Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Smart Cars

Ausarbeitung im Zuge des Praktikums „Parallele Programmierung“

Rami Aly und Christoph Hüter

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Agenda

I. Problembeschreibung
II. Modellbeschreibung
III. Sequenzielles Programm
    I. Preprocessing
    II. Routentabelle und Simulation
    III. Visualisierung
    IV. Laufzeiten
IV. Parallelisierung
    I. Routingtabelle
        I. Methode a)
        II. Methode b)
    II. Bewegung der Autos/Simulation
V. Performanceanalyse und Skalierbarkeit
VI. Verbesserungen und Ausblick

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Kontext

Gegeben: Karte eines Straßennetzes, Autos mit Start- und Zielorten, die zu unterschiedlichen Zeitpunkten die Karte betreten

Ziel: Konstruktion von optimalen Wegen und intelligente Berechnung eines möglichst optimalen Weges von Fahrzeugen zu einem Ziel in komplexen Straßennetzen

- Intelligent bedeutet hier, dass Autos stark befahrende Straßen und Staus verhindern, um somit ihr Ziel schneller erreichen.
- Hohe Komplexität und Skalierbarkeit des Problems gute Voraussetzung für Parallelisierung

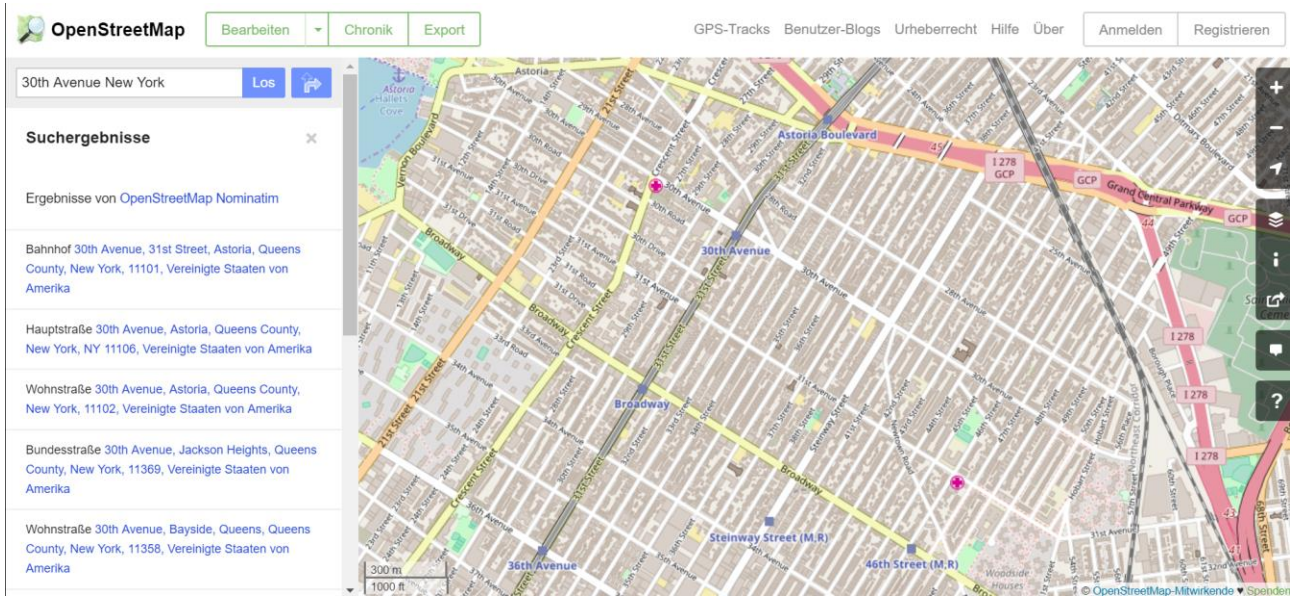
Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Problem- und Modellbeschreibung

- Gerichteter Graph (V, E)
- Kante hat Kapazitätsfunktion e
- Kantengewichtsfunktion f, abhängig von e
- Spawner $S \subseteq V$ Eintrittspunkt für Marken
- Ampelfunktonalität von Knoten übernommen

Sequentielles Programm - Preprocessing

- Kartendatenbank OpenStreetMap(Geofabrik)
- Pythonscript zum Extrahieren der Knoten und relevanten Straßen
- Anpassung und Normierung der Koordinaten und Längen



VertexID	PosX	PosY
…	….	…

VertexID start	VertexID end
….	….

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Lösungsansatz

1. Routentabelle und Timetable
2. Bewegung und Simulation der Autos

Sequentielles Programm - RoutingTable

- Erzeuge Graph aus extrahierten Informationen im C++ Programm
- Randknoten werden als Spawner ausgewählt (beliebig änderbar)
- Benutze A* Start Algorithmus und Luftlinie als Heuristik zur Berechnung (Boost Graph Library [BGL])

Pfade:

ID	5	18
5	()	(.....cba)
18	(abc....)	()

Kosten:

ID	5	18
5	0	Distanz({1,2})
18	Distanz({1,2})	0

Nachbarn:

ID	K_nearest
5	$\{n_1,....,n_k\}$
18	$\{n_1,....,n_k\}$
19	$\{n_1,....,n_k\}$

Sequentielles Programm - Simulation

- Spawner erzeugen Marken mit Ziel
- Suche Pfad mit niedrigstem Wert:

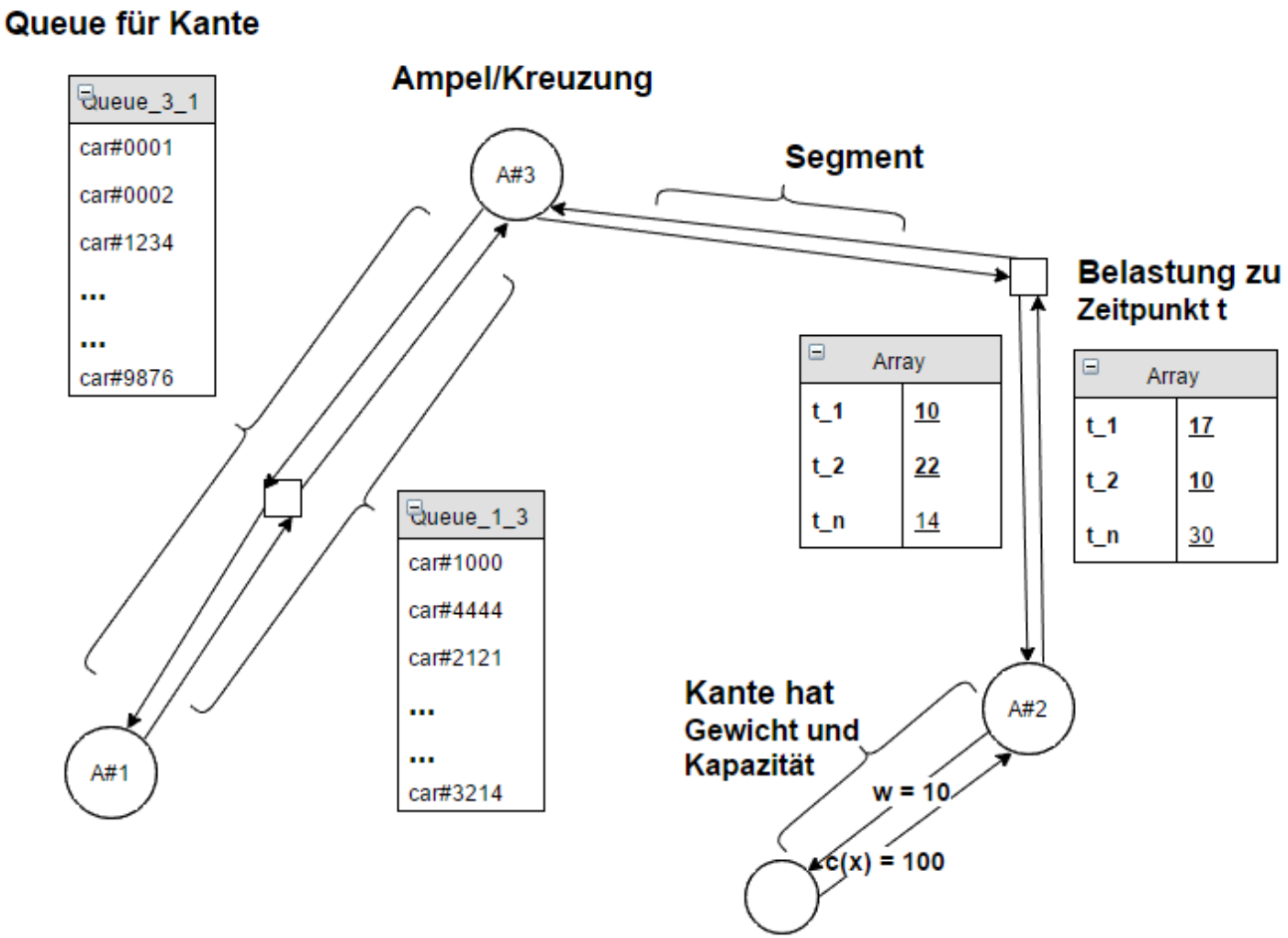
$$\theta(v,w) = c[v,w] + \sum_{e \in RT[v,w].E} f_e(t_e) + dist(w_m, w)$$

- Jede Kante speichert in einer Timetable für jeden Zeitpunkt voraussichtliche Belegung
- Nach Wahl der Route aktualisiere die Timetable für alle betroffenen Kanten mit Heurisitk

Sequentielles Programm - Simulation

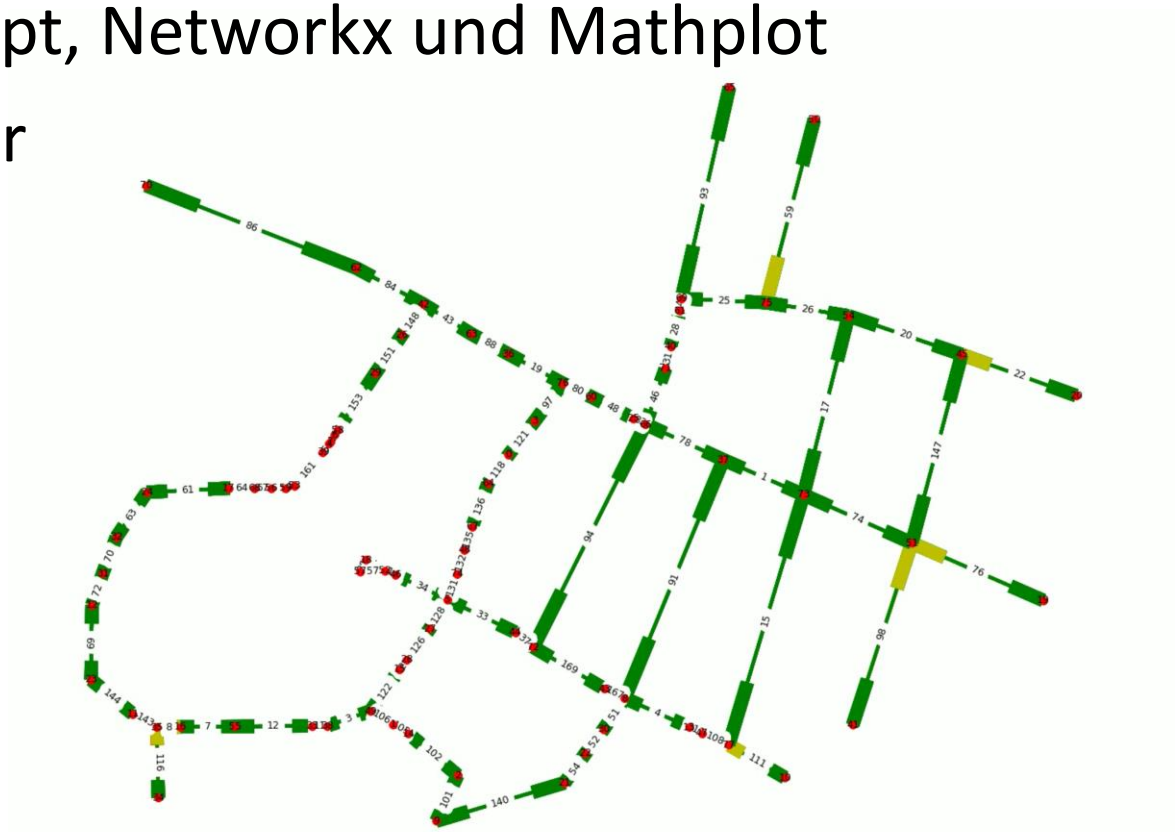
- Bewegung von Marke innerhalb der Kante wird über eine Schlange geregelt
- Das Wechseln der Kante wird von dem verbundenen Knoten gesteuert
  - Bedingung: Ampel grün und noch Platz auf der Kante
- Bewegung der Autos geschieht in zwei Schritten:
  - Bewege Marke und merke falls es Edge überschreiten möchte
  - Prüfe ob Marke Edge überschreiten kann und bewege es um verbleibende Strecke
- Verlässt Marke Kante, aktualisiere Gewicht und zerstöre sie ggf.
- Speichere Belegung und Kapazität jeder Kante nach jedem Step in Datei ab

Sequentielles Programm - Simulation



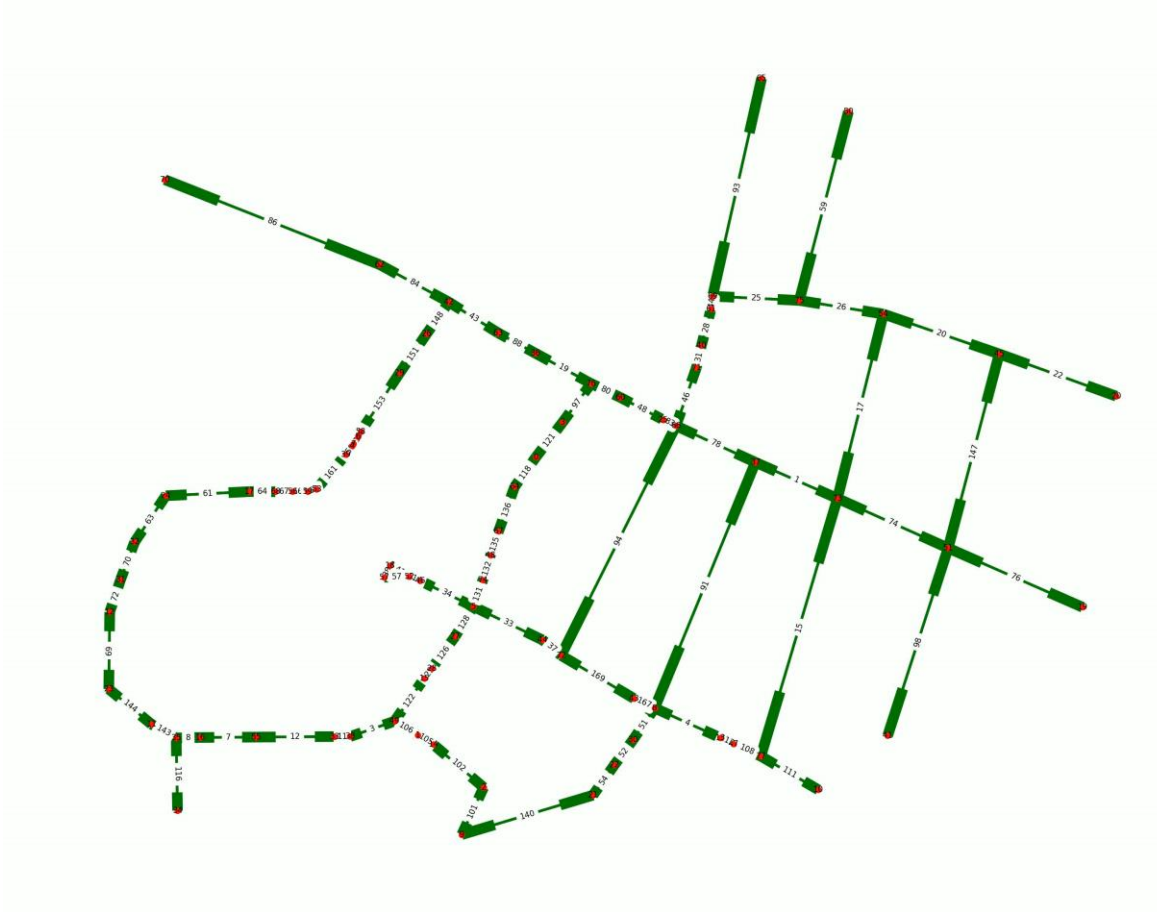
Sequentielles Programm - Visualisierung

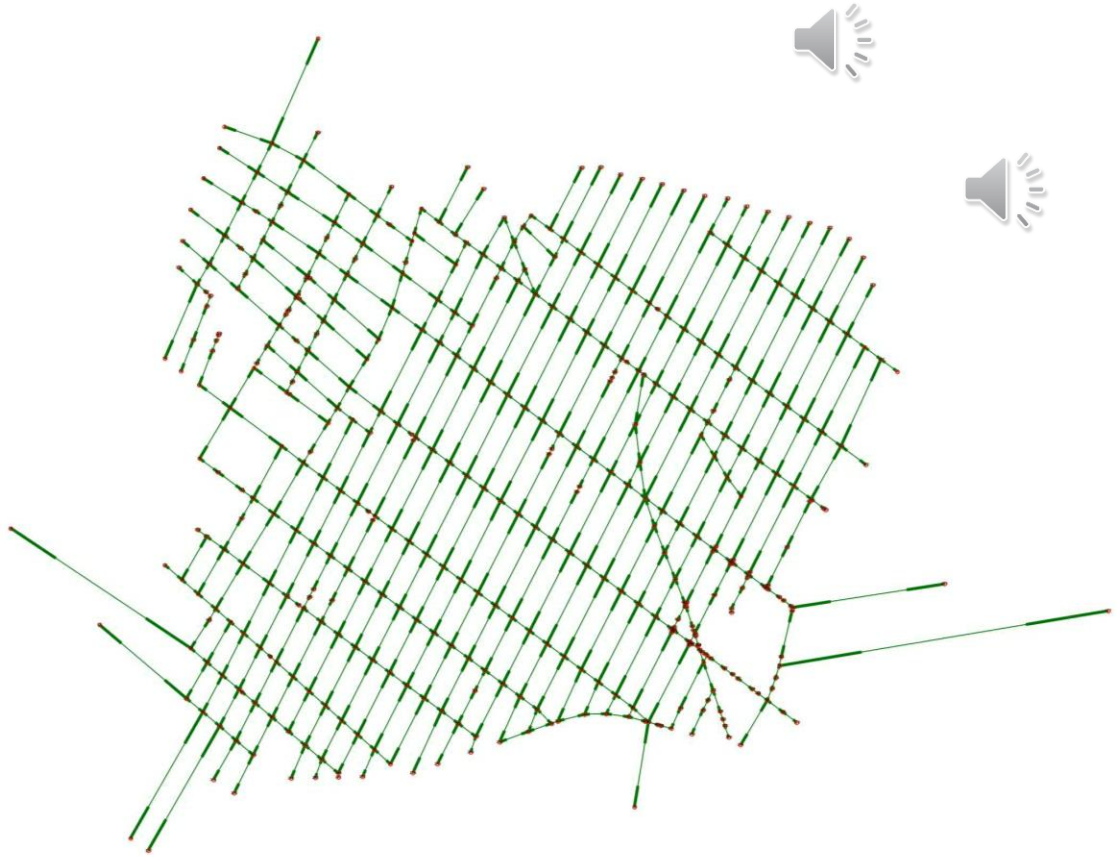
- Visualisierung des Graphens und der Dichte der Autos auf den Kanten.
- Visualisierung mit Pyton Script, Networkx und Mathplot
- Erzeuge Animation aus Bilder



Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Sequenzielles Programm - Visualisierung





Sequentielles Programm - Laufzeiten

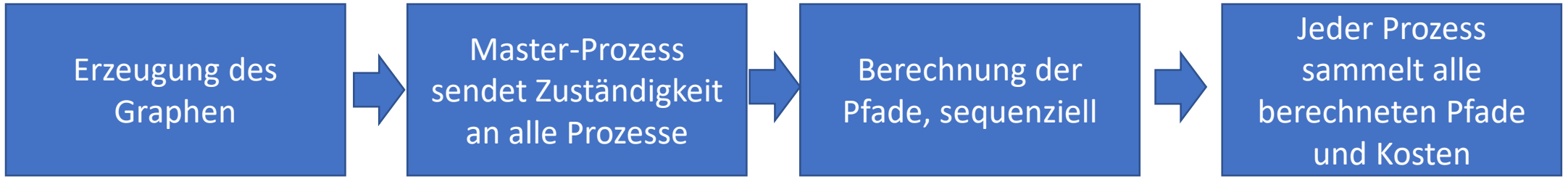
10 Durchläufe Durchschnitt:

- Knoten: 78, Kanten: 170, Steps: 200
  - RoutingTabelle: 0:00:857, Optimiert: 0:00:75
  - Simulation: 00:41

- Knoten: 424, Kanten: 1244, Steps: 200
  - RoutingTabelle: 1:37, Optimiert: 1:06
  - Simulation: 27:23

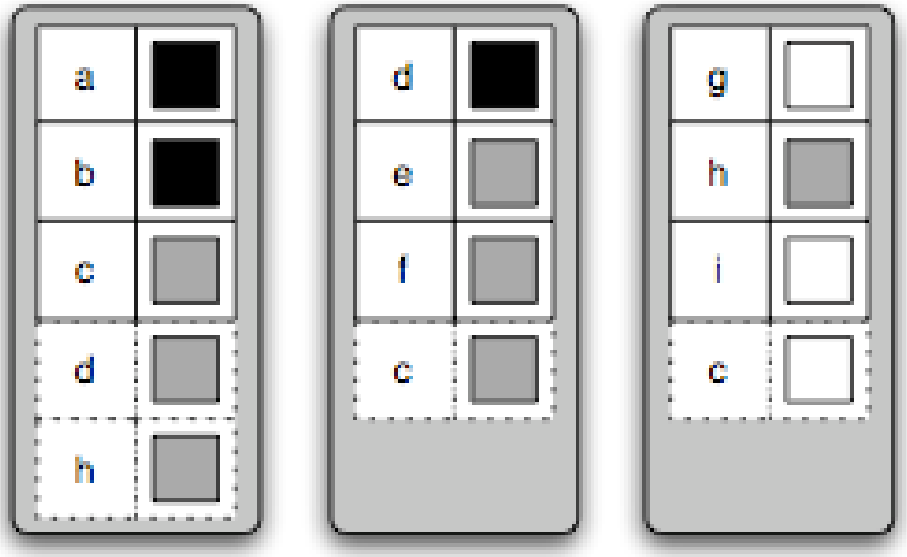
Parallelisierung – RoutingTable, Methode a)

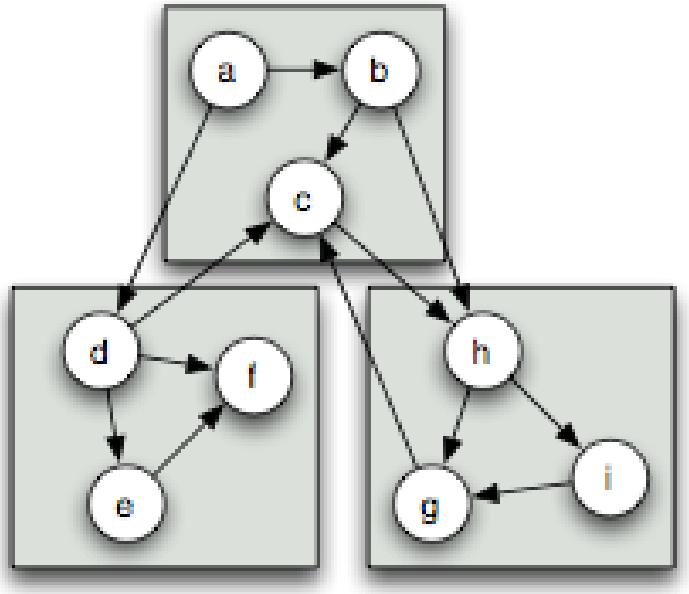
- Berechnung des Pfades sequenziell, parallel für mehrere Spawner durchführen
- Scatterv zur Aufgabenverteillung
- Gatherv für alle Prozesse, zum sammeln der Pfade



Parallelisierung – RoutingTable, Methode b)

- Nutzung der Parallel BGL
- Berechnung des kürzesten Pfades parallel, nacheinander für alle Spawner





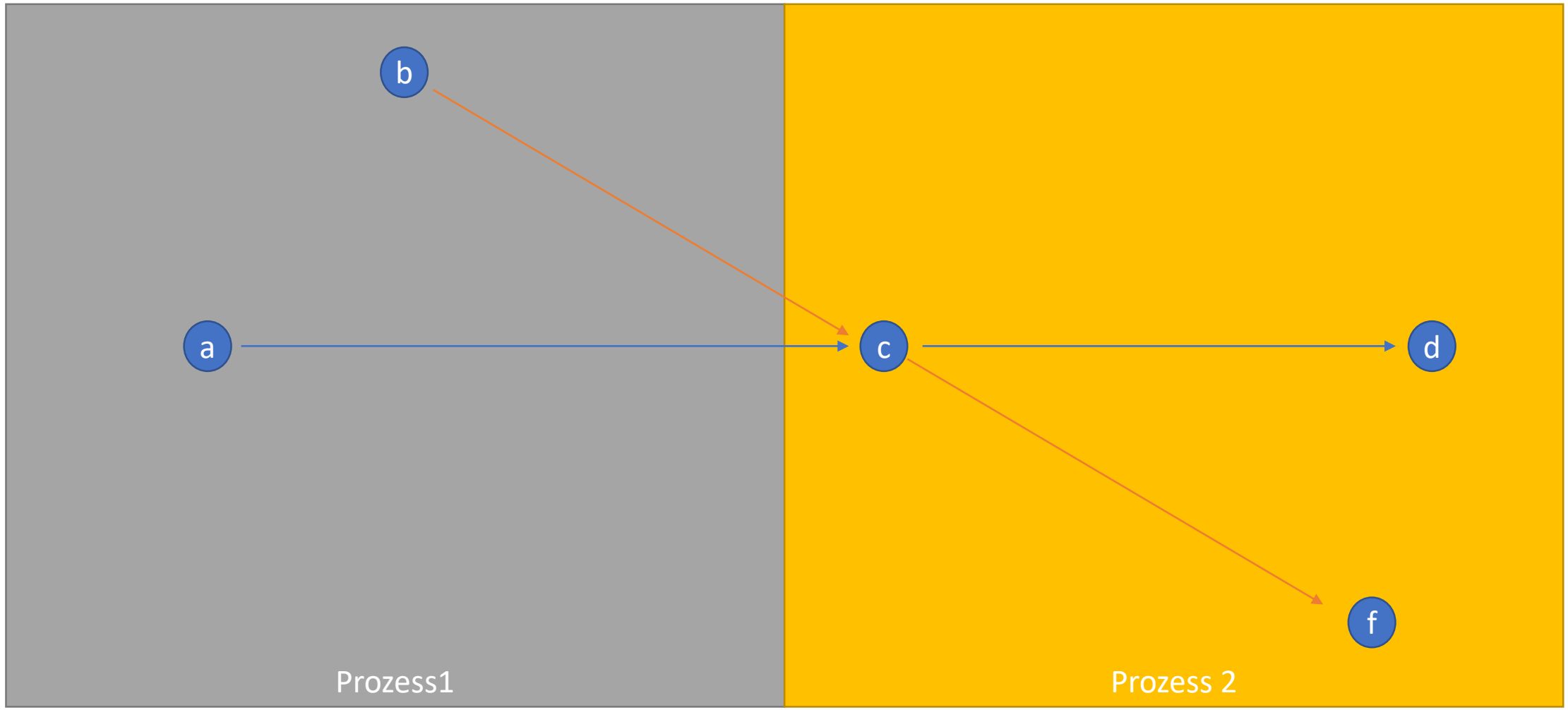
Quelle: http://www.boost.org/doc/libs/1_65_0/libs/graph_parallel/doc/html/overview.html

Parallelisierung - Simulation

- Ordne Prozessen Routen zu
- Falls mehrere Routen gleiche Kante verwenden, ordne eindeutig einem Prozess zu
- Kommunikation, falls Auto Straße eines anderen Prozesses betritt
- Kommunikation, falls beim Spawnen Kantengewicht auf fremder Edge aktualisiert wird

Ziel: Wenig Kommunikation bei Aktualisierung des Kantengewichtes, da Route möglichst konsistent einem Prozess zugeordnet wird (Vorteil gegenüber Zerteilung des Graphs)

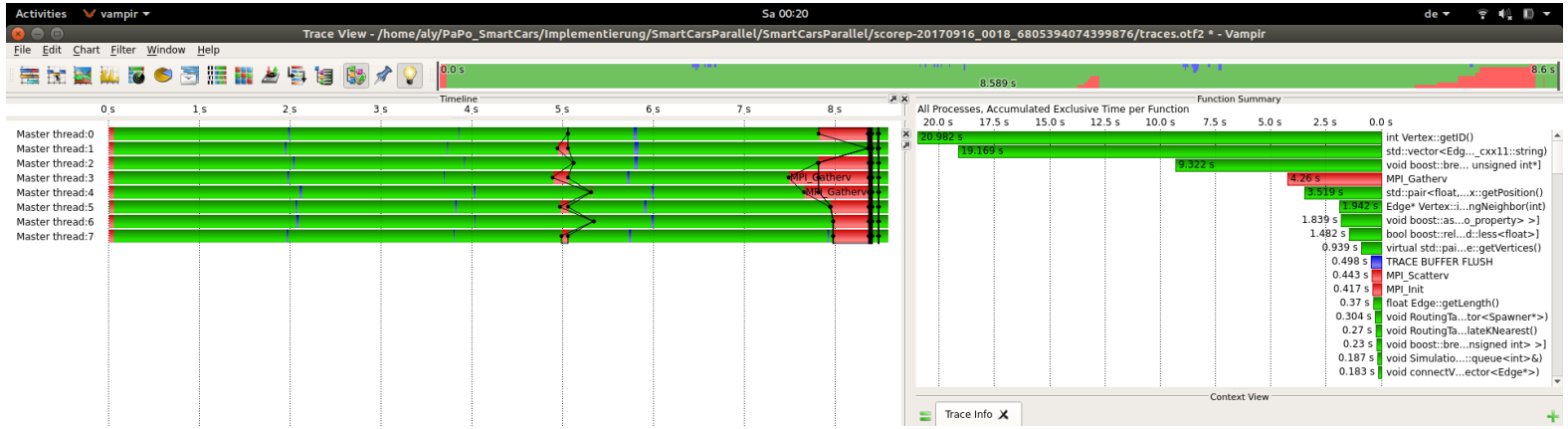
Parallelisierung - Simulation

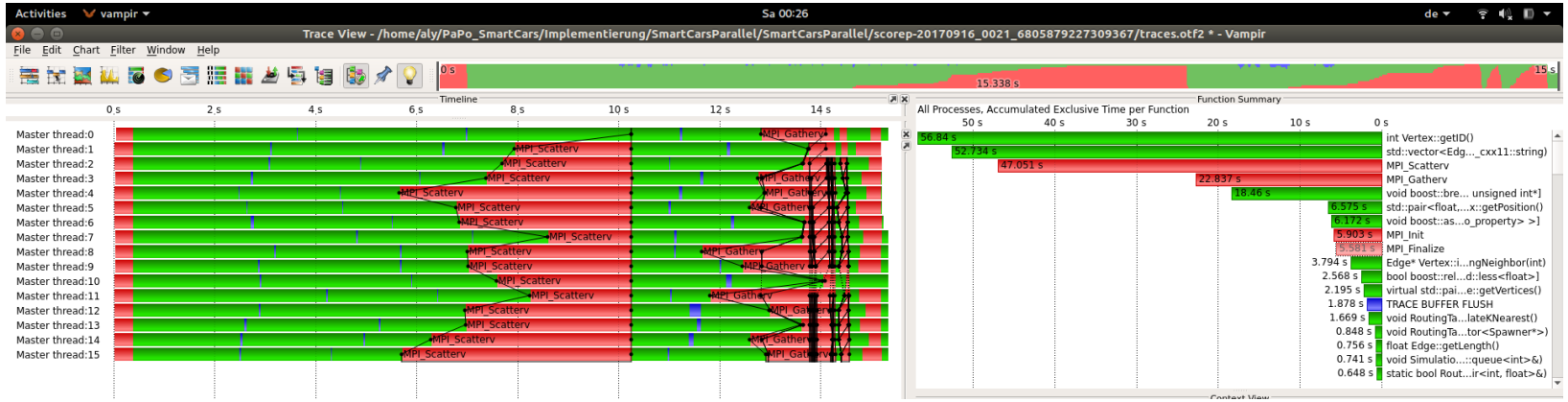


Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Performanceanalyse und Skalierbarkeit

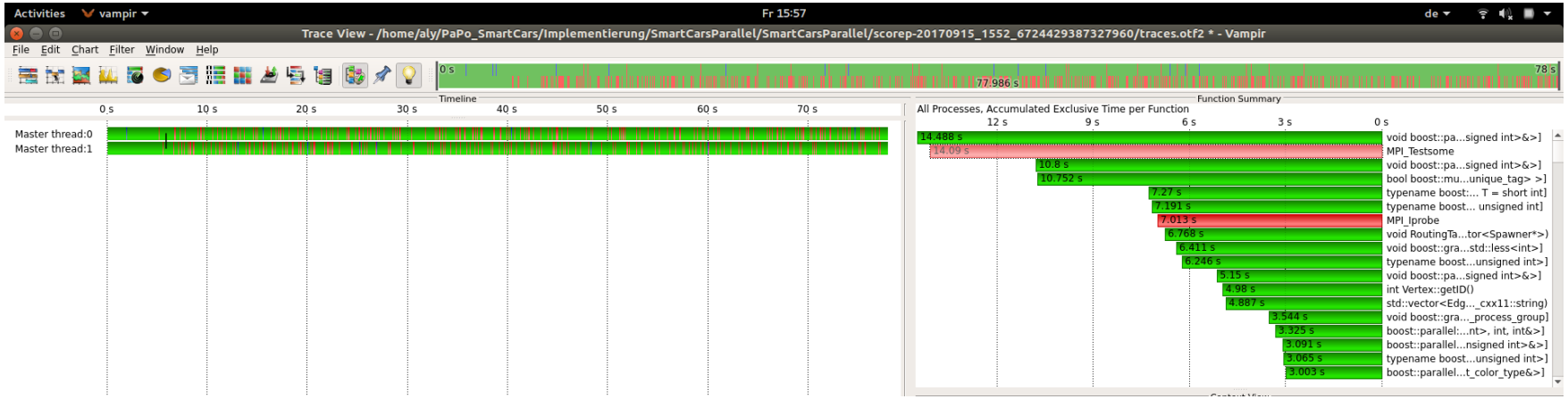
RoutingTable a), Analyse mit Vampir und Oprofile:

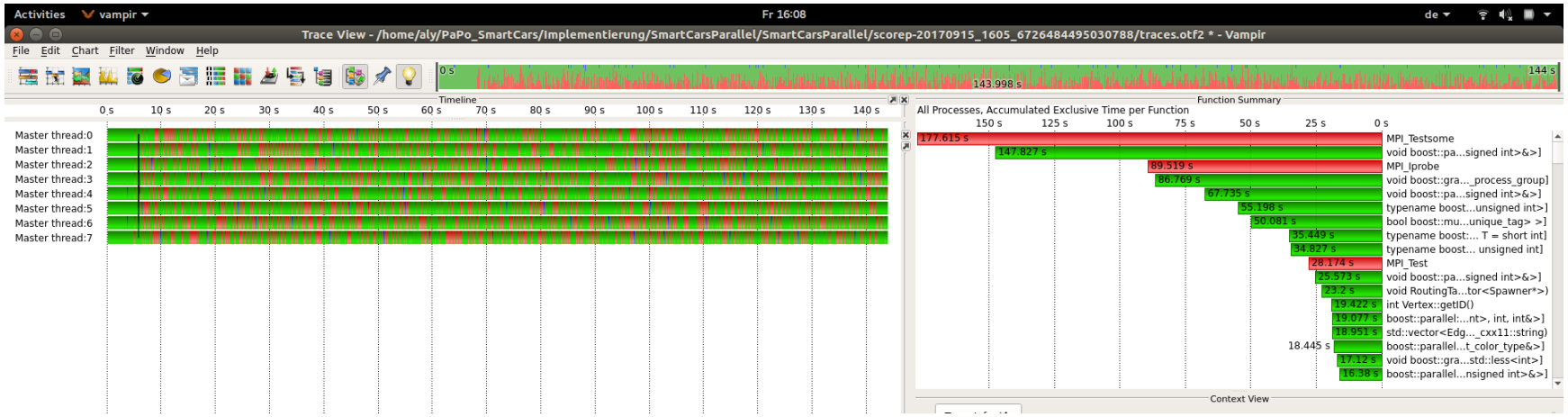




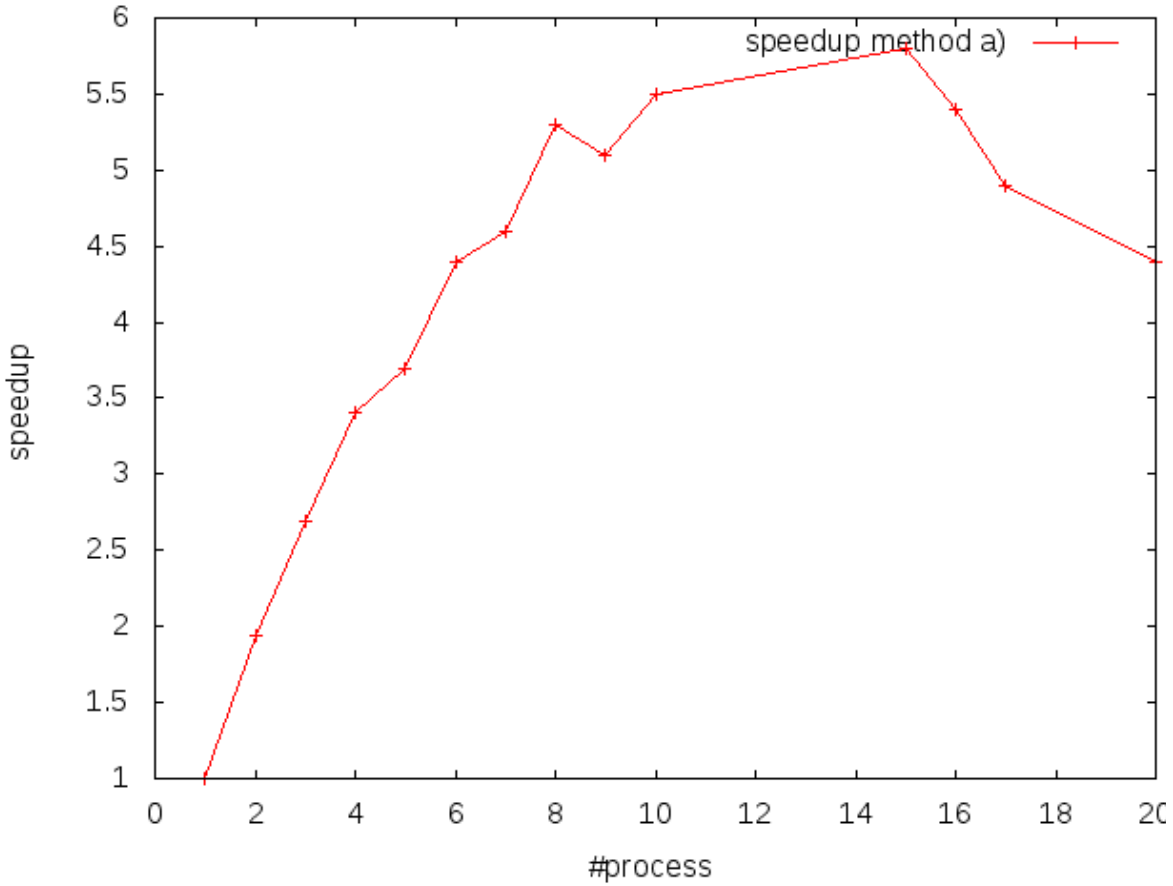
Performanceanalyse und Skalierbarkeit

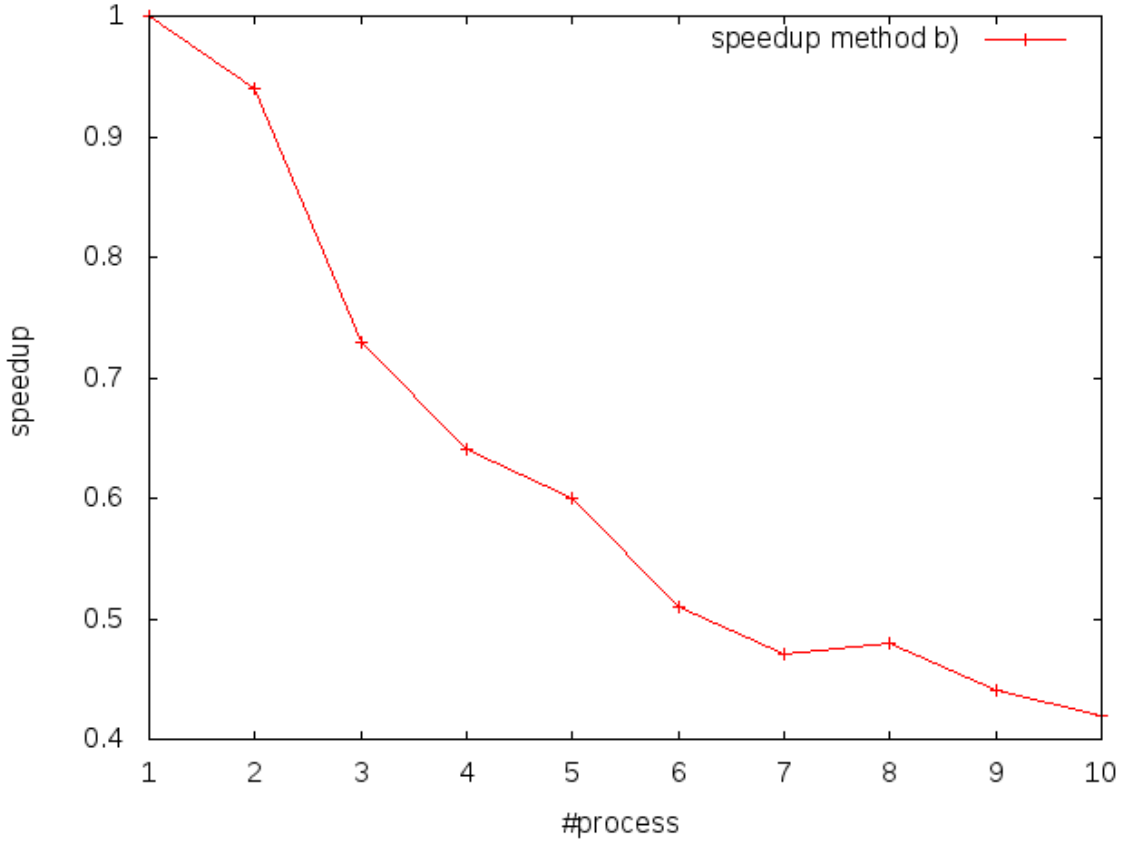
RoutingTable b), Analyse mit Vampir und Oprofile:





Performanceanalyse und Skalierbarkeit





alternative Karte: 16700 Knoten, 39982 Kanten bis 3 Prozesse getestet, bei n = **3** Speedup von **0.984**

Verbesserungen

- Bei der Parallelisierung der RoutingTables auch doppelte Berechnungen filtern
- Bessere Heuristik für TimeTables finden
- Funktion einbauen, dass alle n-ten optimalen Routen gespeichert und verwendet werden
- Parallel BGL: Eager Dijkstra oder Delta Routing verwenden, da bessere strong scaling