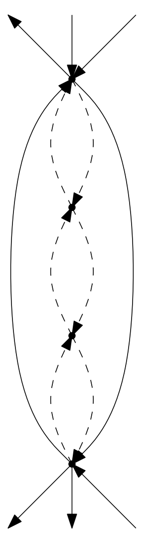
- Knoten mit niedrigem Grad sind unwichtig

Knoten-Reduktion:

- entferne diese Knoten iterativ
- füge neue Kanten (Abkürzungen) hinzu, um die Abstände zwischen verbleibenden Knoten zu erhalten

Kanten-Reduktion:

- behalte nur relevante Shortcuts
- lokale Suche während oder nach Knoten-reduktion

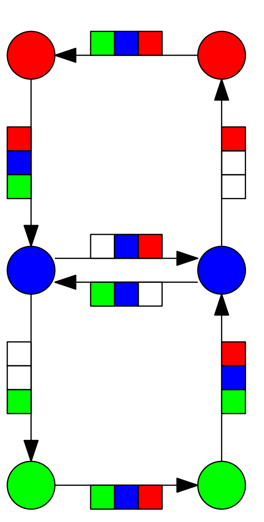


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Arc-Flags [Lauther04],[Möhring et al. 05,06]

Idee:

- partitioniere den Graph in *k* Zellen
- hänge ein Label mit *k* Bits an jede Kante
- zeigt ob *e* wichtig für die Zielzelle ist
- modifizierter Dijkstra überspringt unwichtige Kanten



Diskussion:

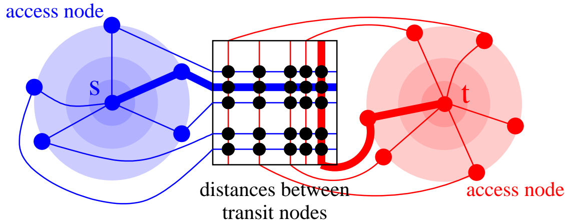
+ einfacher Suchalgorithmus
+ schnell
– lange Vorberechnungszeit
– keine Vorteile in Zielzelle

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Table-Lookups

Idee:

- speichere Distanztabellen
- nur für “wichtige” Teile des Graphen
- Suchen laufen nur bis zur Tabelle
- harmoniert gut mir hierarchischen Techniken



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Core-ALT

Landmarken	Bidirektionale Suche	Kontraktion	Arc-Flags	Table-Lookups

L2
t
s
L1

s
t

Core-ALT

Motivation:

- ALT ist robust gegenüber der Eingabe
- aber hoher Speicherverbrauch

Hauptidee:

- beschränke ALT nur im Kern

Core-ALT

(Landmarken, bidirektionale Suche, Kontraktion)

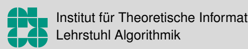
Idee

- begrenze Beschleunigungstechnik auf kleinen Subgraphen (Kern)

s ● ● t

Vorberechnung

- kontrahiere Graphen zu einem Kern
- Landmarken nur im Kern

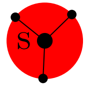


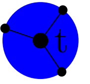
Core-ALT

(Landmarken, bidirektionale Suche, Kontraktion)

Idee

- begrenze Beschleunigungstechnik auf kleinen Subgraphen (Kern)



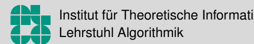


Vorberechnung

- kontrahiere Graphen zu einem Kern
- Landmarken nur im Kern

Anfrage

- Initialphase: normaler Dijkstra

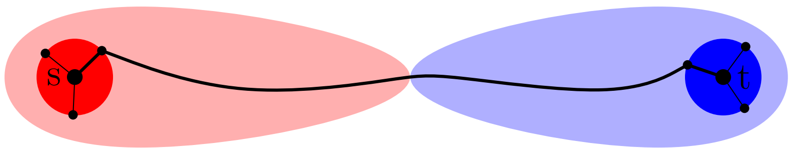


Core-ALT

(Landmarken, bidirektionale Suche, Kontraktion)

Idee

- begrenze Beschleunigungstechnik auf kleinen Subgraphen (Kern)



Vorberechnung

- kontrahiere Graphen zu einem Kern
- Landmarken nur im Kern

Anfrage

- Initialphase: normaler Dijkstra
- benutze Landmarken nur im Kern

Proxy Knoten

Problem:

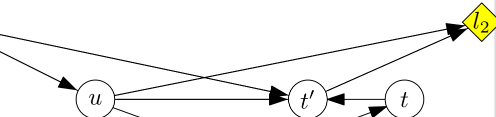
- ALT braucht Potential von jedem Knoten zu t und s
- s und/oder t könnten außerhalb des Kerns liegen
- somit keine Abstandswerte von den Landmarken zu s und t

Proxy Knoten

Problem:

- ALT braucht Potential von jedem Knoten zu t und s
- s und/oder t könnten außerhalb des Kerns liegen
- somit keine Abstandswerte von den Landmarken zu s und t

Lösung:



- bestimme Proxy-Knoten t' für t (s analog), t' im Kern
- neue Ungleichungen:

$$d(u,t) \geq d(u,l_2) - d(t',l_2) - d(t,t')$$
$$d(u,t) \geq d(l_1,t') - d(l_1,u) - d(t,t')$$

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Einfluss Kontraktion

input	c	h	PREPRO. core nodes	$\|E\|$ incr.	time [min]	space [B/n]	QUERY #settled nodes	#entry nodes	time [ms]
Europe	0.0	0	100.00%	0.00%	68	512.0	25 324	1.0	19.61
	0.5	10	35.48%	10.23%	21	187.7	10 925	3.2	8.02
	1.0	20	6.32%	14.24%	7	38.4	2 233	8.2	2.16
	2.0	30	3.04%	11.41%	9	21.8	1 382	13.3	1.55
	2.5	50	1.88%	9.16%	11	15.4	1 394	18.6	1.34
	3.0	75	1.29%	7.80%	12	12.2	1 963	24.2	1.43
	5.0	100	0.86%	6.94%	18	9.8	3 126	34.0	1.67
USA	0.0	0	100.00%	0.00%	93	512.0	68 861	1.0	48.87
	0.5	10	28.90%	11.40%	20	154.2	21 544	3.4	16.61
	1.0	20	8.29%	12.68%	11	48.9	7 662	7.1	6.96
	2.0	30	3.21%	10.53%	12	22.5	3 338	12.6	4.11
	2.5	50	2.06%	8.00%	13	16.1	2 697	17.1	3.01
	3.0	75	1.45%	6.39%	14	12.6	2 863	22.0	2.85
	5.0	100	0.86%	4.50%	21	9.3	3 416	30.2	2.35

Einfluss Anzahl Landmarken

$\|L\|$	no cont. (c=0.0, h=0) PREPRO. time [min]	space [B/n]	QUERY #settled nodes	time [ms]	low cont. (c=1.0, h=20) PREPRO. time [min]	space [B/n]	QUERY #settled nodes	time [ms]
8	26.1	64	163 776	127.8	7.1	10.9	12 529	10.25
16	85.2	128	74 669	53.6	9.4	14.9	5 672	5.77
32	27.1	256	40 945	29.4	6.8	23.0	3 268	2.97
64	68.2	512	25 324	19.6	8.5	36.2	2 233	2.16

$\|L\|$	med: c=2.5, h=50 PREPRO. time [min]	space [B/n]	QUERY #settled nodes	time [ms]	high: c=5.0, h=100 PREPRO. time [min]	space [B/n]	QUERY #settled nodes	time [ms]
8	10.1	7.0	4 431	3.98	17.8	5.9	4 106	2.51
16	11.0	8.2	2 456	2.33	18.3	6.5	3 500	2.23
32	10.0	10.6	1 704	1.66	17.7	7.6	3 264	2.01
64	10.5	15.4	1 394	1.34	18.0	9.8	3 126	1.67

Lokale Anfragen

ALT (16 landmarks)
CALT low contraction (64 landmarks)
CALT medium contraction (64 landmarks)

Query Time [ms]

1000
100
10
1
0.1
0.01

2^{6} 2^{7} 2^{8} 2^{9} 2^{10} 2^{11} 2^{12} 2^{13} 2^{14} 2^{15} 2^{16} 2^{17} 2^{18} 2^{19} 2^{20} 2^{21} 2^{22} 2^{23} 2^{24}

Dijkstra Rank

Whd.: Multi-Level Arc-Flags

Problem:

- Anfragen in einer Zelle ohne Beschleunigung
- viele Real-Welt Anfragen sind lokal

Multi-Level Arc-Flags:

- Multi-Level Partition
- berechne partielle Flaggen

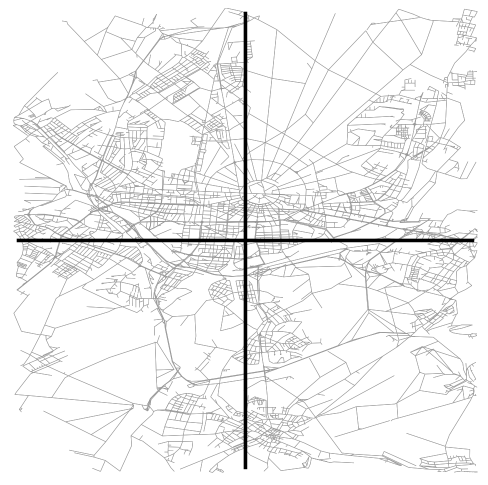
Whd.: Multi-Level Arc-Flags

Problem:

- Anfragen in einer Zelle ohne Beschleunigung
- viele Real-Welt Anfragen sind lokal

Multi-Level Arc-Flags:

- Multi-Level Partition
- berechne partielle Flaggen



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

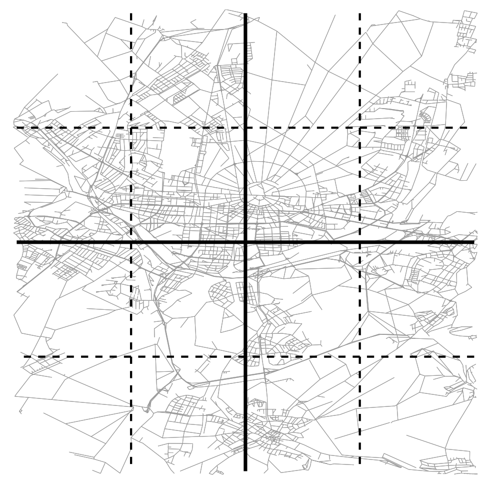
Whd.: Multi-Level Arc-Flags

Problem:

- Anfragen in einer Zelle ohne Beschleunigung
- viele Real-Welt Anfragen sind lokal

Multi-Level Arc-Flags:

- Multi-Level Partition
- berechne partielle Flaggen



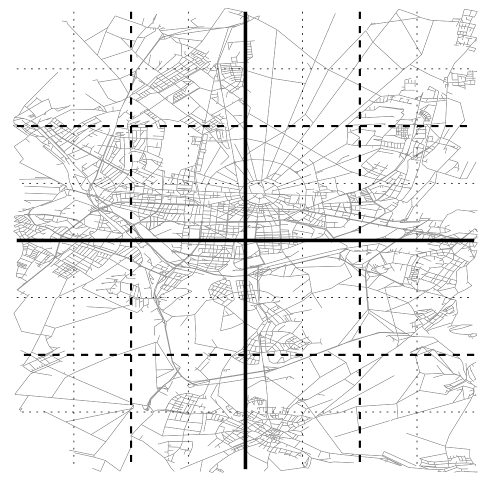
Whd.: Multi-Level Arc-Flags

Problem:

- Anfragen in einer Zelle ohne Beschleunigung
- viele Real-Welt Anfragen sind lokal

Multi-Level Arc-Flags:

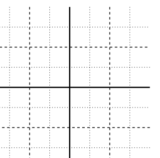
- Multi-Level Partition
- berechne partielle Flaggen



Whd.: Multi-Level Arc-Flags

Vorberechnung:

- oberster Level wie gehabt
- auf unteren Leveln:
    - von jedem Randknoten führe Dijkstra aus, bis alle Knoten der Superzelle abgearbeitet worden sind
    - setze Flaggen für alle Kanten (u, v) für die u in Superzelle
    - Hinweis: es reicht nicht, nur den Subgraphen der Superzelle zu betrachten (Übungsaufgabe)



Anfragen:

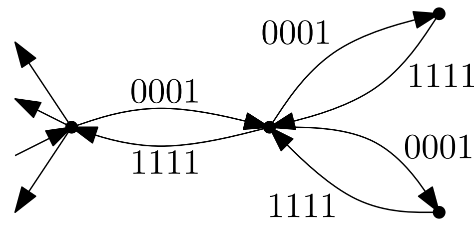
- bestimme gemeinsamen Level l von u und t
- werte Flaggen auf dem Level l aus

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Wdh.: Automatisches Setzen von Flaggen

Beobachtung:

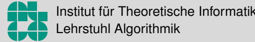
- Für manche Kanten kann man die Flaggen automatisch setzen



Angehangene Bäume:

- Kanten zur Wurzel hin haben alle Flaggen gesetzt
- Kanten von Wurzel weg haben nur eine Flagge gesetzt
- Also: Entferne Bäume vor Arcs-Flag-Berechnung aus Graph
- Knotenzahl verringert sich um 1 Drittel

Anmerkung: Alle Knoten eines angehangenen Baumes müssen zur gleichen Zelle wie dessen Wurzel gehören.



SHARC

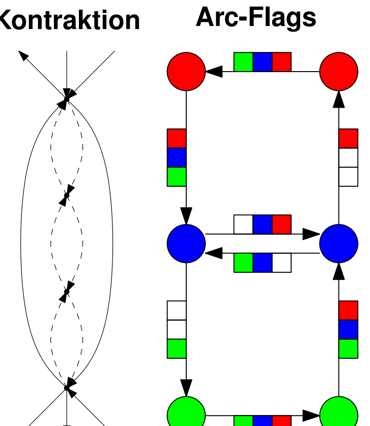
Landmarken

Bidirektionale Suche

Kontraktion

Arc-Flags

Table-Lookups



SHARC

Motivation:

- unidirektional
- Vorberechnungszeiten von Arc-Flags reduzieren

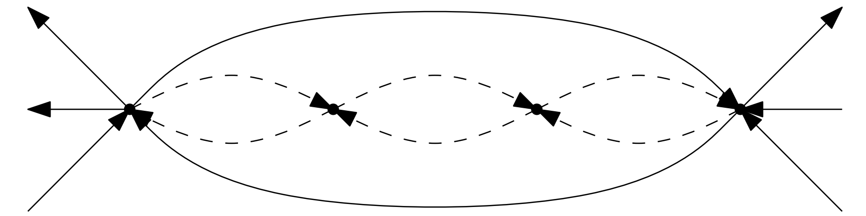
Ideen:

- Multi-Level Arc-Flags
- Integration von Kontraktion
- Verfeinern von Flaggen

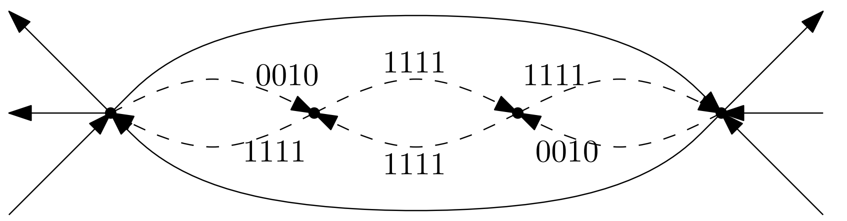
Integration von Kontraktion

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Integration von Kontraktion



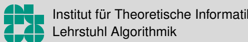
Integration von Kontraktion



Idee:

- sub-optimale flaggen für entfernte Kanten
- in die Komponente, nur own-cell Flagge
- sonst volle Flaggen

- Vorberechnung nur auf kontrahierten Graph
- sub-optimale Flaggen nur am Anfang und Ende der Query
- Überspringen von Kanten automatisch durch Arc-Flags Query



SHARC (Kontraktion, Arc-Flags)

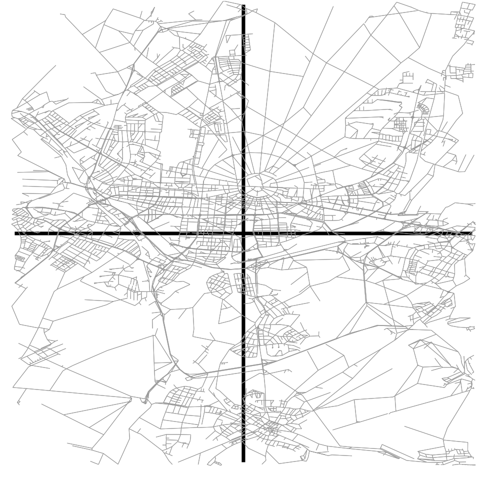
Vorberechnung:

- Multi-Level-Partition

SHARC (Kontraktion, Arc-Flags)

Vorberechnung:

- Multi-Level-Partition

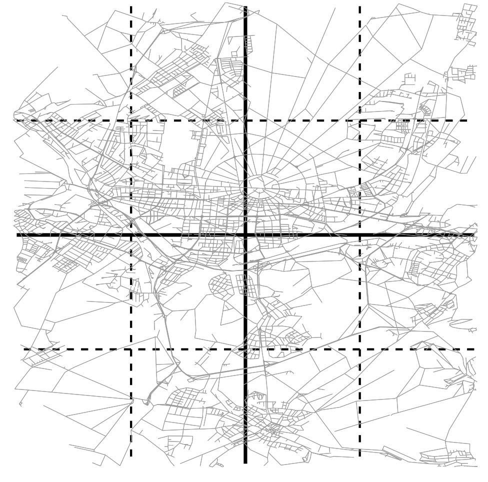


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

SHARC (Kontraktion, Arc-Flags)

Vorberechnung:

- Multi-Level-Partition

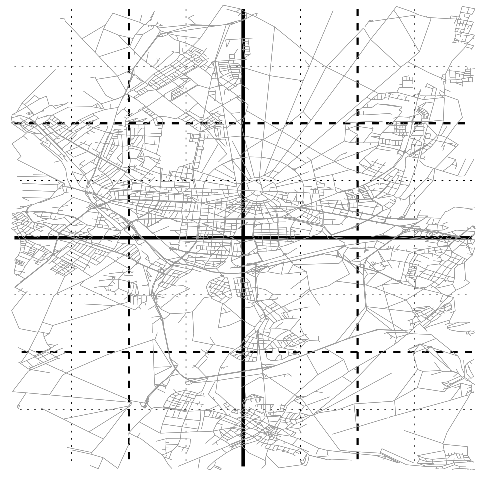


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

SHARC (Kontraktion, Arc-Flags)

Vorberechnung:

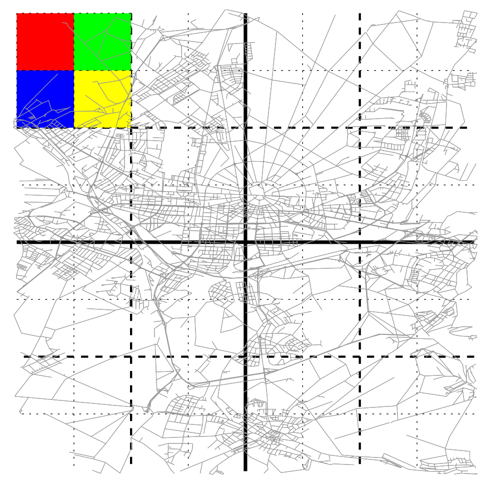
- Multi-Level-Partition
- iterativer Prozess:
  - kontrahiere Subgraphen



SHARC (Kontraktion, Arc-Flags)

Vorberechnung:

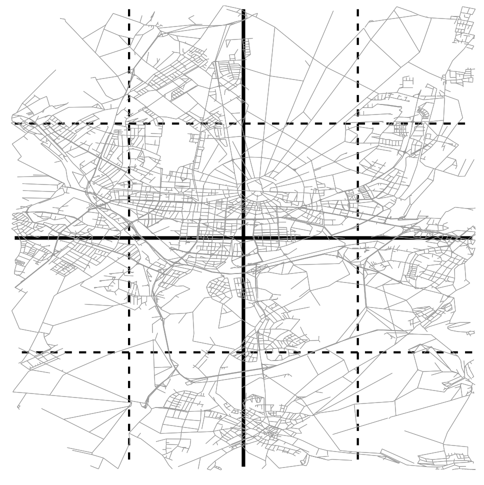
- Multi-Level-Partition
- iterativer Prozess:
    - berechne Flaggen



SHARC (Kontraktion, Arc-Flags)

Vorberechnung:

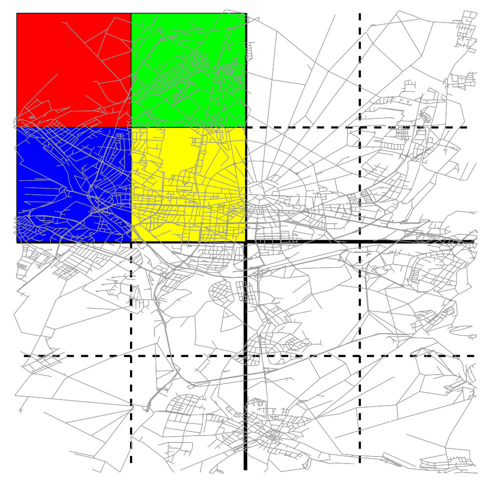
- Multi-Level-Partition
- iterativer Prozess:
  - kontrahiere Subgraphen



SHARC (Kontraktion, Arc-Flags)

Vorberechnung:

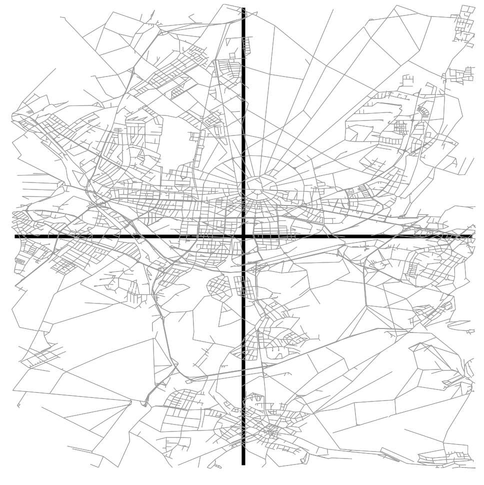
- Multi-Level-Partition
- iterativer Prozess:
  - berechne Flaggen



SHARC (Kontraktion, Arc-Flags)

Vorberechnung:

- Multi-Level-Partition
- iterativer Prozess:
  - kontrahiere Subgraphen

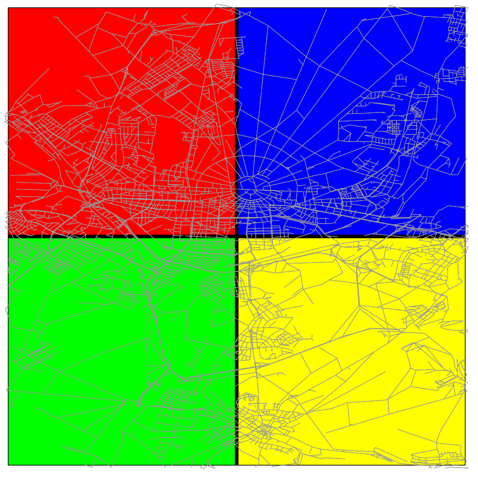


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

SHARC (Kontraktion, Arc-Flags)

Vorberechnung:

- Multi-Level-Partition
- iterativer Prozess:
  - kontrahiere Subgraphen
  - berechne Flaggen



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

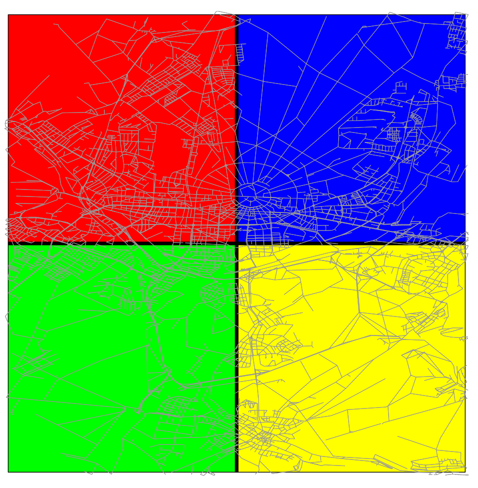
SHARC (Kontraktion, Arc-Flags)

Vorberechnung:

- Multi-Level-Partition
- iterativer Prozess:
  - kontrahiere Subgraphen
  - berechne Flaggen

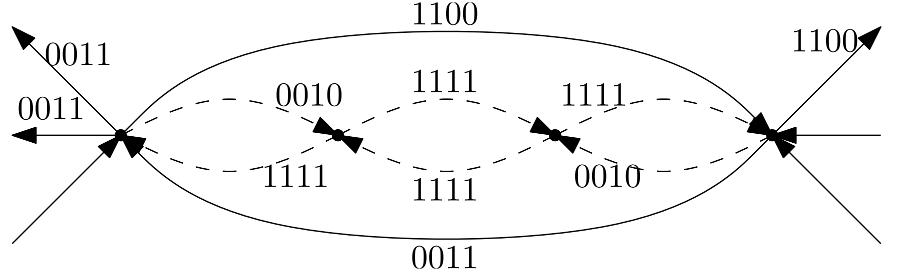
Anfragen:

- unidirektional
- hierarchisch (Kontraktion)
  LowestCommonSupercell(u,t)
- zielgerichtet (Arc-Flags)



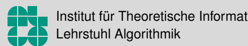
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Verfeinerung



Beobachtung:

- Flaggen schlechter als nötig
- geht es besser?



Verfeinerung

1100
0011
1100
0011
0010
1110
1110
0011
0011
0010
0011
0011

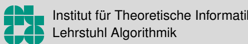
Beobachtung:

- Flaggen schlechter als nötig
- geht es besser?

Idee:

- verfeinere Flaggen
- propagiere Flaggen von wichtigen zu unwichtigen Kanten
- mittels lokaler Suche

$\Rightarrow$ sehr gute Flaggen



Partielle Flaggen

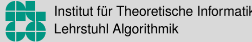
Partielle Flaggenberechnung:

- Flaggen auf unteren Leveln unwichtig
- nur wenige werden am Ende relaxiert
- setze einfach alle Flaggen auf 1
- offensichtlich korrekt

Partielle Verfeinerung:

- verfeinere nur Knoten auf hohen Leveln
- dadurch zu Beginn einer Anfrage schlechte Flaggen

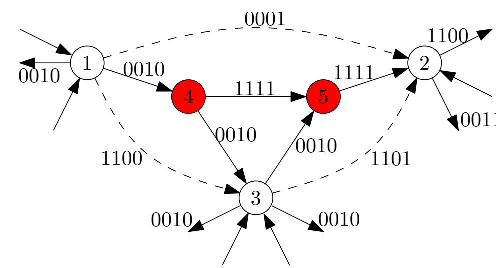
Gutes Setup: Flaggenberechnung: Level 4-5, Verfeinerung: 1-5
(Level 0: Originalgraph, Level 5: gröbste Partition)



Kompression

Beobachtung:

- Shortcut entspricht einem Pfad
- manche Shortcuts erscheinen unwichtig



Idee:

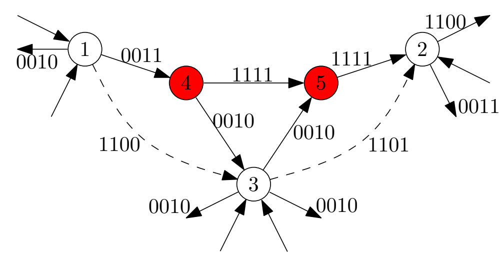
- entferne (manche) Shortcuts nach Vorberechnung
- vererbe Flaggen an erste Kante des Pfades
- Extremfall: entferne alle Shortcuts

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Kompression

Beobachtung:

- Shortcut entspricht einem Pfad
- manche Shortcuts erscheinen unwichtig



Idee:

- entferne (manche) Shortcuts nach Vorberechnung
- vererbe Flaggen an erste Kante des Pfades
- Extremfall: entferne alle Shortcuts

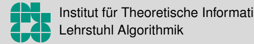
PARTITION										PREPRO		QUERY		
#cells per level									⌀#nodes	time	space	#settled	#rel.	time
l_0	l_1	l_2	l_3	l_4	l_5	l_6	l_7	#total cells	per cell	[h:m]	[B/n]	nodes	edges	[μs]
128	-	-	-	-	-	-	-	128	140 704.5	1:52	6.0	78 429	178 103	23 306
8	120	-	-	-	-	-	-	960	18 760.6	1:14	9.8	11 362	26 323	3 049
4	4	120	-	-	-	-	-	1 920	9 380.3	1:24	10.6	5 982	14 128	1 637
4	8	116	-	-	-	-	-	3 712	4 851.9	1:25	10.8	3 459	8 372	983
8	8	112	-	-	-	-	-	7 168	2 512.6	1:36	11.5	2 182	5 389	667
16	16	96	-	-	-	-	-	24 576	732.8	2:12	13.1	1 217	3 169	428
4	4	4	116	-	-	-	-	7 424	2 425.9	1:20	11.2	2 025	5 219	625
4	4	8	112	-	-	-	-	14 336	1 256.3	1:14	11.6	1 320	3 544	441
4	8	16	100	-	-	-	-	51 200	351.8	1:17	13.1	819	2 357	319
4	4	4	4	112	-	-	-	28 672	628.1	1:12	12.0	957	2 827	360
4	4	8	8	104	-	-	-	106 496	169.1	1:18	13.7	700	2 153	294
4	4	4	16	100	-	-	-	102 400	175.9	1:16	13.7	703	2 162	295
4	8	8	16	92	-	-	-	376 832	47.8	1:30	16.0	663	2 046	288
4	4	4	4	4	108	-	-	110 592	162.9	1:15	13.6	695	2 263	299
4	**4**	**4**	**4**	**8**	**104**	**-**	**-**	**212 992**	**84.6**	**1:21**	**14.5**	**654**	**2 116**	**290**
4	4	4	8	8	100	-	-	409 600	44.0	1:28	16.1	645	2 087	290
4	4	4	8	16	92	-	-	753 664	23.9	1:31	17.7	646	2 028	289
4	4	8	8	16	88	-	-	1 441 792	12.5	1:50	19.8	663	2 085	296
4	4	4	4	4	4	104	-	425 984	42.3	1:27	15.6	649	2 209	299
4	4	4	4	8	8	96	-	1 572 864	11.5	1:46	19.3	637	2 092	294
4	4	4	8	8	16	84	-	5 505 024	3.3	2:00	21.5	655	2 035	294
4	4	4	4	4	4	4	100	1 638 400	11.0	1:43	19.2	650	2 247	308
4	4	4	4	4	4	8	96	3 145 728	5.7	1:56	20.0	627	2 113	293
4	4	4	4	4	8	8	92	6 029 312	3.0	1:51	21.1	649	2 121	300
4	4	4	4	4	8	16	84	11 010 048	1.6	2:03	21.5	648	2 035	296

Einfluss Kontraktion

	PREPRO		QUERY		
c	time [h:m]	space [B/n]	#settled nodes	#relaxed edges	time [μs]
1.0	1:40	15.1	1 572	3 705	578
1.5	1:20	14.8	886	2 464	348
2.0	1:20	14.7	714	2 171	301
2.5	1:21	14.5	654	2 116	290
3.0	1:23	14.6	622	2 109	286

Beobachtung:

- Einfluss der Kontraktion begrenzt



Einfluss Flaggenberechnung

ARC-FLAGS		PREPRO		QUERY		
core levels	refinement levels	time [h:m]	space [B/n]	#settled nodes	#relaxed edges	time [μs]
-	-	0:16	12.8	204 518	960 653	76 640
5	5	0:24	13.2	23 313	70 225	6 021
5	4-5	0:24	13.2	6 583	23 038	1 843
5	3-5	0:25	13.3	2 394	11 547	856
5	2-5	0:27	13.6	1 350	8 721	611
5	1-5	0:29	13.7	1 127	8 091	553
4-5	4-5	0:30	13.7	6 186	18 170	1 626
4-5	3-5	0:30	13.7	2 042	6 683	648
4-5	2-5	0:31	13.7	993	3 883	405
4-5	**1-5**	**0:34**	**13.7**	**784**	**3 338**	**355**
3-5	3-5	0:35	13.8	1 974	5 962	615
3-5	2-5	0:37	14.2	933	3 161	371
3-5	1-5	0:39	14.2	729	2 629	323
2-5	2-5	0:44	14.3	900	2 862	354
2-5	1-5	0:46	14.3	696	2 335	305
1-5	1-5	0:54	14.5	684	2 236	300
0-5	0-5	1:21	14.5	654	2 116	290

Beobachtung:

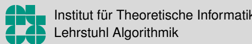
- partielle Berechnung reicht aus
- Verfeinerung wichtiger als Berechnung
- Ungenauigkeit am Ende der Anfrage eher verzeihbar als am Anfang (Suchraum fächert sich auf)

stripped SHARC

	SHARC				stripped SHARC			
	PREPRO		QUERY		PREPRO		QUERY	
	time	space	#settled	time	time	space	#settled	time
c	[h:m]	[B/n]	nodes	[ms]	[h:m]	[B/n]	nodes	[ms]
0.50	7:32	13.8	10 876	3.38	7:48	7.8	14 697	4.76
0.75	4:29	14.3	5 420	1.99	4:43	7.7	62 303	26.03
1.00	1:45	15.1	1 997	0.90	2:03	7.5	1 891 320	1 096.48

Beobachtung:

- volles Entfernen nur bei niedriger Kontraktion



Zufallsanfragen

	Europe				USA			
	PREPRO		QUERY		PREPRO		QUERY	
	time [h:m]	space [B/n]	#settled nodes	time [μs]	time [h:m]	space [B/n]	#settled nodes	time [μs]
generous SHARC	1:21	14.5	654	290	0:58	18.1	865	376
economical SHARC	0:34	13.7	784	355	0:38	17.2	1 230	578
stripped SHARC	7:48	7.8	14 697	4 762	6:41	9.2	38 817	12 719
bidirectional SHARC	2:38	21.0	125	65	2:34	23.1	254	118

Lokale Anfragen

CHASE

Landmarken

Bidirektionale Suche

s
t

Kontraktion

Arc-Flags

Table-Lookups

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

CHASE

Motivation:

- CHs sind ungerichtet
- weitere Beschleunigung von CH durch zielgerichtetes Verfahren

Hauptideen:

- Flaggen auf Suchgraphen berechnen
- nur auf oberen Teil der Hierarchie

Erster Ansatz

Vorberechnung:

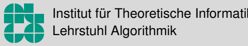
- CH-Vorberechnung
- partitioniere Suchgraphen
- berechne Flaggen für Suchgraphen

Anfragen:

- normale CH-query
- mit Flaggen pruning

Problem:

- mehr Randknoten durch Shortcuts
- sehr lange Vorberechnungszeit



Partielle Flaggen

Vorberechnung:

- extrahiere Graphen H mit k wichtigsten Knoten
- führe Vorberechnung für H aus

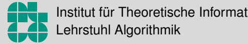
2-Phasen Anfragen:

1. normale CH-query
2. CH + Flaggen query

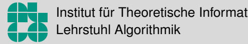
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Beispiel

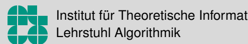
Beispiel



Beispiel

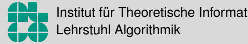


Beispiel



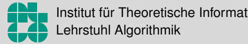
Beispiel

Beispiel

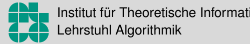


Beispiel

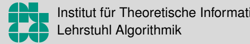
Beispiel



Beispiel



Beispiel



Beispiel

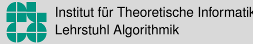
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

CHASE: CH+Arc-flagS

	size of Arc-Flags	0%	0.5%	1%	2%	5%	10%	20%
Prepro.	[h:m]	0:25	0:32	0:41	1:02	1:39	4:04	8:56
	[B/n]	-2.7	0.0	1.9	4.9	12.1	22.2	39.5
Query	# settled	355	111	78	59	45	39	35
	time [μs]	180.0	43.8	30.8	23.1	17.3	14.9	13.0

Beobachtungen:

- zusätzliche 7 Minuten Vorberechnung $\Rightarrow$ zusätzlicher speedup von 4.1 (economical variant)
- eine Stunde $\Rightarrow$ speedup von 10.4
- mehr al 5% lohnen sich nicht

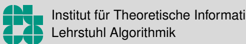


Vergleich

	Preprocessing		Query	
	[h:m]	[B/n]	#settled	[μs]
MLD-3	$<$ 0:01	2	6 074	910
Contraction Hierarchies	0:02	-3	284	90
bidirectional Arc-Flags	17:08	19	2 369	1 600
Transit-Node Routing	1:52	204	—	3.4
CH-TNR	0:34	147	—	3.3
Hub Labels	6:12	967	—	0.2
unidirectional SHARC	0:34	13.7	784	355
bidirectional SHARC	3:32	21	125	65
REAL-(64,16)	2:21	32	679	1 100
eco. CHASE	0:32	0	111	43.8
gen. CHASE	1:39	12	45	17.3

Beobachtung:

- CHASE fast so schnell wie TNR, weniger Vorberechnungsplatz
- SHARC: unidirektionale Query (könnte noch von Nutzen sein)



Lokale Anfragen

pure CH
eco CHASE
gen CHASE

Query Time [µs]

5 10 20 50 100 200

2^6 2^7 2^8 2^9 2^{10} 2^{11} 2^{12} 2^{13} 2^{14} 2^{15} 2^{16} 2^{17} 2^{18} 2^{19} 2^{20} 2^{21} 2^{22} 2^{23} 2^{24}

Dijkstra Rank

Lokale Anfragen

contraction hierarchies
ch + edge flags
transit node routing

Query Time [µs]

5 10 20 50 100 200

2^{6} 2^{7} 2^{8} 2^{9} 2^{10} 2^{11} 2^{12} 2^{13} 2^{14} 2^{15} 2^{16} 2^{17} 2^{18} 2^{19} 2^{20} 2^{21} 2^{22} 2^{23} 2^{24}

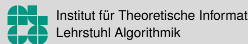
Dijkstra Rank

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Zusammenfassung

Kombinationen

- Basismodule:
  - bidirektionale Suche
  - landmarken
  - reach
  - arc-flags
  - Kontraktion
  - Table-Lookups
- beste Kombinationen: hierarchisch + zielgerichtet
- CH + Arc-Flags sehr effizient



Literatur

Literatur (SHARC und Kombinationen):

- Reinhard Bauer and Daniel Delling:
  SHARC: Fast and Robust Unidirectional Routing
  In: *ACM Journal of Experimental Algorithmics*, 14:2.4, 2009.
- Reinhard Bauer, Daniel Delling, Peter Sanders, Dennis Schieferdecker, Dominik Schultes, and Dorothea Wagner:
  Combining Hierarchical and Goal-Directed Speed-Up Techniques for Dijkstra's Algorithm
  In: *ACM Journal of Experimental Algorithmics*, 15:2.3, 2010.

Was bisher geschah

bisheriger Stoff:

- Punkt-zu-Punkt Abfragen
- statisches Szenario

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Was bisher geschah

Karlsruher Institut für Technologie

bisheriger Stoff:

- Punkt-zu-Punkt Abfragen
- statisches Szenario

Erweiterte Anfragen

- one-to-many
- many-to-many
- OVIs
- one-to-all?

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Was bisher geschah

bisheriger Stoff:

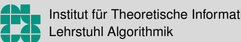
- Punkt-zu-Punkt Abfragen
- statisches Szenario

Erweiterte Anfragen

- one-to-many
- many-to-many
- OVIs
- one-to-all?

Erweiterte Szenarien

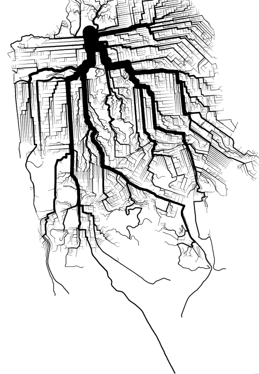
- Abbiegeverbote/-kosten
- Staus
- Alternativen
- Multikriteriell
- Eisenbahn



Kürzeste-Wege Bäume

Anfrage:

- gegeben ein nicht negativ gewichteter gerich
  Knoten *s*
- berechne Distanzen von *s* zu *allen* anderen



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

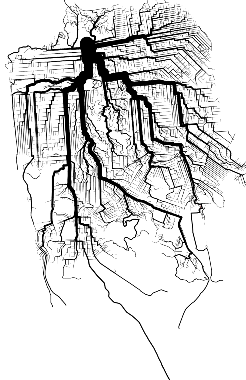
Kürzeste-Wege Bäume

Anfrage:

- gegeben ein nicht negativ gewichteter gerich
  Knoten *s*
- berechne Distanzen von *s* zu *allen* anderen

Lösung:

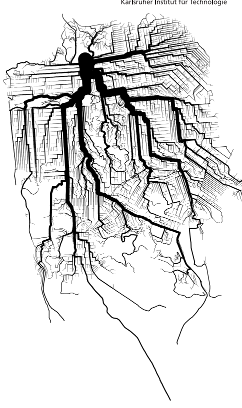
- Dijkstra [Dij59]



Kürzeste-Wege Bäume

Anfrage:

- gegeben ein nicht negativ gewichteter gerich Knoten s
- berechne Distanzen von s zu *allen* anderen



Lösung:

- Dijkstra [Dij59]

Fakten:

- $O(m + n \log n)$ mit Fibonacci Heaps [FT87]
- linear (mit kleiner Konstanten) in Praxis [Gol01]
- Ausnutzung von moderner Hardware schwierig

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Moderne CPU Architektur

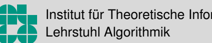
Einige Fakten:

- viele Kerne
- mehr Kerne als Speicherkontroller
- Hyperthreading
- Multi-Sockel System
- steile Speicherhierarchie
- Cache coherency
- keine Register coherency

Intel Nehalem microarchitecture

quadruple associative Instruction Cache 32 KByte, 128-entry TLB-4K, 7 TLB-2/4M per thread
128
Prefetch Buffer (16 Bytes)
Branch Prediction global/bimodal, loop, indirect jmp
Predecode & Instruction Length Decoder
Instruction Queue 18 x86 Instructions Alignment MacroOp Fusion
Complex Decoder
Simple Decoder
Simple Decoder
Simple Decoder
Loop Stream Decoder
Decoded Instruction Queue (28 µOP entries)
Micro Instruction Sequencer
MicroOp Fusion
2 x Retirement Register File
2 x Register Allocation Table (RAT)
Reorder Buffer (128-entry) fused
Reservation Station (128-entry) fused
Port 4
Port 3
Port 2
Port 5
Port 1
Port 0
Store Data
AGU Store Addr. Unit
AGU Load Addr. Unit
Integer/ MMX ALU, Branch
SSE ADD Move
Integer/ MMX ALU
FP ADD
SSE ADD Move
FP MUL
Integer/ MMX ALU, 2x AGU
SSE MUL/DIV Move
128
128
128
Result Bus
Memory Order Buffer (MOB)
128
128
octuple associative Data Cache 32 KByte, 64-entry TLB-4K, 32-entry TLB-2/4M
256
Uncore
Quick Path Inter-connect
4 x 20 Bit 6,4 GT/s
DDR3 Memory Controller
3 x 64 Bit 1,33 GT/s
Common L3-Cache 8 MByte
256 KByte 8-way, 64 Byte Cacheline, private L2-Cache
512-entry L2-TLB-4K

GT/s: gigatransfers per second



Moderne CPU Architektur

Einige Fakten:

- viele Kerne
- mehr Kerne als Speicherkontroller
- Hyperthreading
- Multi-Sockel System
- steile Speicherhierarchie
- Cache coherency
- keine Register coherency

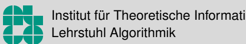
Hauptherausforderungen:

- Parallelisierung
- Speicherzugriff

Intel Nehalem microarchitecture

quadruple associative Instruction Cache 32 KByte, 128-entry TLB-4K, 7 TLB-2/4M per thread
128
Prefetch Buffer (16 Bytes)
Branch Prediction global/bimodal, loop, indirect jmp
Predecode & Instruction Length Decoder
Instruction Queue 18 x86 Instructions Alignment MacroOp Fusion
Complex Decoder
Simple Decoder
Simple Decoder
Simple Decoder
Loop Stream Decoder
Decoded Instruction Queue (28 µOP entries)
Micro Instruction Sequencer
MicroOp Fusion
2 x Retirement Register File
2 x Register Allocation Table (RAT)
Reorder Buffer (128-entry) fused
Reservation Station (128-entry) fused
Port 4, Port 3, Port 2, Port 5, Port 1, Port 0
Store Data
AGU Store Addr. Unit
AGU Load Addr. Unit
Integer/ MMX ALU, Branch
SSE ADD Move
Integer/ ALU
FP ADD
SSE ADD Move
FP MUL
Integer/ MMX ALU, 2x AGU
SSE MUL/DIV Move
128
Result Bus
Memory Order Buffer (MOB)
128
octuple associative Data Cache 32 KByte, 64-entry TLB-4K, 32-entry TLB-2/4M
256
Uncore
Quick Path Inter-connect
4 x 20 Bit 6,4 GT/s
DDR3 Memory Controller
3 x 64 Bit 1,33 GT/s
Common L3-Cache 8 MByte
256 KByte 8-way, 64 Byte Cacheline, private L2-Cache
512-entry L2-TLB-4K

GT/s: gigatransfers per second

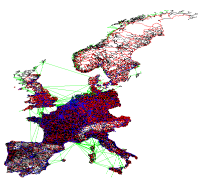


Ausnutzen von Moderner Hardware

Daten Lokalität

- Eingabe: West Europa
- 18M Knoten, 23M Strassen

  Dijkstra: $\approx$ 3.0 s $\Rightarrow$ nicht real-time



Core-i7 workstation (2.66 GHz)

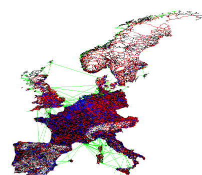
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Ausnutzen von Moderner Hardware

Daten Lokalität

- Eingabe: West Europa
- 18M Knoten, 23M Strassen

Dijkstra:	$\approx$ 3.0 s	$\Rightarrow$ nicht real-time
$n + m$ clock cycles:	$\approx$ 15 ms	$\Rightarrow$ viel schneller



Core-i7 workstation (2.66 GHz)

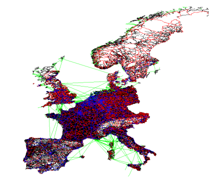
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Ausnutzen von Moderner Hardware

Daten Lokalität

- Eingabe: West Europa
- 18M Knoten, 23M Strassen

Dijkstra:	$\approx$ 3.0 s	$\Rightarrow$ nicht real-time
$n + m$ clock cycles:	$\approx$ 15 ms	$\Rightarrow$ viel schneller
BFS:	$\approx$ 2.0 s	



- Verlangsamung kommt nicht durch Priorityqueue allein

Core-i7 workstation (2.66 GHz)

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Ausnutzen von Moderner Hardware

Karlsruher Institut für Technologie

Daten Lokalität

- Eingabe: West Europa
- 18M Knoten, 23M Strassen

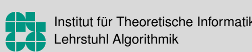
Dijkstra:	$\approx$ 3.0 s	$\Rightarrow$ nicht real-time
$n + m$ clock cycles:	$\approx$ 15 ms	$\Rightarrow$ viel schneller
BFS:	$\approx$ 2.0 s	

- Verlangsamung kommt nicht durch Priorityqueue allein

Core-i7 workstation (2.66 GHz)

Parallelisierung:

- Spekulation
- Δ-stepping [MS03],[MBBC09]
- mehr Operationen als Dijkstra
- keine grosse Beschleunigung auf dünnen Graphen



Ausnutzen von Moderner Hardware

Daten Lokalität

- Eingabe: West Europa
- 18M Knoten, 23M Strassen

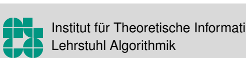
Dijkstra:	$\approx$ 3.0 s	$\Rightarrow$ nicht real-time
$n + m$ clock cycles:	$\approx$ 15 ms	$\Rightarrow$ viel schneller
BFS:	$\approx$ 2.0 s	

- Verlangsamung kommt nicht durch Priorityqueue allein

Core-i7 workstation (2.66 GHz)

Parallelisierung:

- Spekulation
- Δ-stepping [MS03],[MBBC09]
- mehr Operationen als Dijkstra
- keine grosse Beschleunigung auf dünnen Graphen
- Berechnen von mehreren Bäumen ist einfach



Ansatz 1

Idee:

- Umordnen der Knoten im Graphen

Ansatz 1

Idee:

- Umordnen der Knoten im Graphen

algorithm	details	time per tree [ms] random	input	DFS
Dijkstra	binary heap	11159	5859	5180
	Dial	7767	3538	2908
	smart queue	7991	3556	2826
BFS	—	6060	2445	2068

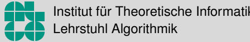
Ansatz 1

Idee:

- Umordnen der Knoten im Graphen

algorithm	details	time per tree [ms] random	input	DFS
Dijkstra	binary heap	11159	5859	5180
	Dial	7767	3538	2908
	smart queue	7991	3556	2826
BFS	—	6060	2445	2068

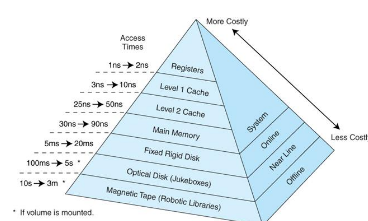
$\Rightarrow$ keine grosse Beschleunigung



Ansatz 2

Dijkstra's Algorithmus:

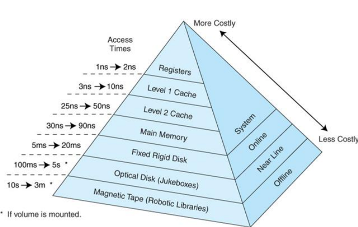
- moderne Hardware nicht voll zu nutzen
- Hauptprobleme:
  - Daten Lokalität
  - Parallelisierung



Ansatz 2

Dijkstra's Algorithmus:

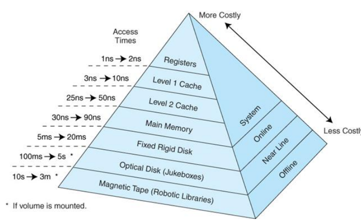
- moderne Hardware nicht voll zu nutzen
- Hauptprobleme:
  - Daten Lokalität
  - Parallelisierung



Fragen:

- hilft Vorberechnung?
- wie?
- Ansatzpunkt?

Ansatz 2



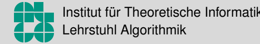
Dijkstra's Algorithmus:

- moderne Hardware nicht voll zu nutzen
- Hauptprobleme:
  - Daten Lokalität
  - Parallelisierung

Fragen:

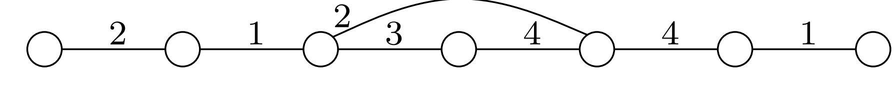
- hilft Vorberechnung?
- wie?
- Ansatzpunkt?

PHAST: Hardware-Accelerated Shortest path Trees



Contraction Hierarchies

preprocessing:

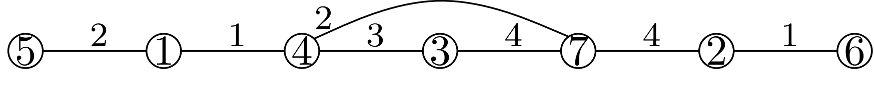


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Contraction Hierarchies

preprocessing:

- ordne Knoten nach Wichtigkeit



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Contraction Hierarchies

preprocessing:

- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu

Contraction Hierarchies

preprocessing:

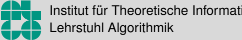
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Contraction Hierarchies

preprocessing:

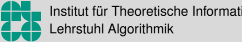
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



Contraction Hierarchies

preprocessing:

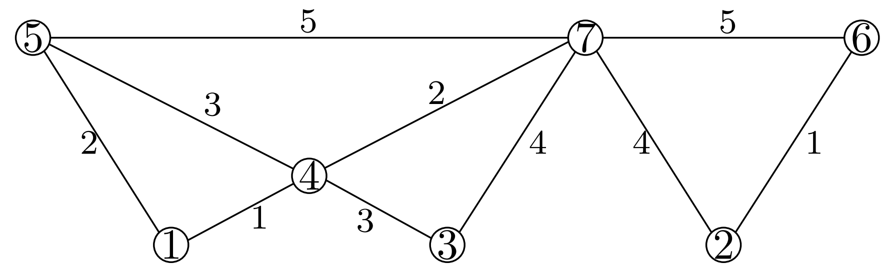
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu

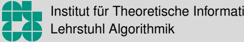


Contraction Hierarchies

preprocessing:

- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu

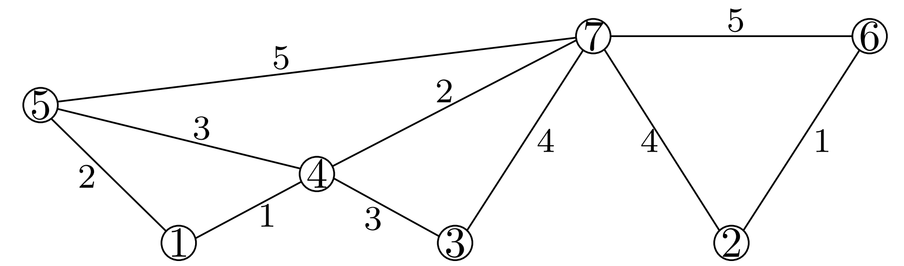




Contraction Hierarchies

preprocessing:

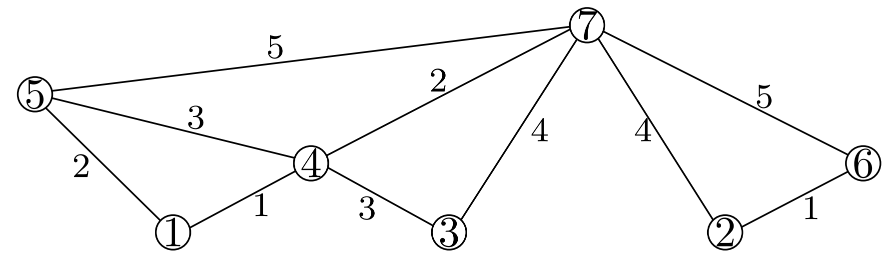
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu

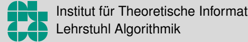


Contraction Hierarchies

preprocessing:

- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu

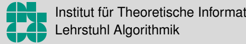




Contraction Hierarchies

preprocessing:

- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu
- Levelzuordnung (ca. 150 in Strassennetzwerken)



Contraction Hierarchies

Punkt-zu-Punkt Anfragen

- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten
- besucht nur 500 Knoten

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Contraction Hierarchies

Punkt-zu-Punkt Anfragen

- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten
- besucht nur 500 Knoten

level

s t

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Contraction Hierarchies

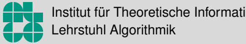
Punkt-zu-Punkt Anfragen

- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten
- besucht nur 500 Knoten

Contraction Hierarchies

Punkt-zu-Punkt Anfragen

- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten
- besucht nur 500 Knoten



Contraction Hierarchies

Punkt-zu-Punkt Anfragen

- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten
- besucht nur 500 Knoten

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Contraction Hierarchies

Punkt-zu-Punkt Anfragen

- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten
- besucht nur 500 Knoten

Korrektheit:

- es gibt einen wichtigsten Knoten auf dem Pfad
- dieser wird von Vorwärts- und Rückwärtssuche gescannt

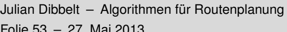
Neuer Anfragealgorithmus

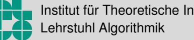
one-to-all Suche von *s*:

level

5 7 s 4 6 1 3 2

5 2 5 3 4 4 2 1 3 1





Neuer Anfragealgorithmus

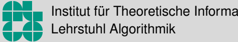
one-to-all Suche von *s*:

- vorwärts CH Suche von *s* ($\approx$ 0.05 ms)

Neuer Anfragealgorithmus

one-to-all Suche von *s*:

- vorwärts CH Suche von s ($\approx$ 0.05 ms)
- setze Distanzen d für alle erreichten Knoten

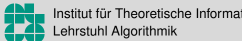


Neuer Anfragealgorithmus

one-to-all Suche von s:

- vorwärts CH Suche von s ($\approx$ 0.05 ms)
- setze Distanzen d für alle erreichten Knoten
- bearbeite alle Knoten u in inverser Levelordnung:
    - checke eingehende Kanten (v, u) mit $lev(v) > lev(u)$
    - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$

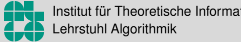
level



Neuer Anfragealgorithmus

one-to-all Suche von *s*:

- vorwärts CH Suche von s ($\approx$ 0.05 ms)
- setze Distanzen d für alle erreichten Knoten
- bearbeite alle Knoten u in inverser Levelordnung:
  - checke eingehende Kanten (v, u) mit $lev(v) > lev(u)$
  - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$

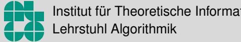


Neuer Anfragealgorithmus

one-to-all Suche von s:

- vorwärts CH Suche von s (≈ 0.05 ms)
- setze Distanzen d für alle erreichten Knoten
- bearbeite alle Knoten u in inverser Levelordnung:
  - checke eingehende Kanten (v, u) mit $lev(v) > lev(u)$
  - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$

level



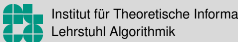
Neuer Anfragealgorithmus

one-to-all Suche von *s*:

- vorwärts CH Suche von *s* ($\approx$ 0.05 ms)
- setze Distanzen *d* für alle erreichten Knoten
- bearbeite alle Knoten *u* in inverser Levelordnung:
  - checke eingehende Kanten (v, u) mit $lev(v) > lev(u)$
  - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$

level

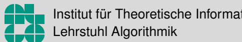
s



Neuer Anfragealgorithmus

one-to-all Suche von s:

- vorwärts CH Suche von s ($\approx$ 0.05 ms)
- setze Distanzen d für alle erreichten Knoten
- bearbeite alle Knoten u in inverser Levelordnung:
  - checke eingehende Kanten (v, u) mit $lev(v) > lev(u)$
  - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$

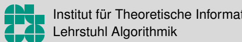


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Neuer Anfragealgorithmus

one-to-all Suche von s:

- vorwärts CH Suche von s ($\approx$ 0.05 ms)
- setze Distanzen d für alle erreichten Knoten
- bearbeite alle Knoten u in inverser Levelordnung:
  - checke eingehende Kanten (v, u) mit $lev(v) > lev(u)$
  - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$



Neuer Anfragealgorithmus

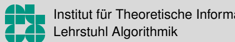
one-to-all Suche von s:

- vorwärts CH Suche von s ($\approx$ 0.05 ms)
- setze Distanzen d für alle erreichten Knoten
- bearbeite alle Knoten u in inverser Levelordnung:
  - checke eingehende Kanten (v, u) mit $lev(v) > lev(u)$
  - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$

level

s

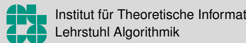
5 7 2 5 3 2 4 4 5 3 4 6 2 1 0 1 3 7 1 1 3 3 2 6



Neuer Anfragealgorithmus

one-to-all Suche von s:

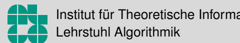
- vorwärts CH Suche von s ($\approx$ 0.05 ms)
- setze Distanzen d für alle erreichten Knoten
- bearbeite alle Knoten u in inverser Levelordnung:
  - checke eingehende Kanten (v, u) mit $lev(v) > lev(u)$
  - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$
- top-down Bearbeitung ohne Priority Queue (ca. 2.0 s)



Neuer Anfragealgorithmus

one-to-all Suche von *s*:

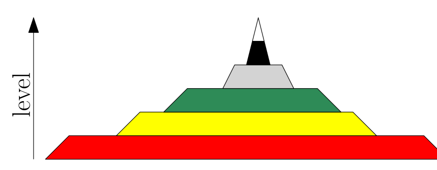
- vorwärts CH Suche von *s* ($\approx$ 0.05 ms)
- setze Distanzen *d* für alle erreichten Knoten
- bearbeite alle Knoten *u* in inverser Levelordnung:
  - checke eingehende Kanten (v, u) mit $lev(v) > lev(u)$
  - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$
- top-down Bearbeitung ohne Priority Queue (ca. 2.0 s)
- genauso schnell wie BFS. Warum das ganze?



Analyse

Beobachtung:

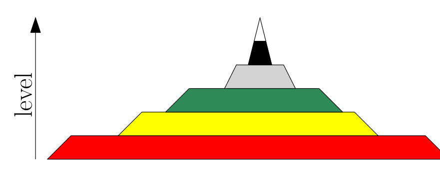
- top-down Prozess ist der Flaschenhals



Analyse

Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient

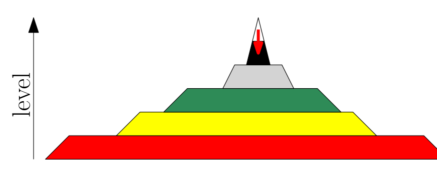


$\mathrm{dist}(u)$:

Analyse

Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient



r

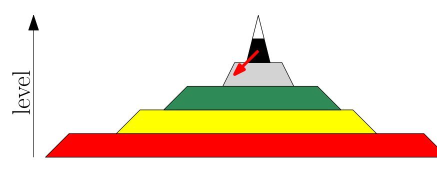
w

dist(u):

Analyse

Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient



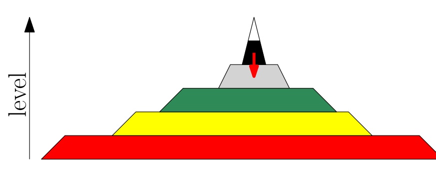
w r

dist(u):

Analyse

Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient



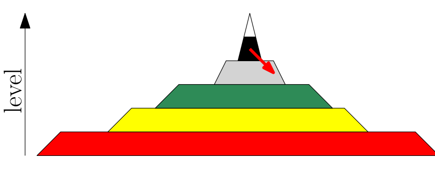
w r

$\mathrm{dist}(u)$:

Analyse

Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient



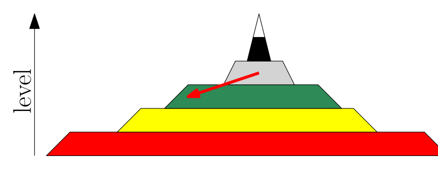
r w

$\mathrm{dist}(u)$:

Analyse

Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient



w

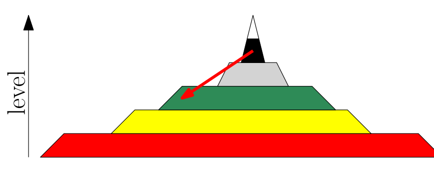
r

dist(u):

Analyse

Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient



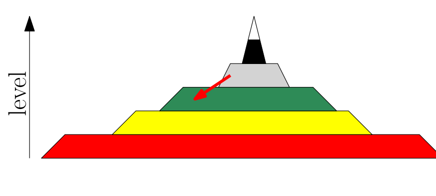
w r

$\mathrm{dist}(u)$:

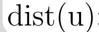
Analyse

Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient
- Zugriffsmuster sind unabhängig von s



r w



Analyse

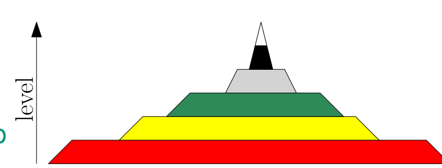
Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient
- Zugriffsmuster sind unabhängig von s

Idee:

- speicher $G_\uparrow$ und $G_\downarrow$ separat
- Umordnung der Knoten, Kanten, und Distanzlabel nach Level

$\Rightarrow$ lesen der Kanten und schreiben der Distanzen wird zu einem sequenziellen Sweep



$\mathrm{dist}(u)$:

Analyse

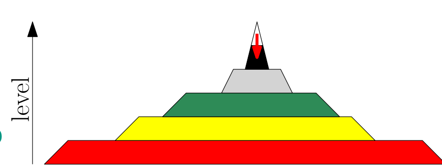
Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient
- Zugriffsmuster sind unabhängig von s

Idee:

- speicher $G_\uparrow$ und $G_\downarrow$ separat
- Umordnung der Knoten, Kanten, und Distanzlabel nach Level

$\Rightarrow$ lesen der Kanten und schreiben der Distanzen wird zu einem sequenziellen Sweep



r w

dist(u):

Analyse

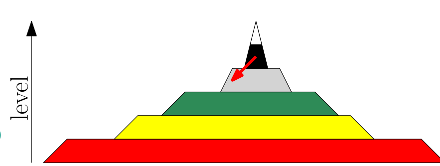
Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient
- Zugriffsmuster sind unabhängig von s

Idee:

- speicher $G_\uparrow$ und $G_\downarrow$ separat
- Umordnung der Knoten, Kanten, und Distanzlabel nach Level

$\Rightarrow$ lesen der Kanten und schreiben der Distanzen wird zu einem sequenziellen Sweep



r w

dist(u):

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Analyse

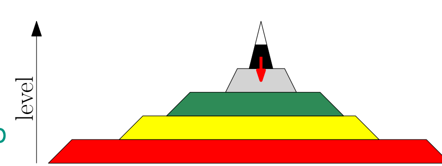
Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient
- Zugriffsmuster sind unabhängig von s

Idee:

- speicher $G_\uparrow$ und $G_\downarrow$ separat
- Umordnung der Knoten, Kanten, und Distanzlabel nach Level

$\Rightarrow$ lesen der Kanten und schreiben der Distanzen wird zu einem sequenziellen Sweep



r w

dist(u):

Analyse

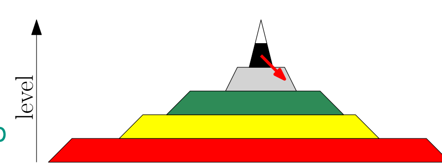
Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient
- Zugriffsmuster sind unabhängig von s

Idee:

- speicher $G_{\uparrow}$ und $G_{\downarrow}$ separat
- Umordnung der Knoten, Kanten, und Distanzlabel nach Level

$\Rightarrow$ lesen der Kanten und schreiben der Distanzen wird zu einem sequenziellen Sweep



r w

dist(u):

Analyse

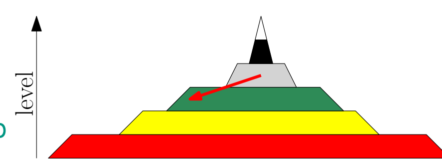
Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient
- Zugriffsmuster sind unabhängig von s

Idee:

- speicher $G_\uparrow$ und $G_\downarrow$ separat
- Umordnung der Knoten, Kanten, und Distanzlabel nach Level

$\Rightarrow$ lesen der Kanten
und schreiben der Distanzen
wird zu einem sequenziellen Sweep



r w

dist(u):

Analyse

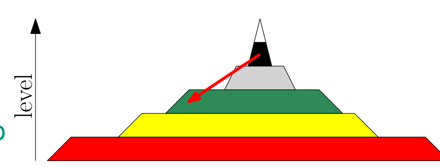
Beobachtung:

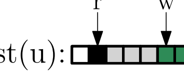
- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient
- Zugriffsmuster sind unabhängig von s

Idee:

- speicher $G_\uparrow$ und $G_\downarrow$ separat
- Umordnung der Knoten, Kanten, und Distanzlabel nach Level

$\Rightarrow$ lesen der Kanten und schreiben der Distanzen wird zu einem sequenziellen Sweep





Analyse

Beobachtung:

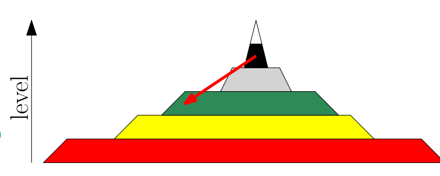
- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient
- Zugriffsmuster sind unabhängig von s

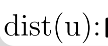
Idee:

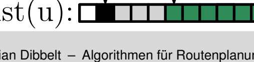
- speicher $G_\uparrow$ und $G_\downarrow$ separat
- Umordnung der Knoten, Kanten, und Distanzlabel nach Level

$\Rightarrow$ lesen der Kanten und schreiben der Distanzen wird zu einem sequenziellen Sweep

$\Rightarrow$ 172 ms pro Baum







Analyse

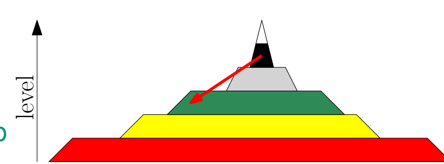
Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient
- Zugriffsmuster sind unabhängig von s

Idee:

- speicher $G_\uparrow$ und $G_\downarrow$ separat
- Umordnung der Knoten, Kanten, und Distanzlabel nach Level

$\Rightarrow$ lesen der Kanten und schreiben der Distanzen wird zu einem sequenziellen Sweep



$\Rightarrow$ 172 ms pro Baum

- aber lesen der Distanzen immer noch ineffizient

r w

dist(u):

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

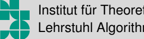
Szenario: Multiple Startknoten

level

s_0

Szenario: Multiple Startknoten

Idee:



Szenario: Multiple Startknoten

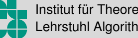
Idee:

- *k* Vorwärtssuchen

Szenario: Multiple Startknoten

Idee:

- *k* Vorwärtssuchen



Szenario: Multiple Startknoten

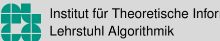
Idee:

- *k* Vorwärtssuchen

Szenario: Multiple Startknoten

Idee:

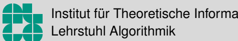
- *k* Vorwärtssuchen
- ein sweep (update aller *k* Werte)
- speicher Distanzlabel pro Knoten



Szenario: Multiple Startknoten

Idee:

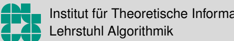
- *k* Vorwärtssuchen
- ein sweep (update aller *k* Werte)
- speicher Distanzlabel pro Knoten



Szenario: Multiple Startknoten

Idee:

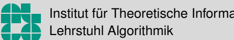
- *k* Vorwärtssuchen
- ein sweep (update aller *k* Werte)
- speicher Distanzlabel pro Knoten



Szenario: Multiple Startknoten

Idee:

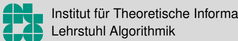
- *k* Vorwärtssuchen
- ein sweep (update aller *k* Werte)
- speicher Distanzlabel pro Knoten



Szenario: Multiple Startknoten

Idee:

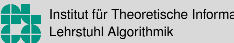
- *k* Vorwärtssuchen
- ein sweep (update aller *k* Werte)
- speicher Distanzlabel pro Knoten



Szenario: Multiple Startknoten

Idee:

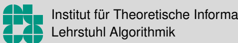
- *k* Vorwärtssuchen
- ein sweep (update aller *k* Werte)
- speicher Distanzlabel pro Knoten



Szenario: Multiple Startknoten

Idee:

- *k* Vorwärtssuchen
- ein sweep (update aller *k* Werte)
- speicher Distanzlabel pro Knoten



Szenario: Multiple Startknoten

Idee:

- *k* Vorwärtssuchen
- ein sweep (update aller *k* Werte)
- speicher Distanzlabel pro Knoten
- 96.8 ms pro Baum ($k = 16$)

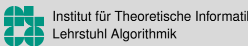
Szenario: Multiple Startknoten

Idee:

- *k* Vorwärtssuchen
- ein sweep (update aller *k* Werte)
- speicher Distanzlabel pro Knoten
- 96.8 ms pro Baum ($k = 16$)

SSE:

- 128-bit Register
- Basisoperationen (min, add) für vier 32-bit Integer parallel
- scanne 4 Distanzlabel auf einmal



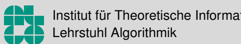
Szenario: Multiple Startknoten

Idee:

- k Vorwärtssuchen
- ein sweep (update aller k Werte)
- speicher Distanzlabel pro Knoten
- 96.8 ms pro Baum ($k = 16$)

SSE:

- 128-bit Register
- Basisoperationen (min, add) für vier 32-bit Integer parallel
- scanne 4 Distanzlabel auf einmal
- 37.1 ms pro Baum ($k = 16$)



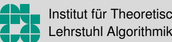
Szenario: Multiple Startknoten

Idee:

- k Vorwärtssuchen
- ein sweep (update aller k Werte)
- speicher Distanzlabel pro Knoten
- 96.8 ms pro Baum ($k = 16$)

SSE:

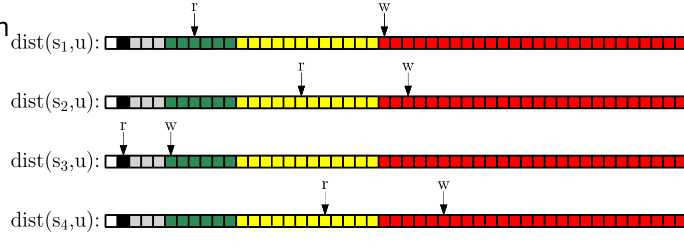
- 128-bit Register
- Basisoperationen (min, add) für vier 32-bit Integer parallel
- scanne 4 Distanzlabel auf einmal
- 37.1 ms pro Baum ($k = 16$)
- Sandy Bridge Architektur: 256-bit Register



Parallelisierung

ganz einfach

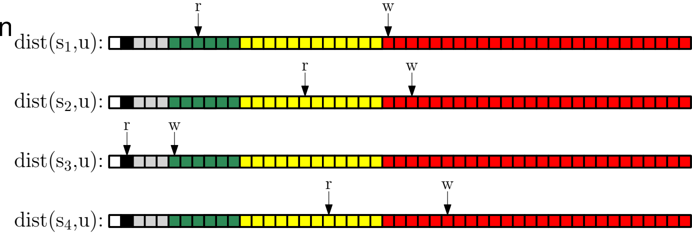
- nach Startknoten



Parallelisierung

ganz einfach

- nach Startknoten



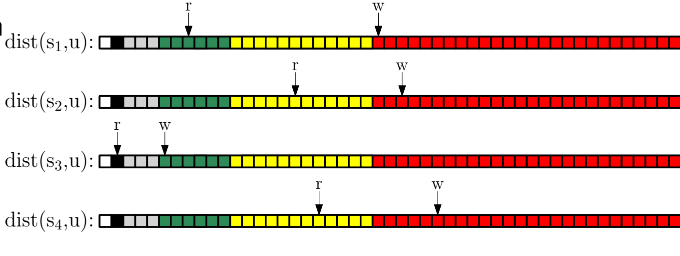
Ergebnisse:

- 16 Startenknoten pro Sweep (updates via SSE)
- multi-core nach Startknoten ⇒ 64 Startknoten parallel (4 cores)
- 18.8 ms per Baum

Parallelisierung

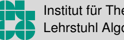
ganz einfach

- nach Startknoten



Ergebnisse:

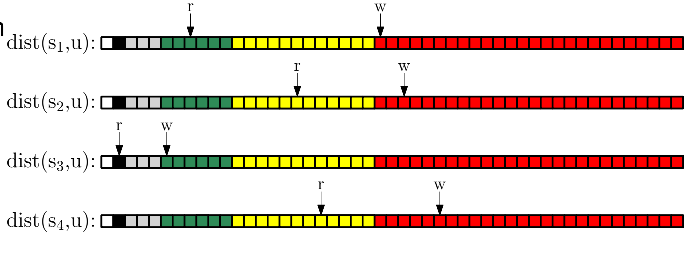
- 16 Startenknoten pro Sweep (updates via SSE)
- multi-core nach Startknoten ⇒ 64 Startknoten parallel (4 cores)
- 18.8 ms per Baum
- Warum kein perfekter Speedup?



Parallelisierung

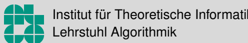
ganz einfach

- nach Startknoten



Ergebnisse:

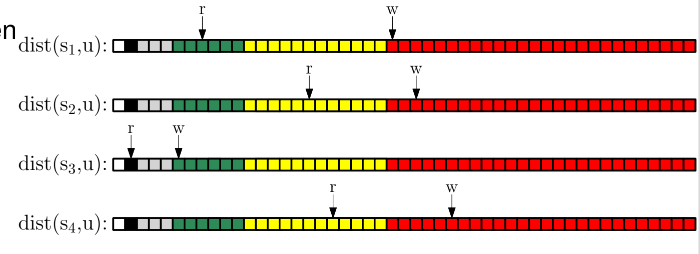
- 16 Startenknoten pro Sweep (updates via SSE)
- multi-core nach Startknoten ⇒ 64 Startknoten parallel (4 cores)
- 18.8 ms per Baum
- Warum kein perfekter Speedup?
- lower bound tests zeigen: nah an Speicherbandbreite



Parallelisierung

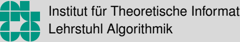
ganz einfach

- nach Startknoten

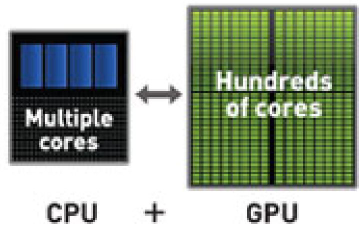


Ergebnisse:

- 16 Startenknoten pro Sweep (updates via SSE)
- multi-core nach Startknoten ⇒ 64 Startknoten parallel (4 cores)
- 18.8 ms per Baum
- Warum kein perfekter Speedup?
- lower bound tests zeigen: nah an Speicherbandbreite
- kann eine GPU helfen?



GPU Architektur

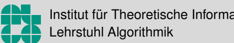


Intel Xeon X5680:

- 3.33 GHz
- 32 GB/s Speicherbandbreite
- 6 Kerne

NVIDIA GTX 580:

- 772 MHz, 1.5 GB RAM
- 192 GB/s Speicherbandbreite
- 16 Kerne, 32 parallele Threads (ein Warp) pro Kern ⇒ 512 parallele Threads
- eingeschränkte Berechnungen



GPU Architektur

Karlsruher Institut für Technologie

einige Fakten:

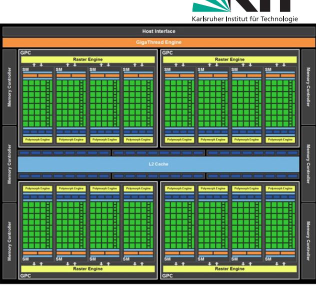
- viele Kerne (bis zu “512”)
- schnellerer Speicher (5x schneller als CPU)
- aber Haupt- $\rightarrow$ GPU Transfer langsam ($\approx$ 20x)
  $\Rightarrow$ minimiere Datentransfer
- keine Cache coherency
  $\Rightarrow$ Berechnungen sollten unabhängig sein
- Single Instruction Multiple Threads Modell
  (Gruppen von Tread folgen gleichen Instruktionen)
- barrel processing um Speicherlatenz zu verbergen
  $\Rightarrow$ tausende von unabhängigen Threads (!) nötig
- Zugriff einer Threadgruppe (Warp) nur effizient für bestimme Zugriffsarten
  (zum Beispiel sequentiell)

GPU Architektur

Karlsruher Institut für Technologie

einige Fakten:

- viele Kerne (bis zu “512”)
- schnellerer Speicher (5x schneller als CPU)
- aber Haupt- $\rightarrow$ GPU Transfer langsam ($\approx$ 20x)
  $\Rightarrow$ minimiere Datentransfer
- keine Cache coherency
  $\Rightarrow$ Berechnungen sollten unabhängig sein
- Single Instruction Multiple Threads Modell
  (Gruppen von Tread folgen gleichen Instruktionen)
- barrel processing um Speicherlatenz zu verbergen
  $\Rightarrow$ tausende von unabhängigen Threads (!) nötig
- Zugriff einer Threadgruppe (Warp) nur effizient für bestimme Zugriffsarten
  (zum Beispiel sequentiell)

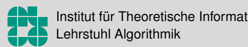


$\Rightarrow$ **eingeschränktes Rechenmodell**
$\Rightarrow$ **Keine generelle Beschleunigung von 1000x gegenüber CPUs**

GPHAST - Ideen

Beobachtungen:

- Auswärtsuche ist schnell
- Flaschenhals ist der lineare Sweep
- Speicherbandbreite das Problem



GPHAST - Ideen

Beobachtungen:

- Auswärtsuche ist schnell
- Flaschenhals ist der lineare Sweep
- Speicherbandbreite das Problem

Idee:

- speicher CH und Distanzarray auf der GPU
- Auswärtssuche auf der CPU
- kopiere Suchraum zur GPU (weniger als 2 kB)
- linearen Sweep auf der GPU

GPHAST - Ideen

Beobachtungen:

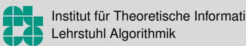
- Auswärtsuche ist schnell
- Flaschenhals ist der lineare Sweep
- Speicherbandbreite das Problem

Idee:

- speicher CH und Distanzarray auf der GPU
- Auswärtssuche auf der CPU
- kopiere Suchraum zur GPU (weniger als 2 kB)
- linearen Sweep auf der GPU

Problem:

- nicht genug Speicher auf GPU um tausende von Bäumen parallel zu bearbeiten
- wir müssen eine einzelne Baumberechnung parallelisieren



Paralleler Linearer Sweep

Beobachtung:

- beim Scannen von Level i:
  - nur eingehende Kanten von level $> i$ wichtig
  - schreiben von Distanzlabeln in Level i, lesen von Level $> i$
  - Distanzlabel für Level $> i$ sind korrekt
- scannen eines Level-i Knoten ist unabhängig von anderen Level-i Knoten

w

$\mathrm{dist}(u)$:

Paralleler Linearer Sweep

Beobachtung:

- beim Scannen von Level i:
  - nur eingehende Kanten von level $> i$ wichtig
  - schreiben von Distanzlabeln in Level i, lesen von Level $> i$
  - Distanzlabel für Level $> i$ sind korrekt
- scannen eines Level-i Knoten ist unabhängig von anderen Level-i Knoten

r

w

$\mathrm{dist}(u)$:

Paralleler Linearer Sweep

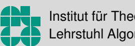
Beobachtung:

- beim Scannen von Level i:
  - nur eingehende Kanten von level $> i$ wichtig
  - schreiben von Distanzlabeln in Level i, lesen von Level $> i$
  - Distanzlabel für Level $> i$ sind korrekt
- scannen eines Level-i Knoten ist unabhängig von anderen Level-i Knoten

r

w

dist(u):



Paralleler Linearer Sweep

Beobachtung:

- beim Scannen von Level *i*:
  - nur eingehende Kanten von level $> i$ wichtig
  - schreiben von Distanzlabeln in Level *i*, lesen von Level $> i$
  - Distanzlabel für Level $> i$ sind korrekt
- scannen eines Level-*i* Knoten ist unabhängig von anderen Level-*i* Knoten

r
w
dist(u):

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

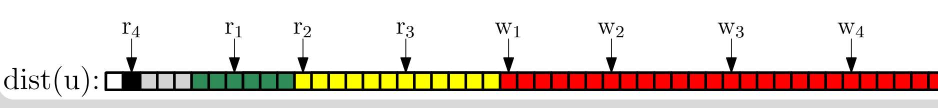
Paralleler Linearer Sweep

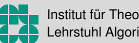
Beobachtung:

- beim Scannen von Level i:
  - nur eingehende Kanten von level $> i$ wichtig
  - schreiben von Distanzlabeln in Level i, lesen von Level $> i$
  - Distanzlabel für Level $> i$ sind korrekt
- scannen eines Level-i Knoten ist unabhängig von anderen Level-i Knoten

Idee:

- scanne alle Knoten auf Level i parallel
- Synchronization nach jedem Level
- ein Thread pro Knoten





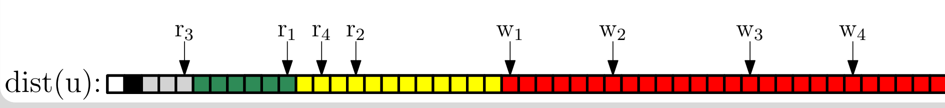
Paralleler Linearer Sweep

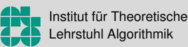
Beobachtung:

- beim Scannen von Level i:
  - nur eingehende Kanten von level $> i$ wichtig
  - schreiben von Distanzlabeln in Level i, lesen von Level $> i$
  - Distanzlabel für Level $> i$ sind korrekt
- scannen eines Level-i Knoten ist unabhängig von anderen Level-i Knoten

Idee:

- scanne alle Knoten auf Level i parallel
- Synchronization nach jedem Level
- ein Thread pro Knoten





Paralleler Linearer Sweep

Beobachtung:

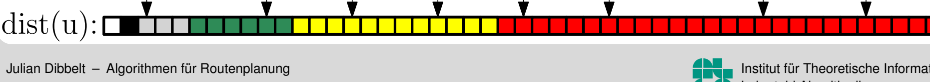
- beim Scannen von Level *i*:
  - nur eingehende Kanten von level $> i$ wichtig
  - schreiben von Distanzlabeln in Level *i*, lesen von Level $> i$
  - Distanzlabel für Level $> i$ sind korrekt
- scannen eines Level-*i* Knoten ist unabhängig von anderen Level-*i* Knoten

Idee:

- scanne alle Knoten auf Level *i* parallel
- Synchronization nach jedem Level
- ein Thread pro Knoten

results:

- 5.5 ms auf NVIDIA GTX 580
- Beschleunigung von 511 gegenüber Dijkstra



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Paralleler Linearer Sweep

Beobachtung:

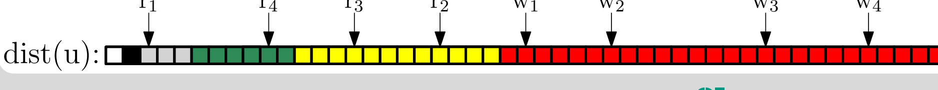
- beim Scannen von Level *i*:
  - nur eingehende Kanten von level $> i$ wichtig
  - schreiben von Distanzlabeln in Level *i*, lesen von Level $> i$
  - Distanzlabel für Level $> i$ sind korrekt
- scannen eines Level-*i* Knoten ist unabhängig von anderen Level-*i* Knoten

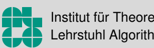
Idee:

- scanne alle Knoten auf Level *i* parallel
- Synchronization nach jedem Level
- ein Thread pro Knoten

results:

- 5.5 ms auf NVIDIA GTX 580
- Beschleunigung von 511 gegenüber Dijkstra
- (mehrere Bäume: 2.2 ms)

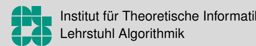




GPHAST

Eigenschaften:

- Haupt- $\rightarrow$ GPU Transfer langsam
  $\Rightarrow$ PHAST kopiert nur 2 kB pro Baum
- keine Cache coherency
  $\Rightarrow$ Berechung ist unabhängig in einem Level
- Single Instruction Multiple Threads innerhalb eines Warps
  $\Rightarrow$ Durchschnittsgrad ist klein
- barrel processing gegen DRAM-Latenz
  $\Rightarrow$ Levelgrösse $>>$ 10000
- Zugriff innerhalb eines Warps nur für bestimmte Zugriffsmuster effizient
  $\Rightarrow$ Wir greifen sequentiell-parallel auf arrays zu

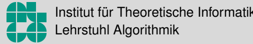


GPHAST

Eigenschaften:

- Haupt- $\rightarrow$ GPU Transfer langsam
  $\Rightarrow$ PHAST kopiert nur 2 kB pro Baum
- keine Cache coherency
  $\Rightarrow$ Berechung ist unabhängig in einem Level
- Single Instruction Multiple Threads innerhalb eines Warps
  $\Rightarrow$ Durchschnittsgrad ist klein
- barrel processing gegen DRAM-Latenz
  $\Rightarrow$ Levelgrösse $>>$ 10000
- Zugriff innerhalb eines Warps nur für bestimmte Zugriffsmuster effizient
  $\Rightarrow$ Wir greifen sequentiell-parallel auf arrays zu

$\Rightarrow$ **PHAST passt ins GPU Rechenmodell**



Initialisierung

Beobachtung:

- initialisieren der Knotenarrays nach jedem Lauf zu langsam (ca. 10 ms)
- Counteransatz zuviel Speicherverbrauch

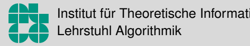
Initialisierung

Beobachtung:

- initialisieren der Knotenarrays nach jedem Lauf zu langsam (ca. 10 ms)
- Counteransatz zuviel Speicherverbrauch

Idee:

- benutze Marker für CH Einträge
- Aufwärtssuche setzt den Marker
- während Sweep, wenn Marker nicht gesetzt, interpretiere als ∞
- Sweep entfernt Marker



PHAST

PHAST auf 4-Kern Workstation (Core-i7 920)

sources/ sweep	time per tree [ms]					
	1 core		2 cores		4 cores	
1	171.9		86.7		47.1	
4	121.8	(67.6)	61.5	(35.5)	32.5	(24.4)
8	105.5	(51.2)	53.5	(28.0)	28.3	(20.8)
16	96.8	(37.1)	49.4	(22.1)	25.9	(18.8)

Werte in Klammern mit SSE aktiviert

GPHAST

PHAST auf Nvidia GTX 580

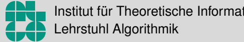
trees / sweep	memory [MB]	time [ms]
1	395	5.53
2	464	3.93
4	605	3.02
8	886	2.52
16	1448	2.21

weitere Eingaben

		Europe		USA	
algorithm	device	time	distance	time	distance
Dijkstra	4-core workstation	947.72	609.19	1269.12	947.75
	12-core server	288.81	177.58	380.40	280.17
	48-core server	168.49	108.58	229.00	167.77
PHAST	4-core workstation	18.81	22.25	27.11	28.81
	12-core server	7.20	8.27	10.42	10.71
	48-core server	4.03	5.03	6.18	6.58
GPHAST	GTX 580	2.21	3.88	3.41	4.65

Beobachtung:

- Beschleunigung für Distanzmetrik geringer



All-Pairs Shortest Paths

Eingabe: Europa mit Reisezeiten

algorithm	device	time	energy [MJ]
Dijkstra	4-core workstation	197d	
	12-core server	60d	
	48-core server	35d	
PHAST	4-core workstation	94h	
	12-core server	36h	
	48-core server	20h	
GPHAST	GTX 580		

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

All-Pairs Shortest Paths

Eingabe: Europa mit Reisezeiten

algorithm	device	time	energy [MJ]
Dijkstra	4-core workstation	197d	
	12-core server	60d	
	48-core server	35d	
PHAST	4-core workstation	94h	
	12-core server	36h	
	48-core server	20h	
GPHAST	GTX 580	11h	

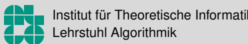
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

All-Pairs Shortest Paths

Eingabe: Europa mit Reisezeiten

algorithm	device	time	energy [MJ]
Dijkstra	4-core workstation	197d	
	12-core server	60d	
	48-core server	35d	
PHAST	4-core workstation	94h	
	12-core server	36h	
	48-core server	20h	
GPHAST	GTX 580	11h	

4-core workstation without GPU: 163 watts
4-core workstation with GPU: 375 watts
12-core server: 332 watts
48-core server: 747 watts

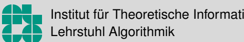


All-Pairs Shortest Paths

Eingabe: Europa mit Reisezeiten

algorithm	device	time	energy [MJ]
Dijkstra	4-core workstation	197d	2780.6
	12-core server	60d	1725.9
	48-core server	35d	2265.5
PHAST	4-core workstation	94h	55.2
	12-core server	36h	43.0
	48-core server	20h	54.2
GPHAST	GTX 580	11h	14.9

4-core workstation without GPU: 163 watts
4-core workstation with GPU: 375 watts
12-core server: 332 watts
48-core server: 747 watts



Baumrekonstruktion

bis jetzt:

- nur Distanzen berechnet, nicht Bäume

Baumrekonstruktion

bis jetzt:

- nur Distanzen berechnet, nicht Bäume

Idee:

- iteration über alle Kanten
- setze $p(u) = v$ wenn $d(v) + \text{len}(v, u) = d(u)$

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Baumrekonstruktion

bis jetzt:

- nur Distanzen berechnet, nicht Bäume

Idee:

- iteration über alle Kanten
- setze $p(u) = v$ wenn $d(v) + \text{len}(v, u) = d(u)$
- speicher Kanten als Triples $(u, v, \text{len}(u, v))$
- ein Thread pro Kante

Baumrekonstruktion

bis jetzt:

- nur Distanzen berechnet, nicht Bäume

Idee:

- iteration über alle Kanten
- setze $p(u) = v$ wenn $d(v) + \text{len}(v, u) = d(u)$
- speicher Kanten als Triples $(u, v, \text{len}(u, v))$
- ein Thread pro Kante
- ein *linearer* Sweep über den Graphen
- erhöht Berechnungszeit um einen Faktor 2

Nächste Termine

Mittwoch, 29.5.2013 (Julian)

Institut für Theoretische Informatik
Lehrstuhl Algorithmik