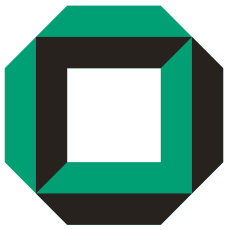
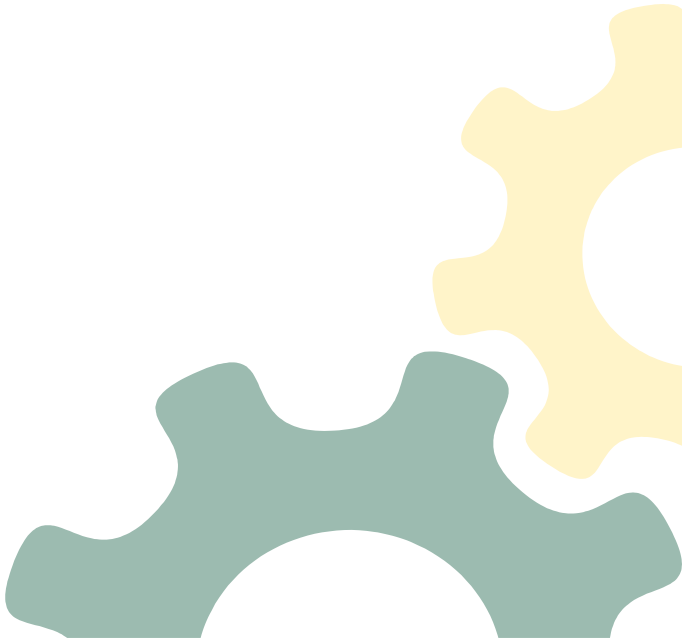


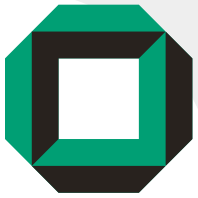
Universität Karlsruhe (TH)
Forschungsuniversität · gegründet 1825

Bereit für die Multicore-Revolution?

Walter F. Tichy

Fakultät für **Informatik**
Lehrstuhl für Programmiersysteme





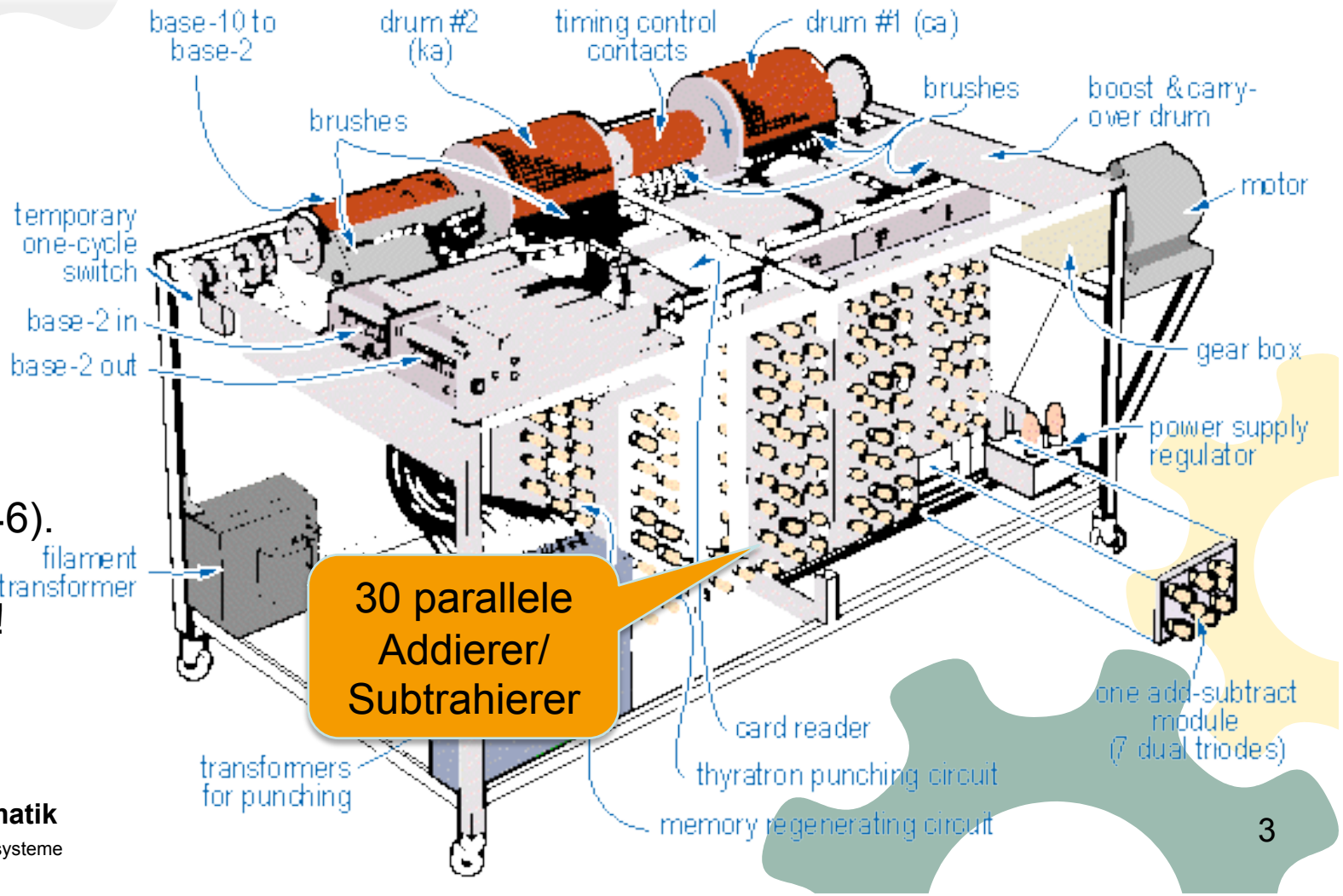
Wir erleben gerade eine Umwälzung der Informatik

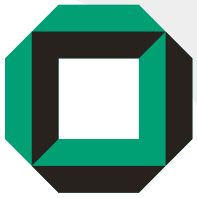
- Von Anfang an war die Informatik geprägt vom sequenziellen Rechner.
- Parallelität war nur in Nischen präsent:
  - Numerisches Rechnen
  - Betriebssysteme, Datenbanken
  - Parallelität auf Instruktionsebene
- Mit Mehrkern-Chips wird Parallelität flächendeckend zur Verfügung stehen.
- Es ist bereits schwierig, einen Rechner mit nur einem Prozessor zu kaufen.

Einige wichtige Parallelrechner

Atanasoff-Berry-Rechner von 1942

Erster digitaler, elektronischer Rechner.
Vor ENIAC (1946).
Schon 30-fach parallel!

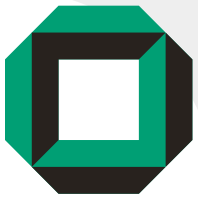




Einige wichtige Parallelrechner

Illiac-IV: SIMD, verteilter Speicher

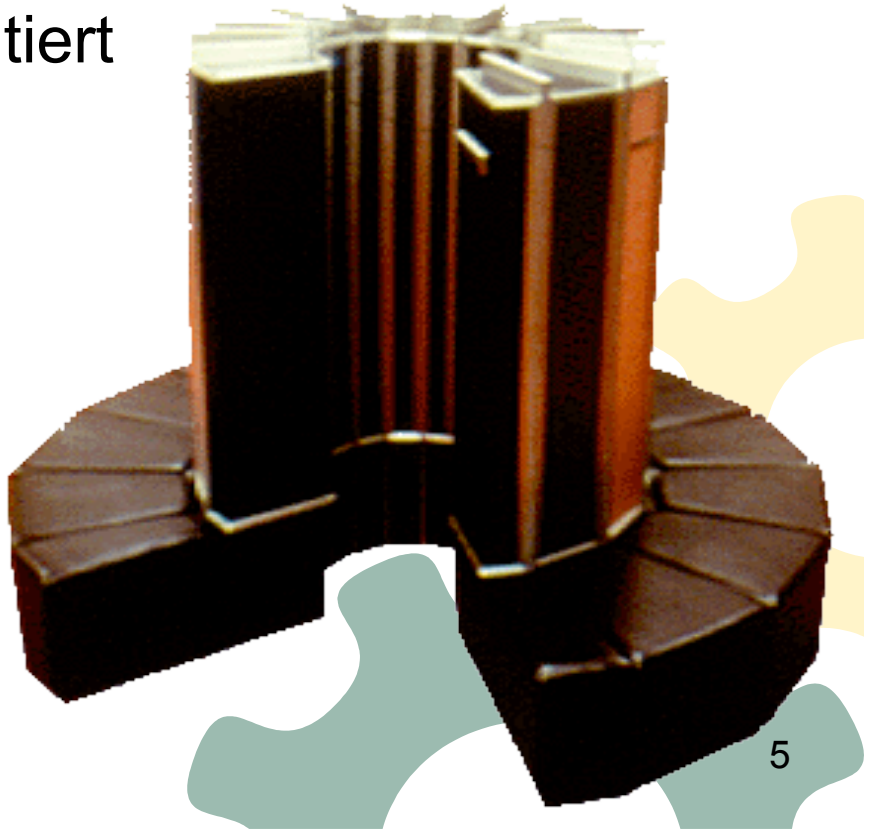
- Nur ein Exemplar gebaut: 1976
  64 Prozessoren
- Schnellster Rechner bis 1981



Wichtige Parallelrechner

Cray-1 Vektorrechner

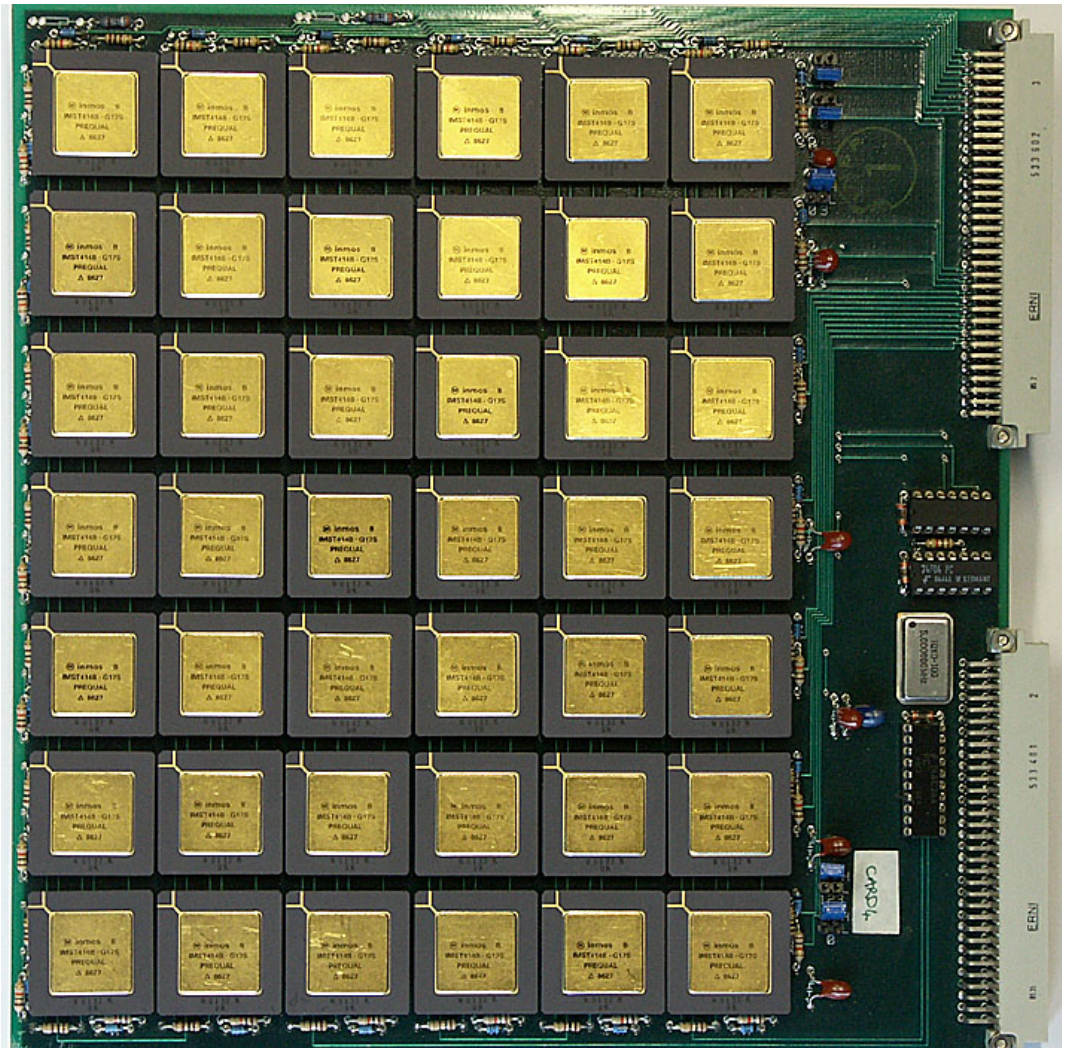
- gemeinsamer Speicher
- Vektorregister mit 64 Elementen
- Vektorinstruktionen implementiert mit Fließbändern.
- Erstmals geliefert 1976
- Damals der zweitschnellste Rechner nach Illiac-VI



Wichtige Parallelrechner

Transputer

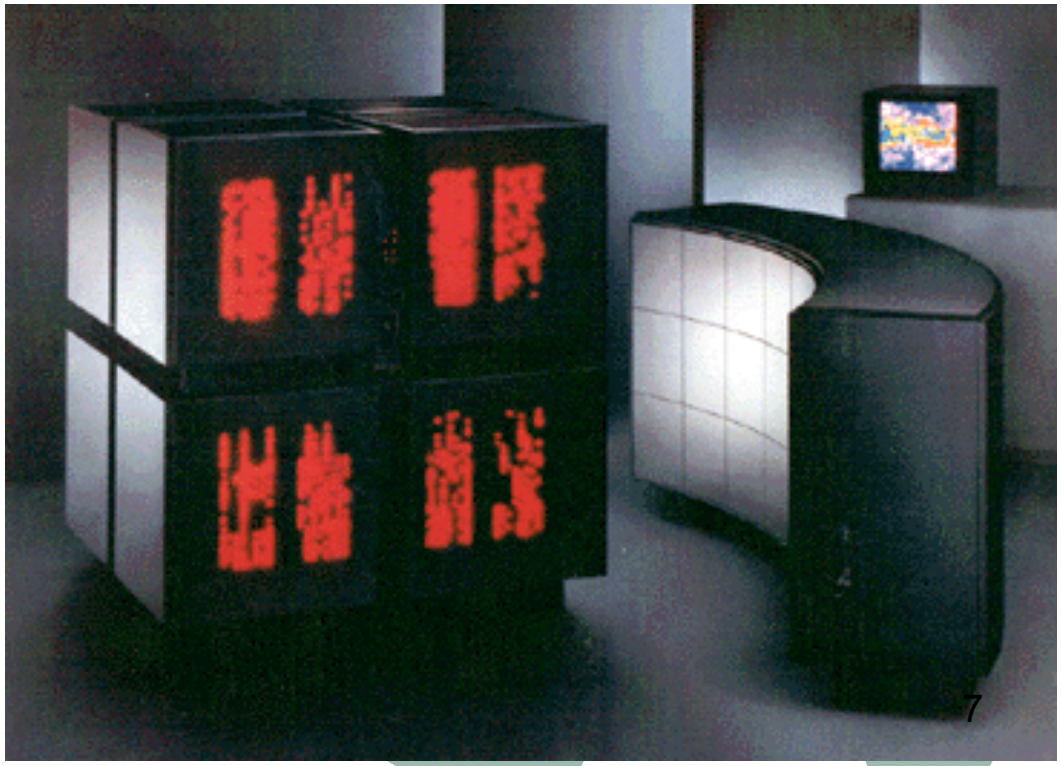
- MIMD-Rechner mit verteiltem Speicher
- Prozessoren mit vier schnellen Verbindungen
- Erstmals geliefert 1984
- Bis zu 1024 Prozessoren
- Occam als Programmiersprache
- Entwickelt von Inmos, Bristol, GB



Wichtige Parallelrechner

CM-1 Connection Machine

- SIMD-Rechner, verteilter Speicher
- 65.536 Prozessoren (1-Bit)
- Hyperkubus als Verbindungsnetz
- Erstmals geliefert 1986
- Erster massiv paralleler Rechner



Wichtige Parallelrechner

Rechnerbündel (Cluster Computers)

- Standard PCs verbunden mit Standard-Netzen
- MIMD-Rechner mit verteiltem Speicher
- Kostenvorteile durch Komponenten aus dem Massenmarkt
- Heute die schnellsten Rechner
- Hunderttausende von Prozessoren
- Siehe top500.org

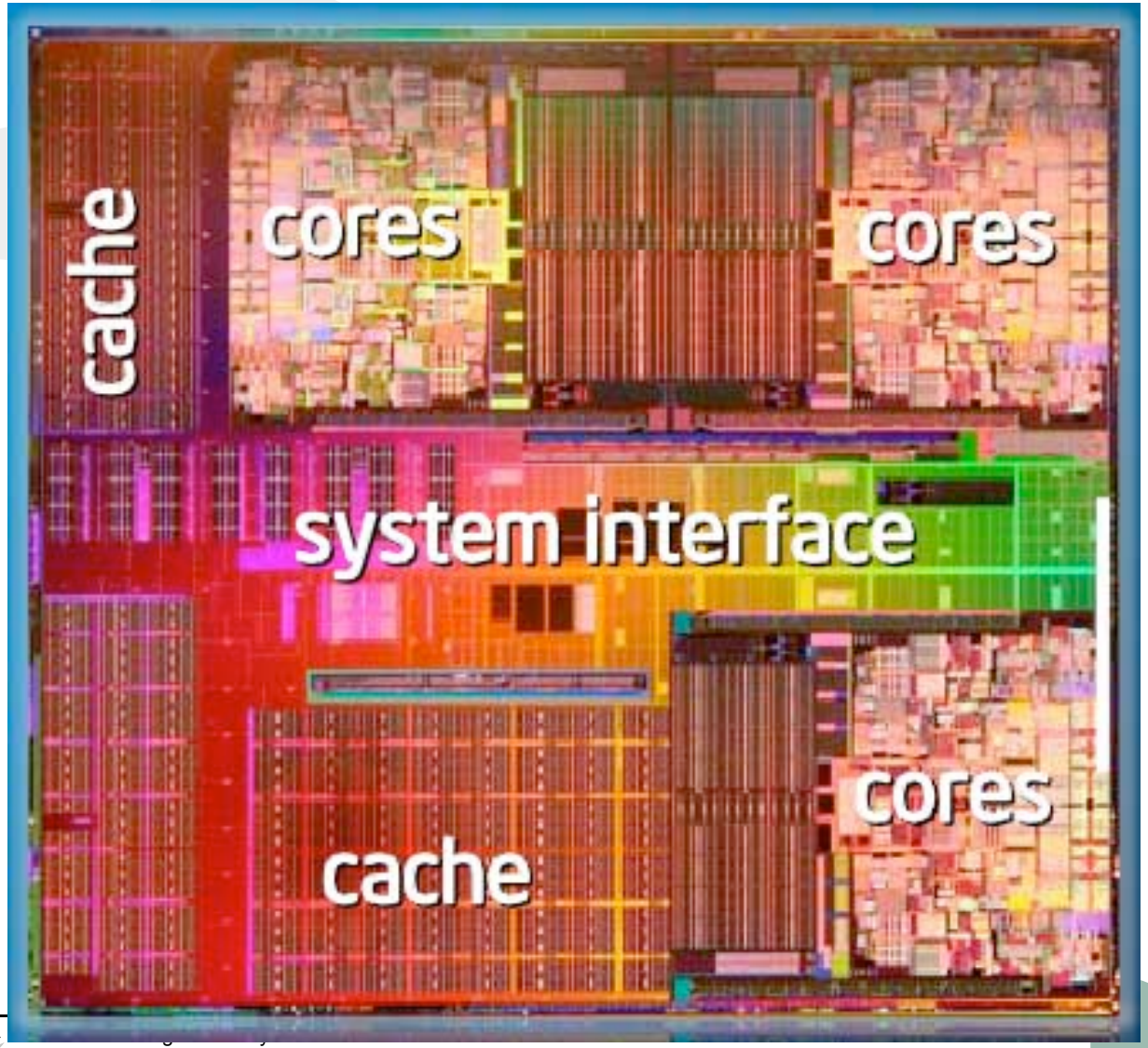
Nasa Avalon mit 140 Alpha Knoten, 1998

Ihr Laptop—
ein Parallelrechner?

Preise Juni 2007

			
Inspiron™ 6400 15" Notebook für vielseitige Unterhaltung & 1 GB RAM.	**Inspiron™ 1520** Stylischer Denker, der es geniesst seine Qualitäten zeigen zu können und gerne im Mittelpunkt steht.	**Inspiron™ 1720** Unterhaltung & Spaß garantiert! Technologie genau angepasst für Ihren Lifestyle. Jetzt in 8 Farben.	**Inspiron™ 1520** Schlank, elegant und noch etwas schlauer und stärker. Jetzt in 8 Farben.
~~729 €~~ 659 € inkl. MwSt., zzgl. 78 € Versand	~~999 €~~ 899 € inkl. MwSt., zzgl. 78 € Versand	1.049 € inkl. MwSt. und Versand	~~1.129 €~~ 1.079 € inkl. MwSt., zzgl. 78 € Versand
Prozessor Intel® Pentium® Dual-Core T2080 Prozessor (1,73 GHz, 533 MHz, 1 MB L2-Cache)	**Prozessor** Intel® Core™ 2 Duo T5450 Prozessor (1,66 GHz, 667 MHz, 2 MB L2-Cache)	**Prozessor** Intel® Core™ 2 Duo T5250 Prozessor (1,5 GHz, 667 MHz, 2 MB L2-Cache)	**Prozessor** Intel® Core™ 2 Duo T7100 Prozessor (1,8 GHz, 800 MHz, 2 MB L2-Cache)

Intel Dunnington
6 Prozessoren auf einem Chip



3 x 2 Xeon Prozessoren

(nicht HW-mehrfädig)

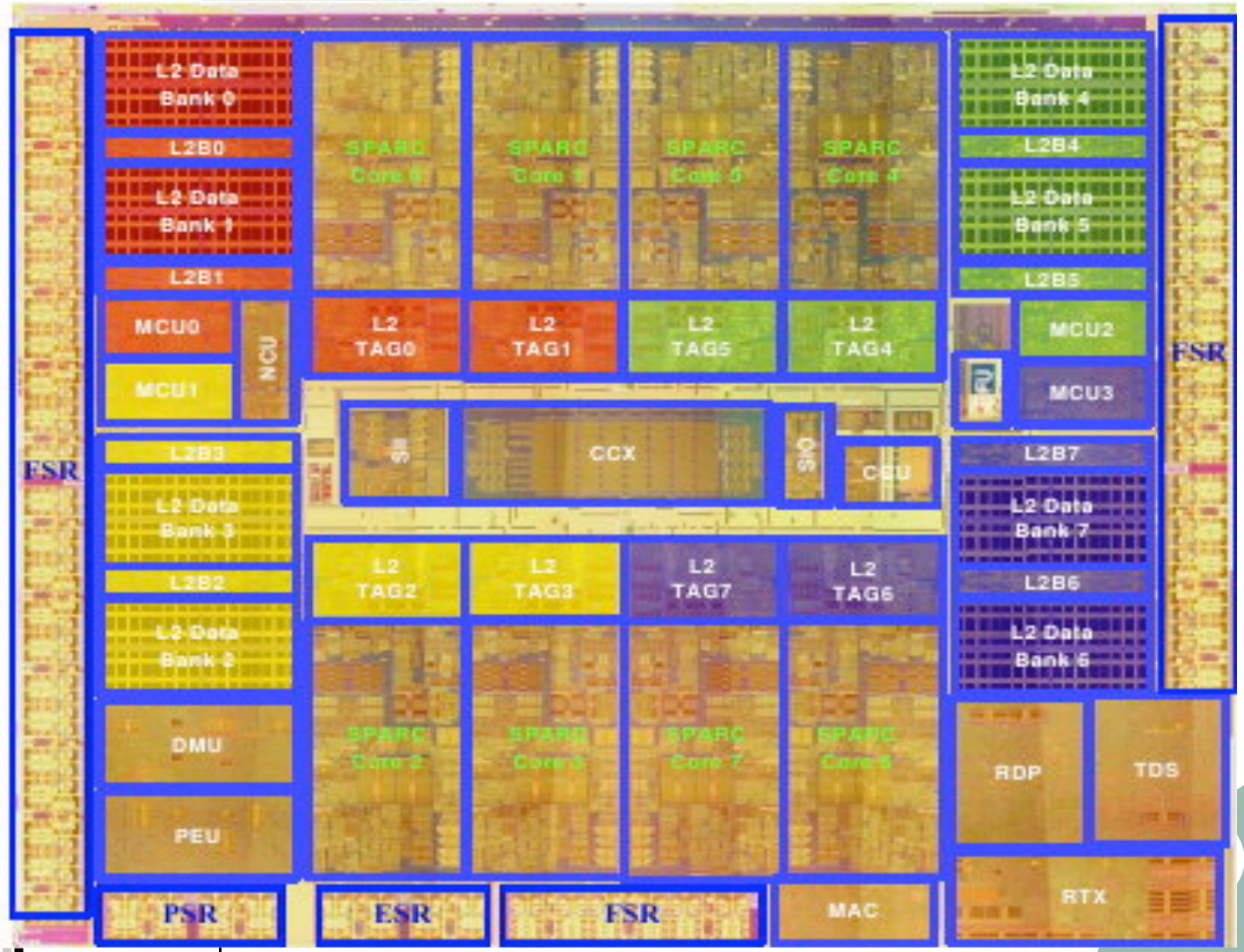
2,6 GHz

130 W

45nm Technik

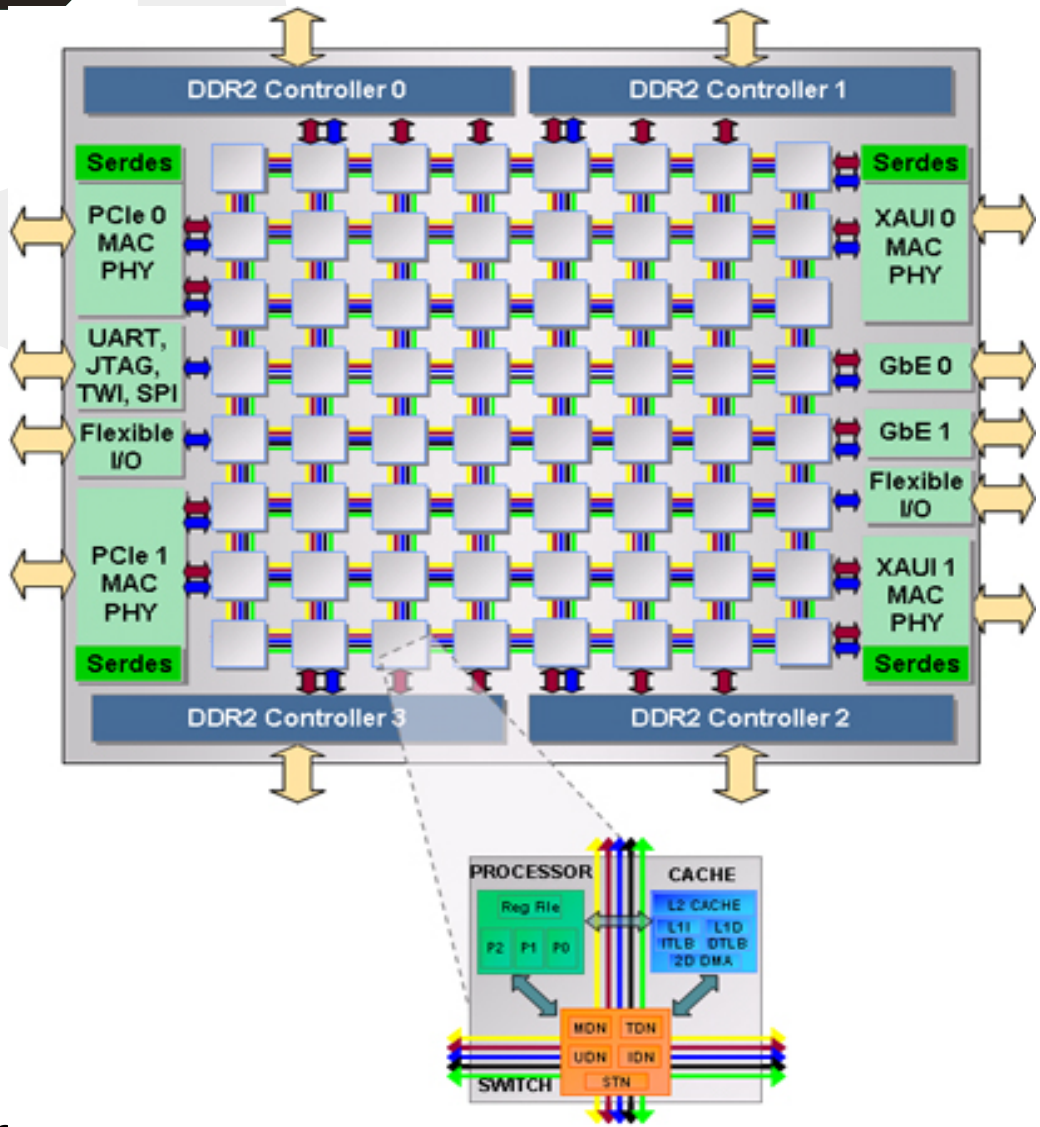
Lieferbar 2008

Sun Niagara 2:
8 Prozessoren auf 3,42 cm^2



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

Tilera´s TILE64



64 VLIW Prozessoren plus Gitter auf einem Chip

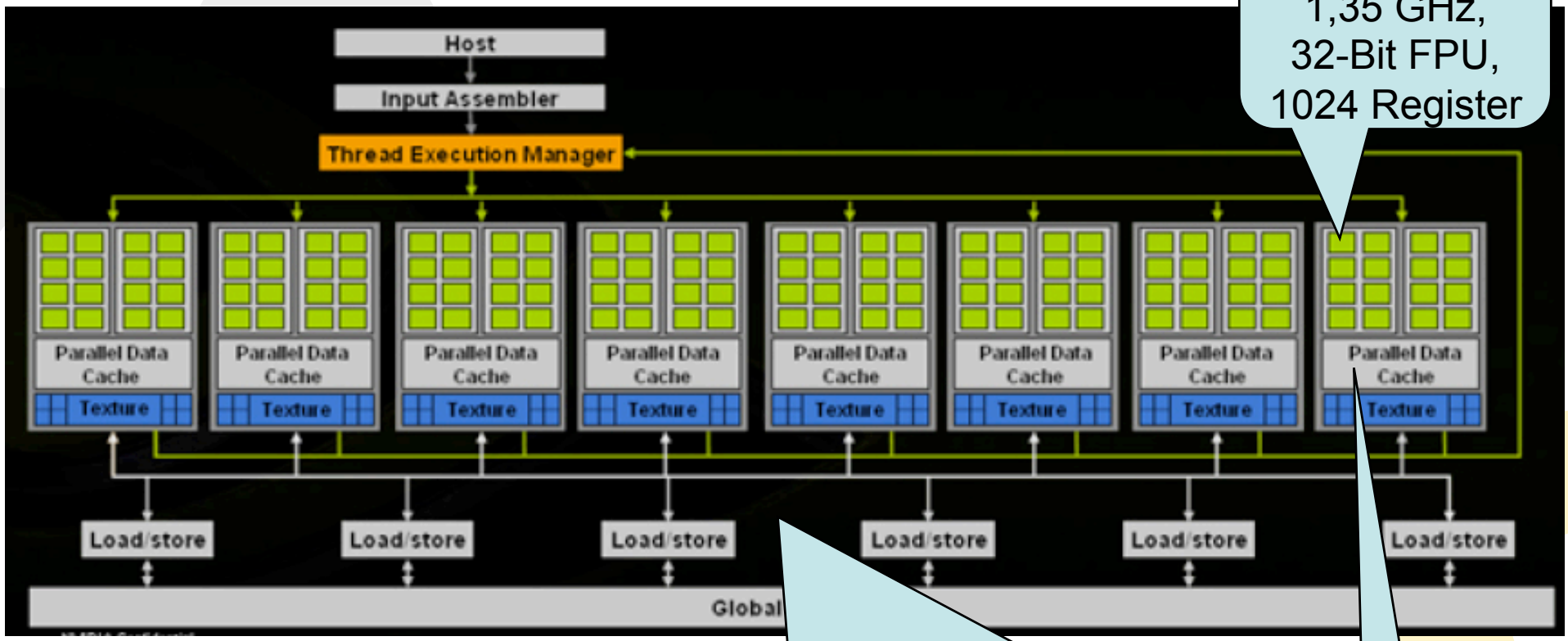
Für eingebettete Anwendungen (Netz- und Videoanwendungen)

700 MHz

22 W

Lieferbar 2007

Nvidia GeForce 8 Graphics Processing Unit



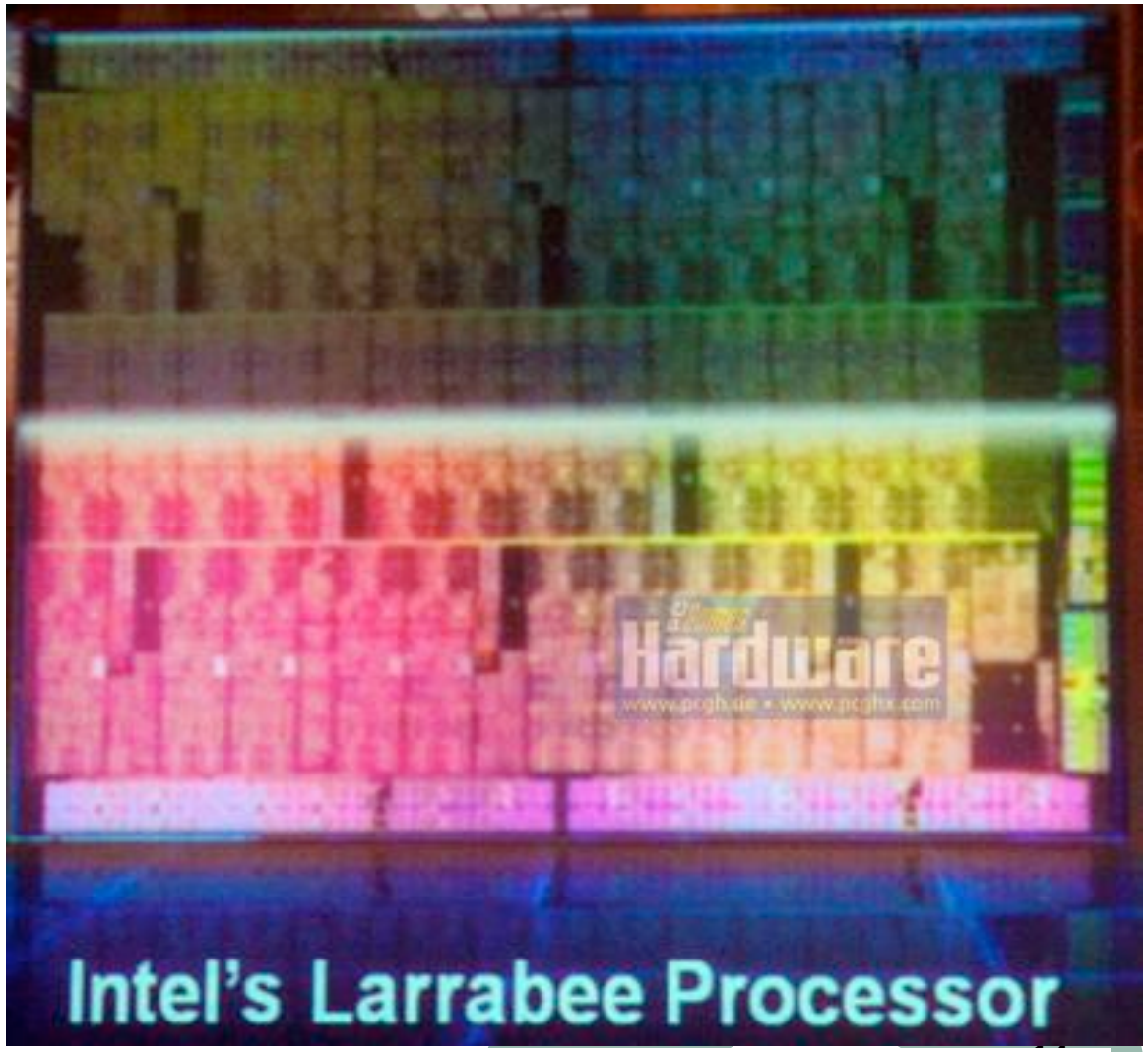
Prozessor,
1,35 GHz,
32-Bit FPU,
1024 Register

16 KB

128 Prozessoren insgesamt, jeder mit 96 Fäden in HW
Insgesamt 12288 HW-Fäden!
SIMD

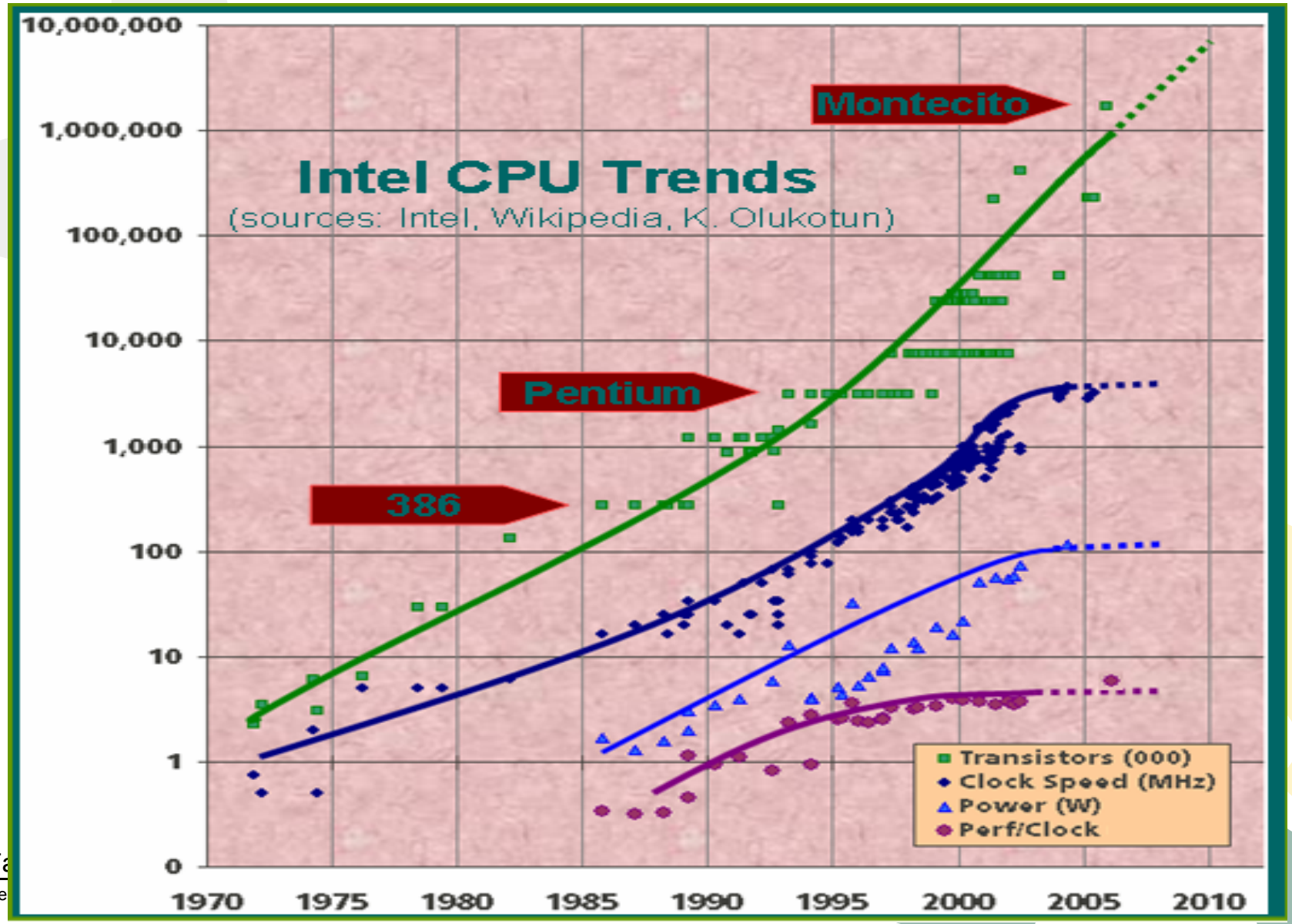
Intels Larrabee: 32 Pentiums auf einem Chip

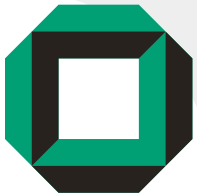
- 32 x86 Prozessoren (45nm),
- 48 x86 Prozessoren (32 nm)
- Cache-kohärent,
- Ringverbindung
- 64-bit Arithemetik
- 4 Registersätze pro Prozessor
- Vektorbefehle für Grafik
- Angekündigt für 2010



Source: www.pcgameshardware.com, May 12, 2009

Was ist passiert?

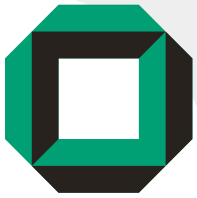




Moore´sche Regel, neue Version

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

Parallelrechner werden in naher Zukunft flächendeckend zur Verfügung stehen.



Was tun mit all den Kernen?

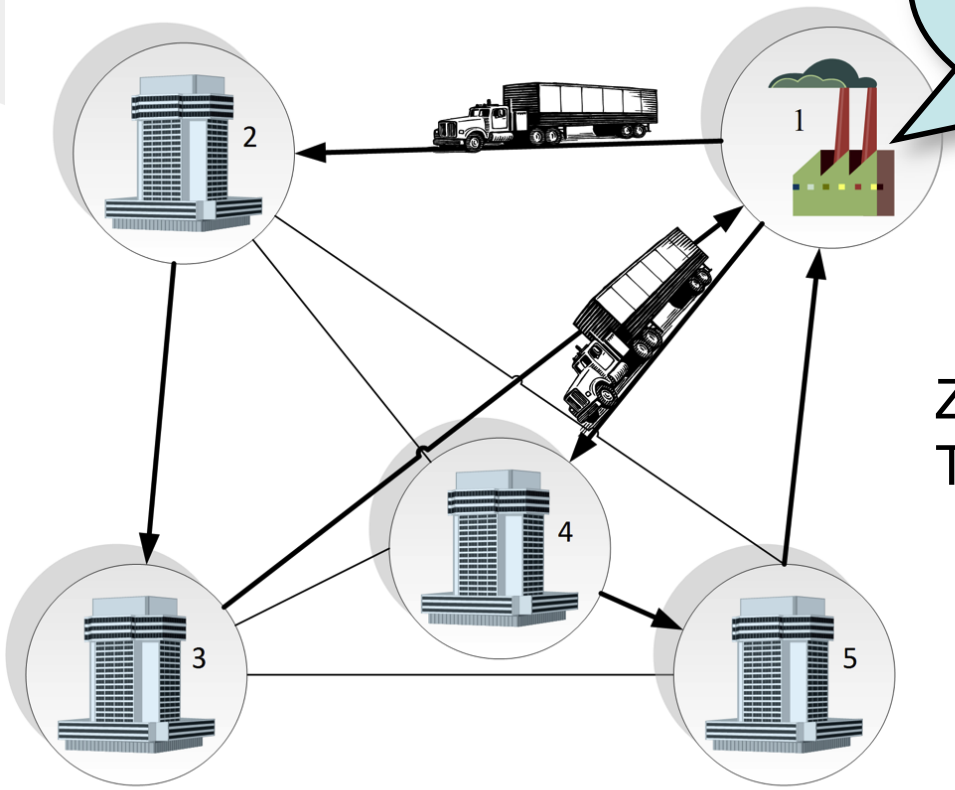
- „Wer braucht 100 Prozessoren für M/S Word?“
  - Mangel an Vorstellungskraft, Informatik-Ausbildung?
  - Gesucht werden überzeugende Anwendungen, die Hunderte von Prozessoren gebrauchen können.
- Wovon könnten alle Nutzer von PCs, eingebetteten Systemen und Mobiltelefonen profitieren?

Beispiel 1: Transportproblem (Diplomarbeit bei SAP)

Welche Aufträge?

Auf welchen Fahrzeugen?

Über welche Strecken?



Gegeben:
- Lieferaufträge
- Fahrzeuge
- Streckennetz

Ziel: kostenoptimaler Transportplan

Verallgemeinerung des Problems des Handlungsreisenden

Warum parallelisieren?

Reale Transportszenarien

Szenarien	1	2	3
Aufträge	804	1177	7040
Ladedimensionen	3	2	4
Aufladestationen	1	1	3
Abladestationen	31	559	1872
Zwischenstationen	0	5	0
Fahrzeuge	281	680	2011
Fahrzeugtypen	7	3	10
Zeitfenster		1	64

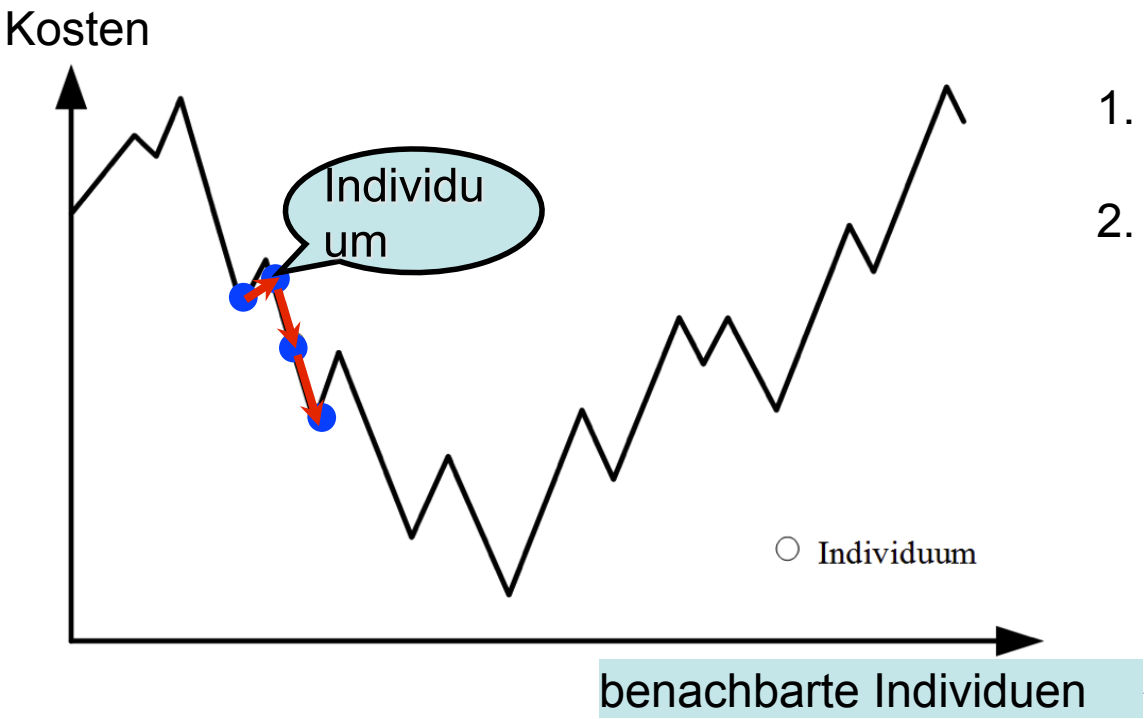
Noch zu berücksichtigen:
Ruhezeiten der Fahrer,
Ladezeiten, Anhänger,
mit/ohne Kühlung,
Fähren, Schiffe....

Sequenzielle Lösung:

- -150.000 Zeilen C++
- -Evolutionärer Algorithmus

mehrere Stunden Rechenzeit für gute Lösungen nötig!

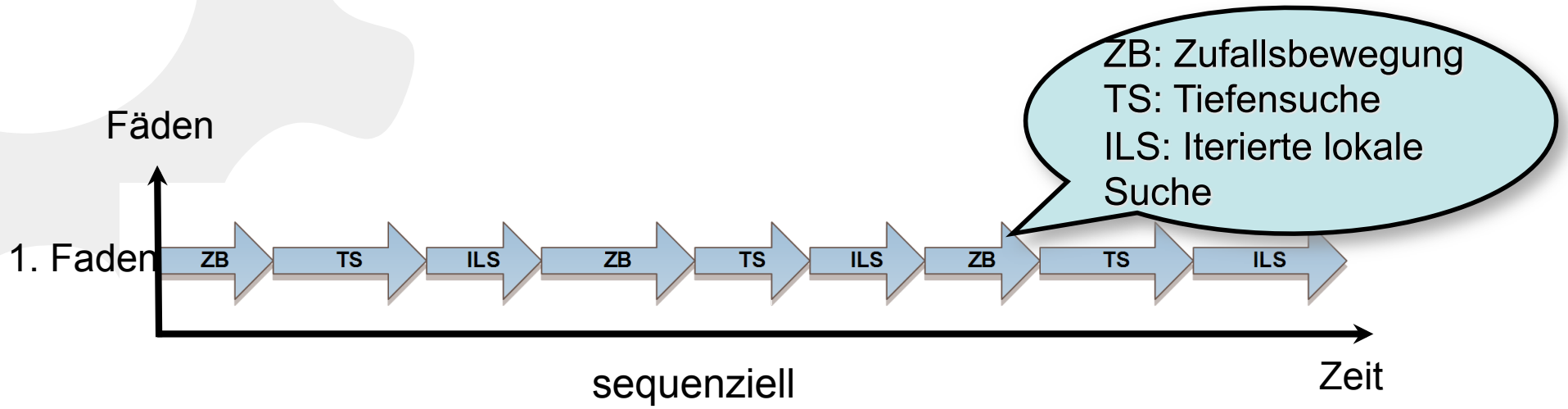
Suchalgorithmus

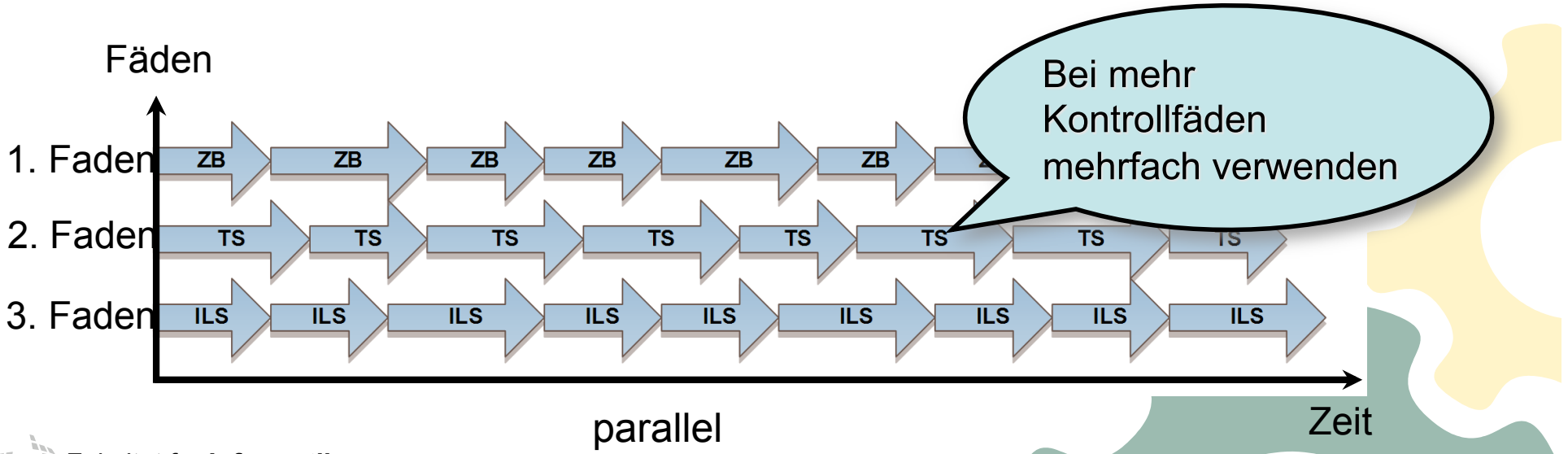


1. Beginne mit initialer Lösung

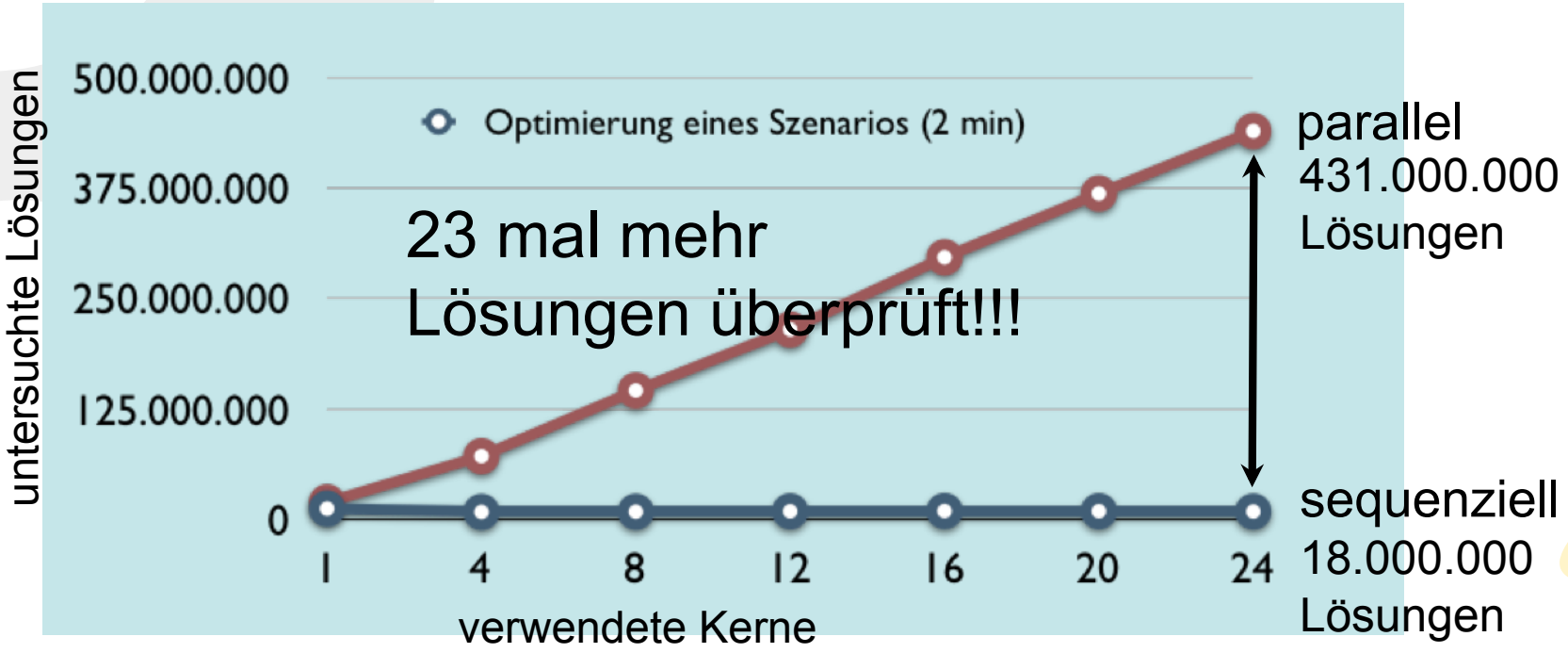
2. Solange Kostenschranke nicht erreicht:
   - Optimiere Lösung mit lokalen Verbesserungen (genaue Untersuchung der Nachbarschaft)
   - Entkommen aus lokalem Minimum durch zufälligem Sprung im Lösungsraum (Führt zu breiter Abdeckung des Suchraums)

Prinzipieller Ablauf



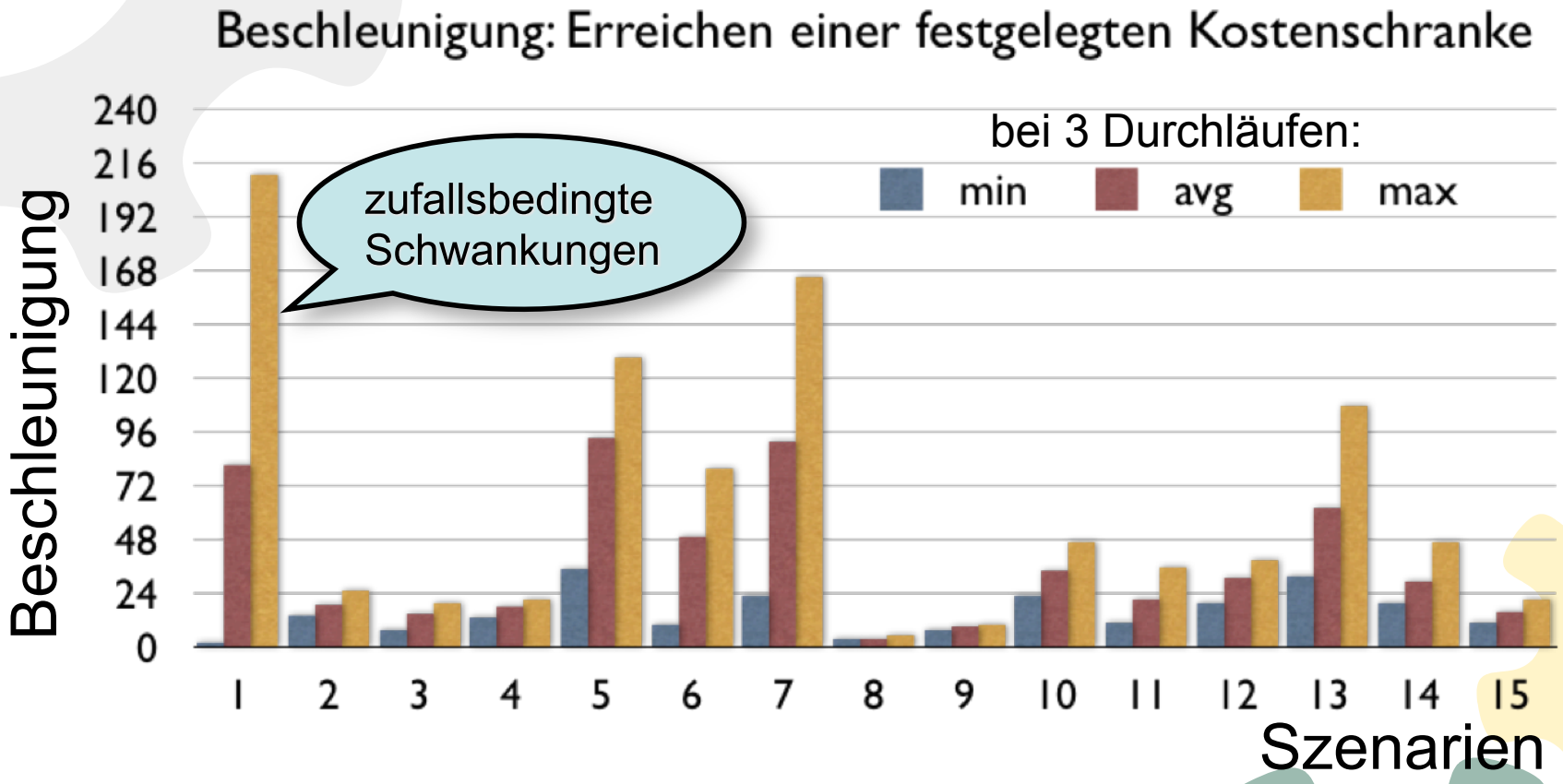


Untersuchte Lösungen in 2 Minuten

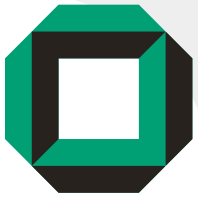


Auf Rechner mit 4 Intel Dunnington Chips
(4 * 6 Prozessor-Kerne)

Beschleunigung bei 24 Fäden



Beschleunigung bei Durchführung aller Szenarien: 17,37
Superlineare Bescheunigung bei frühzeitigem Finden guter Kandidaten

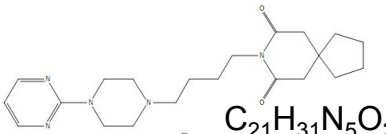


Was ist das generelle Problem bei der Parallelisierung?

- Für den Erfolg müssen Beschleunigung, Programmier-Produktivität und Qualität gleichzeitig gemeistert werden.
  - Parallelisierung nur bei Beschleunigung interessant
  - Produktivität und Qualität dürfen nicht schlechter werden!
- Sprachen, Debugger, Analysewerkzeuge sind unzureichend für Parallelität (Thread ≅ Goto?)
- Die meisten Programmierer sind unzureichend auf Parallelität vorbereitet.

Beispiel 2: Analyse von Massenspektrogrammen

Massenspektrogramme



$C_{21}H_{31}N_5O_2$

Wirkstoff

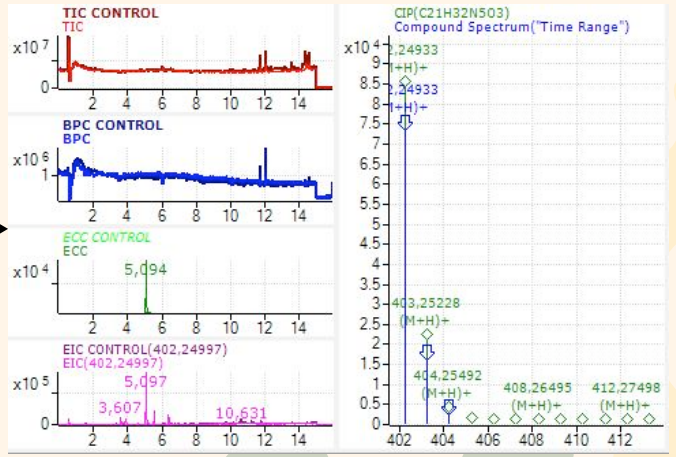
Agilent Technologies

Laden der Spektrogramme

Laden eines Wirkstoffes

1 2 3 4 … n

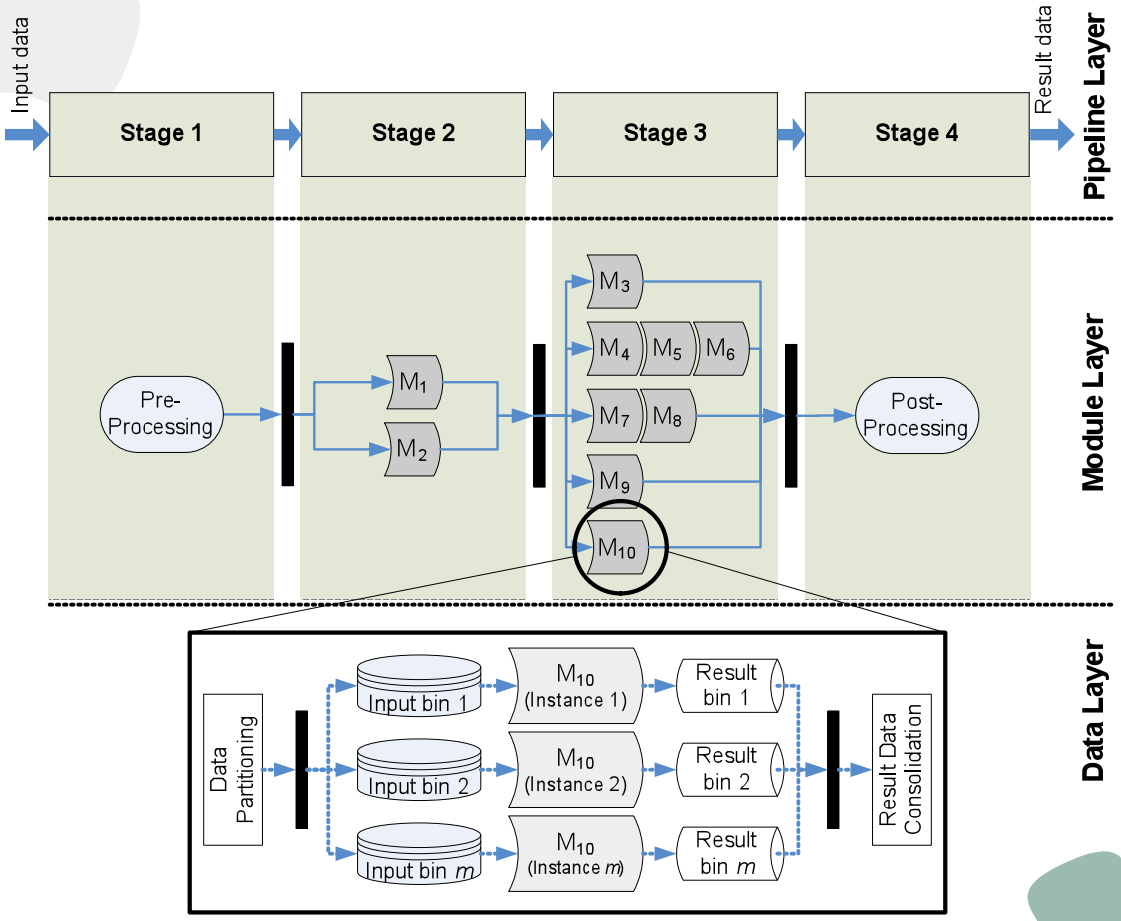
Mehrstufiges Algorithmen-Fließband:
Suchen von Metaboliten in den Spektrogrammen
Parallelisierungspotential

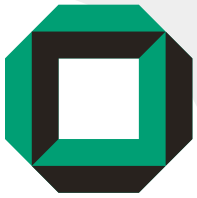


Ergebnis: Zeitabhängige Darstellung der Metaboliten

Desktop-Applikation

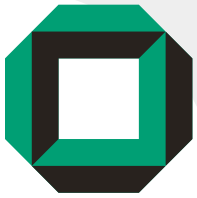
Mehrschichtige Parallelisierung mit Entwurfs-Mustern





Auto-Tuning

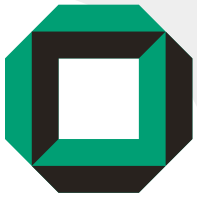
- **Problem:** Finde Optimum leistungsrelevanter Parameter
- Parameter sind Plattform- und Algorithmen-abhängig
  - Anzahl Kerne,
  - Anzahl Fäden,
  - Parallelitätsebenen,
  - Anzahl Fließbandstufen, Fließbandstruktur,
  - Anzahl Arbeiter in Master/Worker, Lastverteilung,
  - Größe der Datenpartitionen,
  - Wahl des Algorithmus
- Einstellung von Hand zu langwierig
- Lass den Rechner das machen!



Auto-Tuning (2)

- **Lösung:** Automatischer Parameter-Optimierer
  - Als Bibliothek, die einfach in eine Anwendung integrierbar ist, und die Parameterkomb. Durchprobiert
  - Annotationen in Atune-IL zeigt variierbare Parameter an.
  - Suchraum ist riesig, also Stichproben, Lernverfahren, intelligente Suche, etc. notwendig.
  - Leistungsunterschiede zwischen bester und schlechtester Konfiguration der Massenspektrogramm-Analyse: Faktor 1,9 (bei einer Gesamtbeschleunigung von 3,1 auf 8 Kernen)
  - bei einer Graphersetzungsanwendung trägt Autotuning sogar den Faktor 4,2 bei Gesamtbeschleunigung 7,7 bei.

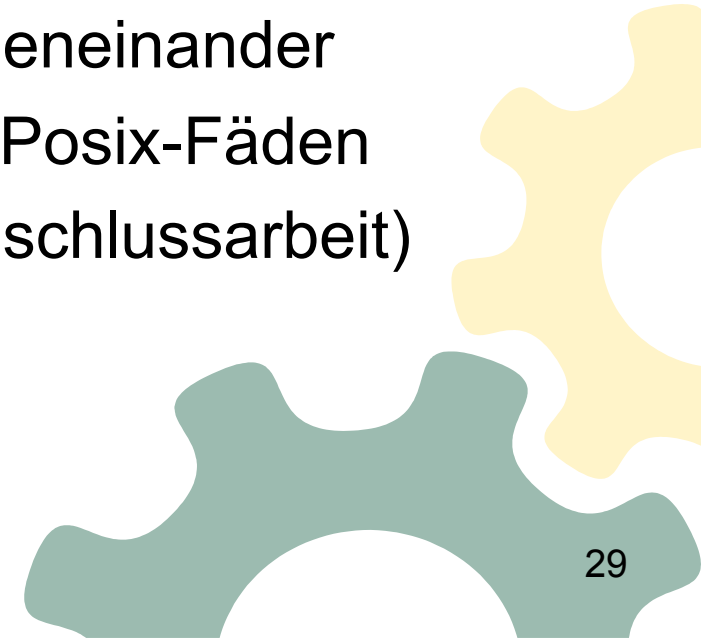
Getestet auf 8-Kern-Rechner (2x Intel Xeon E5320 Quad-Core 1,86 GHz)



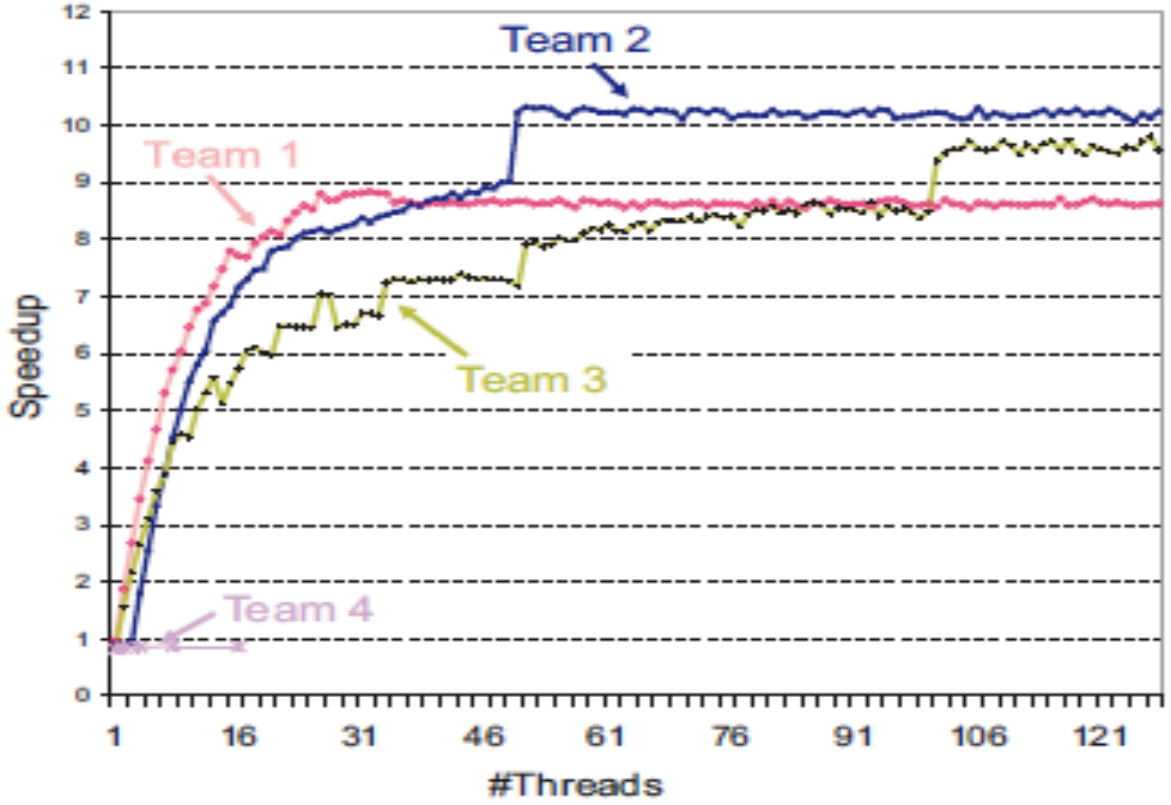
Beispiel 3: BZip2

- Kompressionsprogramm
- Vielfach im täglichen Gebrauch
- Ca 8000 Zeilen, quelloffen

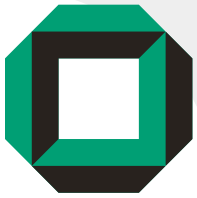
- Im Wettbewerb parallelisiert
  - 4 Teams von je zwei Personen gegeneinander
  - 3 Monate Training in OpenMP und Posix-Fäden
  - Wettbewerb dauerte 3 Wochen (Abschlussarbeit)



Beschleunigung

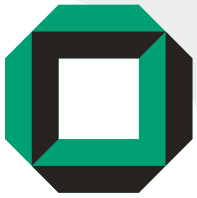


Gewinner erreichte 10-fachen Beschleunigung auf Sun Niagara T1 (acht Prozessoren, 32 HW-Fäden).



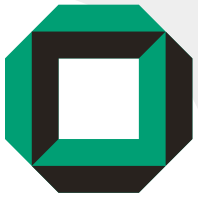
Was hat funktioniert?

- Massive Umstrukturierung war erforderlich
  - Teams, die hier wenig investierten, hatten keinen Erfolg.
  - Die Gewinner parallelisierten erst am Tag vor der Abgabe; die übrigen Zeit von 3 Wochen wurde nur dazu benutzt, das Programm für die Parallelisierung vorzubereiten.
  - Viele Abhängigkeiten, Seiteneffekte und Optimierungen für den sequentiellen Ablauf mussten beseitigt werden, bevor überhaupt Parallelisierung möglich war.



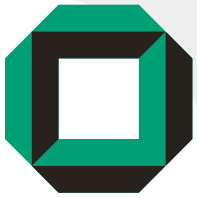
Was hat nicht funktioniert?

- Parallelisierungskonstrukte nach und nach hinzu zu fügen hat bei keinem Team funktioniert.
- Nur den Laufzeit-kritischen Pfad zu parallelisieren reichte nicht.
- Feingranulare Parallelisierung innerer Schleifen hat keine nennenswerte Beschleunigung erzielt.
  - Parallelisierte Einheiten müssen größere Arbeitsabschnitte umfassen.
- Algorithmen waren so speziell, dass geeignete, parallelisierte Bibliotheksroutinen kaum zu erwarten sind.



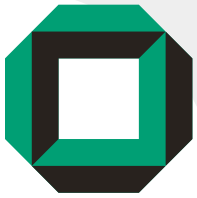
Parallelisierung ist keine Geheimkunst

- Arbeite nach Plan! Herumprobieren reicht nicht.
- Entwickle Hypothesen und Abschätzungen, wo Parallelisierung am meisten Gewinn bringen könnte!
- Berücksichtige mehrere Parallelisierungsebenen!
- Benutze Parallelisierungsmuster!
  - Auftraggeber/Auftragnehmer, Fließband, Gebietszerlegung, Erzeuger/Verbraucher.
- Verzweifle nicht bei der Restrukturierung!
- Ärmel hochkrempeln und anfangen!



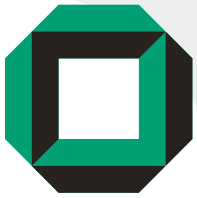
Wozu die Rechenleistung einsetzen?

- Intuitive Schnittstellen, mit Bild- und Sprachverarbeitung
- Vorausschauende Anwendungen, die „ahnen“, was der Benutzer will
- Eingehende Modellierung des Benutzers und der Umgebung
- Berechnungen, die heute zu langsam sind
- Erhöhung der Zuverlässigkeit



Forschungsthemen Parallele Softwaretechnik

- Bessere Programmiersprachen zum direkten Ausdruck paralleler Abläufe
- Übersetzer und Optimierer
- Ablaufplanung (Prozess/Prozessor)
- Parallele Entwurfsmuster und Architekturen
- Parallele Algorithmen und Bibliotheken
- Automatische Suche nach Wettlaufsituationen, Synchronisierungsfehler
- Autotuning, auto-Skalierung, Adaption
- Leistungsvorhersagen für parallele Architekturen
- Werkzeuge zur Restrukturierung und Parallelisierung
- Neue Anwendungsklassen
- Ihr Lieblingsthema/Technik/Expertise angewandt auf Multicore



XJava: Parallelitätsmuster kompakt ausgedrückt

Operator „=>“ verknüpft Prozesse in Fließbandstruktur, wie in Unix.

```
compress(File in, File out) {
  read(in)  =>  compress()  =>  write(out);
}
```

Liest Datei, gibt Folge von Blöcken aus

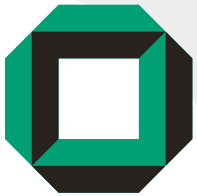
Datenstrom

Liest Blöcke, gibt kompr. Blöcke aus

Datenstrom

Schreibt Blöcke in Datei

Filter laufen parallel, bis Ende der Eingabe.
Geeignet für Fließbänder, Auftraggeber/Arbeiter, Erzeuger/Verbraucher.
Datenströme sind typisiert und typsicher.



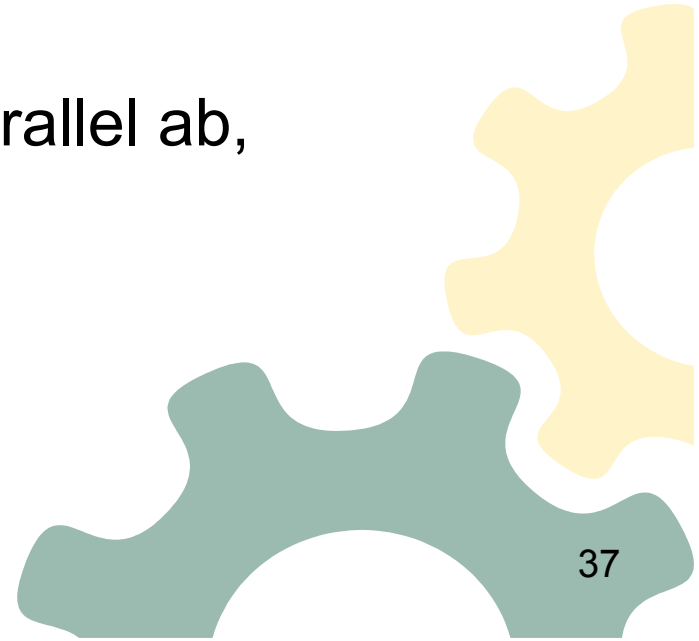
XJava

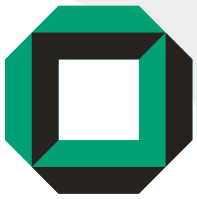
Operator „|||“ lässt Prozesse parallel laufen:

```
compress(f1, f1out) ||| compress(f2, f2out);
```

Methoden laufen in eigenen Fäden parallel ab, implizite Barriere am Schluss.

Für Prozess- und Datenparallelismus.





Auftraggeber/Arbeiter in XJava

Eingabetyp
=>
Ausgabetyp

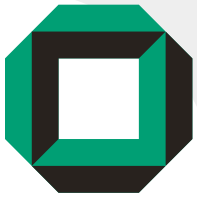
- Deklarationen für 1 Auftraggeber, 3 Arbeiter:

```
void => X master() { /* Auftraggeber*/ }
X => void w()      { /* Arbeiter*/ }
X => void gang()   { w() ||| w() ||| w(); }
```

- Deklaration für *i* Arbeiter (dynamisch):

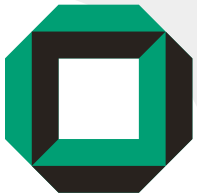
```
X => void gang()   { w():i; }
```

- **`master() => gang()`**
  `master` übergibt den `w` in `gang` im Umlaufverfahren Elemente vom Typ X.
- **`master() =>* gang()`**
  übergibt alle Elemente an alle Arbeiter.



XJava Erweiterungen

- Einfach zu verstehende Parallelität
- Voll integriert in Java
- Typsicher
- Einfacher zu handhaben als Fäden oder Bibliotheken
- Weniger Code, weniger Fehlerquellen
- Keine Synchronisationsfehler bei Übergabe von Daten über Puffer (Fließbänder)
- Vollautomatisches Autotuning möglich



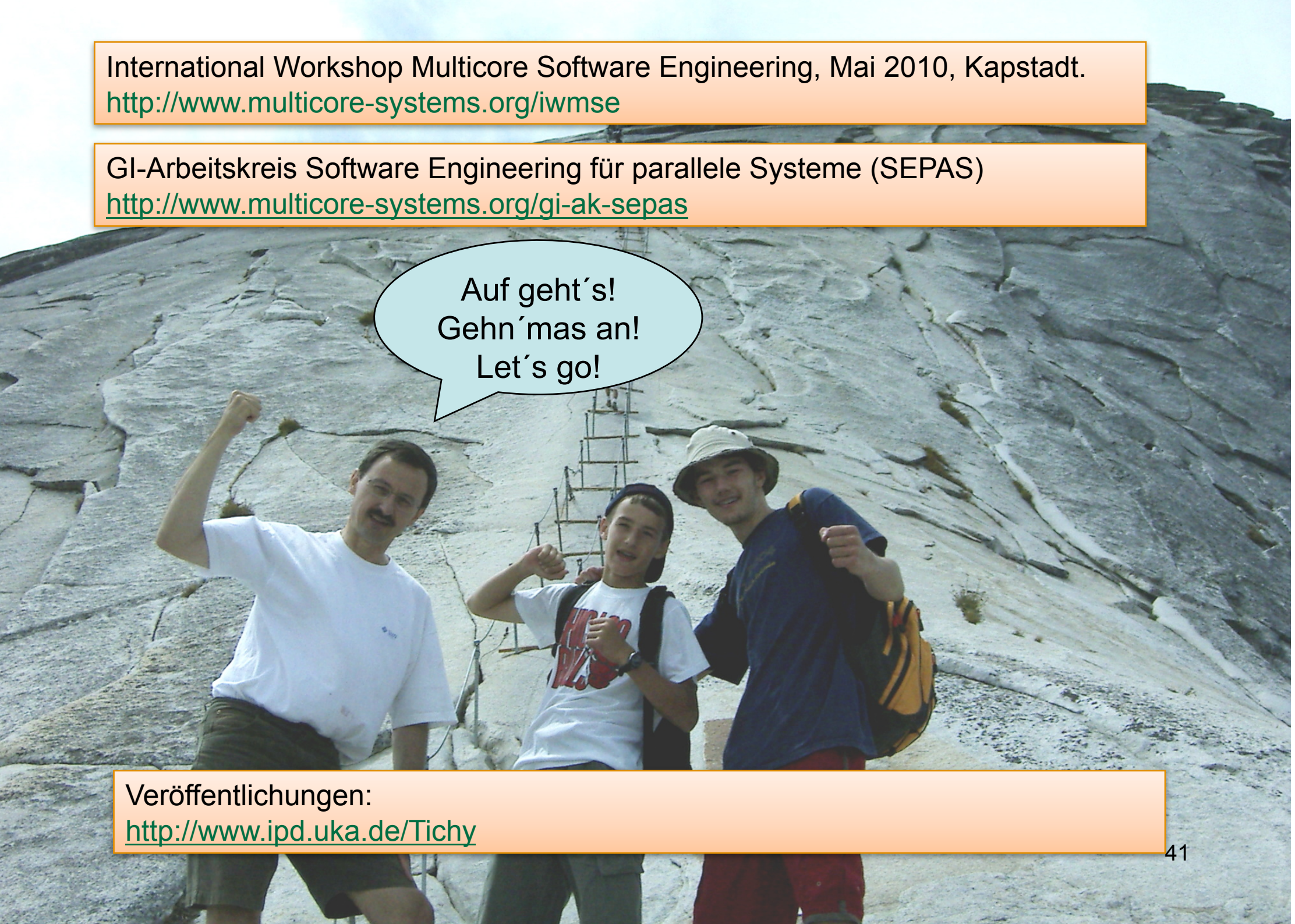
Schluss

- Zukünftige Leistungssteigerungen nur über Parallelität
- Ziel: Beschleunigung von Anwendungen bei gleichbleibender Produktivität, Qualität
- … während sich die Anzahl der Prozessorkerne pro Chip alle zwei Jahre verdoppelt.
- Viele der Grundlagen der Informatik müssen neu überdacht werden.

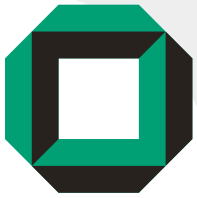
„Neuerfindung der Softwaretechnik“

International Workshop Multicore Software Engineering, Mai 2010, Kapstadt.
http://www.multicore-systems.org/iwmse

GI-Arbeitskreis Software Engineering für parallele Systeme (SEPAS)
http://www.multicore-systems.org/gi-ak-sepas



Veröffentlichungen:
http://www.ipd.uka.de/Tichy



Wie kann ich mich bereit machen?

- Java Programmierer: studiere java.util.concurrent
- C/C++-Programmierer: hole Pthreads und OpenMP
  - Aufgabe 1: implementiere Erzeuger/Verbraucher-Muster
  - Aufgabe 2: Parallelisiere Vektor-Vektor-Addition
  - Aufgabe 3: Parallelisiere Matrix-Vektor-Multiplikation; vergleiche Leistung bei 1, 2, 4, 8, 16, 32 Fäden.
  - Aufgabe 4: Parallelisiere numerische Integration
  - Wir lehren das bereits im 2. Semester.
  - Lösungen auf Anfrage erhältlich.
- Dann parallelisiere eine erste, kleine Anwendung.
  - Z.B. Google auf Ihrem Rechner (Indexerstellung)