Karlsruher Institut für Technologie

Kontrollflussbasierte Leistungsschätzung zur Parallelisierung

Sergej Poimzew
Betreuer: Korbinian Molitorisz

IPD Tichy, Fakultät für Informatik

Gliederung der Arbeit

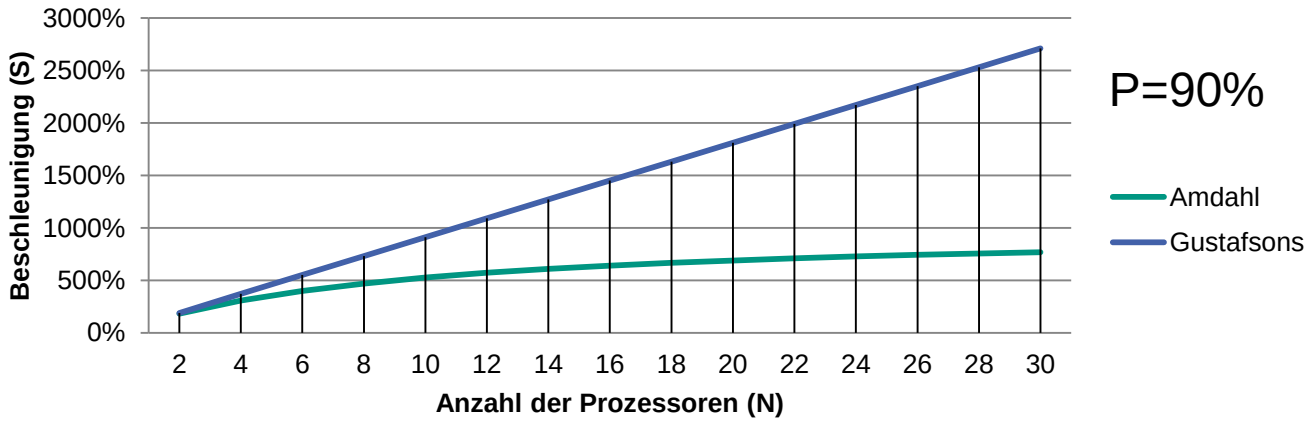
- Einführung
- Verwandte Arbeiten
- Ziele und Anforderungen
- Eigener Ansatz
  - Analysebeispiel
  - Statische und dynamische Analyse
  - Leistungsschätzung
  - Evaluierung
- Zusammenfassung und Ausblick

Einführung

- Heute besitzen fast alle Rechner mehrere Kerne auf einem Chip
- Moderne Software soll parallel arbeiten und alte Systeme sollen umgeschrieben werden
- Software zu parallelisieren ist schwierig und erfordert eine Voruntersuchung: wie und an welchen Stellen soll parallelisiert werden?
- Performanzevaluierung und -schätzung sollen helfen die richtigen Parallelisierungsstellen (Methoden, Schleifen) im Programm zu finden

Verwandte Arbeiten (1)

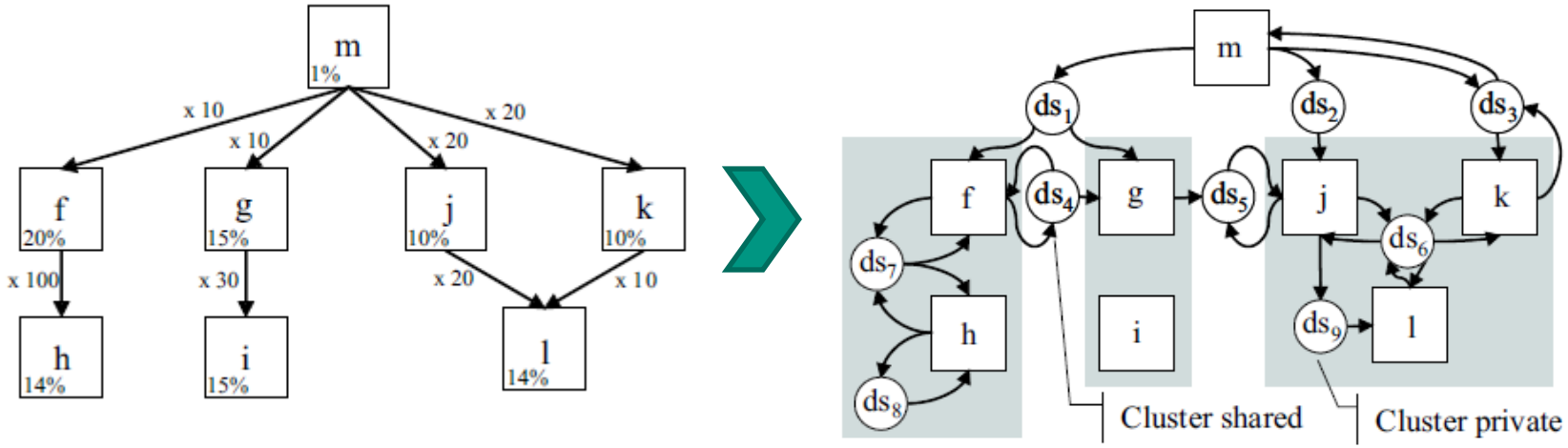
- Amdahlsches Gesetz: $$S = \frac{1}{(1-P) + o(N) + \frac{P}{N}} < \frac{1}{(1-P)}$$
- Gustafsons Gesetz: $$S = (1-P) + P * N$$
- Grundprinzip: Aufteilung des Programms in sequentielle (1-P) und parallele Teile (P)
- Parallele Teil P wird auf N Prozessoren schneller Ausgeführt (Amdahl) oder es wird in die gleiche Zeit mehr Arbeit gemacht (Gustafson)



Verwandte Arbeiten (2)

- “Estimating Parallel Performance, A Skeleton-Based Approach”
  - Entwurfsmusterbasierte Zeitschätzung
  - Drei Entwurfsmuster: Parallel Map, Divide and Conquer, Iteration

- “Function Level Parallelism Driven by Data Dependencies”
  - Parallelitätsdetektion durch Analyse des Kontroll- und Datenflusses
  - Statische und dynamische Analyse auf Methodenebene



Quelle: [RV07]

Ziele und Anforderungen

- Es sollen sequenzielle Programme untersucht und der möglicher Leistungszuwachs durch Parallelisierung abgeschätzt werden.

- **Ziel 1**: Präzise Leistungsschätzung
  - Formelerweiterung von Amdahl und Gustafson
  - Schätzung auf Methodenebene
  - Skalierbarkeit der Schätzergebnisse auf größere Datenmengen

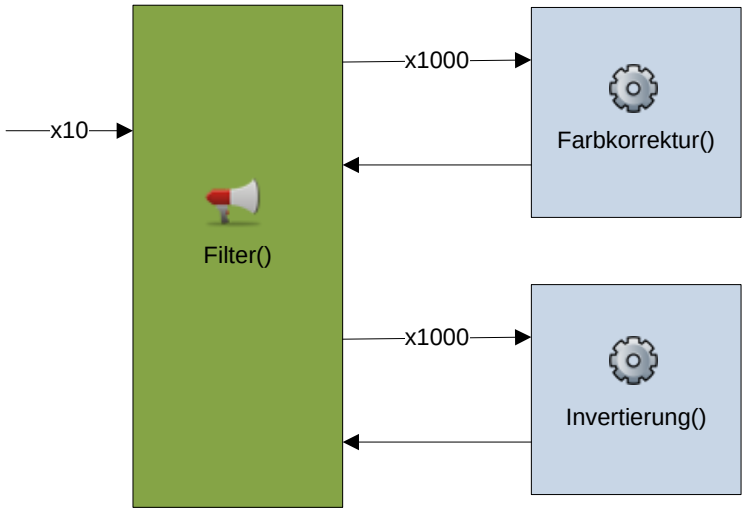
- **Ziel 2:** Eine werkzeugunterstützte Lösung
  - Automatisierte Codeanalyse zur Gewinnung von Laufzeitdaten
  - Anpassbarkeit: Verwendung anderer Analysewerkzeuge
  - Manuelles Ändern der Parameter durch Entwickler möglich

Analysebeispiel

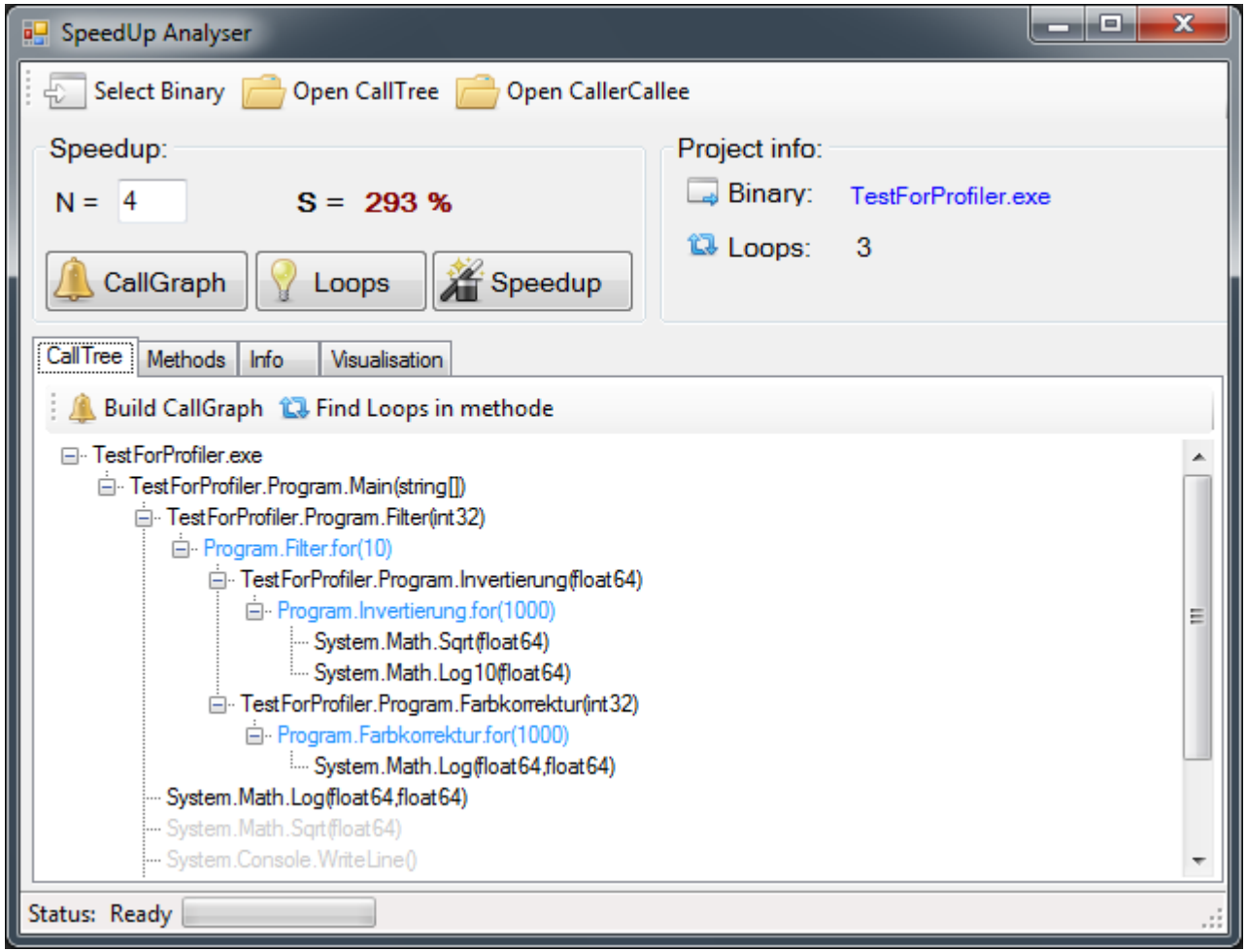
- Beispiel „Bildverarbeitung“
  - Filter-Methode wird auf mehrere Eingabebilder angewendet
  - Ein Filter hat zwei Bildbearbeitungsschritte, die nacheinander angewendet werden
  - Die Bildbearbeitungsschritte können problemlos parallelisiert werden

- **Frage**
  Was soll parallelisiert werden –
  der Filter oder einzelne Schritte?

- **Lösung**
  Eine vorherige Leistungsschätzung
  kann an diese Stelle hilfreich sein!



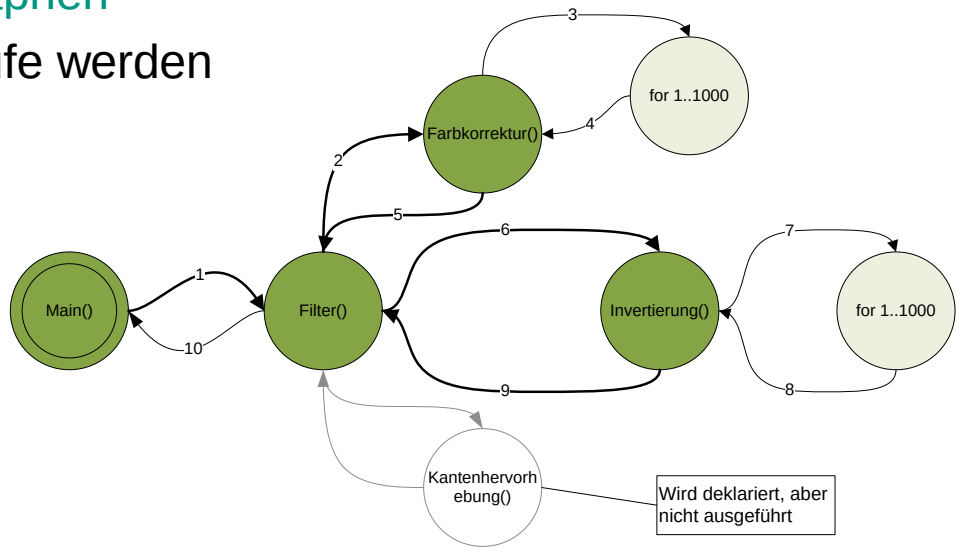
Analysebeispiel (2) - LoopEst



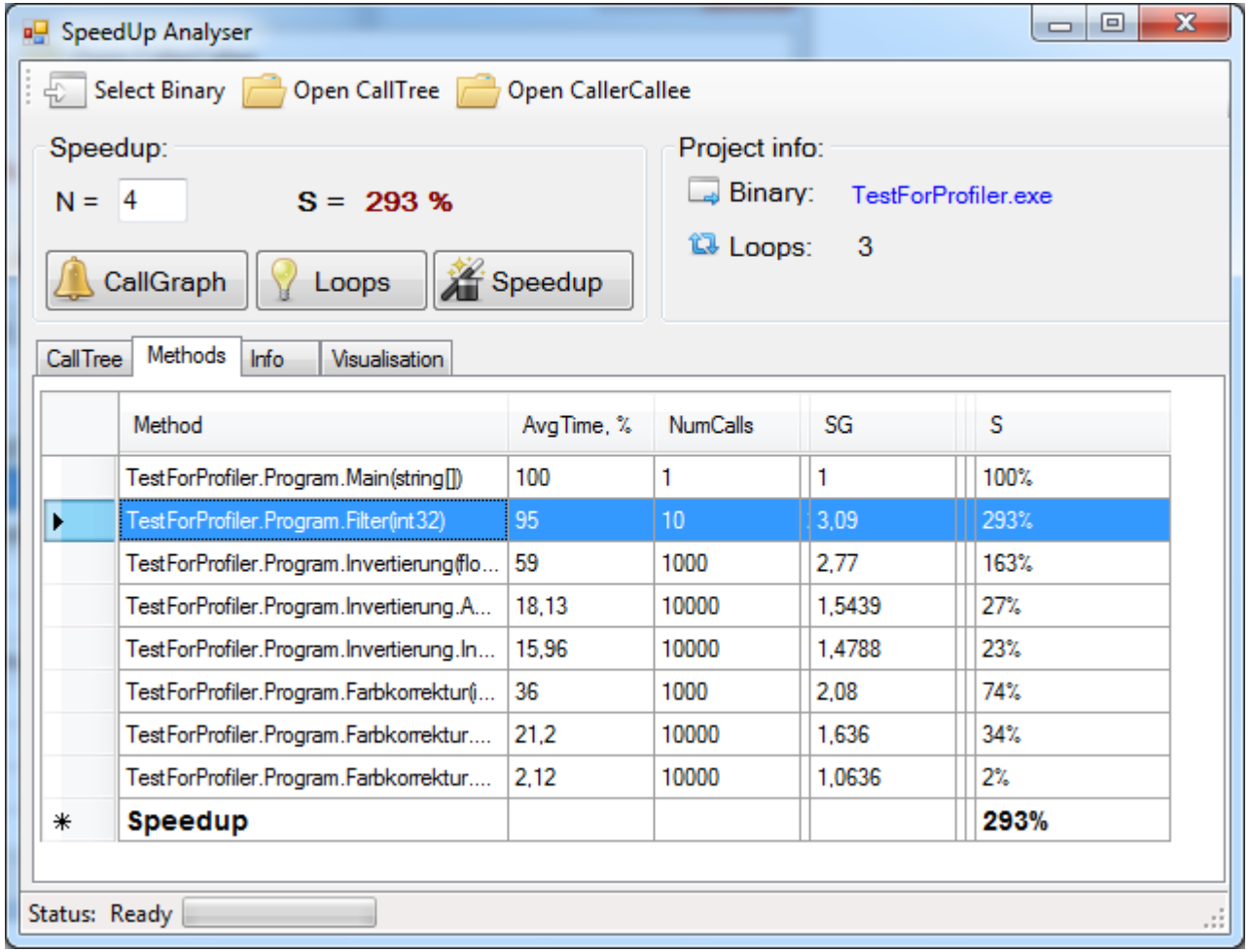
Statische und dynamische Analyse (1)

- Statische Analyse
  - Es wird **Common Compiler Infrastructure (CCI)** verwendet
  - Attributbasiertes Erkennen von Schleifen (eine Instrumentierung ist nötig)

- Dynamische Analyse
  - Es wird **Microsoft Visual Studio Profiler** verwendet
  - Aufbau eines Ausführungsgraphen
  - Laufzeit und Anzahl der Aufrufe werden für allen Methoden bestimmt

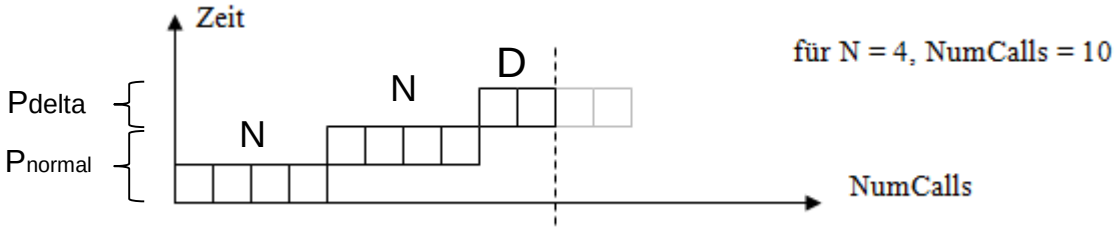


Leistungsschätzung – LoopEst (1)



Leistungsschätzung – LoopEst (2)

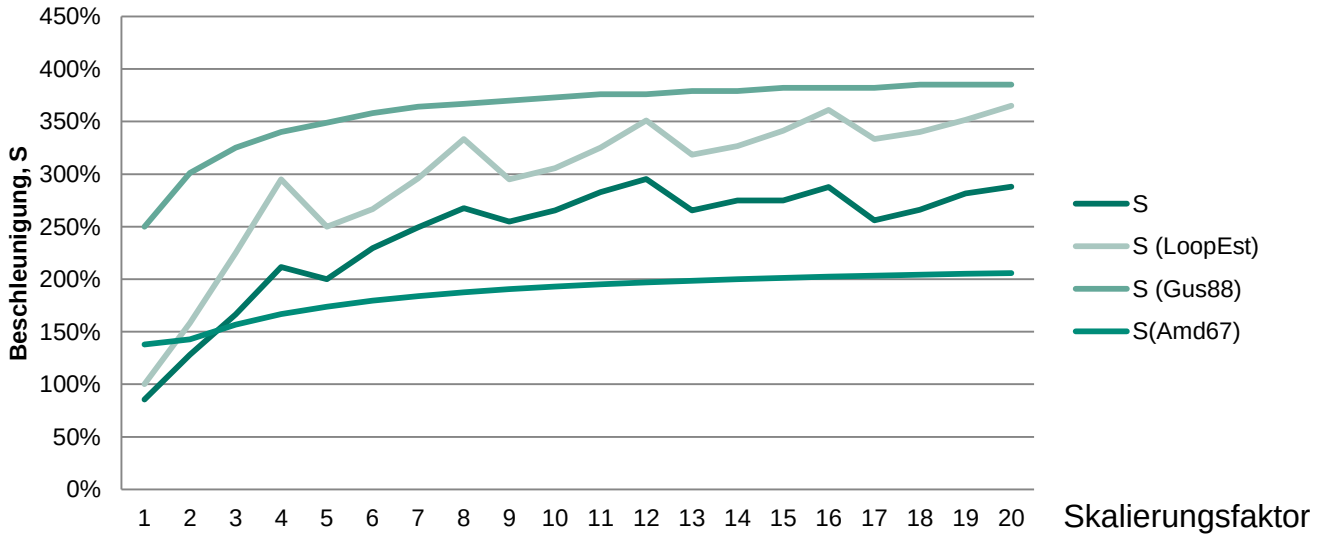
- Jede Methode wird separat abgeschätzt
- Formelerweiterung von Gustafson durch neuen Parameter *NumCalls* (Anzahl der Iterationen)
- Man unterscheidet drei Fälle im Verhältnis *NumCalls* zu *N*:
  - 1. Fall: NumCalls < N: $S = (1 - P) + P * (NumCalls)$
  - 2. Fall: NumCalls % N = 0: $S = (1 - P) + P * N$
  - 3. Fall: NumCalls % N ≠ 0: $S = (1 - P) + (P_{normal} * N) + (P_{delta} * D)$



- Einzelne Beschleunigungsraten werden anschließend gewichtet und zusammenaddiert: $S_{gesamt} = \sum S_m * G_m$

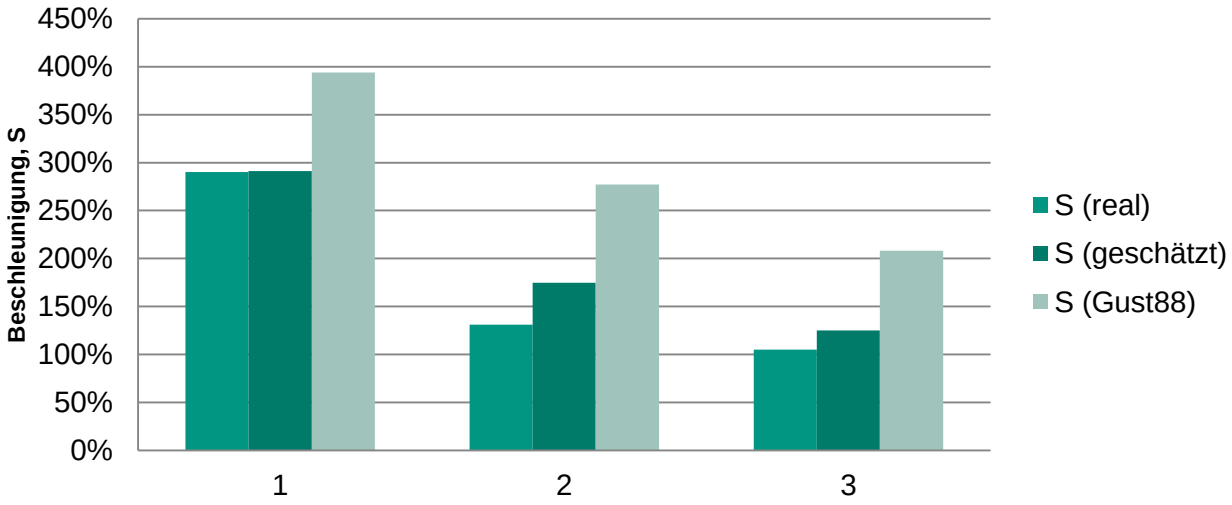
Leistungsschätzung (2) - Skalierung

- **Annahme:** Wenn der sequentielle Teil konstant bleibt, parallele Teil mit der wachsenden Eingabegröße wächst.
- Skalierung von P: $$P_{neu} = \frac{P_{alt} * Skalierungsfaktor}{(1 - P_{alt}) + P_{alt} * Skalierungsfaktor}$$
- Evaluierung (Matrizenmultiplikation): P = 50%



Evaluierung (1)

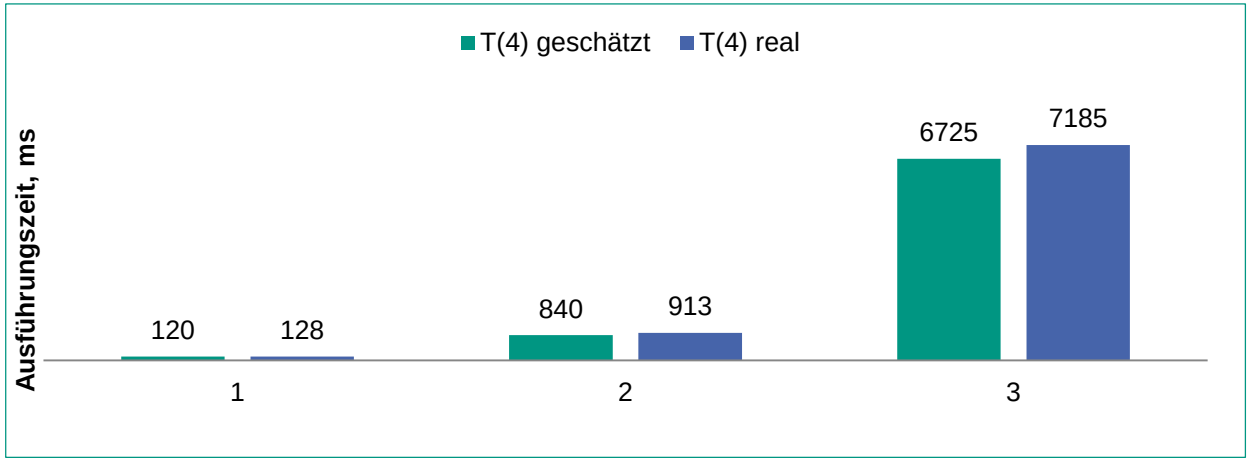
- Evaluierung von „ Bildbearbeitungs“-Beispiel
  - Es wurden drei Parallelisierungsmöglichkeiten betrachtet
    1 – mehrere Bilder wurden parallel verarbeitet (äußere Schleife, P = 95%)
    2 – Farbkorrektur() wurde parallel ausgeführt (P = 59%)
    3 – Invertierung() wurde parallel ausgeführt (P = 36%)
  - Abgeschätzte Laufzeiten wurden mit den tatsächlichen direkt verglichen



Evaluierung (2)

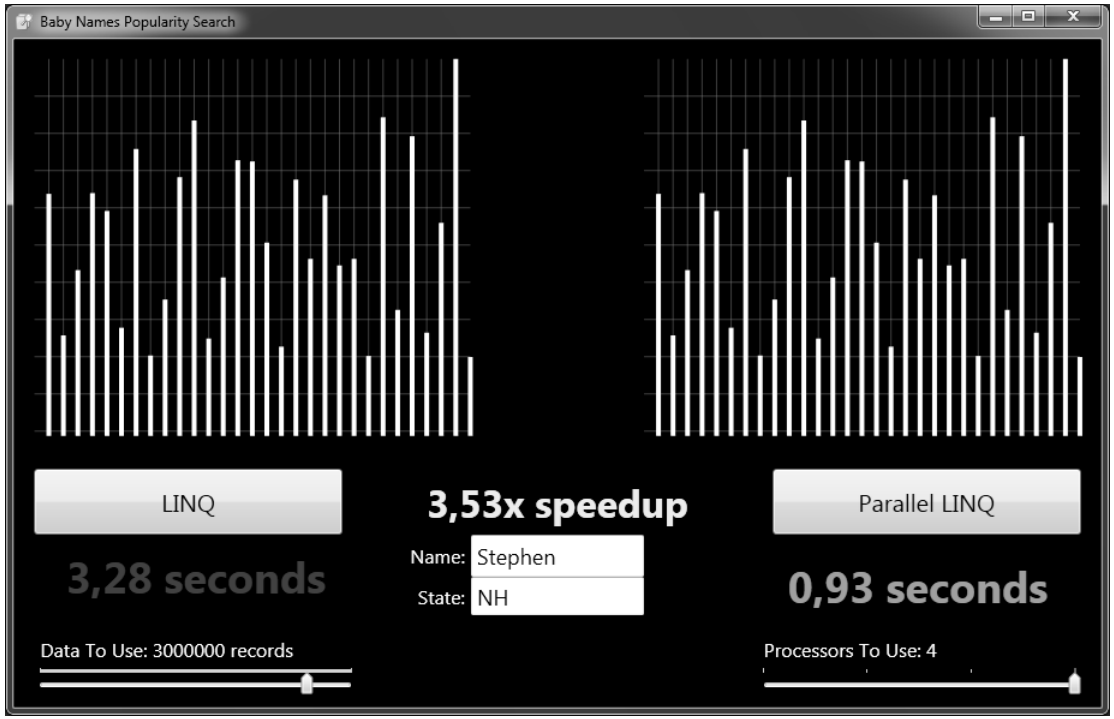
- Matrizenmultiplikation – Schätzung durch Skalierung
  - T_{seq} – sequentielle Zeit $T_{par}(4)$ – parallele Zeit, 4 Kerne

Matrix	T_{seq}	$T_{par}(4)$ geschätzt	$T_{par}(4)$ gemessen	P (geschätzt)
1. 250x250	375 ms	120 ms	128 ms	95%
2. 500x500	3020 ms	840 ms	913 ms	97%
3. 1000x1000	24050 ms	6725 ms	7185 ms	98%



Evaluierung (3)

- **BabyNames**: ein schwieriges Beispiel – viele Methoden, GUI, PLINQ
  - Abgeschätzte Beschleunigung = 315%
  - Tatsächliche Beschleunigung = 353%



Zusammenfassung und Ausblick

- Es wurden verschiedene Ansätze verwendet und kombiniert:
  - Erweiterung der Formel von Gustafson durch einen neuen Parameter
  - Anwendung auf Methodenebene und nicht auf das gesamte Programm
  - Es wird der Kontrollfluss allgemein betrachtet und keine Entwurfsmuster
  - Statische und dynamische Analyse wurden kombiniert

- Weiterentwicklungsbedarf:
  - Automatische Erkennung von Schleifen und Rekursionen
  - Berücksichtigung der Datenabhängigkeiten

- LoopEst sagt wie groß der mögliche Leistungszuwachs durch Parallelisierung sein kann und wo man am besten parallelisieren soll.

Vielen Dank!

Fragen?

Reevaluating Amdahl's Law and Gustafson's Law [Shi96]

- Gustafson revealed that it was indeed possible to achieve more than 1000 fold speedup using 1024 processors [4]. This appeared to have "broken" the Amdahl's Law and to have justified massively parallel processing.