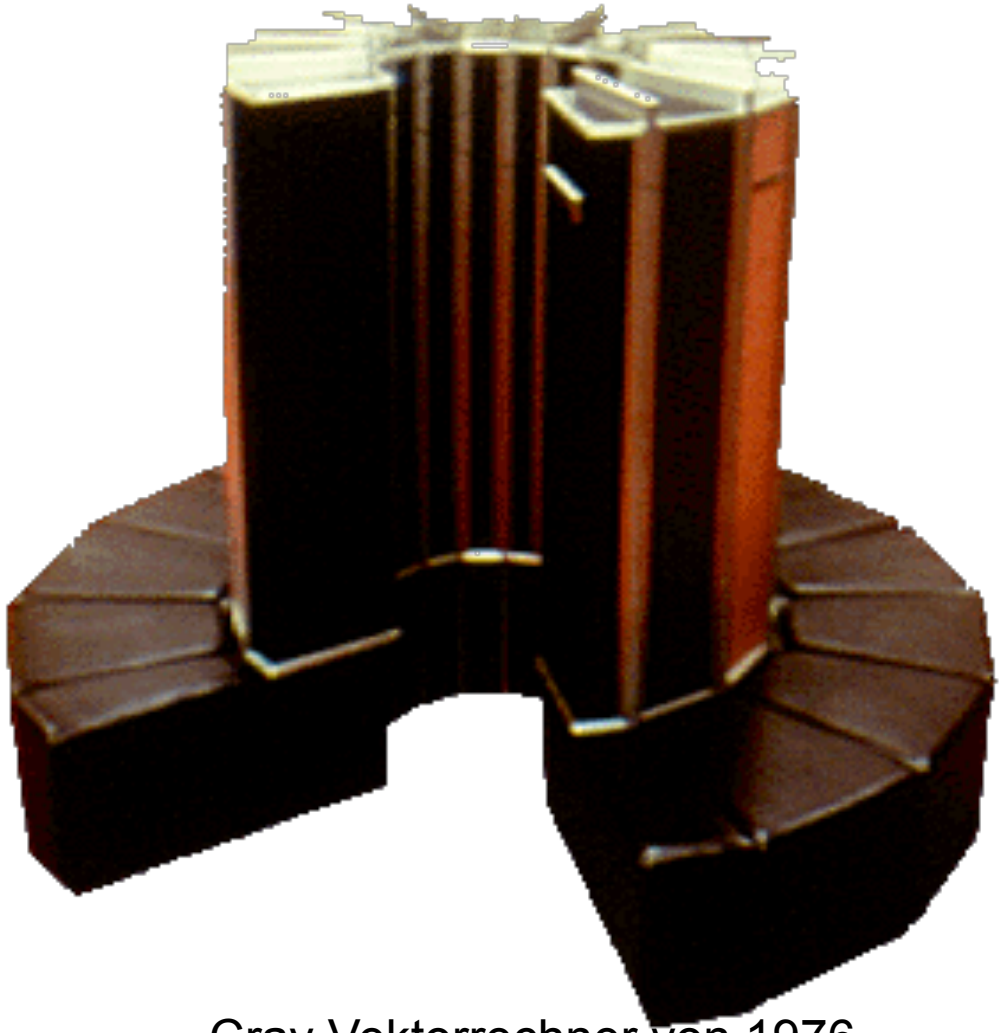
Die Multicore-Transformation

Walter F. Tichy

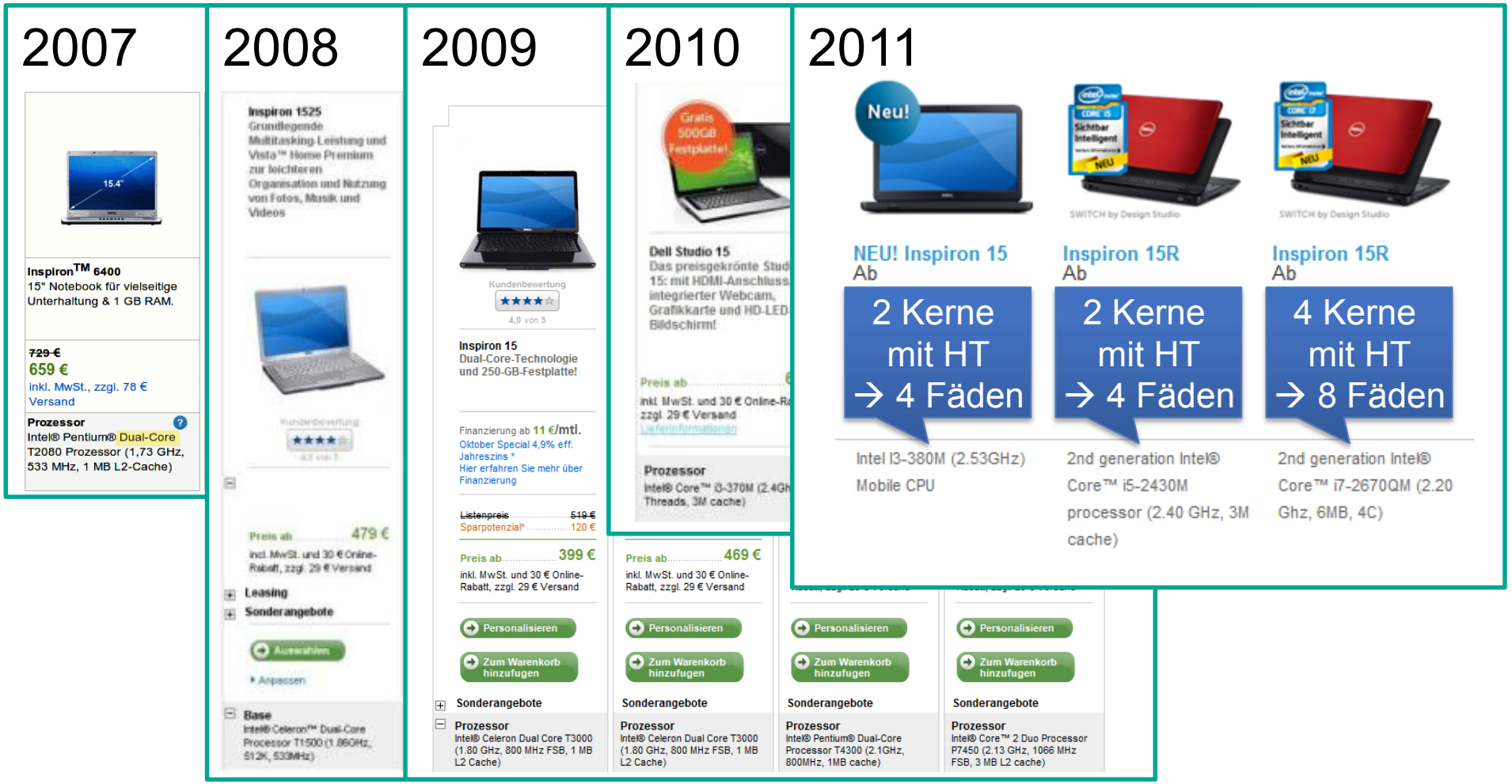
IPD Tichy, Fakultät für Informatik

Parallelrechner der ersten Generation



Cray Vektorrechner von 1976

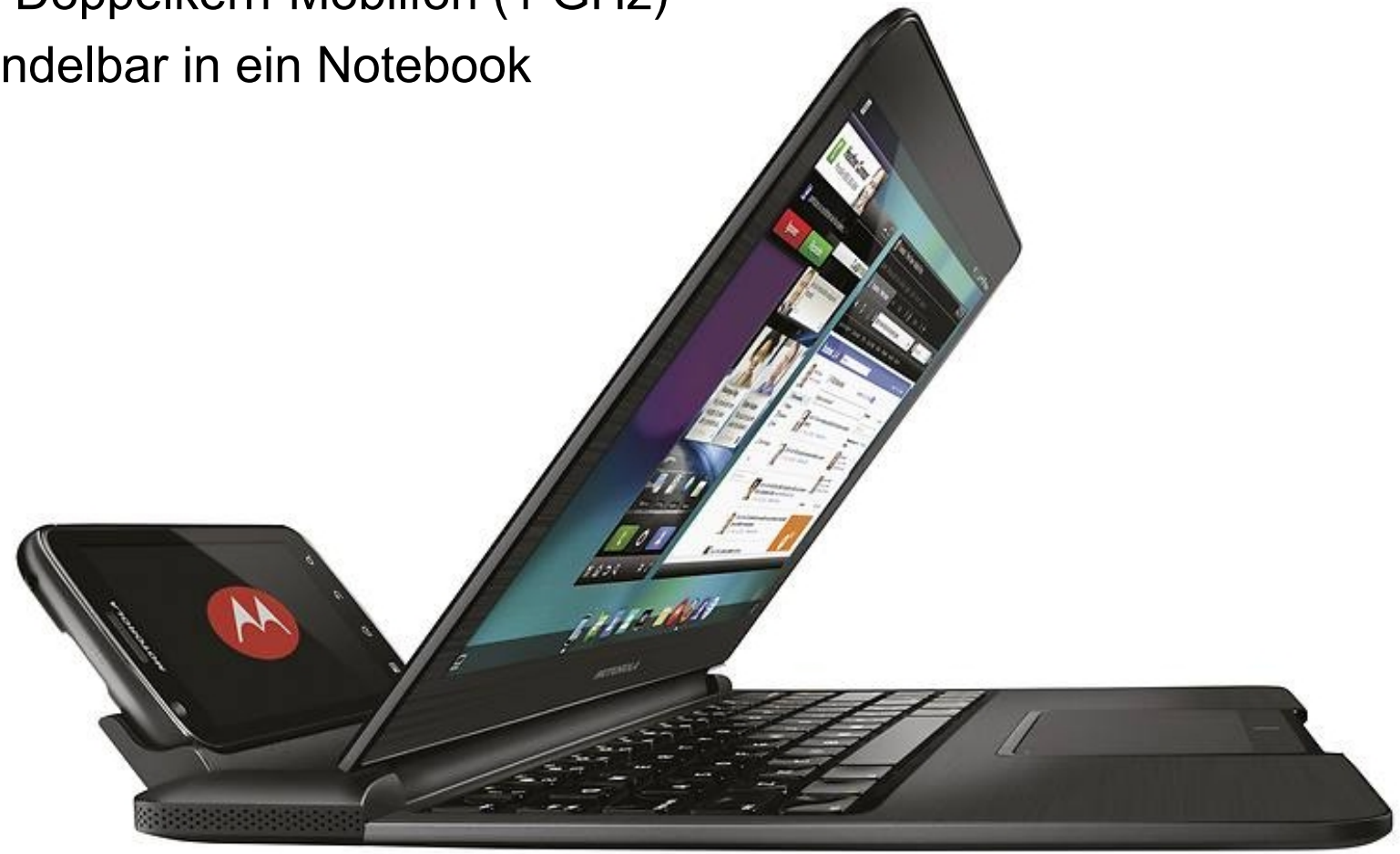
Ihr Laptop – ein Parallelrechner !?



Quelle: Dell

Mobiltelefone mit Mehrkernprozessoren

- **Motorola Atrix (Q1 2011)**
  - Erstes Doppelkern-Mobilfon (1 GHz)
  - Umwandelbar in ein Notebook



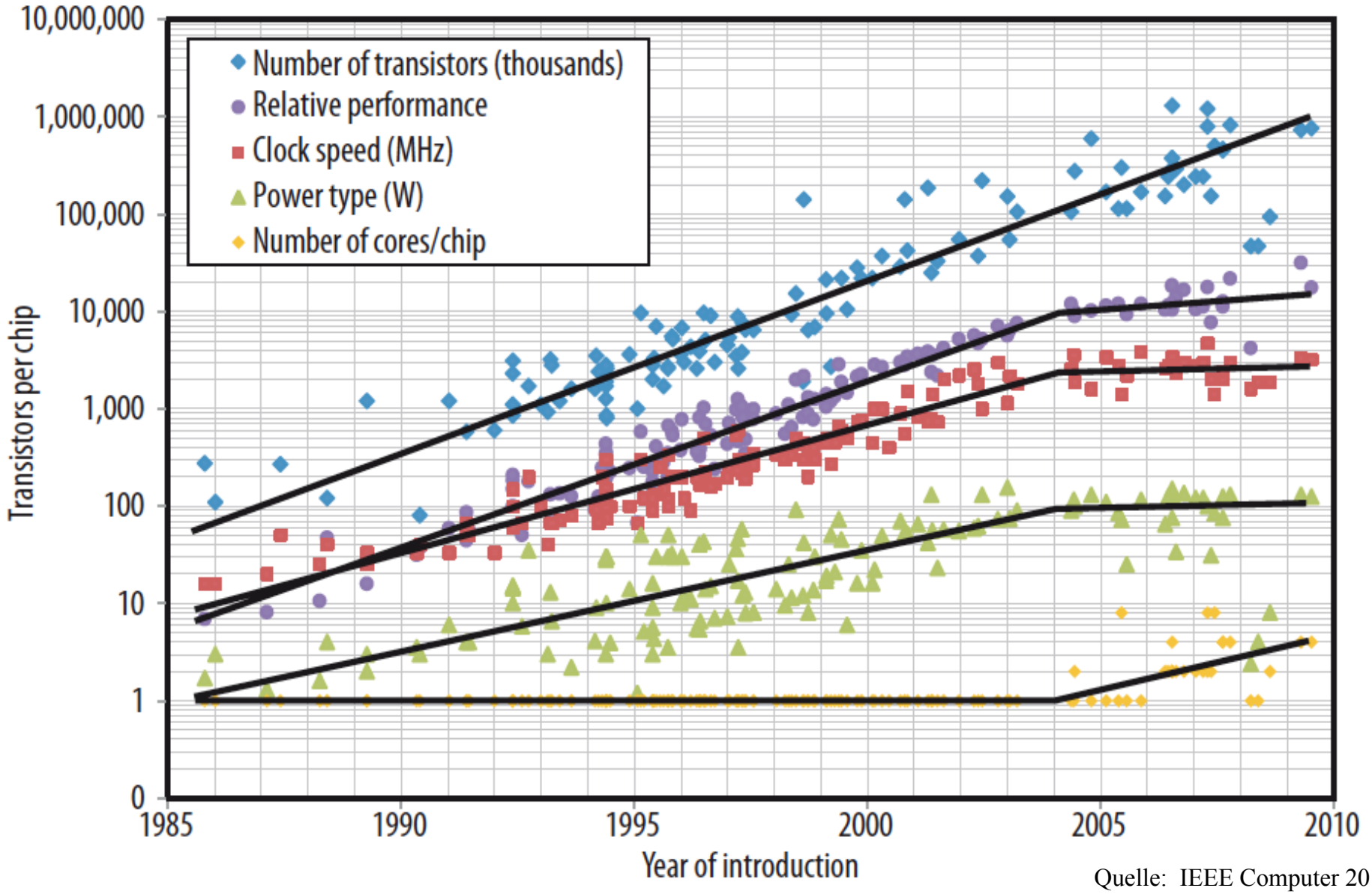
Quelle: http://www.luxuryissues.com/

Samsung, Apple u.a. folgten

Samsung Galaxy S2

Intel SCC: ein Chip mit 48 Kernen

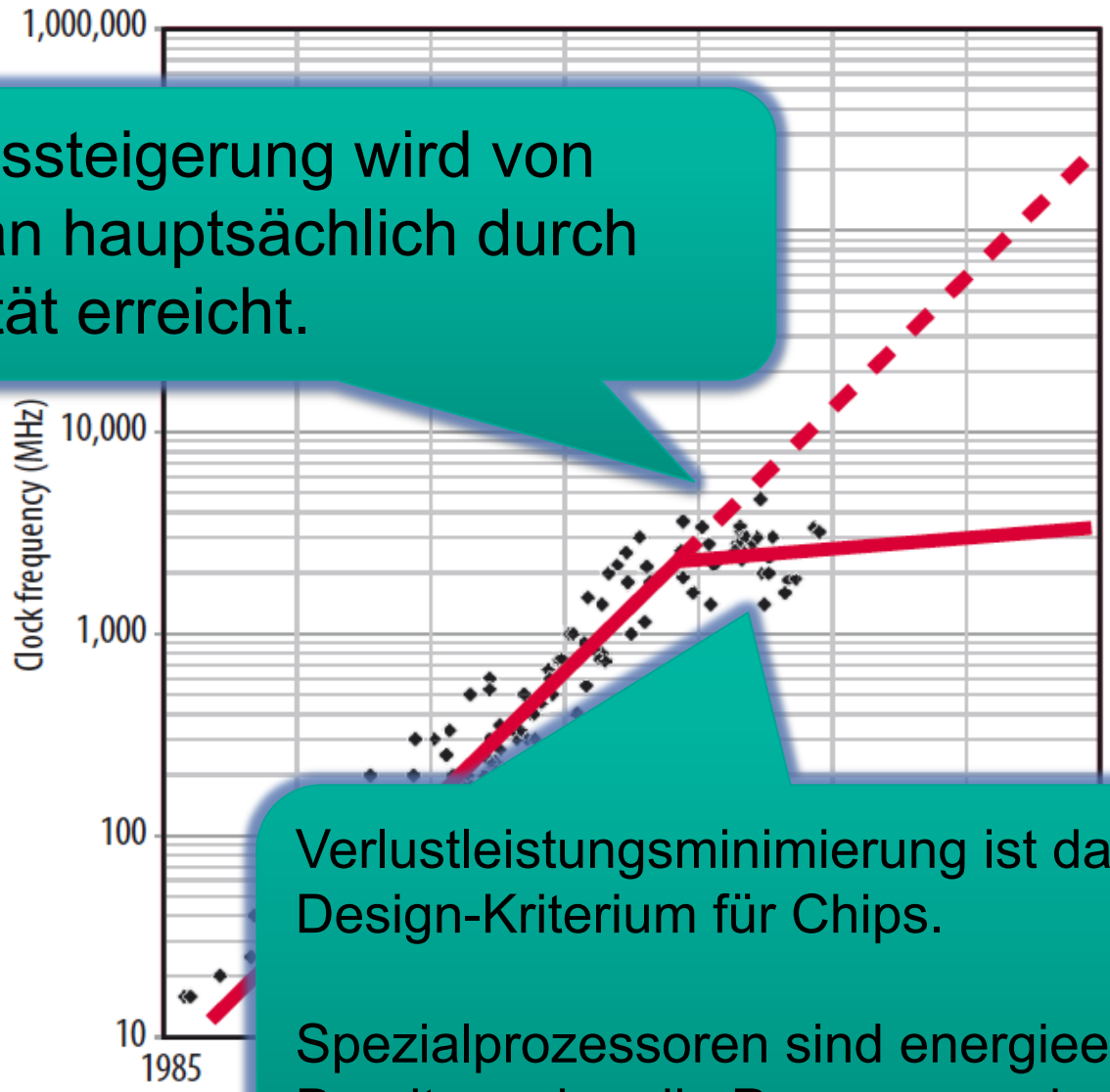
48



Quelle: IEEE Computer 2011

Figure 1. Transistors, frequency, power, performance, and processor cores over time. The original Moore's law projection of increasing transistors per chip remains unabated even as performance has stalled.

Leistungssteigerung wird von 2004/5 an hauptsächlich durch Parallelität erreicht.



Verlustleistungsminimierung ist das überragende Design-Kriterium für Chips.

Spezialprozessoren sind energieeffizienter. Damit werden die Prozessoren heterogen.

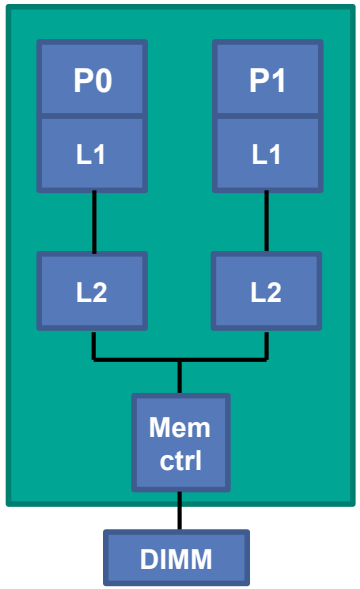
Figure 2. Historical growth in single-processor performance and a forecast of processor performance to 2020, based on the ITRS roadmap. A dashed line represents expectations if single-processor performance had continued its historical trend.

Quelle:
IEEE Computer 2011

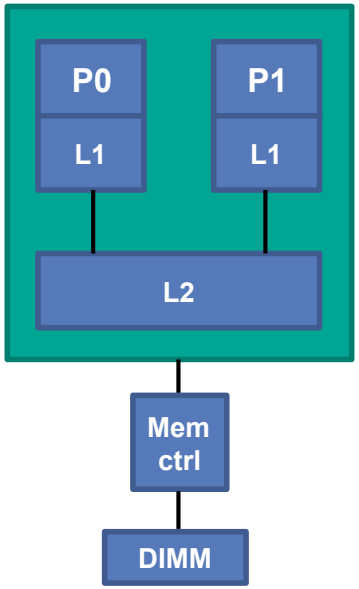
Mehrkern-Architekturen

- Dual-Core und Quad-Core

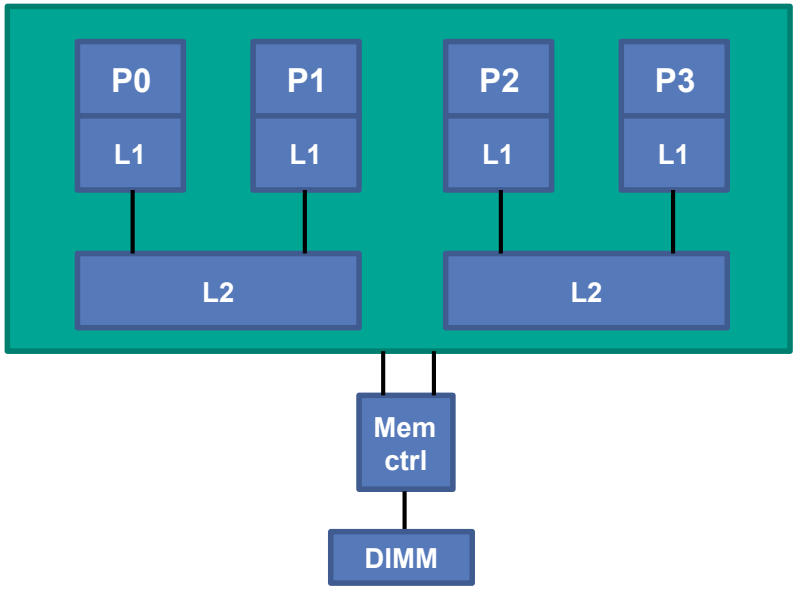
AMD Opteron



Intel Xeon



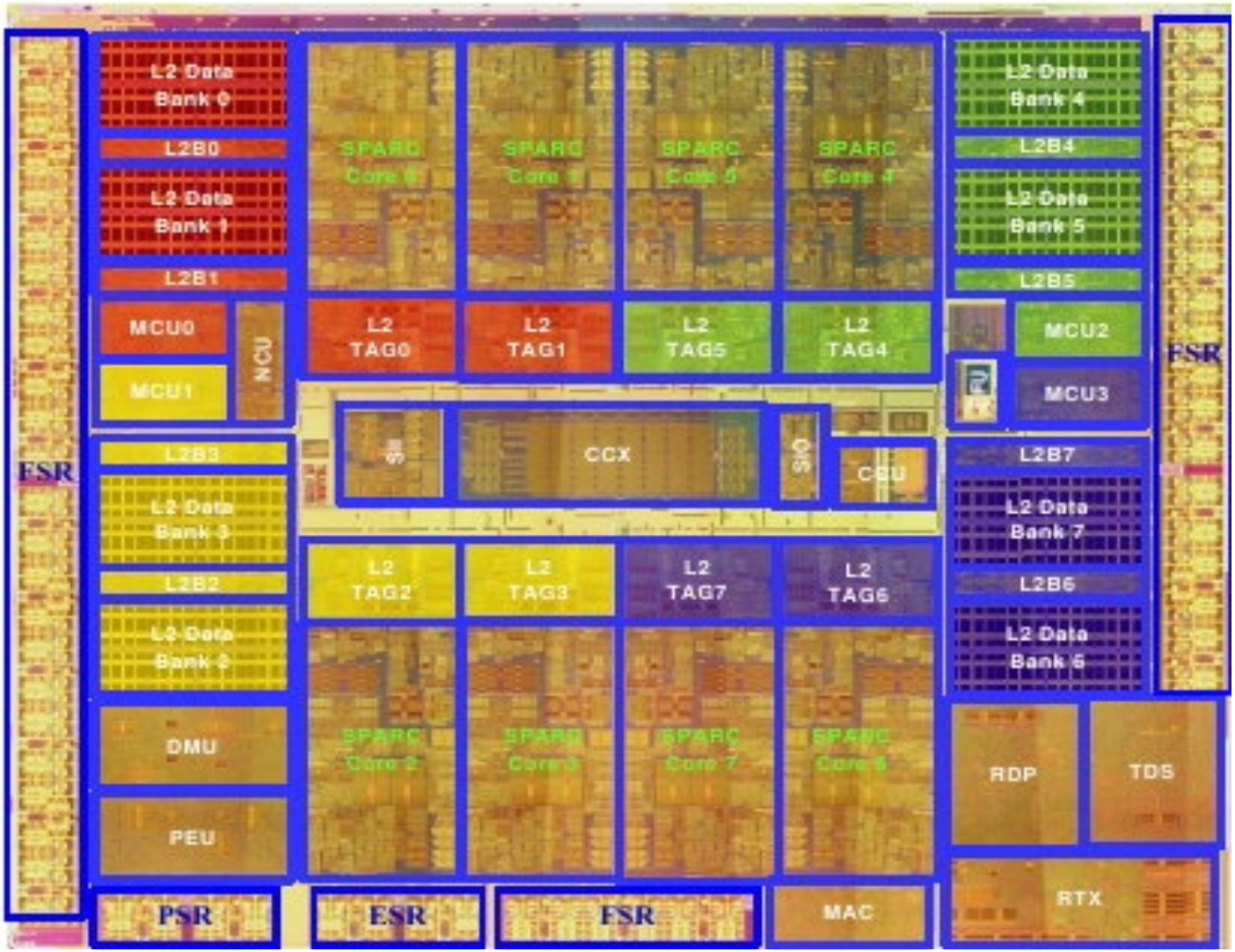
Intel Quad-Core



- Getrennte L1/L2-Caches, was zu divergenten Kopien der gleichen Variablen führt.
- Cache-Kohärenz nur bei bestimmten Ereignissen, z.B. beim Schreiben in den Speicher oder durch Speicherbarrieren.

N. Aggarwal et al., Isolation in Commodity Multicore Processors, IEEE Computer, 2007, 40, 49-59

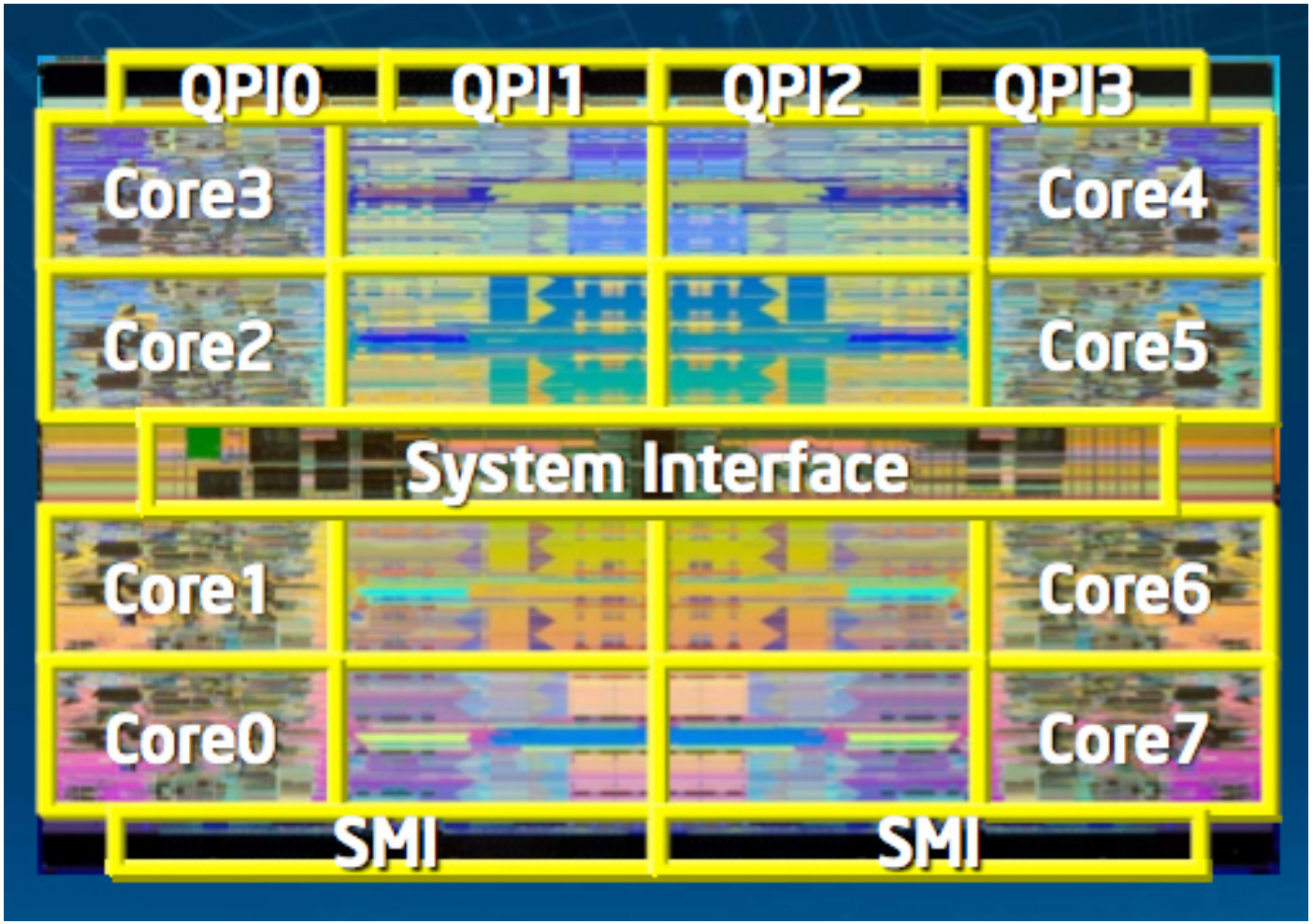
Sun Niagara 2:
2007: 8 Prozessoren auf 3,42 cm2



- 8 Sparc Prozessoren
- 8 HW-Fäden pro Prozessor
- 8x9 Kreuzschiene
- 1,4 GHz
- 75 W
- 65nm Technik
- Lieferbar 2007
- Erste Version 2005

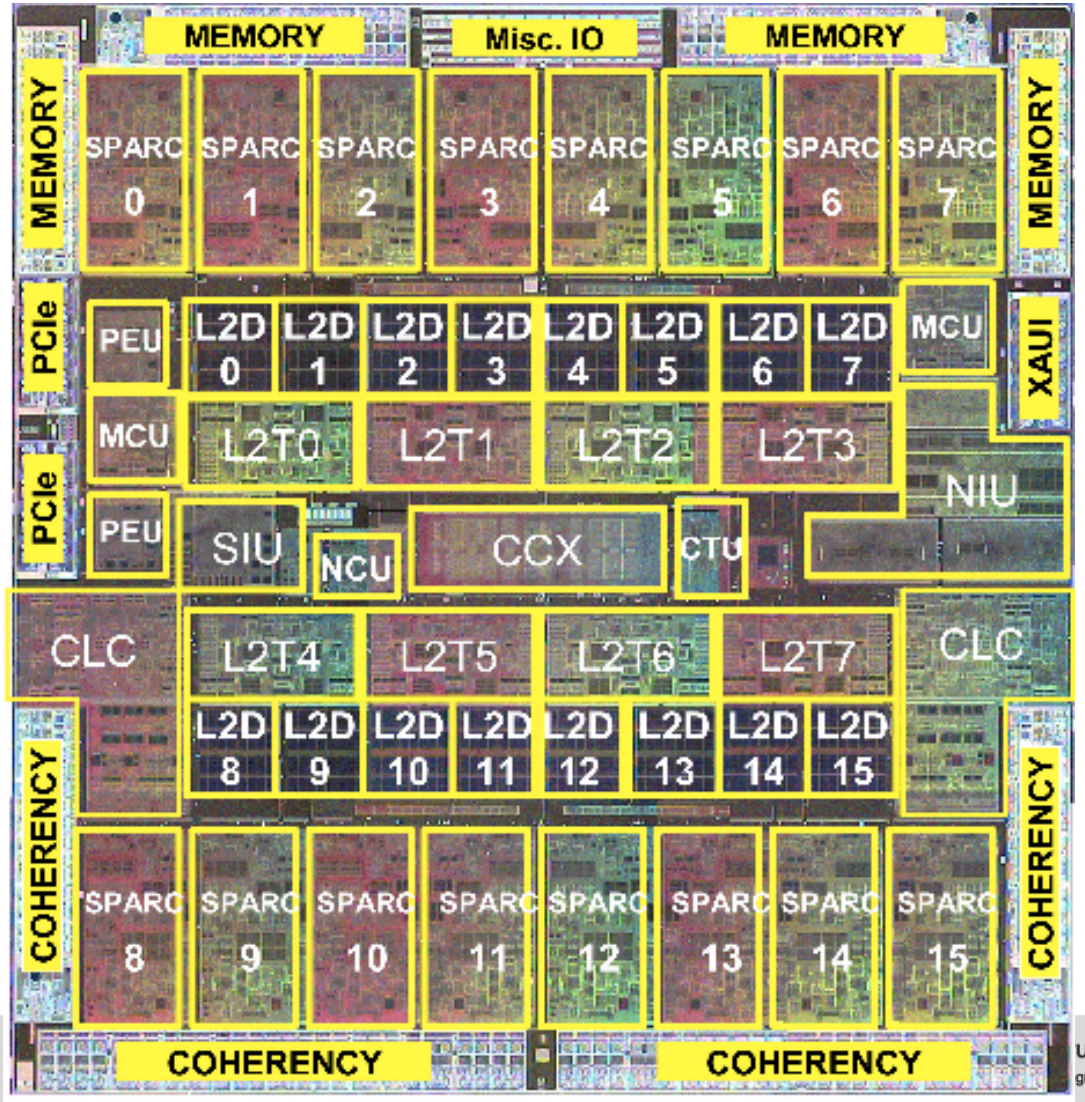
Mehrkerner übernehmen

- 2010: Intel Nehalem EX mit 8 Kernen, 16 Fäden

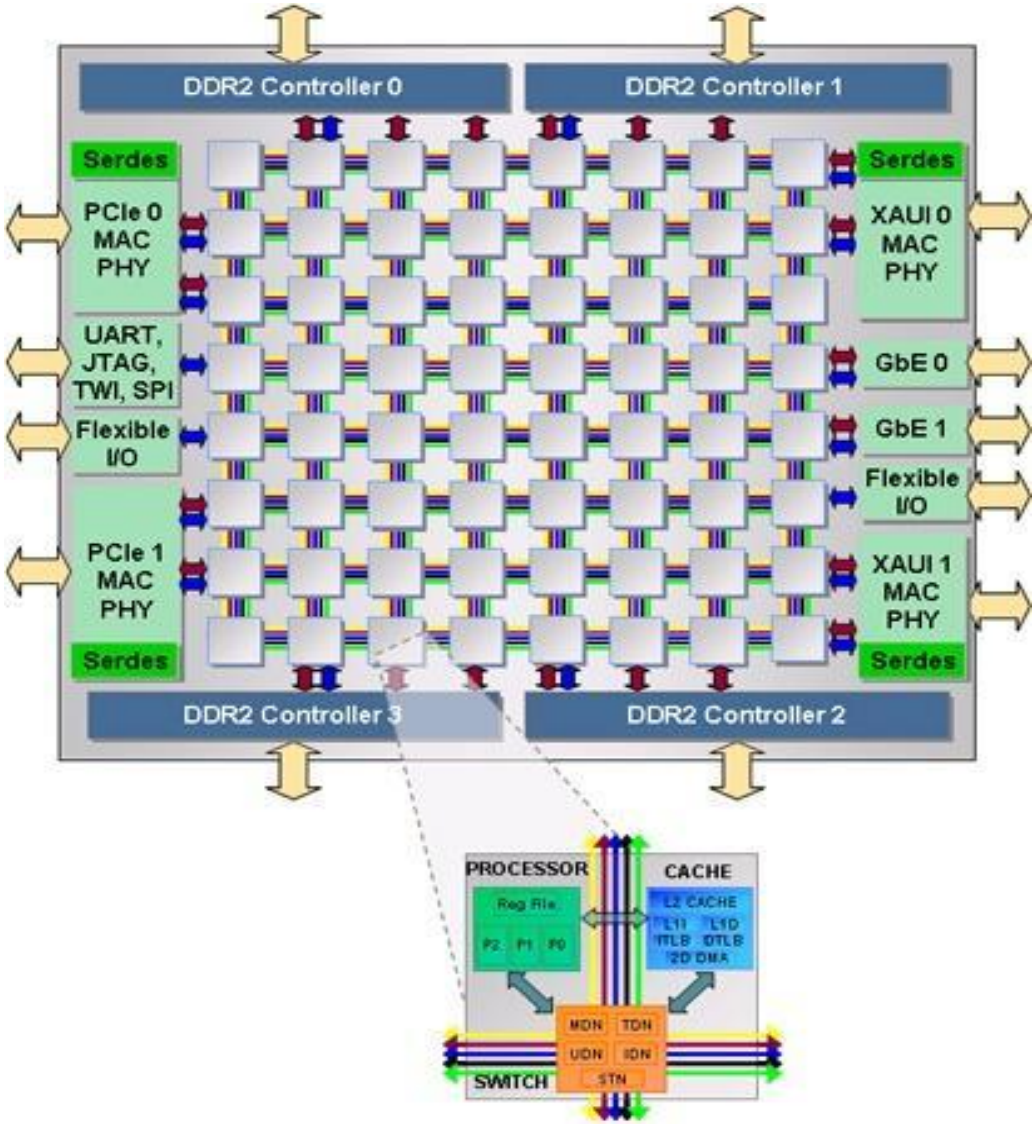


Mehrkerner übernehmen

- 2010: Oracle/SUN T3
- 16 Prozessoren,
- 64 Fäden,
- 1,67 GHz,
- 10^9 Transistoren,
- Bis zu 4 Chips zusammenschaltbar



Tilera TILE64



64 VLIW Prozessoren plus Gitter auf einem Chip

Für eingebettete Anwendungen (Netz- und Videoanwendungen)

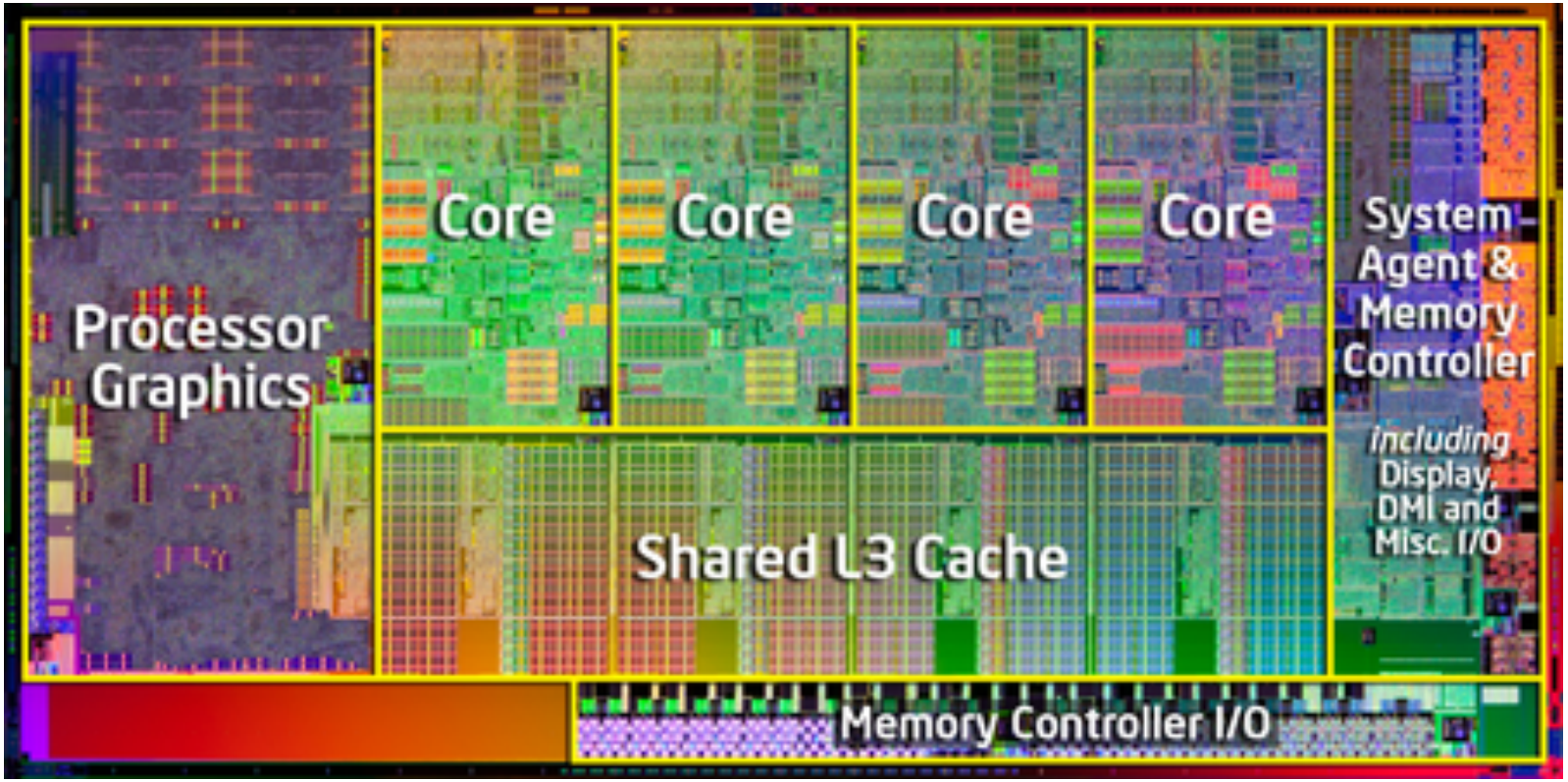
700 MHz

22 W

Lieferbar 2007

2012
: 100 Prozessoren

2011: Intel Sandy Bridge



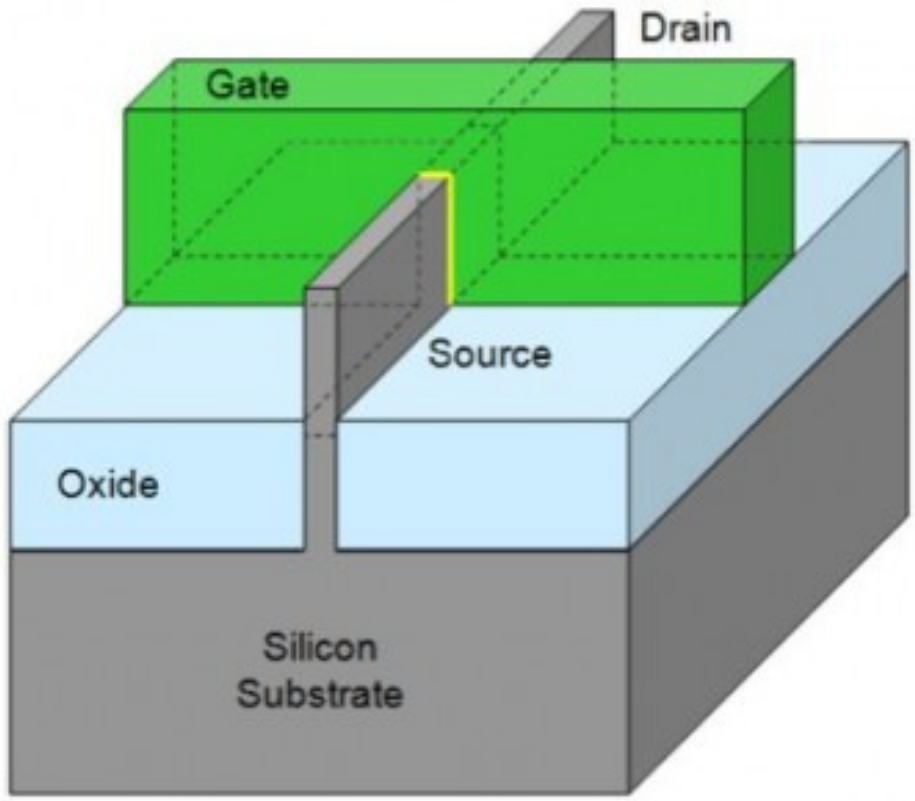
2011: 4 CPUs, 12 graphische Ausführungseinheiten, 32 nm
Ivy Bridge: 4 CPUs, 16 graphische Ausführungseinheiten, 22nm (3D transistor)

2011

Wo rechnen wir?
Heterogenes Chip: drastisch unterschiedliche Befehlssätze.
Dynamischer Wechsel? Lastabhängig?
Wann/warum/wie oft?
Und wie schreiben wir die Software dafür?

Wie schnell rechnen wir?
Einzelne Kerne können langsamer getaktet werden.
Kalte Kerne können kurzfristig 25% übertaktet werden.
Ablaufplanung/Resourcenzuteilung dafür?

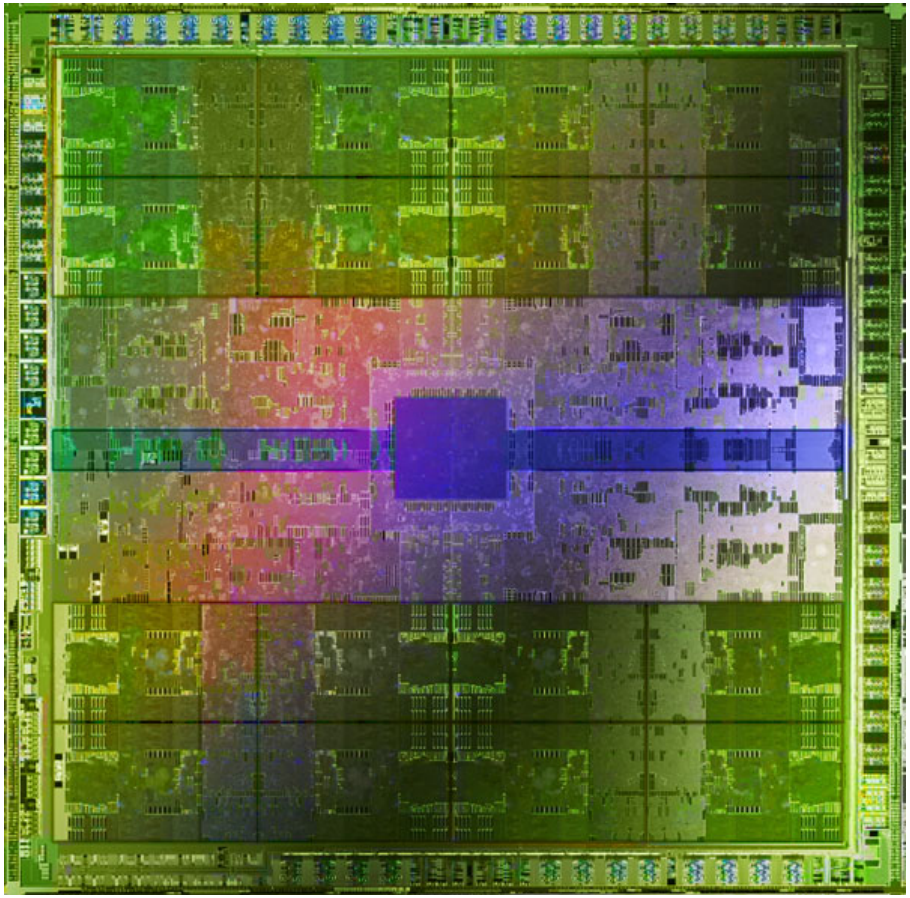
Intels 3D-Transistor



Das Gatter umgibt den Kanal von drei Seiten.
Dadurch verringerte Leckströme, niedrigere Spannung, höhere Frequenzen, kleinere Abmessungen.

Vielkerner in Graphik-Prozessoren

- 2010: Nvidia GeForce GTX 480M (Fermi) mit 352 Kernen



Moore´sche Regel, neue Version

Verdopplung der Anzahl Prozessoren pro Chip mit jeder Chip-Generation, bei etwa gleicher Taktfrequenz

- Parallelrechner werden in naher Zukunft flächendeckend zur Verfügung stehen.

Was sind die Folgen? (1)

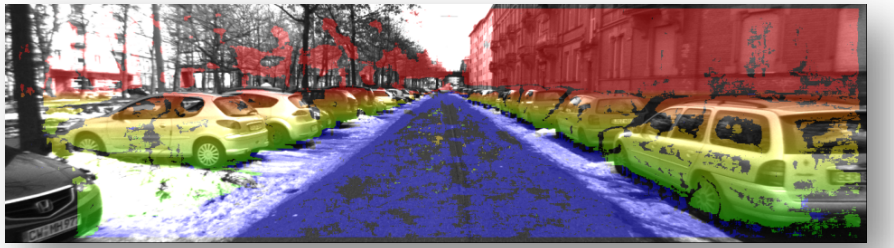
- Hauptproduktlinie der Prozessor-Hersteller sind Multikern-Chips.
- Server werden bereits seit 2005 mit Multikern-Chips ausgeliefert.
- Laptops werden seit 2006 mit Doppelprozessor-Chips ausgestattet (Dell, Apple, u.s.w.)
- Eingebettete Anwendungen (Steuerungsanwendungen, z.B. in Autos) werden auf Mehrkerner konsolidiert werden.

Beispiel: Echtzeit Videoanalyse

- Tiefenschätzung aus Stereokamera (Auflösung 1400x400)

  - Dual Intel Xeon CPU (12 Kerne)
    - ~100 Bilder/s (Verlustleistung190 W)
  - NVIDIA GTX470 GPU
    - ~100 Bilder/s (Verlustleistung 210 watts)
  - Bis jetzt: Speziallösung mit FPGAs

- Weitere Verarbeitung
  - Fahrbahnerkennung
  - Kollisionswarnung
  - …



Was sind die Folgen? (2)

- Leistungssteigerungen von alltäglichen Anwendungen wird durch Parallelisierung erreicht.
- Parallelismus wird zum Normalfall.
- Informatiker müssen Parallelismus beherrschen lernen.

Was sind die Folgen? (3)

- Die Softwarehersteller müssen rasch auf Parallelverarbeitung umstellen, um Wettbewerbsfähigkeit zu erhalten.
  - Umstellung von existierenden Anwendungen
  - Erstellung neuer, paralleler Anwendungen
- An der Informatik des KIT bilden wir seit 2009 ab dem 2. Semester (SWT 1) in Parallelprogrammierung aus.
- Beherrschung der Parallelprogrammierung wird zur Job-Garantie werden.

Herausforderung an die Softwaretechnik

Parallele Software erstellen
zu vergleichbaren Kosten und Qualität wie sequenzielle Software

Programmieransätze & Terminologie

Def.: parallel [griechisch: παράλληλος]
Allgemein: nebeneinander verlaufend, in gleichem Abstand [Brockhaus]

Informatik: gleichzeitig ablaufend

- Anmerkung: In der Informatik-Literatur wird manchmal zwischen
  - „nebenläufig“ (engl. concurrent; im Sinne von „nicht kausal von einander abhängig“) und
  - „parallel“ (engl. parallel; im Sinne von „simultan“) unterschieden.
- Diese Unterscheidung werden wir an keiner Stelle brauchen und verwenden daher beide Begriffe synonym.

Programmieransätze & Terminologie: Überblick

Grundsätzlich zwei wichtige Alternativen:

- Parallelrechner mit gemeinsamem Speicher
  - Prozessoren haben einen gemeinsamen Speicherbereich, den sie gemeinsam benutzen können. Jeder Prozessor kann jede Speicherzelle ansprechen.
  - Heutige Multikernrechnern
- Parallelrechner mit verteiltem Speicher
  - Jeder Prozessor hat seinen eigenen Speicher, der nur ihm zugänglich ist.
  - Zur Kommunikation schicken sich die Prozessoren Nachrichten
  - Wird hier ***nicht*** behandelt.
  - Kann aber für Multicore in Zukunft wichtig werden.
  - Gemeinsamer Speicher und Cache-Kohärenz erfordern heute schon mehr Energie als das Rechnen aller Kerne.

Programmieransätze & Terminologie: gemeinsamer Speicher (1)

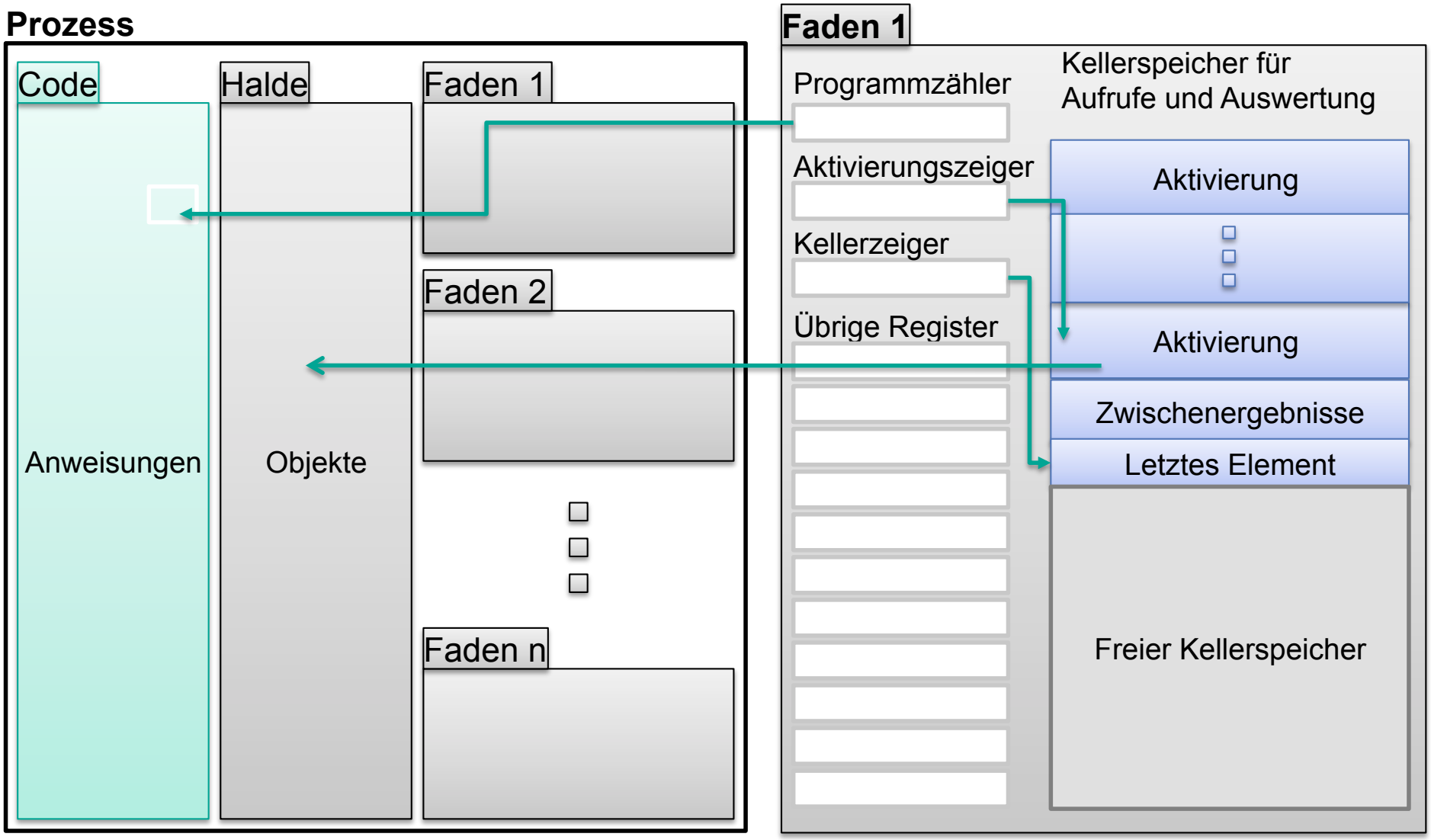
Prozess (engl. Process)

- Enthält Programmressourcen
  - Code-Segment (Instruktionen)
  - Daten-Segment (globale Variablen, Halde)
  - Mind. 1 Kontrollfaden

Kontrollfaden (engl. Thread)

- Instruktionsstrom, der ausgeführt wird
- Jeder Faden hat eigenen
  - Befehlszeiger
  - Keller
  - Register-Kopien
- Teilt sich mit anderen Fäden
  - Adressraum
  - Code/Daten-Segment
  - Andere Ressourcen (z.B. geöffnete Dateien, Sperren, etc.)

Prozesse und Kontrollfäden



Programmieransätze & Terminologie: gemeinsamer Speicher (2)

- Prinzipielles Vorgehen bei gemeinsamen Speicher
  - Prozesse und Fäden werden vom Betriebssystem erzeugt und Kernen zugewiesen
    - In Programmiersprachen eingebaut (z.B. Java Threads, OpenMP) oder über Bibliotheken (z.B. Pthreads, Threading Building Blocks) bereitgestellt.
  - Informationsaustausch über gemeinsam genutzte Variablen
  - Synchronisationskonstrukte koordinieren Ausführung im Falle von Daten- oder Steuerungsabhängigkeiten

Ein erstes Beispiel mit 2 Fäden

```
public class Wettlauf {
        long wert = 0;
        long n;  /* Anzahl Iterationen, von Kommandozeile gesetzt */
```

```
        Thread inkrement = new Thread(new Runnable() {

                public void run() {
                        for (long i = 0; i < n; i++)
                                wert++;
                }
        });
```

```
        Thread dekrement = new Thread(new Runnable() {

                public void run() {
                        for (long i = 0; i < n; i++)
                                wert--;
                }
        });
```

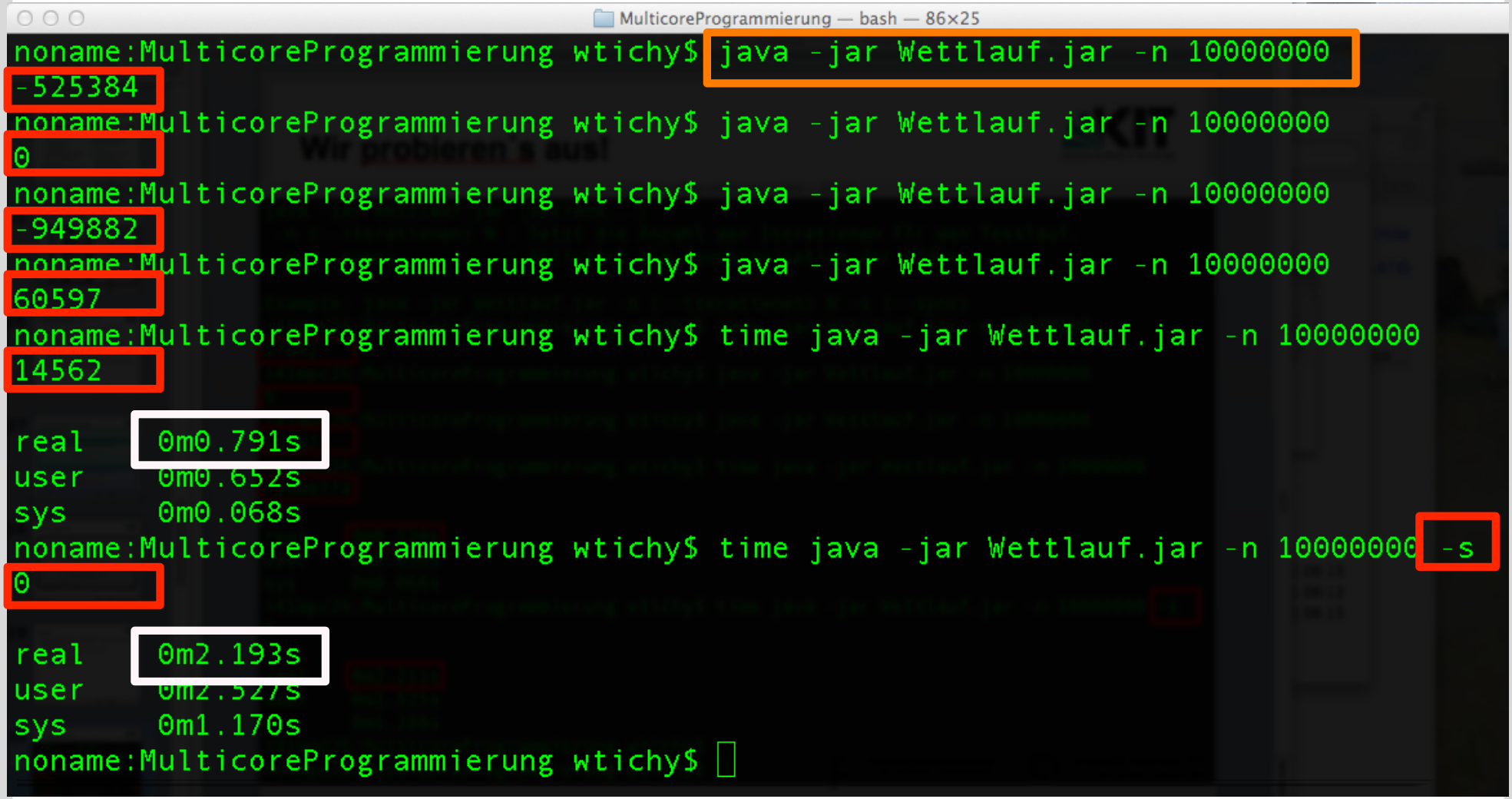
Beispiel fortgesetzt

```
starteTestlauf() {
        inkrement.start();
        dekrement.start();

        /* Auf Ende beider Fäden warten */
        try {
                inkrement.join();
                dekrement.join();

        } catch (InterruptedException e) {
                e.printStackTrace();
        }
        return wert;
}
```

Ergebnis

- Welchen Wert gibt `starteTestlauf` zurück, wenn n mit 10^6 initialisiert ist?
  1. Programm terminiert möglicherweise nicht
  2. 0
  3. entweder $+10^6$ oder -10^6
  4. etwas anderes

Wir probieren´s aus!



Korrektur

```
Thread inkrement = new Thread(new Runnable() {

        public void run() {
                for (long i = 0; i < n; i++)
                        synchronized{ wert++; }
        }
});
```

```
Thread dekrement = new Thread(new Runnable() {

        public void run() {
                for (long i = 0; i < n; i++)
                        synchronized{ wert--; }          }
});
```

Ad-hoc Synchronisation
Was kann hier passieren?

Initial:
o == null
oBereit == false;

Faden 1

```
(1) o = new Object();

(2) oBereit = true;
```

Faden 2

```
(3) while (! oBereit) {/* nichts*/}

(4) o.operation();
```

Übersetzer und Prozessor dürfen unabhängige Anweisungen vertauschen.
Caches könnnen Aktualisierung verzögern.
Es könnte als (2) vor (1) oder (4) vor (3) ausgeführt werden.

Abhilfe in Java: oBereit als volatile deklarieren.
Das verhindert Umordnung um Lese-/Schreibinstruktionen,
die oBereit betreffen.
In anderen Sprachen müssen Speicherbarrieren eingefügt werden.

Mehr zu Speichermodellen:

- Hans J. Boehm, „You Don´t Know Jack About Shared Variables or Memory Models“, CACM 2(55), Feb. 2012, 48-54.
- Manson, Pugh, and Adve, The Java Memory Model, POPL 2005 Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, doi> 10.1145/1040305.1040336
- Das Java Speichermodell, Kap. 17 der Java Sprachspezifikation. http://docs.oracle.com/javase/specs/
- Paul E. Mckenney, Memory Barriers: a Hardware View for Software Hackers, 2009, http://www.rdrop.com/users/paulmck/scalability/paper/whymb.2009.04.05a.pdf

Und was gibt´s sonst noch?

- Datenparallelität
  - gleiche Funktion auf vielen Datenelementen ausführen
- Aufgabenparallelität
  - unabhängige Aufgaben gleichzeitig ausführen
- Fließbandparallelität
  - jeder Filter ein eigener Faden, mit Pufferung dazwischen
- Auftraggeber/-nehmer (Master/worker)
- Map/reduce
- Paralleles Teile-und-Herrsche
- Aufgabenklau (work stealing)
- Futures
- Transaktionaler Speicher

Fallstudie: Berechnung eines invertierten Index

- Suchoperationen arbeiten mit einem invertierten Index: zu jedem Term werden die Dateien aufgeführt, die den Term enthalten.
- Z.B. ist es dann leicht, alle Dateien zu finden, die die Terme „Tichy“ und „Multicore“ enthalten.
- Wir parallelisieren die Erstellung eines invertierten Index für Dateien auf einem Rechner.

Was ist zu parallelisieren?

Drei Phasen:

1. Dateinamen-Erzeugung

   Von einer Wurzel aus den Dateibaum durchforsten

2. Term-Extraktion

   Datei öffnen, Terme auslesen

3. Index-Aufbau

   Paare (<Term>, <Dateiname>) in einen Index einfügen

Welche der drei Phasen ist die langsamste?

Ist die Festplatte der Flaschenhals, so dass eine Parallelisierung kaum lohnt?

Also Messen!

Erste Schritte: Benchmark erstellen, messen

- Zunächst nur ASCII-Dateien
  - Word-Dateien etc. zu aufwendig für den Anfang
  - Da Textdateien schnell zu bearbeiten sind, ergeben sich weniger Möglichkeiten zur Parallelisierung.
- Etwa 50.000 Dateien unterschiedlicher Länge
  - 900 MB reicht für Testläufe
- Sequenziellen Indexgenerator schreiben
- Benchmark vermessen.

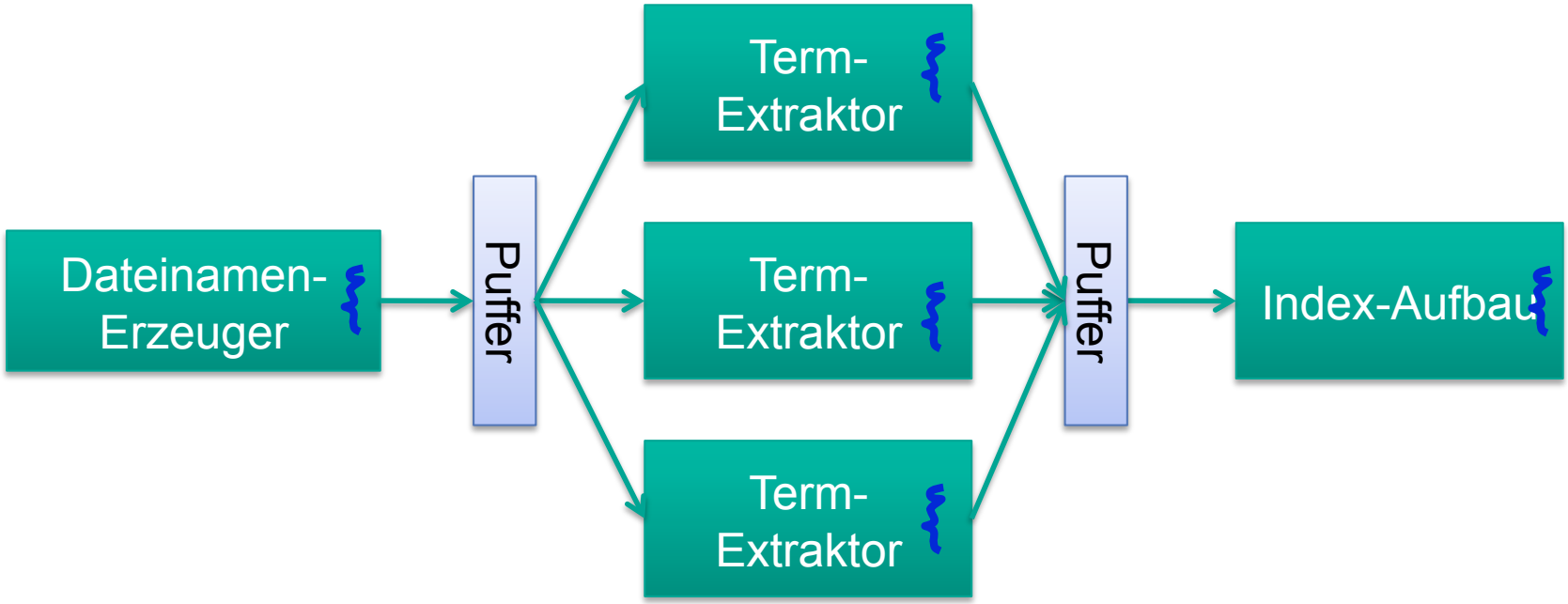
Messergebnisse

	Ausführungszeiten (s) sequenziell			
Plattform	Dateinamen-Erzeugung	Reines Lesen der Dateien	Term-Extraktion	Indexaufbau
(4 Kerne)	5,0	77,0	88,0	22,0
(8 Kerne)	4,0	47,0	61,0	29,0
(32 Kerne)	5,0	73,0	80,0	28,0

Schlussfolgerungen:

1. Dateinamen-Erzeugung unter 5%; nicht parallelisieren!
2. Termextraktion nicht durch Platten-E/A beschränkt, daher: mehrere Termextraktoren gleichzeitig laufen lassen.
3. Auch die Indizes replizieren?
4. Wie erreichen wir Lastbalance?

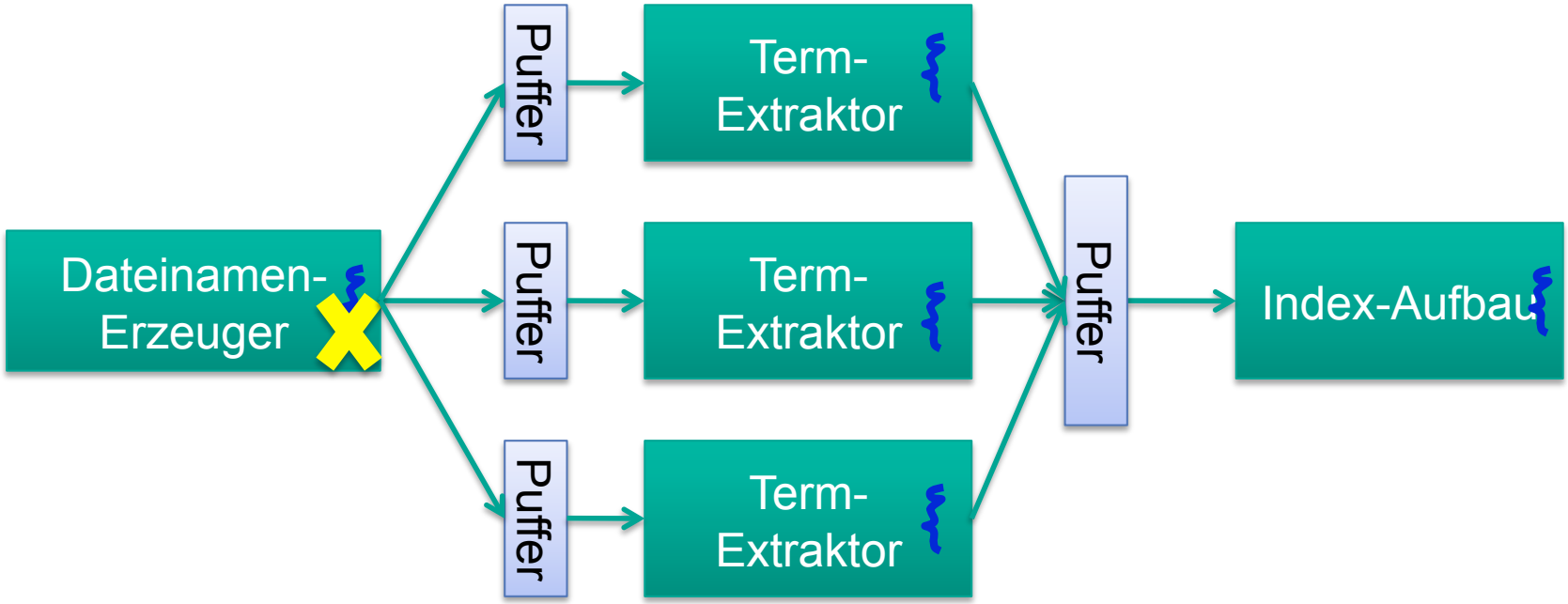
Erster Parallelisierungsversuch



Insgesamt 5 Fäden.

Eine Größenordnung langsamer als sequenzielle Version.
Warum?

Zweiter Versuch



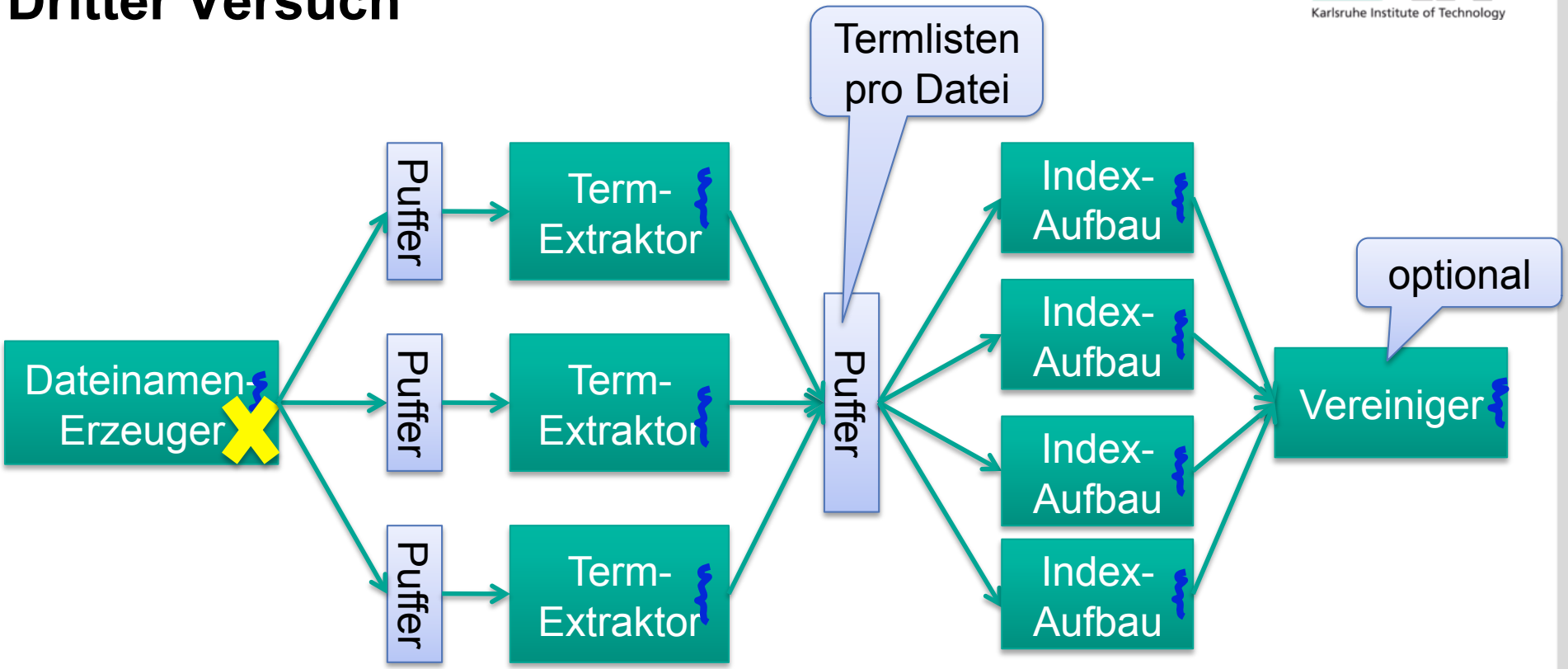
Dateinamen-Erzeuger läuft allein, ohne andere Fäden, und speichert die Ergebnisse komplett in den Puffern der Extraktoren ab (im Umlaufverfahren). Damit dort keine Synchronisation nötig.

Immer noch langsamer als sequenzielle Version.
Warum?

Dritter Versuch

- Jeden einzelnen Term abzulegen ist extrem feingranular: Für Extraktion eines Terms von 10 Buchstaben sind ca. 50 Instruktionen nötig.
- D.h. alle 50 Instruktionen erfolgt ein teurer Monitor-Eintritt und –Austritt mit evtl. Warten, plus ein Signal (notifyAll).
- Duplikate sind weitere Quelle der Ineffizienz. Z.B. kann der Term „Multicore“ in einer Datei mehrmals vorkommen.
  - Einsetzen des Paares („Multicore“, <Dateiname>) heißt, die Liste der Dateinamen, die mit „Multicore“ verbunden sind, linear abzusuchen, ob der Dateiname schon existiert.
- Alternative: Zuerst eine Termliste für eine komplette Datei erzeugen (Duplikate eliminieren), dann die ganze Termliste auf einmal einsetzen. Dann muss auch nicht mehr nach Dateinamen-Duplikaten gesucht werden.

Dritter Versuch



Nur noch Synchronisation am Termlisten-Puffer und eine Barriere vor dem Vereiniger. Sperrengranularität nun OK.

Wieviele Term-Extraktoren und Index-Aufbau-Fäden braucht man?

Ausprobieren! (am besten mit Autotuning)
Plattform-abhängig!

Ergebnisse auf Intel 4-Kerner

	Beste Konfiguration	Ausführungs-Zeit	Beschleunigung
Sequentiell	1	220 s	-
Nur ein Index	3 Extraktoren 1 Index	46,7 s	4,71
Mehrere Indexe plus Vereiniger	3 Extraktoren 5 Indexe 1 Vereiniger	46,9 s	4,70
Ohne Vereiniger	3 Extraktoren 2 Indexe	46,4 s	4,74

4-Kerner: 3 Extraktoren, 2 Indexe am besten
8-Kerner: 6 Extraktoren, 2 Indexe am besten
32-Kerner: 9 Extraktoren, 4 Indexe am besten

Was lernen wir davon?

1. Messen, um die Komponenten mit dem größten Parallelisierungs-Potenzial zu finden!
2. Feingranulare Sperren vermeiden!
3. Achte auf Flaschenhälse (E/A, Puffer, Datenstrukturen)!
4. Berücksichtige unterschiedliche parallele Entwürfe!
5. Benutze Überschlagsrechnungen um Alternativen einzuschätzen!
6. Probiere Alternativen durch, am besten mit einem Autotuner!
7. Durch Misserfolg nicht entmutigen lassen!

Forschungsthema: Werkzeugunterstützung zur Parallelisierung

AutoProfiler

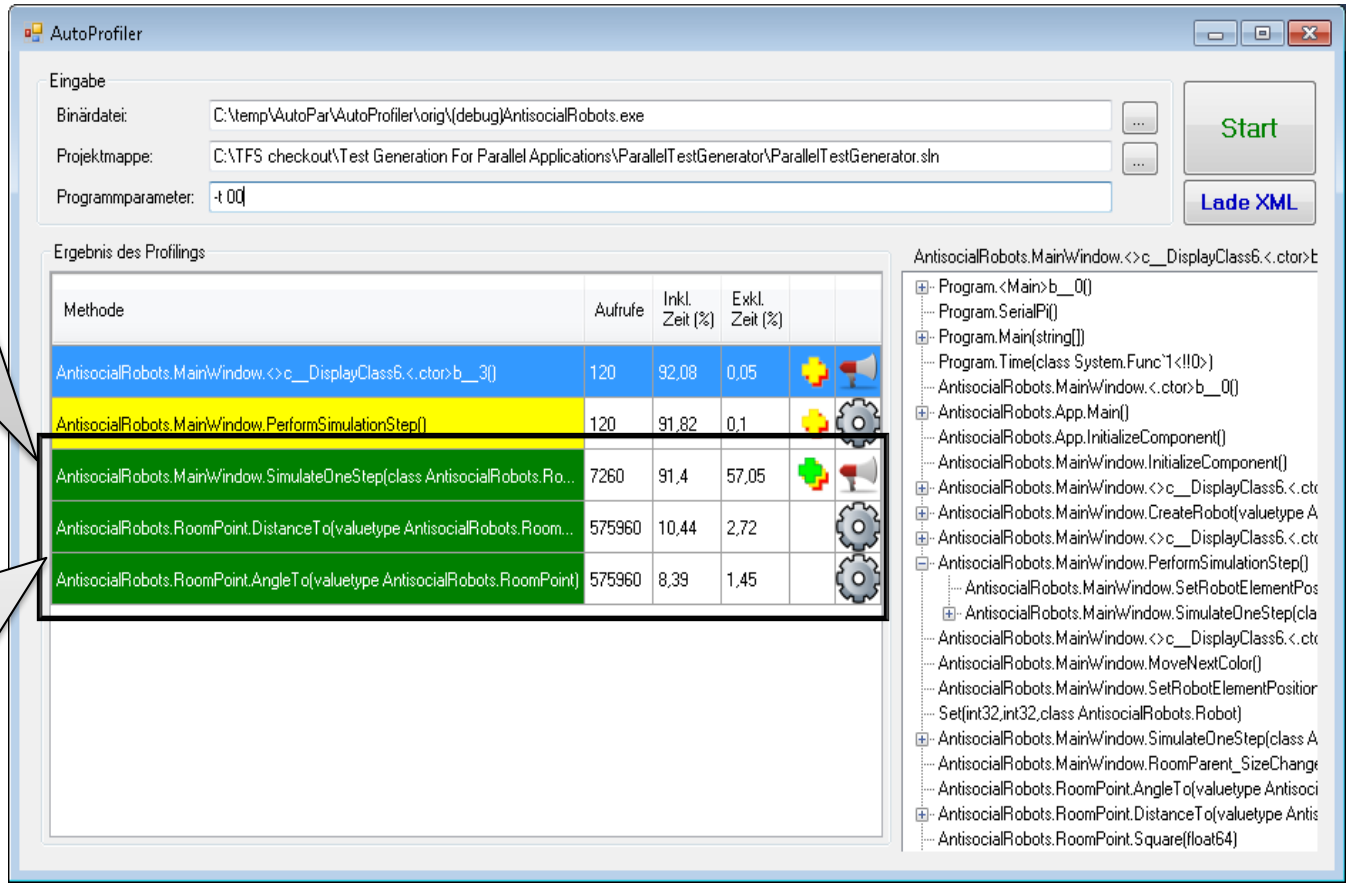
- Parallelisierung durch Erkennung von Entwurfsmustern im sequentiellen Laufzeitprofil

Identifikation einer Master-Methode

- Hoher inklusiver Anteil an Gesamtlaufzeit
- Geringer exklusiver Anteil an Gesamtlaufzeit
- Relativ geringe Aufrufhäufigkeit

Identifikation von zwei Worker-Methoden

- Hoher Anteil an Gesamtlaufzeit
- Relativ hohe Aufrufhäufigkeit



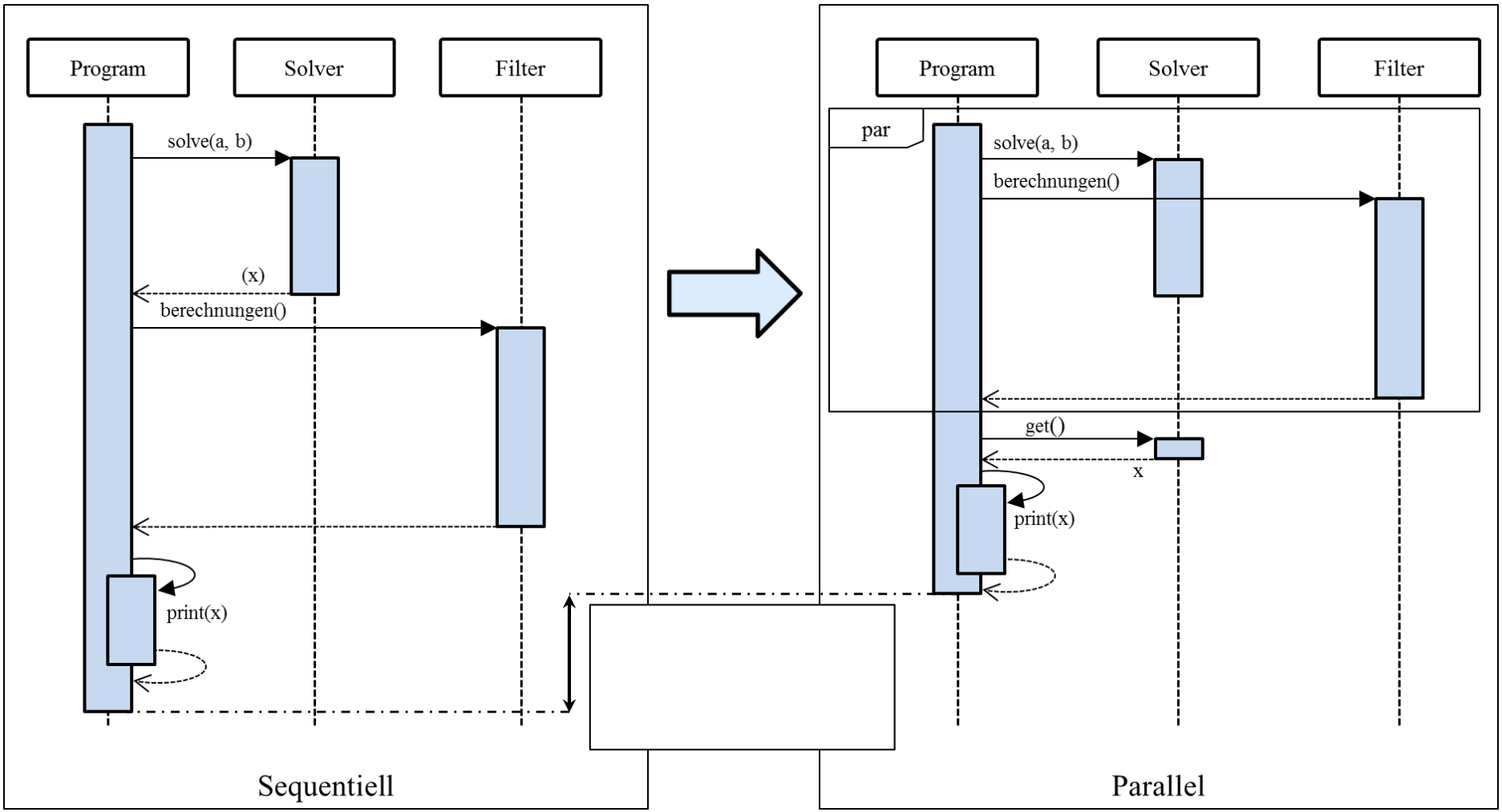
AutoFuture

- Berechnungskonzept Future:
  - Parallele Berechnung eines Ergebnisses, das später gebraucht wird.
  - Beim Zugriff auf das Ergebnis wird dieses entweder geliefert oder der Aufrufer solange blockier

- AutoFuture: automatischer Einsatz von Futures
  - Erkennung asynchron ausführbarere Methoden im Kontrollfluss
  - Aufspalten des Kontrollflusses
  - Zusammenführen an Synchronisationspunkten
  - Heuristik zur Identifikation geeigneter Synchronisationspunkte

AutoFuture

- Laufzeitvorteil durch Parallelberechnung

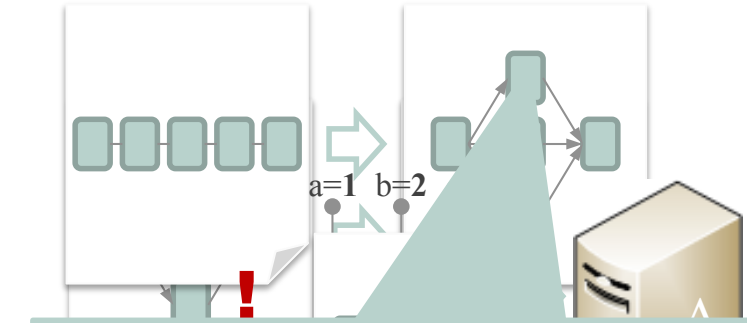


AutoFuture

- Ergebnisse
  - Bisher 5 parallelisierte Anwendungen
  - Beschleunigung von 0,76 – 3,34 auf 4-Kernrechner
- Ergebnisse der Untersuchung der Anwendbarkeit
  - Spekulative Berechnung zu einem früheren Zeitpunkt im Programmablauf über AutoFuture möglich (3 von 30 Fälle)
  - Schleifenparallelisierung durch Verteilen der Iterationen an AutoFutures möglich (10 von 30 Fälle)
  - Parallelausführung von Anweisungsblöcken innerhalb von Methoden durch AutoFutures schwer zu finden (4 von 30 Fälle)

Autotuning

- Parallelisierung ist komplex und fehleranfällig.
- Parallele Programme enthalten eine Reihe von einstellbaren Parametern.
- Manuelle Optimierung ist komplex und zeitaufwändig.
- Jede Plattform erfordert eigene Optimierung.
- **Autotuning**: Lass den Rechner selber optimieren!

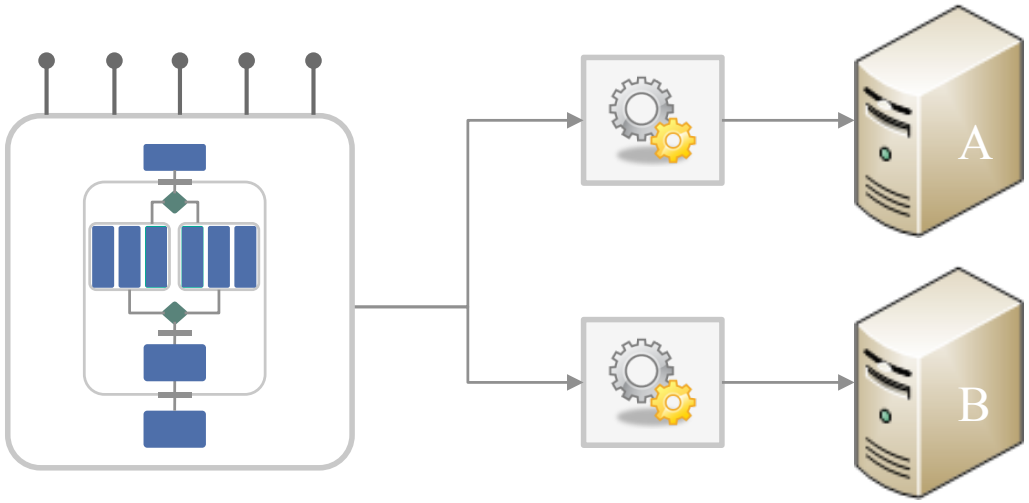


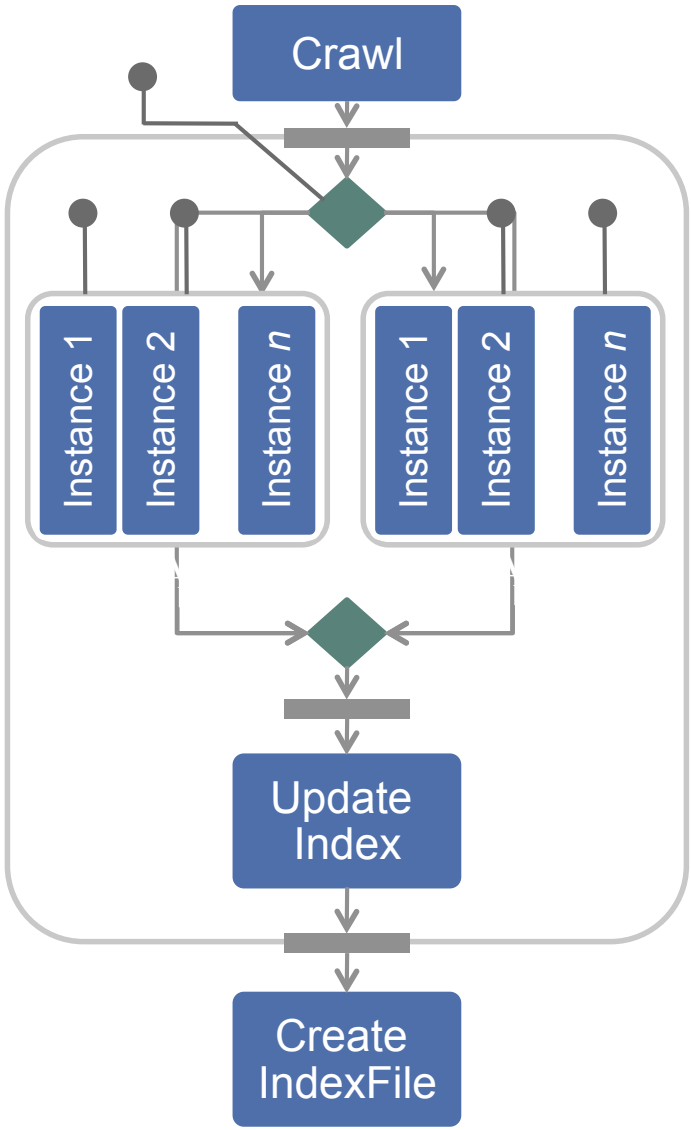
Beispiele für Tuning-Parameter

- Anzahl Fließbandstufen
- Wahl des besten Algorithmus
- Wahl des Kerntyps (CPU/ GPU)
- Größe der Datenpartitionen
- Anzahl Arbeitsfäden
- Lastbalanzierungsstrategie

Beispiel: Erzeugung eines invertierten Index

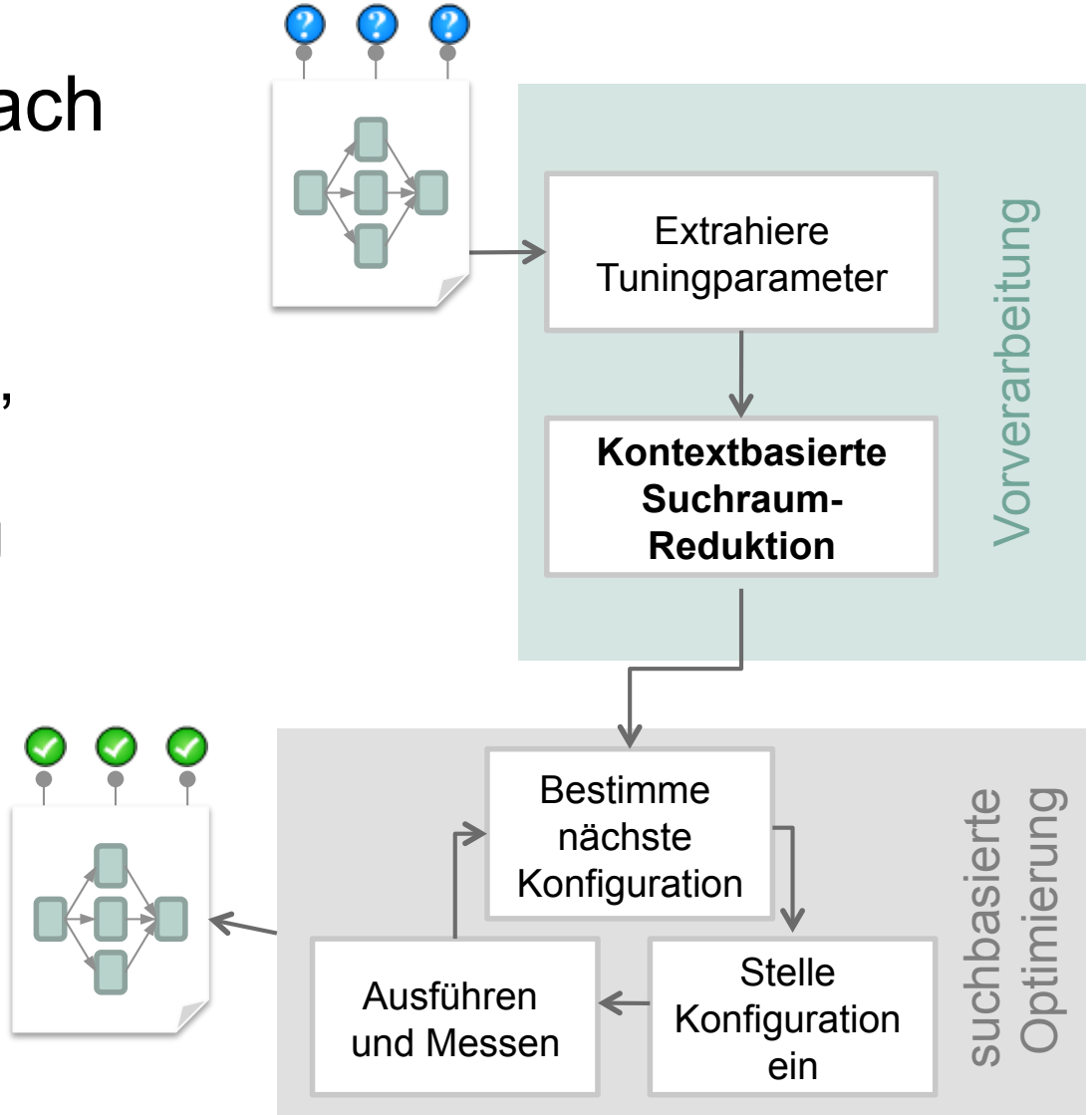
```
TunablePipeline MyDesktopSearch
[source:Crawl;sink:CreateIndexFile]
{
    TunableAlternative
    {
        ParseAlgo1
        ParseAlgo2
    },
    AC_UpdateIndex
}
```





Atune-OPT
Optimisierungsprozess

- Empirische Suche nach bester Konfiguration
  - Suchalgorithmen: zufällige Stichproben, *Gradientensuche*, Schwarmoptimierung

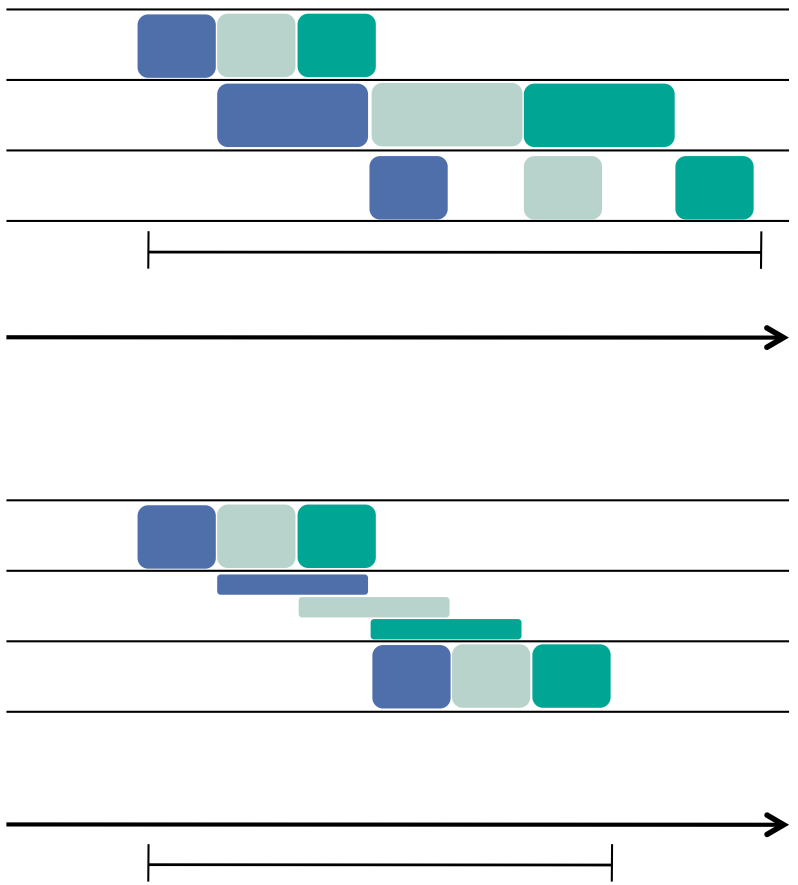


Atune-OPT
Suchraumreduktion

- Ausnutzung der Entwurfsmuster

Beispiel

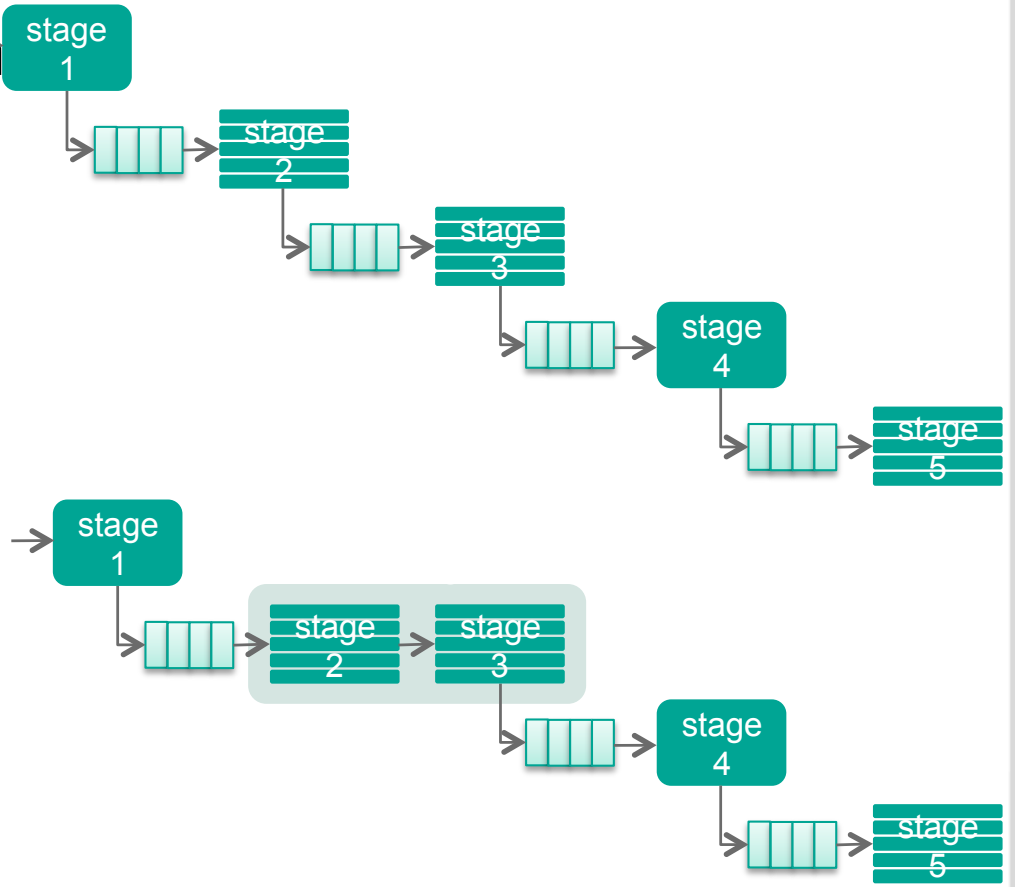
- *Optimierbares Fließband*
- Anstelle alle Kombinationen durchzuprobieren, balanziere Fließband:
- Hier: repliziere langsame Stufen



Atune-OPT

Suchraumreduktion (2)

- Fusioniere hintereinander liegende, schnelle Stufen (spart Puffer, Synchronisation)
- Fusionieren/Replizieren, bis alle Stufen etwa gleichschnell.
- Mögliches Ergebnis: alle Stufen fusioniert, aber mehrere Instanzen des ganzen Fließbandes

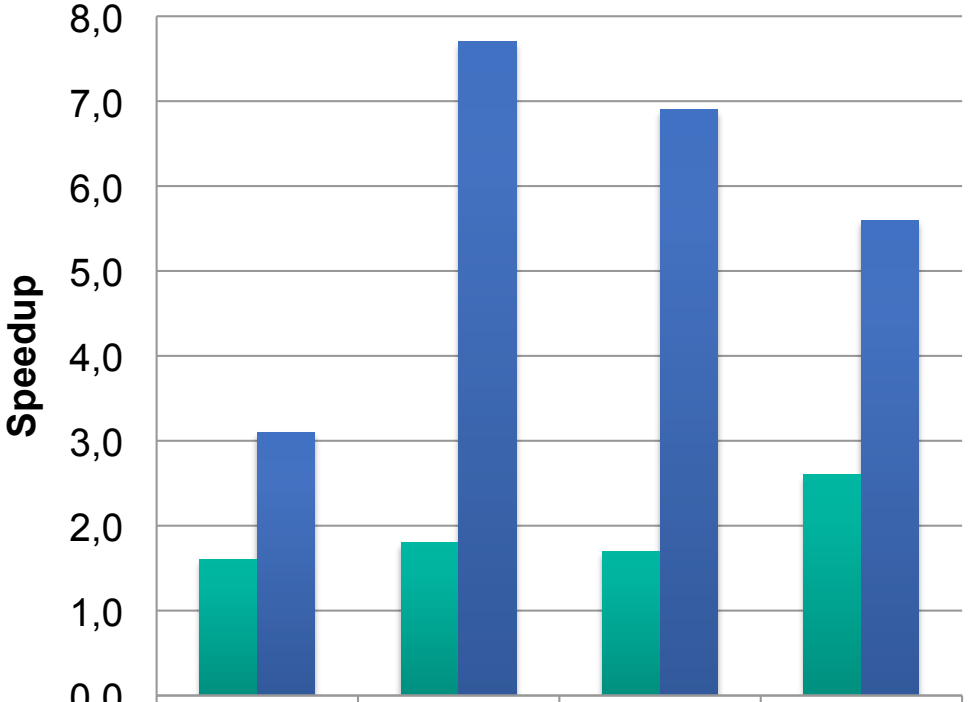


Evaluierung (1)

- Beschleunigung durch Autotuning gegenüber sequenzieller Variante

Metriken

- *Worst* speedup
- *Best* speedup after tuning
- Tuning Performance Gain (TPG)



	MID	GrGen	DS	Video
Worst	1,6	1,8	1,7	2,6
Best	3,1	7,7	6,9	5,6
TPG	194%	428%	406%	215%

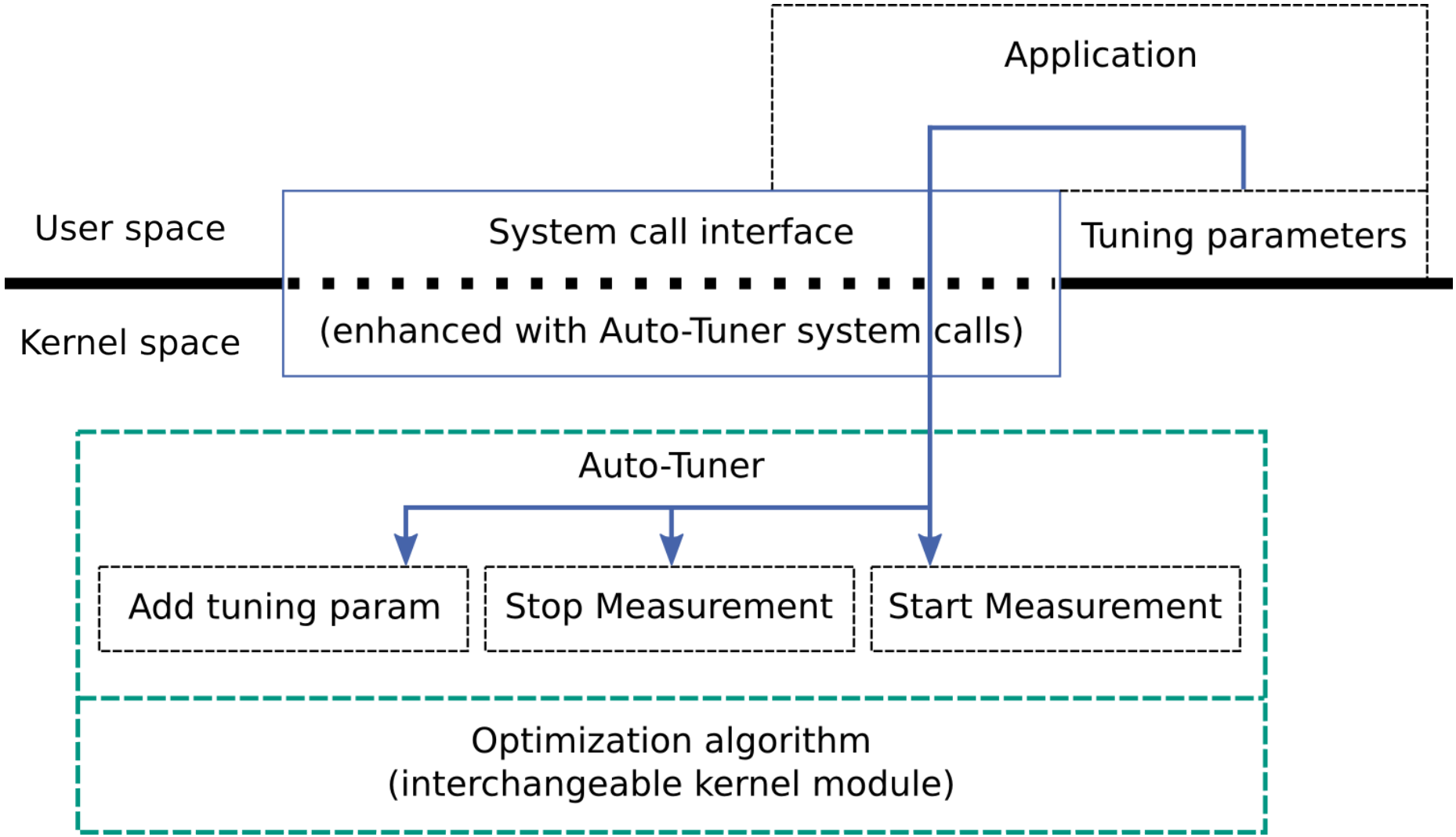
Related Work

- *ATLAS/AEOS* (Whaley et al., 2000)
  - Auto-tuning system for algebraic operations and algorithms
  - Domain specific approach
  - No support for parallel programs
- *Active Harmony* (Tapus et al., 2002)
  - Search-based auto-tuning system for library optimization
  - Comprehensive analysis of search algorithms
  - Not applicable for parallel programs
- *MATE* (Morajko et al., 2007)
  - Model-based tuning system for distributed PVM programs
  - Provides good performance predictions
  - Limited to special program structures
- *Parallel Pattern Language* (Mattson et al., 2004)
  - Structured collection of parallel patterns
  - Provides guideline for parallel programming
  - Optimization is not considered

Weitere Forschung

- Mehr optimierbare Architekturmuster!
- Integration der Muster in Prog.-Sprache (XJava)
- Wende Auto-Tuning auf heterogene Prozessoren an!
- Parameter-Vorhersage
  - Finde gute Startwerte für die Suche
    - Replikationsgrad limitiert durch freie Fäden
    - Benutze work-stealing-Heuristik bei Rekursion
  - Erste Ergebnisse: manchmal bis zu 90% des Optimums vorhersagbar.
- Online Tuning
  - Optimiere während der Programmausführung (statt davor in Testläufen)

Online Auto-Tuning: Architektur



Online Autotuning: Wie´s funktioniert

```
int threadCount = 1;
addParam(&threadCount, 1, 16);

while (calculation not finished) {
   startMeasurement();

   doCalculation(threadCount);

   stopMeasurement();
}
```

Autotuner wird Tuning-Parameter und Werteberich übergeben

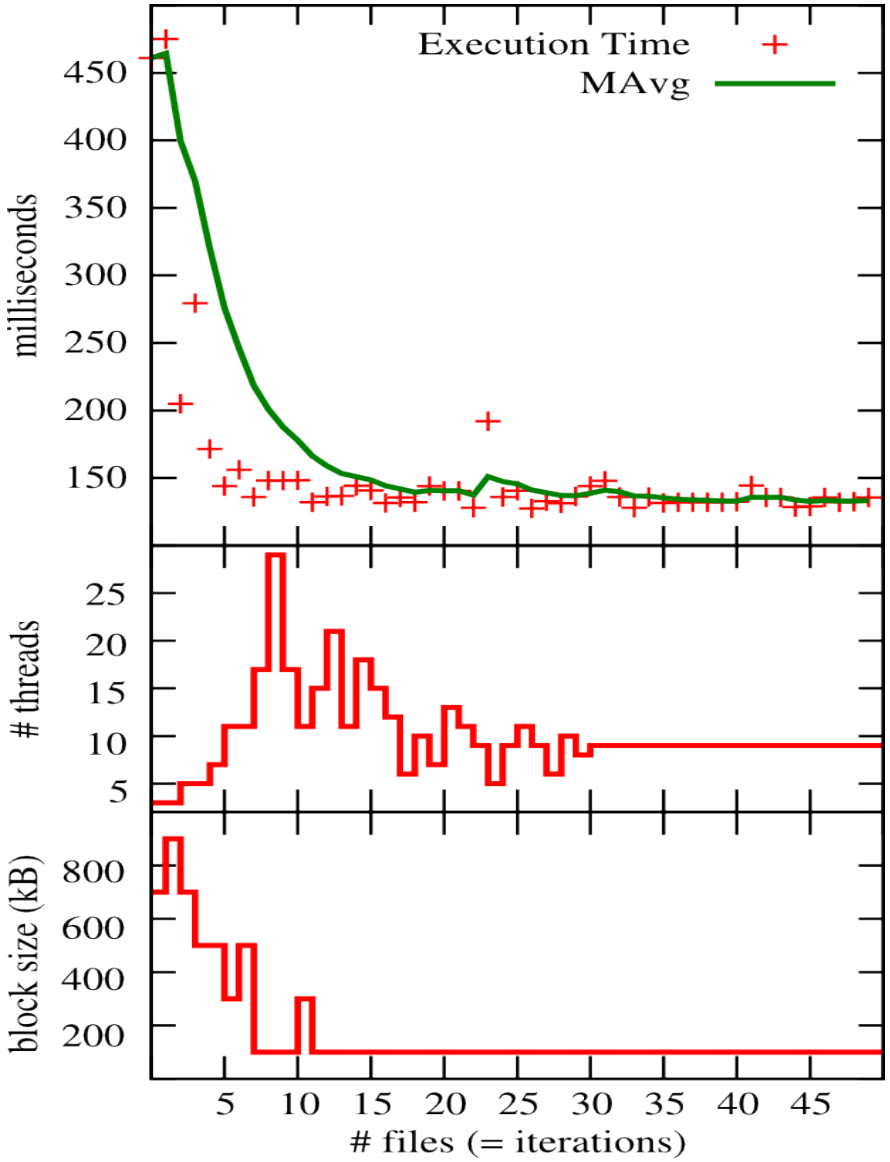
Mess-Schleife

Zeitmessung startet

Ablauf wird gemessen

Zeitmessung endet
Autotuner justiert Tuning-Parameter

Auto-Tuning BZip2



Paralleles BZip2

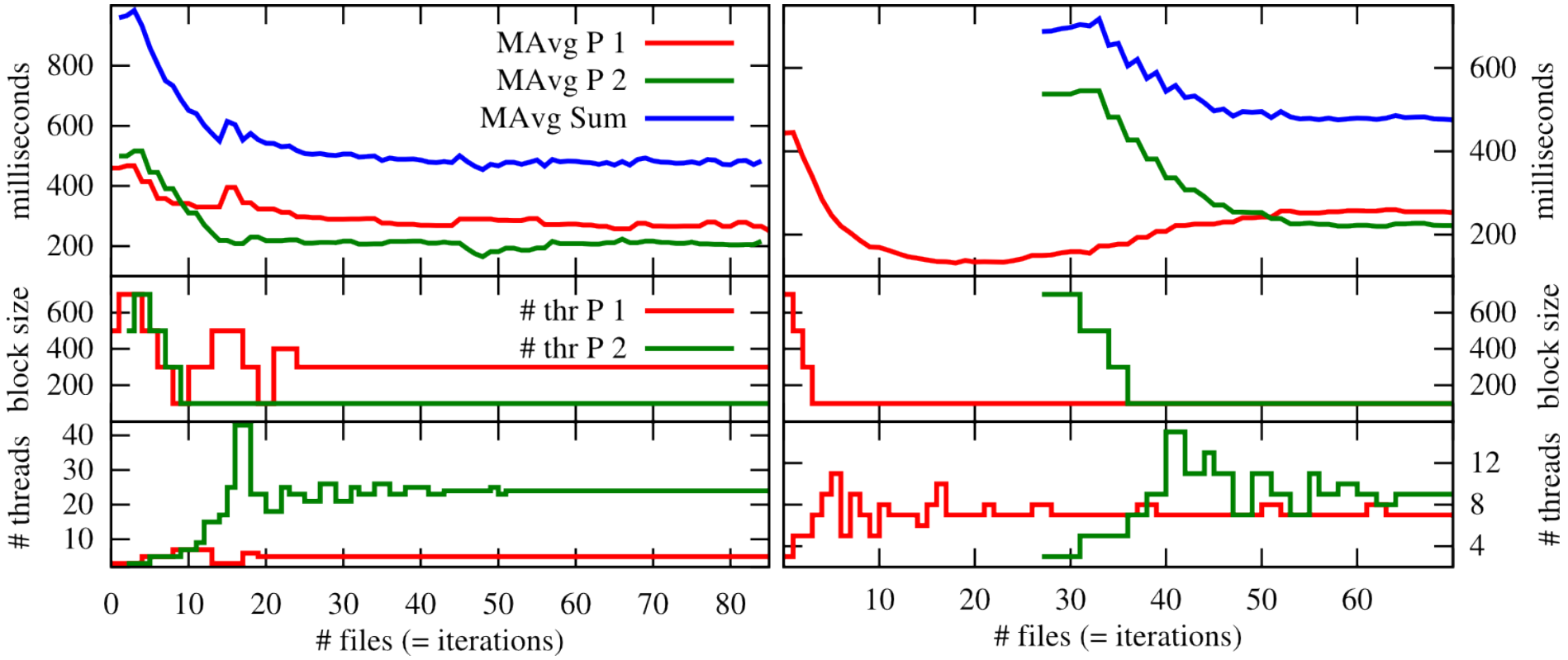
Initiale Parameterwerte:
3 Fäden, Blockgröße 700 kB

Laufzeit ohne Tuning: 22,9 s

Laufzeit mit Tuning: 8 s

Beste Laufzeit (wenn man mit bester Konfiguration startet): 6,5 s

Auto-Tuning: Zwei BZip2-Läufe gleichzeitig



Viele weitere Forschungsthemen

- Mehrkern-SWT ist reif für Metoden/Werkzeug-Entwicklung
- geleitet von empirischer Evaluation
  - Restrukturierung sequenzieller Anwendungen für Parallelisierung
  - Fallstudien zu Parallelisierung
  - Automatische Parallelisierung
  - Autotuning (bessere Suchalgorithmen, Prädiktion)
  - Spracherweiterungen für parallele Programmierung
  - Programmierung heterogene Prozessoren
  - Finden von Wettläufen und Synchronisierungsfehler

  - Ihr Thema!

Zusammenfassung

- Mehrkerner haben bereits übernommen, selbst bei mobilen Geräten.
- SW-Parallelisierung ist eine große Herausforderung.
- Zur Zeit weitgehend Handarbeit.
- Werkzeuge werden die Arbeit mit der Zeit vereinfachen.
- In der Forschung werden jetzt viele neue Ansätze untersucht (Entwurfsmuster, Bibliotheken, Sprachen, Wettlauferkenner, Autotuner, Ablaufplaner, u.v.a.m.)

- Parallelverarbeitung wird zum Normalfall werden.

Aktuelles Buch dazu: