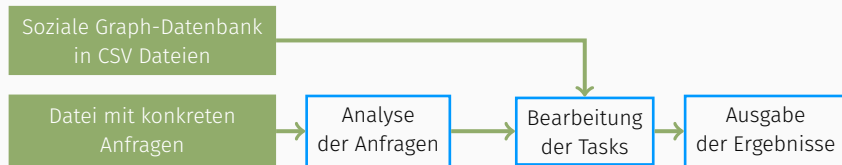


HOCHPERFORMANTE ANALYSEN IN GRAPH-DATENBANKEN

Moritz Kaufmann, Tobias Mühlbauer, Manuel Then,
Andrey Gubichev, Alfons Kemper, Thomas Neumann

5. März 2015

Technische Universität München

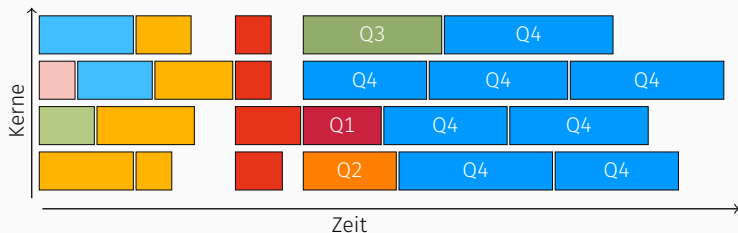


4 Anfragetypen

1. Kürzeste Wege finden
2. Top-K größten Interessengruppen von Personen finden, die nach einem Tag b geboren wurden
3. K-Hop Nachbarschaften von Knoten analysieren
4. Die zentralsten Knoten finden

Parallelisierung
Effiziente Breitensuchen
Schranken zur Suchraumbegrenzung

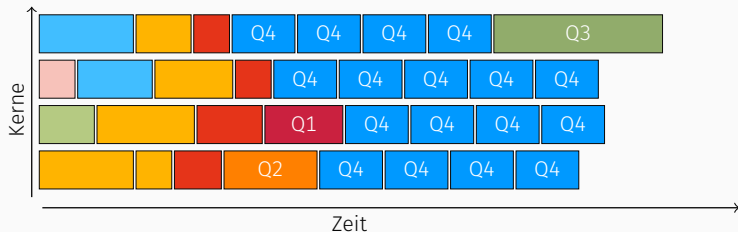
Aufteilung: Laden → Indexe erstellen → Anfragen
+ Dateien aufteilen



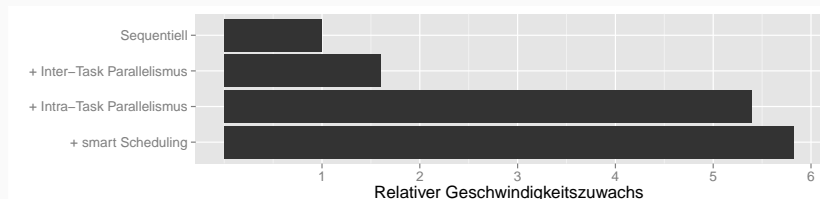
Probleme: *Weiße Flächen bedeuten Ineffizienz!*

- Unterschiedliche Laufzeiten
- Ineffizienz bei den Barrieren

- Besser:**
- + Q4 parallelisieren
 - + Feingranulare Taskabhängigkeiten



Probleme: Priorisierung



Aber weshalb nicht Faktor 8?

- I/O skaliert nicht linear
- Mehrere Threads teilen sich den CPU-Cache

Anfragetyp 2: Eine Breitensuchen pro Interesse
→ 10.000 Breitensuchen pro Anfrage

Anfragetyp 4: Eine Breitensuche pro Knoten
→ 1 Millionen Breitensuchen pro Anfrage

Scalable Graph Exploration on Multicore Processors

Vinay Agarwal¹ Fabrizio Petrini² Davide Paoletti² David A. Bader³
¹IBM TJ Watson, Yorktown Heights, NY 10598, USA
²IBM Computational Science Center, Dublin, Ireland
³College of Computing, Georgia Tech, Atlanta, GA 30332, USA
vinayagarwal@cs.cmu.edu, fpetrini@cs.cmu.edu, paoletti@cs.cmu.edu, bader@cc.gatech.edu

Parallel Breadth-First Search on Distributed Memory Systems

Vinay Agarwal¹ Fabrizio Petrini² Davide Paoletti² David A. Bader³
¹IBM TJ Watson, Yorktown Heights, NY 10598, USA
²IBM Computational Science Center, Dublin, Ireland
³College of Computing, Georgia Tech, Atlanta, GA 30332, USA
vinayagarwal@cs.cmu.edu, fpetrini@cs.cmu.edu, paoletti@cs.cmu.edu, bader@cc.gatech.edu

Parallel Breadth-First Search on Distributed Memory Systems

Aydin Buluc¹ Kamesh Madduri²
¹Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA
²Department of Computer Science, University of California, Berkeley, CA
{ABuluc, KMadduri}@lbl.gov

Parallel Breadth-First Search on Distributed Memory

Aydin Buluc¹ Kamesh Madduri²
¹Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA
²Department of Computer Science, University of California, Berkeley, CA
{ABuluc, KMadduri}@lbl.gov

In-Core Computation of Geometric Hypergraph Centralities

Paolo Boldi¹ Sebastiano Vigna²
¹Dipartimento di Informatica, Università degli Studi di Milano
²Department of Informatics, University of Cambridge
{pboldi, vigna}@cam.ac.uk

Parallel Distributed

Aydin Buluc¹ Kamesh Madduri²
¹Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA
²Department of Computer Science, University of California, Berkeley, CA
{ABuluc, KMadduri}@lbl.gov

Direction-Optimizing Breadth-First Search

Paolo Boldi¹ Sebastiano Vigna²
¹Dipartimento di Informatica, Università degli Studi di Milano
²Department of Informatics, University of Cambridge
{pboldi, vigna}@cam.ac.uk

A Hundred Billion

D. A. Bader and K. Madduri. Designing Multithreaded Algorithms for Breadth-First Search and St-connectivity on the Cray MTA-2. In *ICPP '06*, pages 523-530, 2006.

[12] F. Checconi, F. Petrini, J. Willcock, A. Lumsdaine, A. R. Choudhury, and Y. Sabharwal. Breaking the Speed and Scalability Barriers for Graph Exploration on Distributed-Memory Machines. In *SC '12*, pages 13:1-13:12, 2012.

h-First Search

Sebastiano Vigna
Department of Informatics, University of Cambridge
vigna@cam.ac.uk

Anfragetyp 2: Eine Breitensuchen pro Interesse
→ 10.000 Breitensuchen pro Anfrage

Anfragetyp 4: Eine Breitensuche pro Knoten
→ 1 Millionen Breitensuchen pro Anfrage

Anfragetyp 2: Eine Breitensuchen pro Interesse

→ 10.000 Breitensuchen pro Anfrage

Anfragetyp 4: Eine Breitensuche pro Knoten

→ **1 Millionen Breitensuchen pro Anfrage**

Parallelisierung hat immer Zusatzkosten!

Parallelisierung einzelner Breitensuchen unnötig um Kerne auszulasten.

Suche: Optimierung von Breitensuchen *ohne Parallelisierung* auf großen Graphen.

Liste:

```
1 for (NodeId start : graph) {
2   queue<NodeId> frontier, next;   frontier.push(start);
3   vector<bool> seen(graph.size()); seen[start] = true;
4
5   while (!frontier.empty()) {
6     for (v : frontier) {
7       for (NodeId n : neighbors(v)) {
8         if (seen[n]) continue;
9         next.push(n);
10        seen[n] = true;
11        do_something_with(n);
12      } }
13
14    handle_round_end();
15    frontier = next;
16    next.clear();
17  } }
```

```
1 for (NodeId start : graph) {
2   queue<NodeId> frontier, next;   frontier.push(start);
3   vector<bool> seen(graph.size()); seen[start] = true;
4
5   while (!frontier.empty()) {
6     for (v : frontier) {
7       for (NodeId n : neighbors(v)) {
8         if (seen[n]) continue;
9         next.push(n);
10        seen[n] = true;
11        do_something_with(n);
12      } }
13
14    handle_round_end();
15    frontier = next;
16    next.clear();
17  } }
```

Langsam!

Problem(e):

Zufällige

Speicherzugriffe

→ Cache misses

→ CPU muss warten

Fix: Manuelles Prefetching? Sortierung der Nachbarn? Sortierung der Queues? ...

Einsicht: Zufällige Zugriffsmuster *können nicht vermieden* werden.

Idee: Mehrere Breitensuchen innerhalb einer Iteration ausführen.

```
queue<NodeId> frontier, next;  
vector<bool> seen(graph.size());
```

```
if (!seen[n]) {  
    ...  
}
```

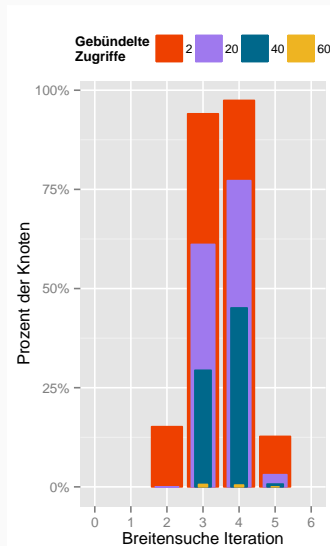


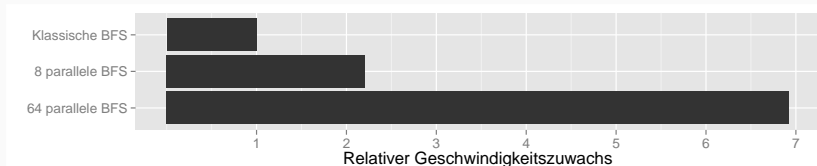
```
queue<NodeId, BfsSet> frontier, next;  
vector<BfsSet> seen(graph.size());
```

```
cur_seen = seen[n];  
for (b: activeBfs) {  
    if (!cur_seen.contains(b)) {  
        ...  
    }  
}
```

Viele große Graphen haben folgende Eigenschaften:

- **Kleiner Durchmesser:** Alle Knoten sind sehr nah beieinander
- Die Anzahl der Nachbarn pro Knoten folgt dem **power law**
- Beispiele: Soziale Netzwerke, Webgraphen, Kommunikationsgraphen





Gut skalierbar bis zu $64 * 8 = 512$ Breitensuchen pro Kern mit SIMD!
Bisherige Benchmarks (Graph500) decken diesen Use-Case nicht ab.

The More the Merrier:
Efficient Multi-Source Graph Traversal

Aufgabe: Finden der *Top-K* größten Interessengruppen von Personen, die nach einem Tag b geboren wurden.

Beispiel: $k=3$, b =Februar 1995

Aktuelle Top-K Werte

Interesse	Größte CC
Fußball	9920
Brasilien	8137
Wandern	4219

Aufgabe: Finden der *Top-K* größten Interessengruppen von Personen, die nach einem Tag *b* geboren wurden.

Beispiel: $k=3$, b =Februar 1995

Aktuelle Top-K Werte

Interesse	Größte CC
Fußball	9920
Brasilien	8137
Wandern	4219

Behelfsindexe

Interesse	Jüngste Person	Personen
Reisen	Mai 1994	12519
Ski fahren	Mai 1996	712
DB	Juni 1990	5
...

Aufgabe: Finden der *Top-K* größten Interessengruppen von Personen, die nach einem Tag *b* geboren wurden.

Beispiel: $k=3$, b =Februar 1995

Aktuelle Top-K Werte

Interesse	Größte CC
Fußball	9920
Brasilien	8137
Wandern	4219

Behelfsindexe

Interesse	Jüngste Person	Personen
Reisen	Mai 1994	12519
Ski fahren	Mai 1996	712
DB	Juni 1990	5
...

Aufgabe: Finden der *Top-K* größten Interessengruppen von Personen, die nach einem Tag *b* geboren wurden.

Beispiel: $k=3$, b =Februar 1995

Aktuelle Top-K Werte

Interesse	Größte CC
Fußball	9920
Brasilien	8137
Wandern	4219

Behelfsindexe

Interesse	Jüngste Person	Personen
Reisen	Mai 1994	12519
Ski fahren	Mai 1996	712
DB	Juni 1990	5
...

Aufgabe: Finden der *Top-K* größten Interessengruppen von Personen, die nach einem Tag *b* geboren wurden.

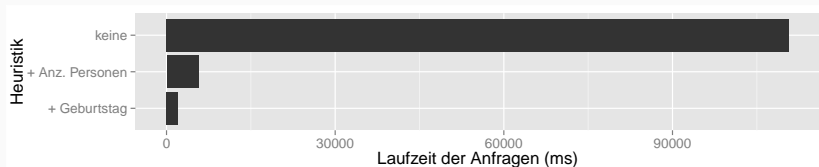
Beispiel: $k=3$, b =Februar 1995

Aktuelle Top-K Werte

Interesse	Größte CC
Fußball	9920
Brasilien	8137
Wandern	4219

Behelfsindexe

Interesse	Jüngste Person	Personen
Reisen	Mai 1994	12519
Ski fahren	Mai 1996	712
DB	Juni 1990	5
...

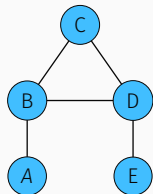


Aufgabe: Finden der k zentralsten Personen im Graph. Zentralität wird bestimmt über die Distanz zu allen anderen Personen.

Gesucht: Eine untere Schranke für die Distanzsumme!
oder: Eine obere Schranke wieviele Knoten mit Distanz k erreicht werden.

Aufgabe: Finden der k zentralsten Personen im Graph. Zentralität wird bestimmt über die Distanz zu allen anderen Personen.

Gesucht: Eine untere Schranke für die Distanzsumme!
oder: Eine obere Schranke wieviele Knoten mit Distanz k erreicht werden.

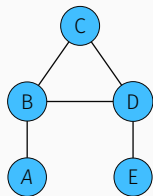


$$\begin{aligned}DSUM(A) &= d(B) + d(C) + d(D) + d(E) \\ &= 1 + 2 + 2 + 3 = 8\end{aligned}$$

$$\begin{aligned}DSUM(A) &= 1 * (|R_1| - |R_0|) + 2 * (|R_2| - |R_1|) + 3 * (|R_3| - |R_2|) \\ &= 1 * (2 - 1) + 2 * (4 - 2) + 3 * (5 - 4) \\ &= 1 * 1 + 2 * 2 + 3 * 1 = 8\end{aligned}$$

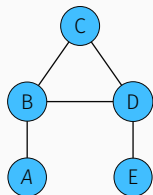
Aufgabe: Finden der k zentralsten Personen im Graph. Zentralität wird bestimmt über die Distanz zu allen anderen Personen.

Gesucht: Eine untere Schranke für die Distanzsumme!
oder: Eine obere Schranke wieviele Knoten mit Distanz k erreicht werden.



R_0	R_1	R_2	R_3
{A}	{A, B}	{A, B, C, D}	{A, B, C, D, E}
{B}	{B, A, C, D}	{B, A, C, D, E}	-
{C}	{C, B, D}	{C, B, D, A, E}	-
{D}	{D, B, C, E}	{D, B, C, E, A}	-
{E}	{E, D}	{E, D, C}	{E, D, B, C, A}

Ansatz: Die Menge der Personen die von A mit $k + 1$ Schritten erreichbar sind, ist durch die Vereinigung der Personen, die seine Nachbarn in k Schritten erreichen können, beschränkt.

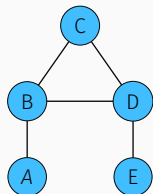


R_0	R_1	R_2	R_3
{A}	{A, B}	{A, B, C, D}	{A, B, C, D, E}
{B}	{B, A, C, D}	{B, A, C, D, E}	-
{C}	{C, B, D}	{C, B, D, A, E}	-
{D}	{D, B, C, E}	{D, B, C, E, A}	-
{E}	{E, D}	{E, D, C}	{E, D, B, C, A}

Annahme: Die Mengen sind unabhängig.

→ Es gilt $|A \cup B| = |A| + |B|$.

→ Nur die Größe muss gespeichert werden.

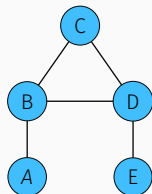


R_0	R_1	R_2	R_3
{A}	{A, B}	{A, B, C, D}	{A, B, C, D, E}
{B}	{B, A, C, D}	{B, A, C, D, E}	-
{C}	{C, B, D}	{C, B, D, A, E}	-
{D}	{D, B, C, E}	{D, B, C, E, A}	-
{E}	{E, D}	{E, D, C}	{E, D, B, C, A}

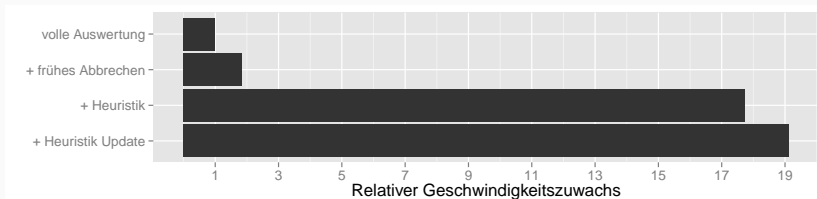
Annahme: Die Mengen sind unabhängig.

→ Es gilt $|A \cup B| = |A| + |B|$.

→ Nur die Größe muss gespeichert werden.



$R_0 \approx R_0 $	$R_1 \approx R_1 $	$R_2 \approx R_2 $
1	2	6 → 5
1	4	15 → 5
1	3	11 → 5
1	4	15 → 5
1	2	6 → 5



- MS-BFS mit einer neuen Sicht auf Graphalgorithmen
- Heuristiken sind für Top-K Anfragen enorm wichtig

Graphanalyzesysteme sind (leider) sehr weit von handgeschriebenen Programmen entfernt!

- Haben wir die passenden Benchmarks?
- Haben wir die passenden Abstraktionen?