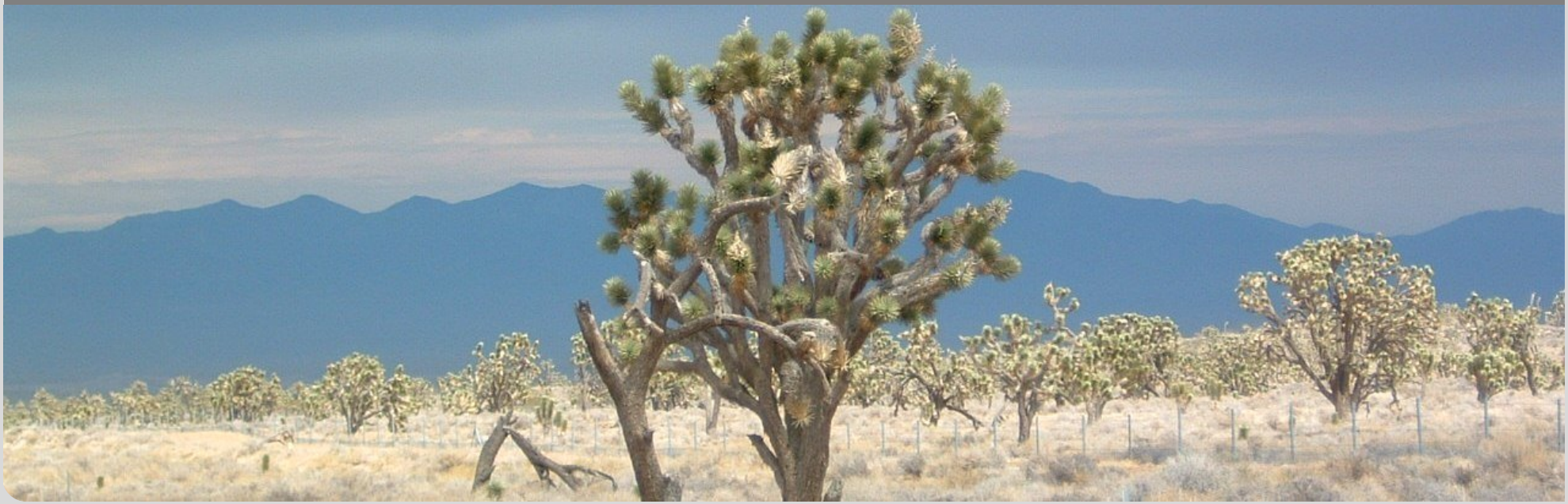
Karlsruher Institut für Technologie

Algorithmen für Routenplanung

10. Sitzung, Sommersemester 2012

Daniel Delling | 4. Juni 2012

MICROSOFT RESEARCH SILICON VALLEY



Was bisher geschah

bisheriger Stoff:

- Punkt-zu-Punkt Abfragen
- statisches Szenario

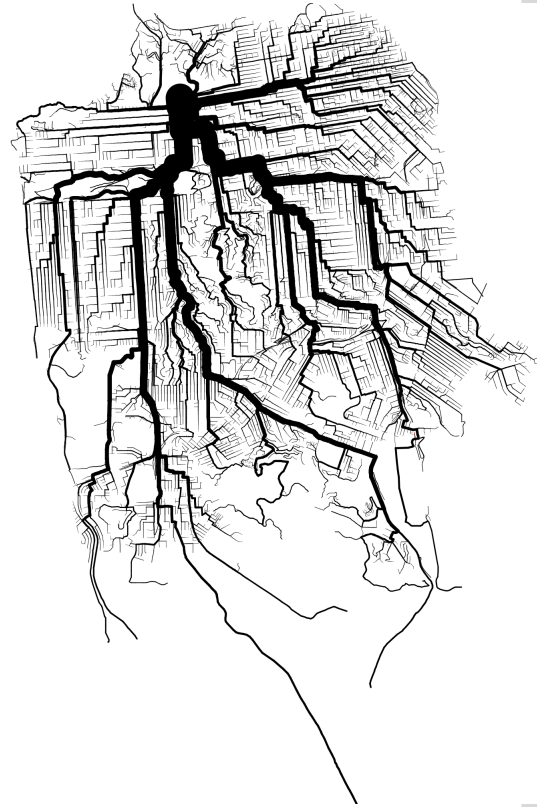
Erweiterte Anfragen

- 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

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

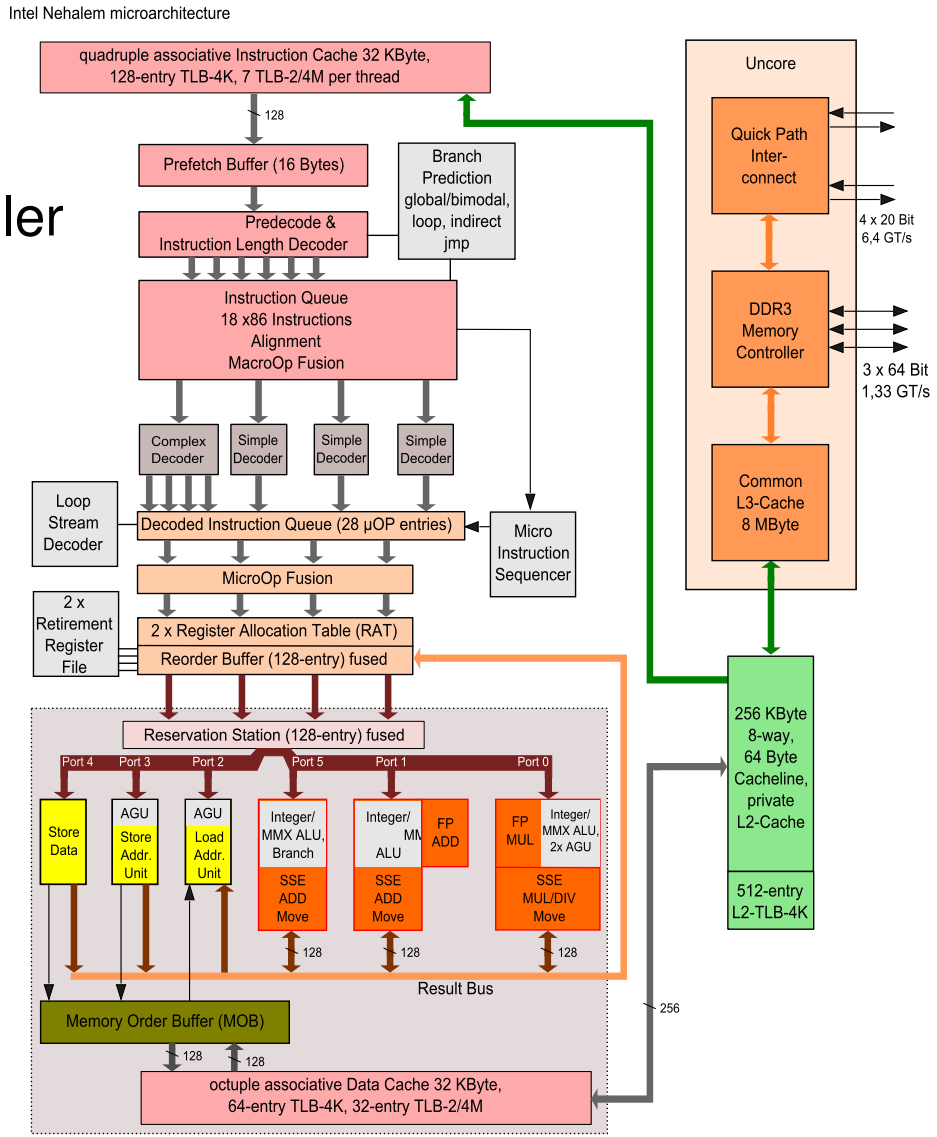
Moderne CPU Architektur

Einige Fakten:

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

Hauptherausforderungen:

- Parallelisierung
- Speicherzugriff

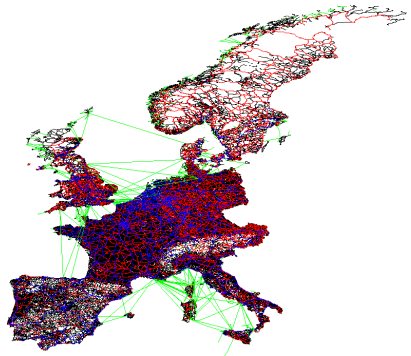


Ausnutzen von Moderner Hardware

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 Daten Struktur

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

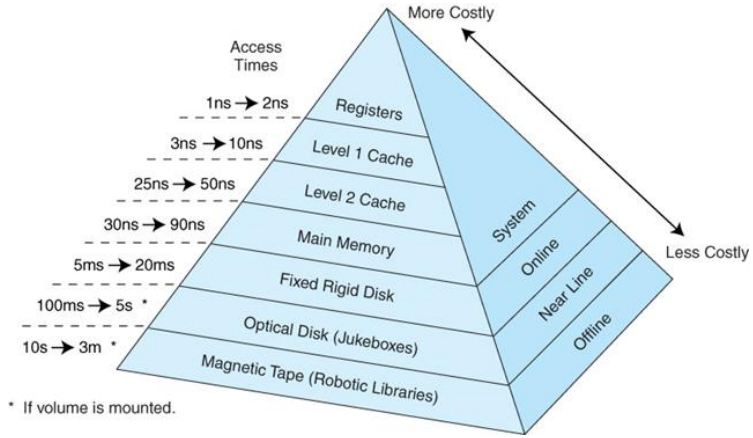
		time per tree [ms]		
algorithm	details	random	input	DFS
Dijkstra	binary heap	11159	5859	5180
	Dial	7767	3538	2908
	smart queue	7991	3556	2826
BFS	—	6060	2445	2068

$\Rightarrow$ keine grosse Beschleunigung

Ansatz 2

Dijkstra's Algorithmus:

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



Fragen:

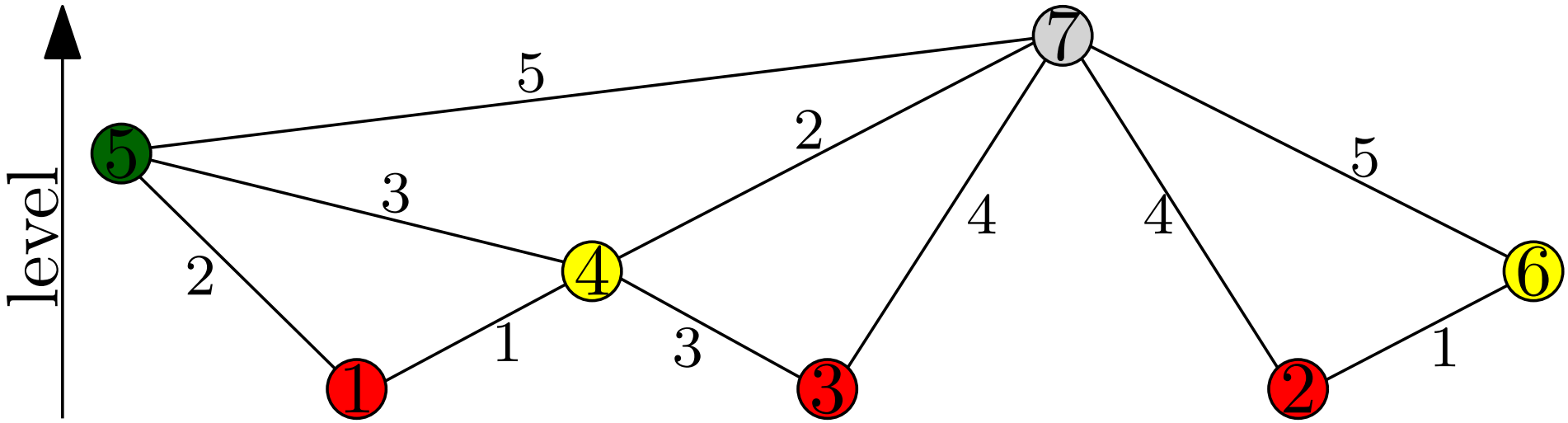
- hilft Vorberechnung?
- wie?
- Ansatzpunkt?

PHAST: Hardware-Accelerated Shortest path Trees

Contraction Hierarchies

preprocessing:

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



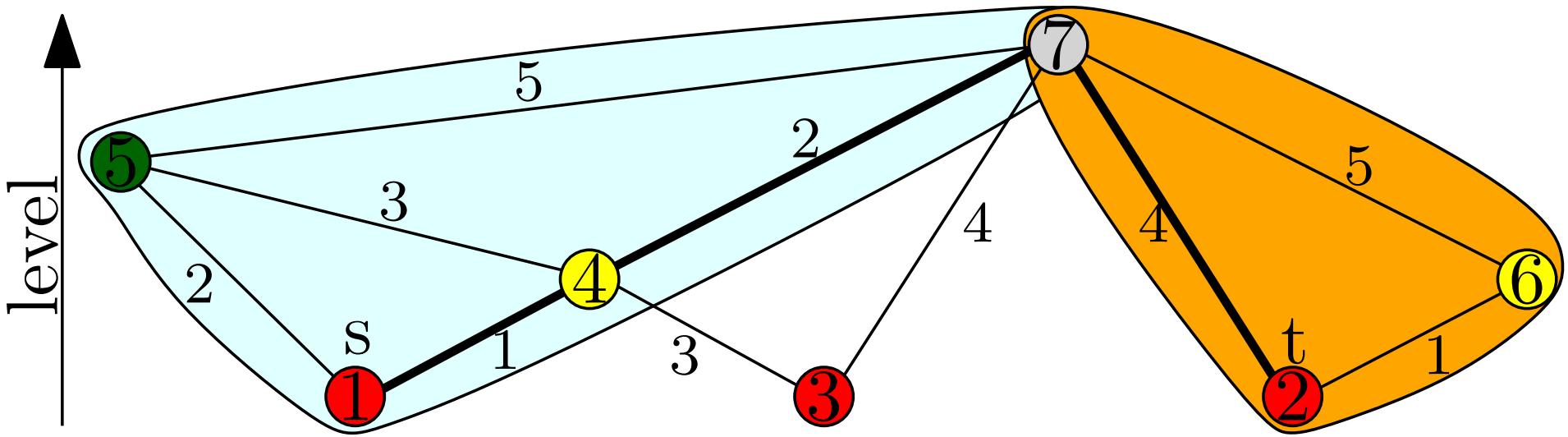
Contraction Hierarchies

Punkt-zu-Punkt Anfragen

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

Korrektheit:

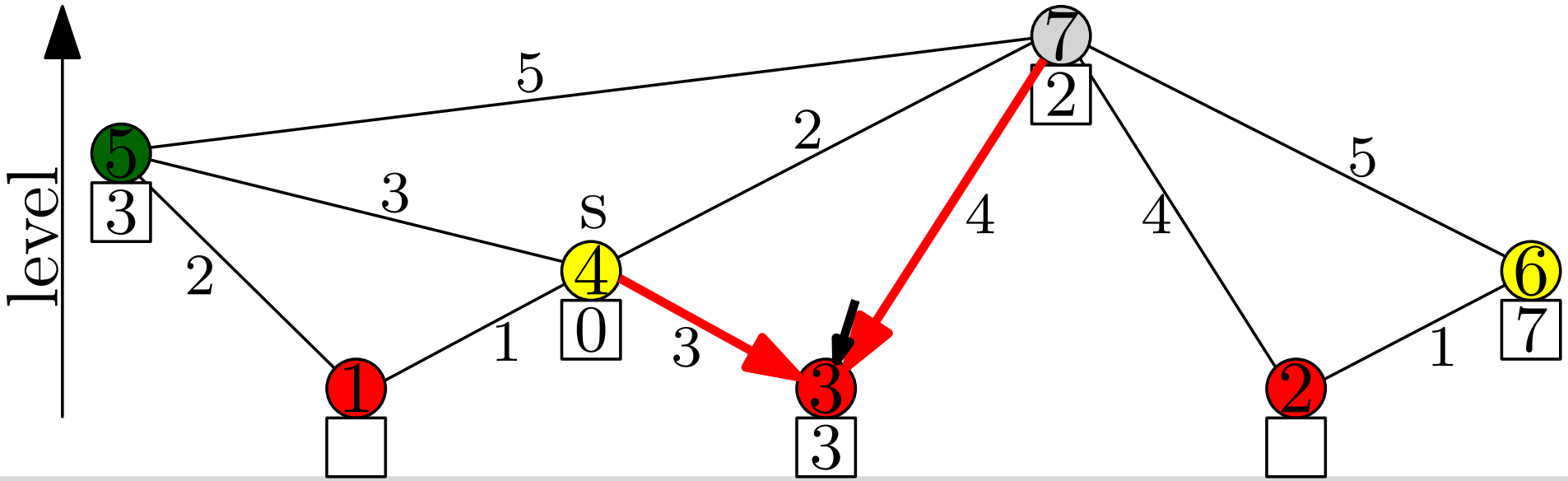
- es gibt einen wichtigsten Knoten auf dem Pfad
- dieser wird von Vorwärts- und Rückwärtssuche gescannt



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:

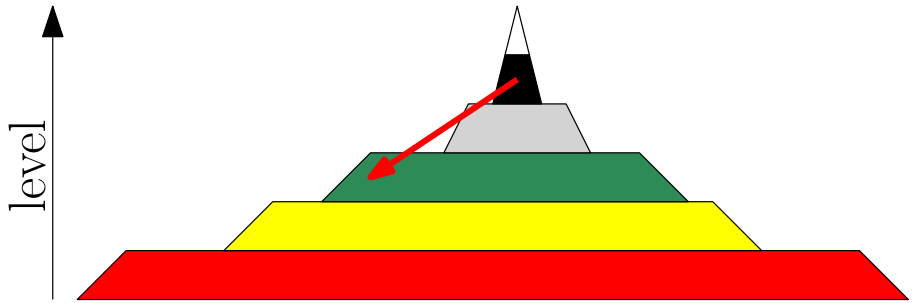
- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch ineffizient
- Zugriffsmuster sind unabhängig von s

Idee:

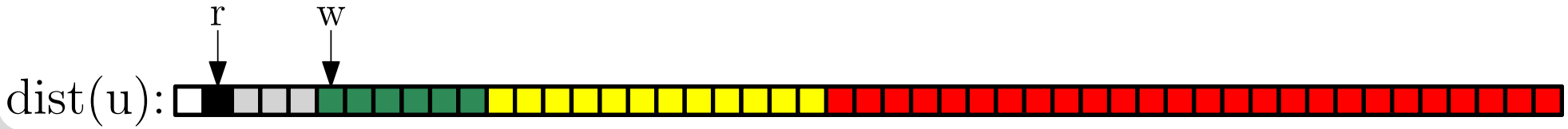
- speicher $G_\uparrow$ und $G_\downarrow$ separat
- Umordnung der Knoten, Kanten, und Distanzlabel nach Level

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

$\Rightarrow$ 172 ms pro Baum



- aber lesen der Distanzen immer noch ineffizient



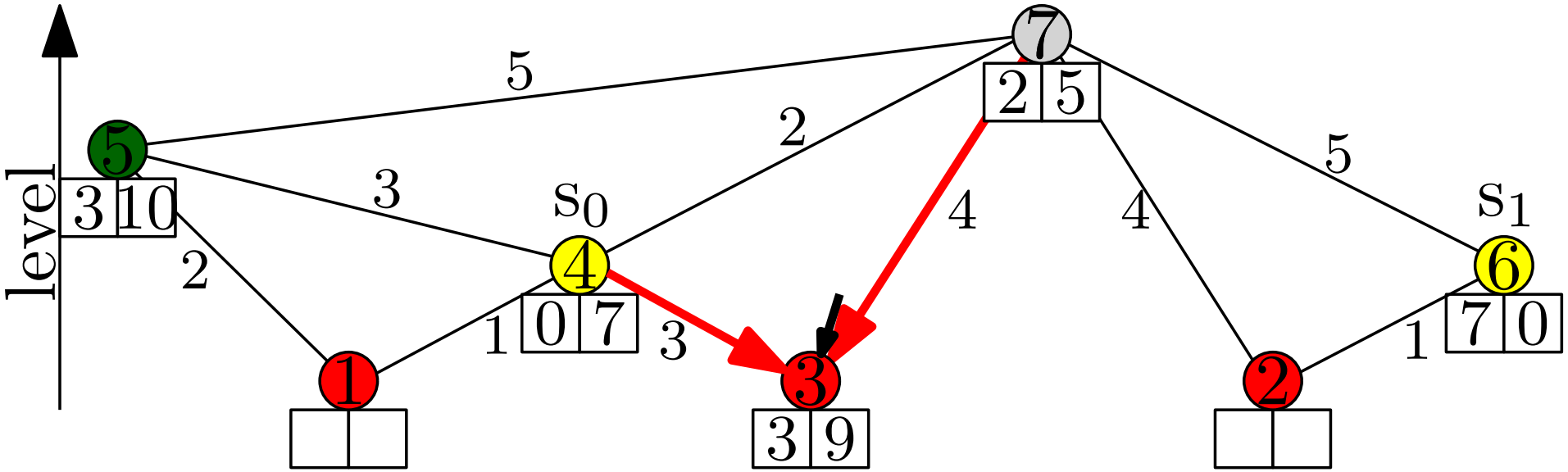
Szenario: Multiple Startknoten

Idee:

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

SSE:

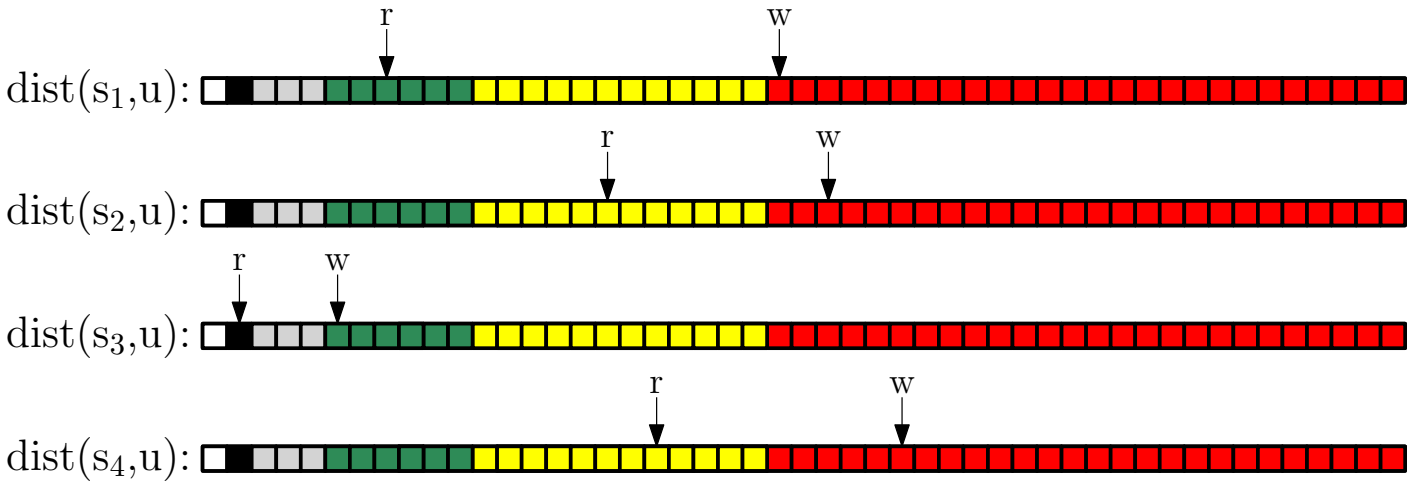
- 128-bit Register
- Basisoperationen (min, add) für vier 32-bit Integer parallel
- scanne 4 Distanzlabel auf einmal
- 37.1 ms pro Baum ($k = 16$)
- Sandy Bridge Architektur: 256-bit Register



Parallelisierung

ganz einfach

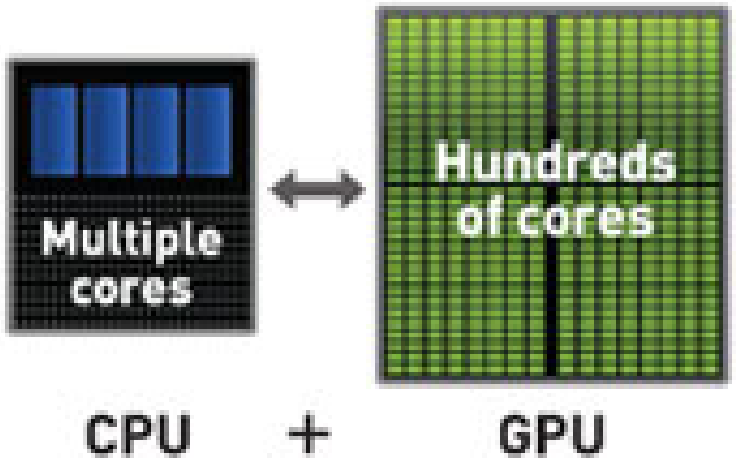
- nach Startknoten



Ergebnisse:

- 16 Startenknoten pro Sweep (updates via SSE)
- multi-core nach Startknoten $\Rightarrow$ 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 $\Rightarrow$ 512 parallele Threads
- eingeschränkte Berechnungen

GPU Architektur

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

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:

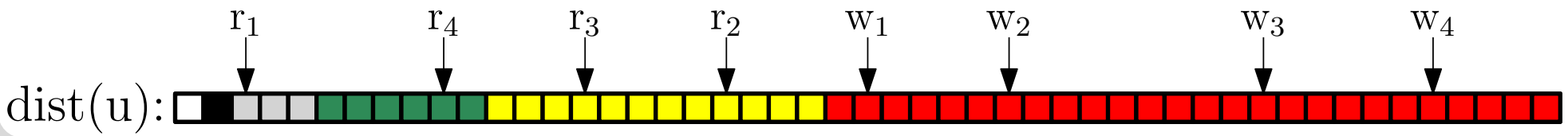
- beim Scannen von Level i:
  - nur eingehende Kanten von level $> i$ wichtig
  - schreiben von Distanzlabeln in Level i, lesen von Level $> i$
  - Distanzlabel für Level $> i$ sind korrekt
- scannen eines Level-i Knoten ist unabhängig von anderen Level-i Knoten

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
  $\Rightarrow$ we 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

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	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

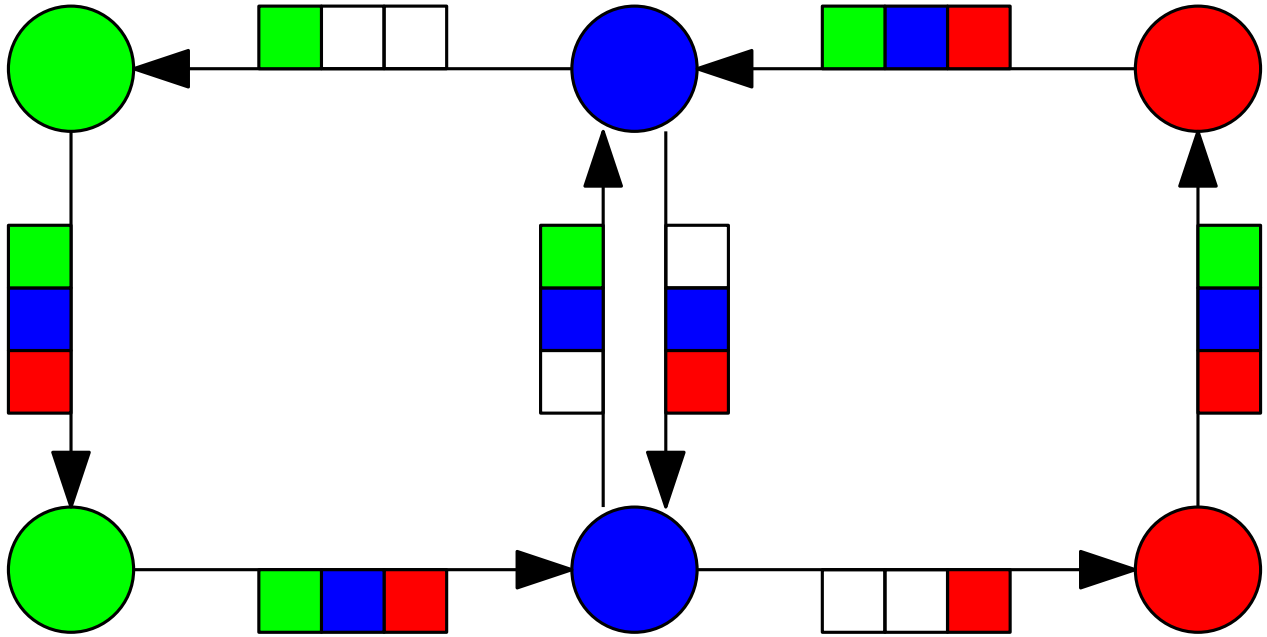
Idee:

- 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

Arc-Flags

Idee:

- setze 1-shell Kanten automatisch
- benutze GPHAST zum Berechnen der Bäume
- setze Flaggen durch zusätzlichen Sweep auf GPU
- Vorberechnung sinkt von 17 Stunden auf 3 Minuten
- ohne Partitionierung (2 Minuten mit PUNCH) und CH Vorberechnung (2 Minuten)



CHASE

Eigenschaften

- Setzen von Flaggen nur für wichtige Kanten (5% wichtigsten Knoten)
- Flaschenhals: Baumberechnung

Idee:

- benutze GPHAST zum Berechnen von Bäumen
- setze Flaggen für alle Kanten
- verzögere Kontraktion von Original-Randknoten

$\Rightarrow$ durch Kontraktion erhöht sich Anzahl Randknoten nicht zu sehr

$\Rightarrow$ Anzahl Randknoten: 22k (11k in G)

$\Rightarrow$ 15 Minuten Vorberechnung, 28 gescannte Knoten, 5.4 μs Anfragzeit

Hybrid GPHAST

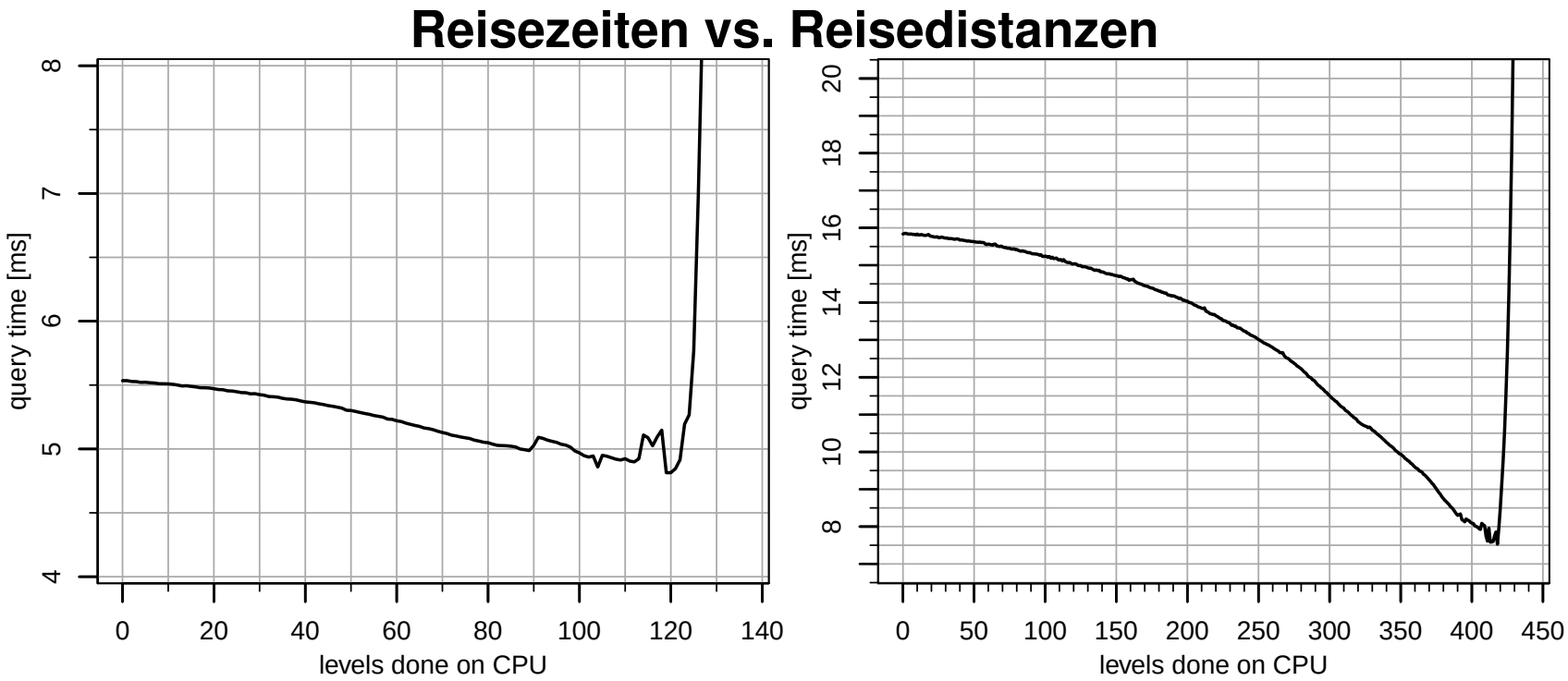
Beobachtung:

- Synchronization des Level kostet Zeit auf der GPU (5 μs pro Level)
- oberen Level sind klein

Idee:

- beginne linearen Sweep auf CPU (bis level k)
- kopiere Suchraum und all Distanzlabel für Knoten oberhalb k zur GPU
- restlicher Scan auf der GPU

Hybrid GPHAST: Ergebnisse



Es lohnt sich, ein Paar Level auf der CPU zu berechnen.

Zusammenfassung

- neuer Algorithmus für kürzeste Wege Bäume
- skaliert auf Modern Architektur
- ein Baum auf GPU: 5.5 ms
  (ungefähr 0.31 ns pro Eintrag)
- real-time Berechnung von kompletten Bäumen
- 16 Bäume auf einer GPU auf einmal: 2.2 ms pro Baum
  (ungefähr 0.13 ns pro Eintrag)
- APSP in 11 Stunden (auf workstation mit einer GPU),
  anstellen von 6 Monaten (auf 4 Kernen)
- erlaub APSP-basierte Berechnungen
- 150 mal Energie-effizienter als Dijkstras Algorithmus
- funktioniert nur, wenn CH funktioniert

Ende

Literatur:

- Daniel Delling, Andrew V. Goldberg, Andreas Nowatzyk, Renato F. Werneck
  PHAST: Hardware-Accelerated Shortest path Trees
  In: *Journal of Parallel and Distributed Computing*, 2012

Nächste Termine

Karlsruher Institut für Technologie

Mittwoch, 6.6.2012

Montag, 11.6.2012
Mittwoch, 13.6.2012
Montag, 18.6.2012
Mittwoch, 20.6.2012