3.– 6. September 2012
in Nürnberg

Herbstcampus

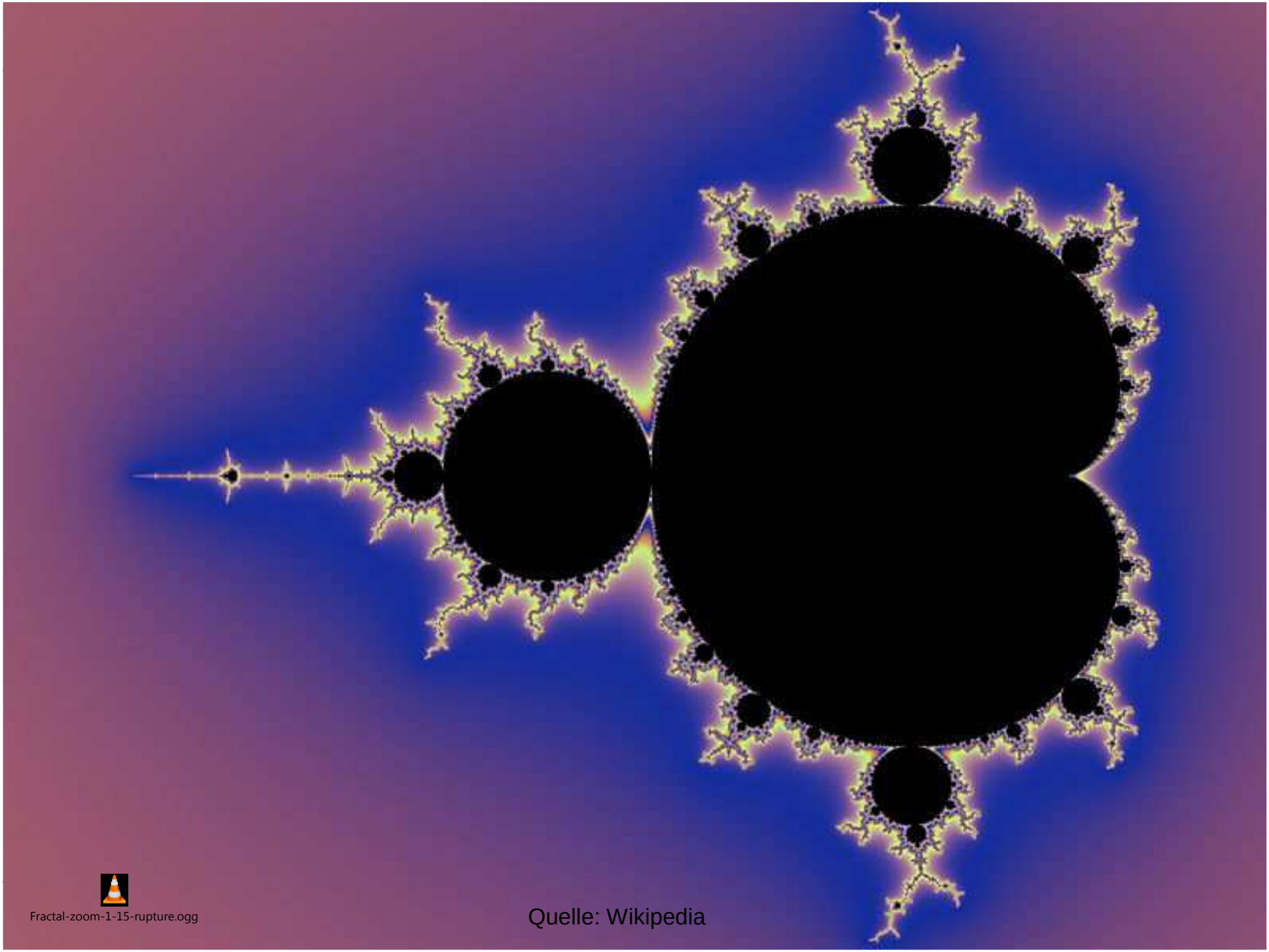
Wissenstransfer
par excellence

# Die Welt ist (leider) kein Apfelmännchen

Von der Parallelisierung realen Codes und warum wir es tun müssen

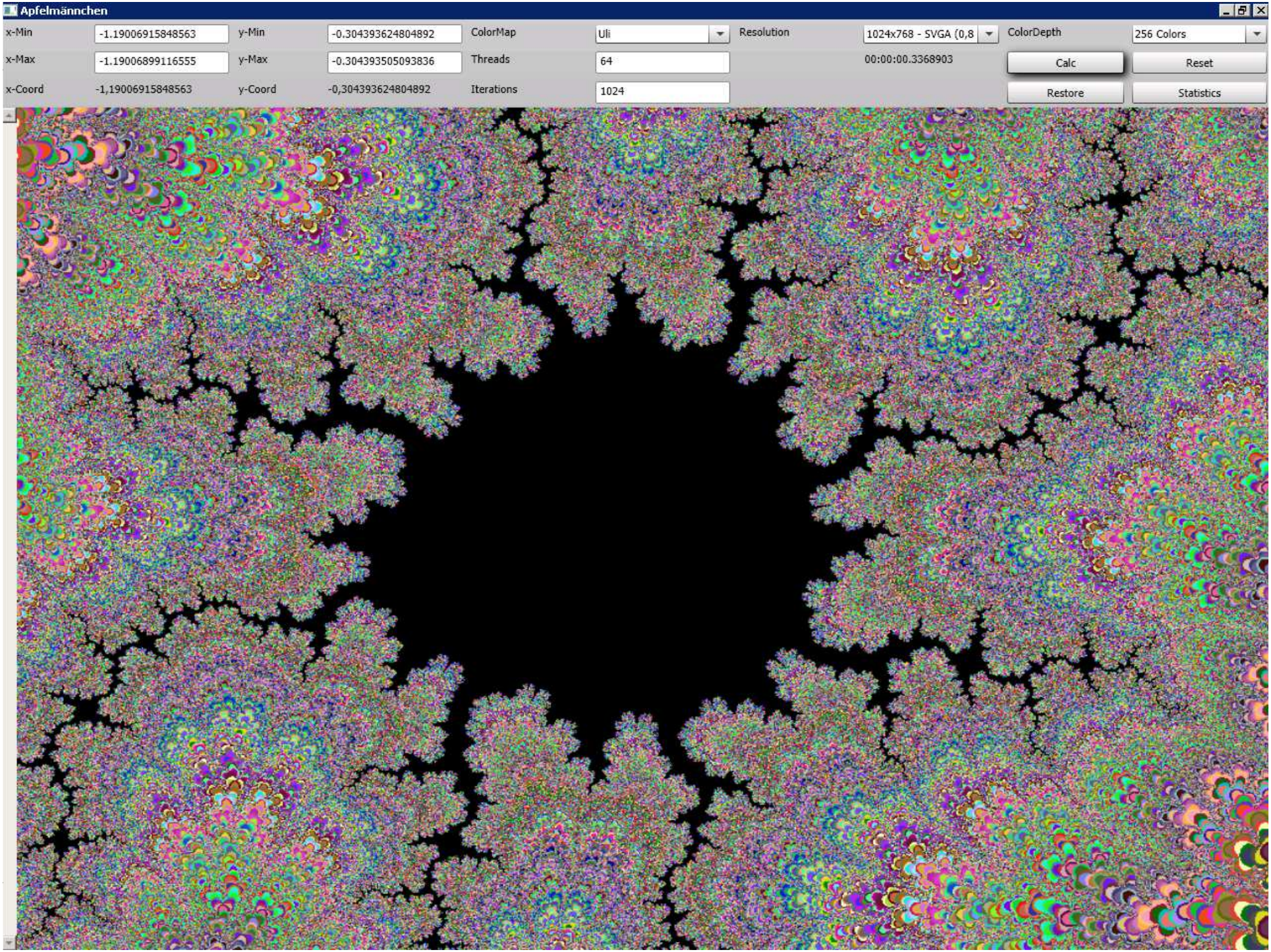

## Dr. Uli Hilburger

NÜRNBERGER Versicherungsgruppe



Fractal-zoom-1-15-rupture.ogg

Quelle: Wikipedia


Apfelmännchen
x-Min
-1.19006915848563
y-Min
-0.304393624804892
ColorMap
Uli
Resolution
1024x768 - SVGA (0,8
ColorDepth
256 Colors
x-Max
-1.19006899116555
y-Max
-0.304393505093836
Threads
64
00:00:00.3368903
Calc
Reset
x-Coord
-1,19006915848563
y-Coord
-0,304393624804892
Iterations
1024
Restore
Statistics

# Apfelmännchen interaktiv

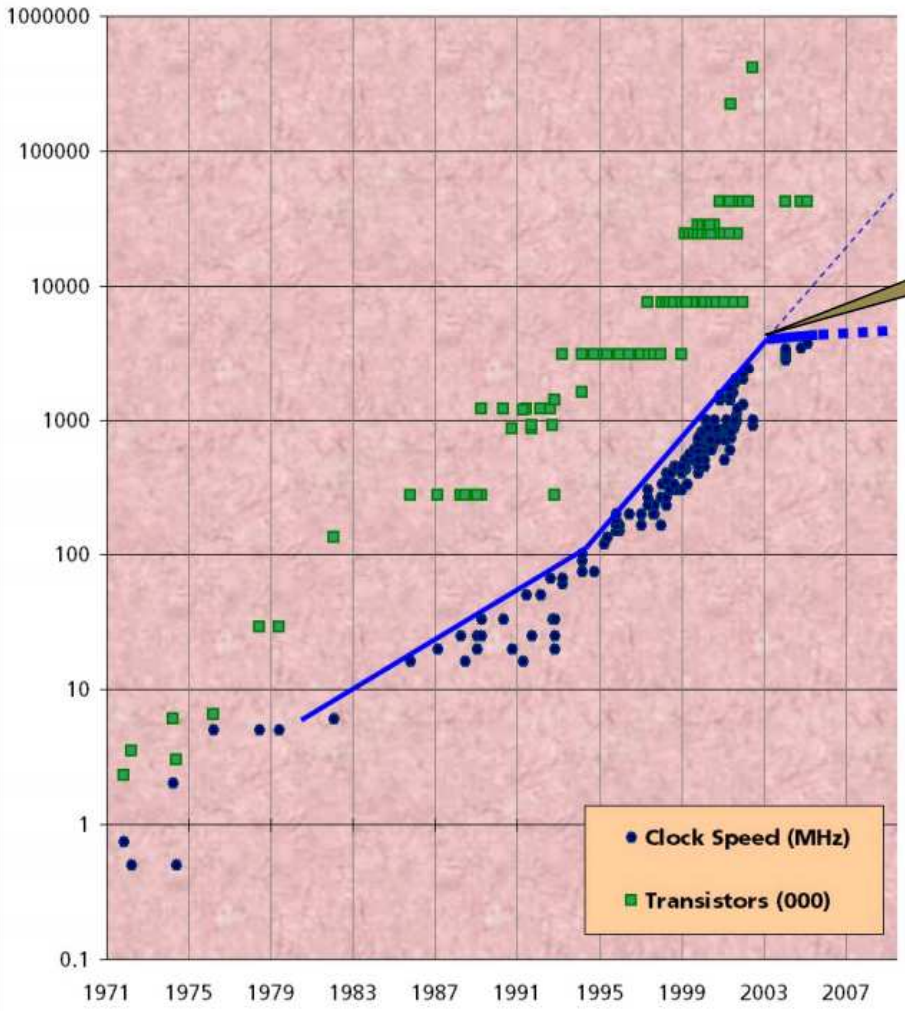
Demo

# Evolution meiner Apfelmännchenprogramme

- 1984 – C64 - 8-Bit-CPU – 1 MHz
- 1990 – PC-286 – 16-Bit-CPU – 12 MHz
- 1991 – PC-287 FPU math. Co-Prozessor
- 1994 – PC-486DX4 – 32-Bit-CPU – 100 MHz
- 2005 – PC-P4 – 64-Bit-CPU – 3000 MHz
- 2012 – Server-X5650 – 64-Bit-CPU – 2670 MHz (2x6 HT)

Fällt Ihnen was auf?
Ich habe keine 16 GHz–CPU sondern 6 Kerne à 2,67 GHz
Egal … ist doch das gleiche !?

# Taktrate von Intel CPUs

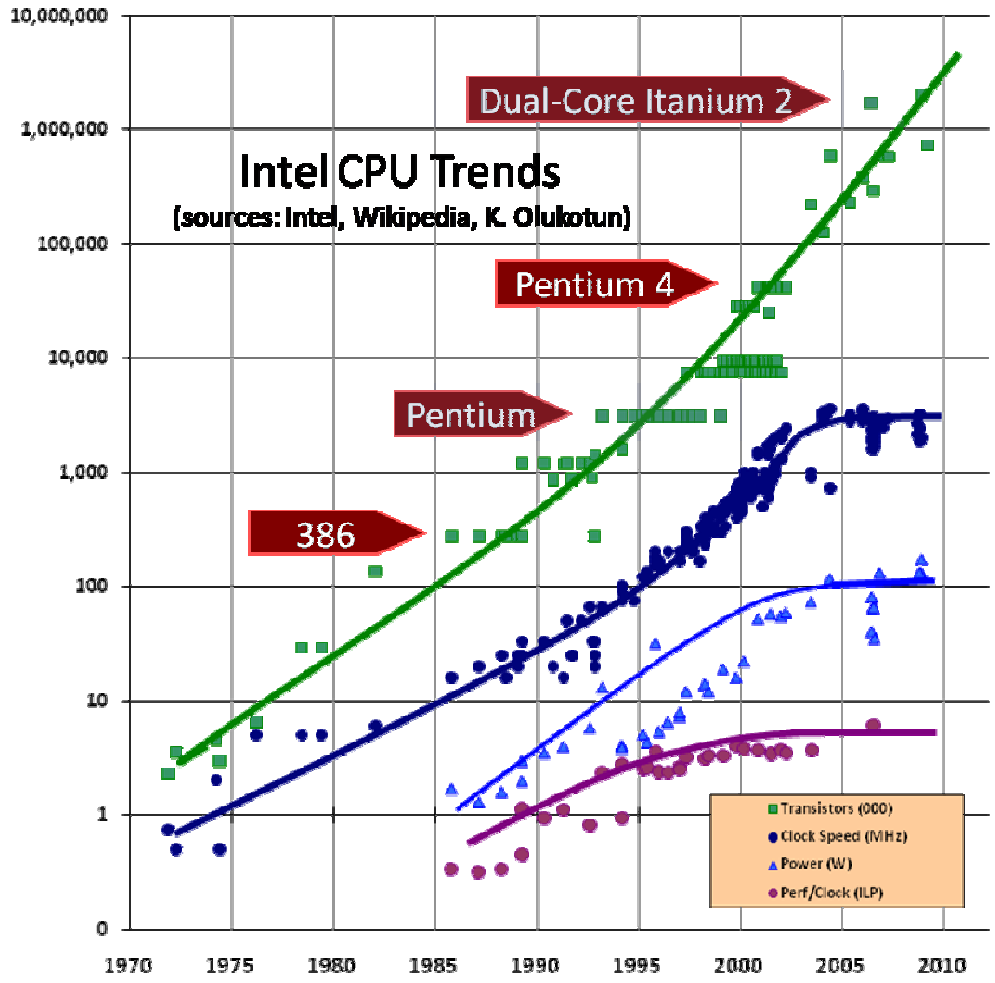



Only very slow linear increase CPU clock speed since 2003

Quelle: http://www.gotw.ca/publications/concurrency-ddj.htm

| Intel Pentium 4 Prozessor | |
|---|---|
| Fläche | 135 mm² = 1,35 cm² = 0,000135 m² |
| Leistungsaufnahme | 115 W bei 3.6 GHz |
| Leistungsdichte (W / m²) | 850 kW / m² |
| Entspricht einem schwarzen Strahler mit T = 1700 °C | |

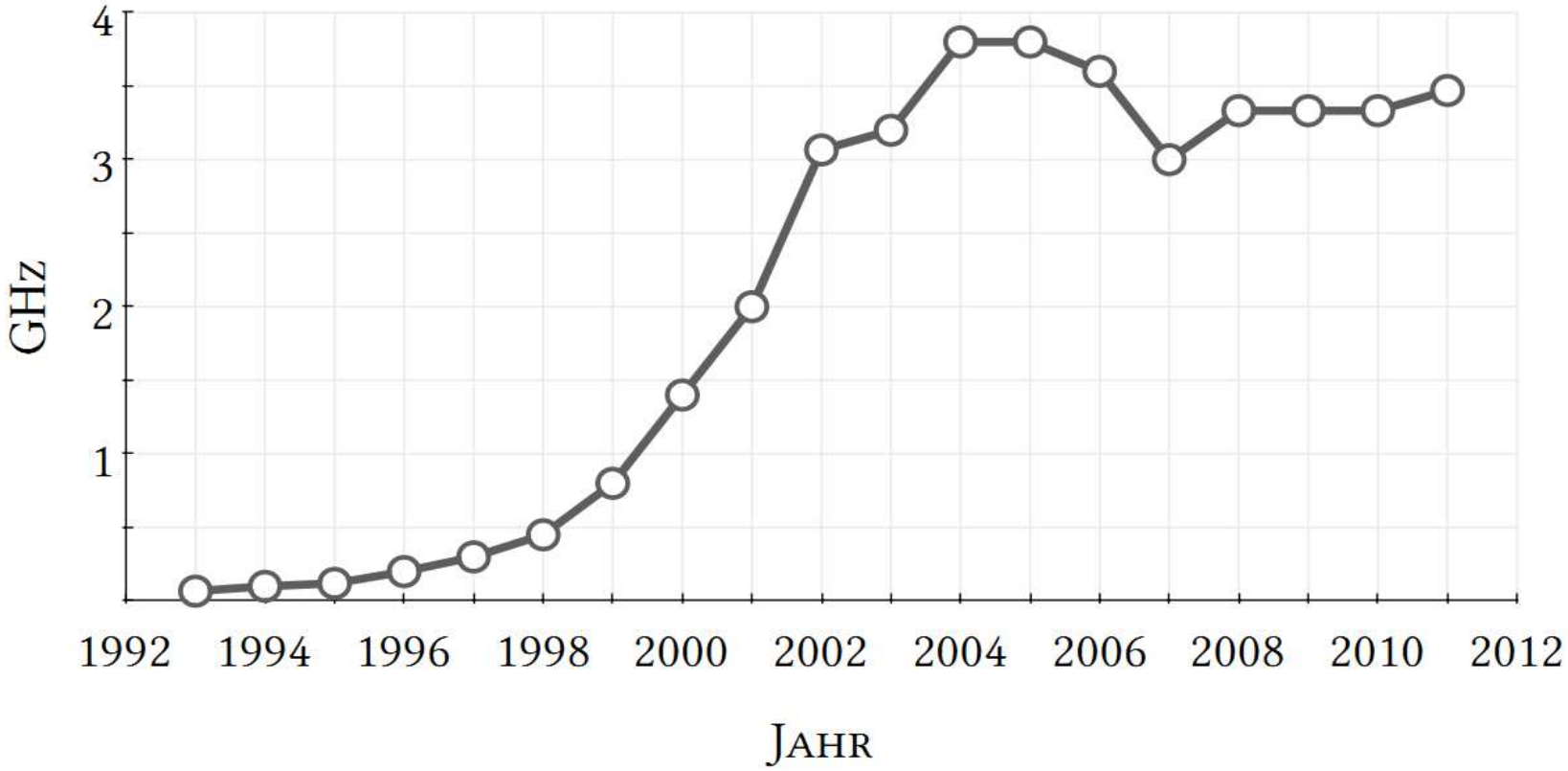
# Update






Quelle: http://www.gotw.ca/publications/concurrency-ddj.htm

# Update




Taktfrequenz der schnellsten Intel-Prozessoren

# Was ist mit den vielen Transistoren passiert?

- Hyper-Threading seit 2002 – P4 42 Mio.
- Dual-Core seit 2005 – Core 2 Duo 291 Mio.
- Quad-Core seit 2006 – Q6600 582 Mio.
- Octa-Core seit 2010 – Xeon 2.300 Mio.
- Deca-Core seit 2011 – Xeon 2.600 Mio.

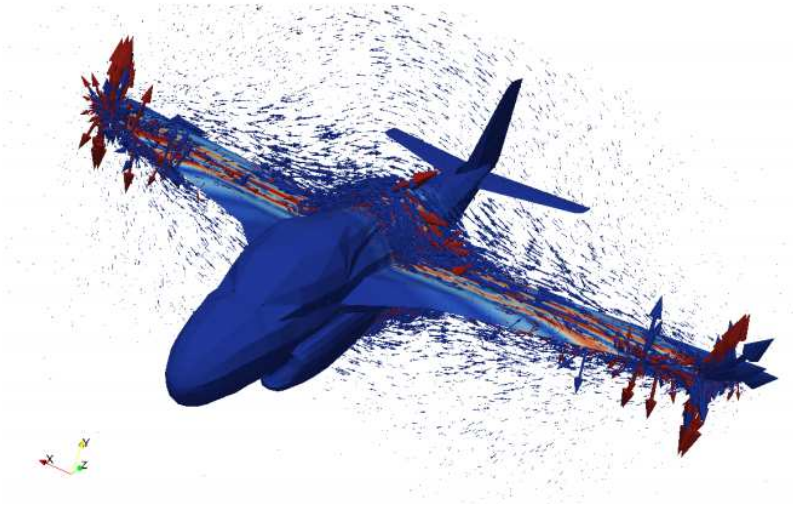
# (Fast) ein Transistor pro Erdbewohner

ATI Radeon HD 7950
Codename: Tahiti XT
Transistoren: 4312 Mio.
Herstellung: 0,028µm
Chiptakt: 800 MHz
Shadertakt: 800 MHz
Speichertakt: 5000 MHz (theoretisch)
Speicher: 3072 MB GDDR-5
Speicherinterface: 384 Bit
Streamprozessoren: 1792
ROPs: 32
TMUs: 112
Pixelfüllrate: 25600 MPixel/s
Texelfüllrate: 89600 MTexel/s
Shaderleistung: 2870.0 GFlops
Speicherbandbreite: 240000 MB/s
Strom (max): 200.0 Watt
Strom (idle): 22.0 Watt

**ATI Radeon HD 7950**


| Specifications | | | |
|---|---|---|---|
| Core Clock: | 800 MHz | Memory Clock: | 1250 MHz |
| Memory Bus: | 384 bit | Memory Size: | 3072 MB |
| Memory Type: | GDDR5 | Interface: | PCI-E |
| ROPs: | 32 | Released: | Jan 2012 |
| Shading Units: | 1792 | | |
| **Graphics Processor** | | | |
| GPU: | Tahiti | Process Size: | 28 nm |
| Transistors: | unknown | DirectX: | 11 |
| Shader Model: | 5 | | |

# CPU versus GPU



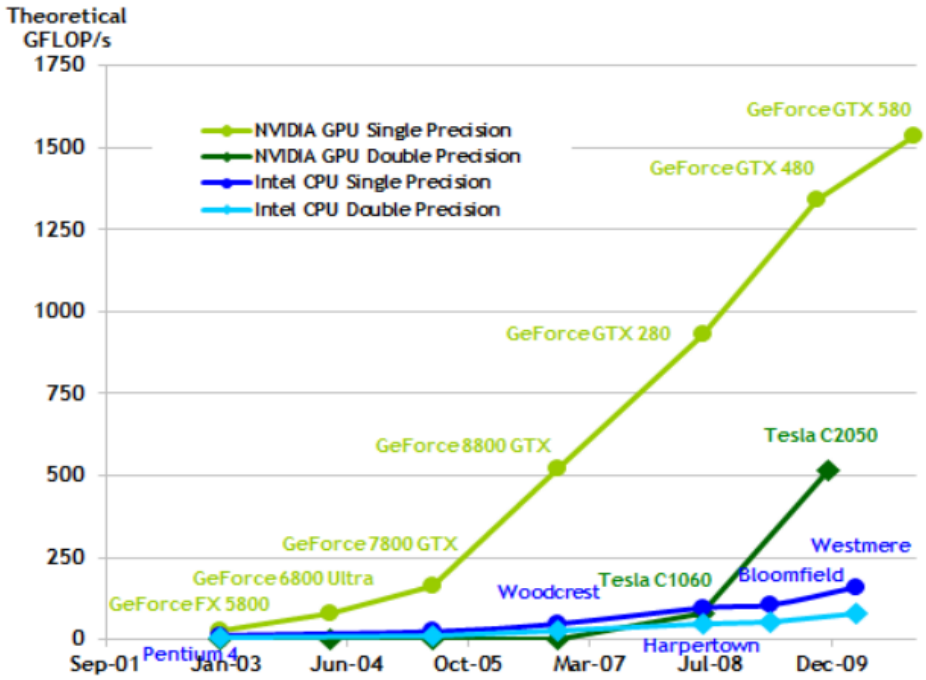

Grün: Recheneinheiten Gelb: Kontrolllogik Orange: Speicher

# CPU versus GPU



## FLOPS Vergleich: NVIDIA vs. Intel






[Quelle: NVIDIA CUDA C Programming Guide]

# Klassische Einsatzgebiete von Parallelrechnen

CERN-LHC-Auswertung von Kollisionsdaten
Kristallzüchtungssimulation (F&E)
Crashtest-Simulationen (KFZ)
Wettervorhersage
SETI(@Home)
Öl-Exploration
Schachprogramme
Biochemie (Wirkstoff-Simulation)
(Echtzeit-)Computertomographie


Spiele Vielkörper-Dynamik (z.B. Rauch-Partikel)
Monte-Carlo-Simulation (Risikoszenarien Finanzmathematik)
Raytracing (realitätsnahe Berechnung von Grafiken und Filmen)
Selbstkonsistente Berechnung von nicht-analytisch lösbaren Gleichungen

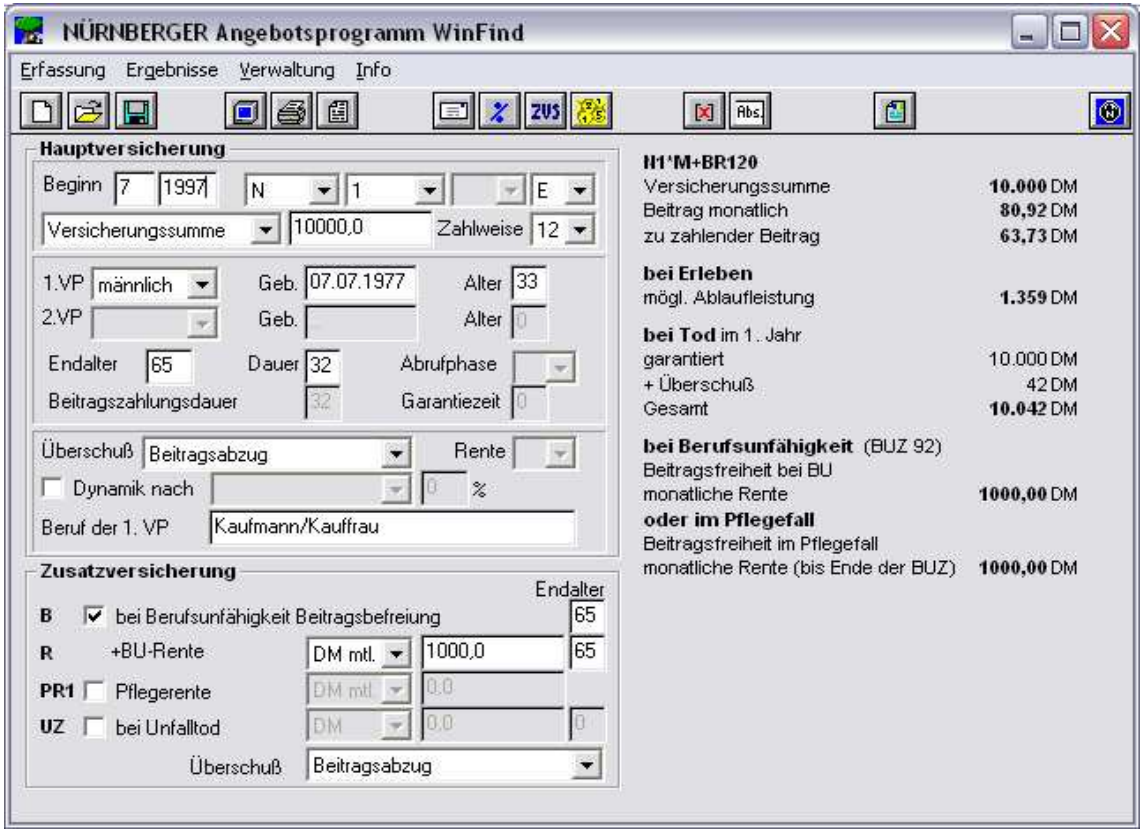
# Und was hat ein Apfelmännchen …

mit einer Versicherung zu tun ?

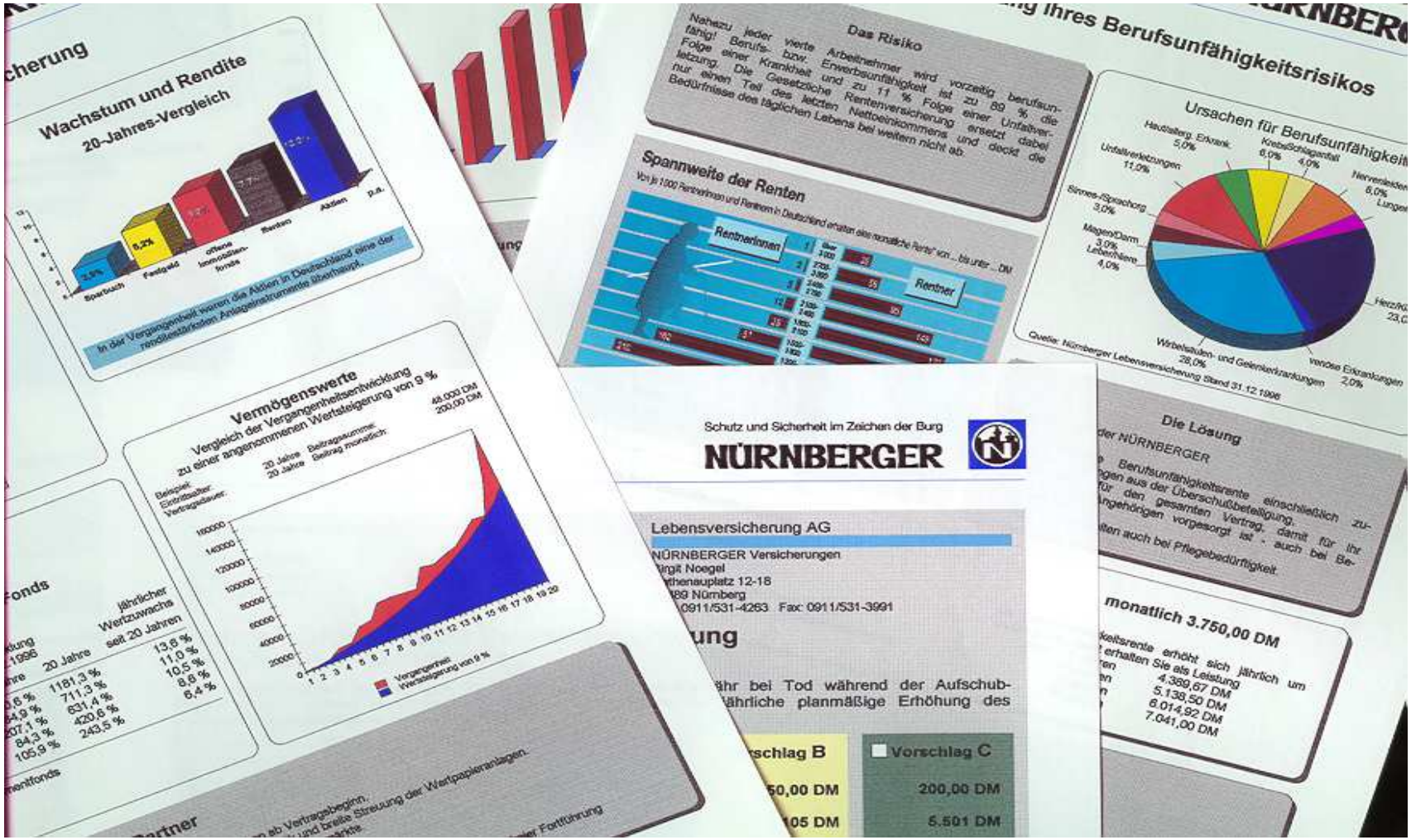
NÜRNBERGER
Beratungstechnologie

# 1981 – der erste Computer im Außendienst

# 1997 – das erste Windows-Programm

# 1997 – Drucken in Prospektqualität



# Wofür wird die Rechenleistung heute genutzt?

- Grafische Oberfläche
- Höhere Auflösung und Farbtiefe
- Spielereien wie Animationen und Übergänge
- Auto-Validierung / -Korrektur
- Tool-Tips + erweiterte Hilfe
- Interaktive Grafiken und Diagramme
- … → Verbesserte „user-experience“
- …
- **Genauere und variantenreichere Rechnung**

# Berechnungsvarianten Lebensversicherung

- Schwierigkeitsgrad steigend:
  - Summenvorgabe (Versicherungssumme) → feste Formel
  - Beitragsvorgabe → Iteration mit Anfangssumme + Formel
  - Ablaufleistungsvorgabe (Garantiesumme + Überschüsse)
- Das kommt erschwerend hinzu:
  - Berechnung mit jährlicher Dynamikerhöhung
  - Überschüsse als Fondsanlage (monatlicher Kauf von Anteilen)
  - Überschüsse als Fonds-Mix (bis zu 10 Wahl-Fonds im Depot)
- Weil es immer noch besser geht:
  - Berechnung diverser Szenarien (Wertentwicklung, Inflation)
  - Hochrechnung Rentenbezugsphase bis 100 mit Überschüssen
  - Steueroptimierung (z.B. Mindesttodesfallsumme über Laufzeit)

# Unser neues Problem mit den Berechnungen

- Kundenerwartung steigt:
  - Interaktivität bei der Berechnung (Schieberegler)
  - Variantenreichtum
  - Hochrechnungen und Prognosen
  - Was-wäre-wenn-Vergleiche
- (Einzel)Prozessor-Geschwindigkeit sinkt:
  - Smartphones/Pads 500 MHz – 1.5 GHz
  - Ultra-/Netbooks 1 – 2 GHz
  - Notebooks/PCs 1.5 – 3.5 GHz
  - Server 2 – 2.5 GHz

# Und da war noch das Kleingedruckte …

- Erzeugung von Druckstücken:
  - Kurzangebote mit 2-3 Seiten
  - Personalisiert + Verlaufsprognosen/Kleingedrucktem bis 100 S.
- Zeitvorgaben:
  - Einzelplatzanwender am PC sollen nicht lange warten müssen
  - SLA-Forderungen von Partnern & Vergleichern für Serverfarm
- Problematisch:
  - Viele Einzeldruckstücke, oft personalisiert / dynamisch berechnet
  - Gemeinsames Inhaltsverzeichnis notwendig
  - Drittanbieter-Software für PDF-Erstellung / -Bearbeitung

# Zusätzliche Herausforderung

- Extrem unterschiedliche (z.T. „unbekannte“) Hardware:
  - Netbook des Maklers: Single-Core-Atom-Prozessor mit 1 GHz
  - Klassischer Heim-PC: Dual-Core-AMD  mit 2 GHz
  - Neuer „Aldi“-PC: Quad-Core mit 2.5 GHz
  - Klassischer Server: Quad-Core-Xeon mit 2 GHz
  - Neuer Server: Deca-Core-Xeon mit 2.4 GHz
  - Die physikalische Hardware im RZ wird zyklisch ausgetauscht, unsere Software läuft auf virtuellen Serverumgebungen.

# Free lunch is over!

Klassische Software wird nicht mehr automatisch mit neuen Prozessoren schneller ☹

Wie können Multi-Prozessoren sinnvoll genutzt werden?

→ Go ( & think) parallel !

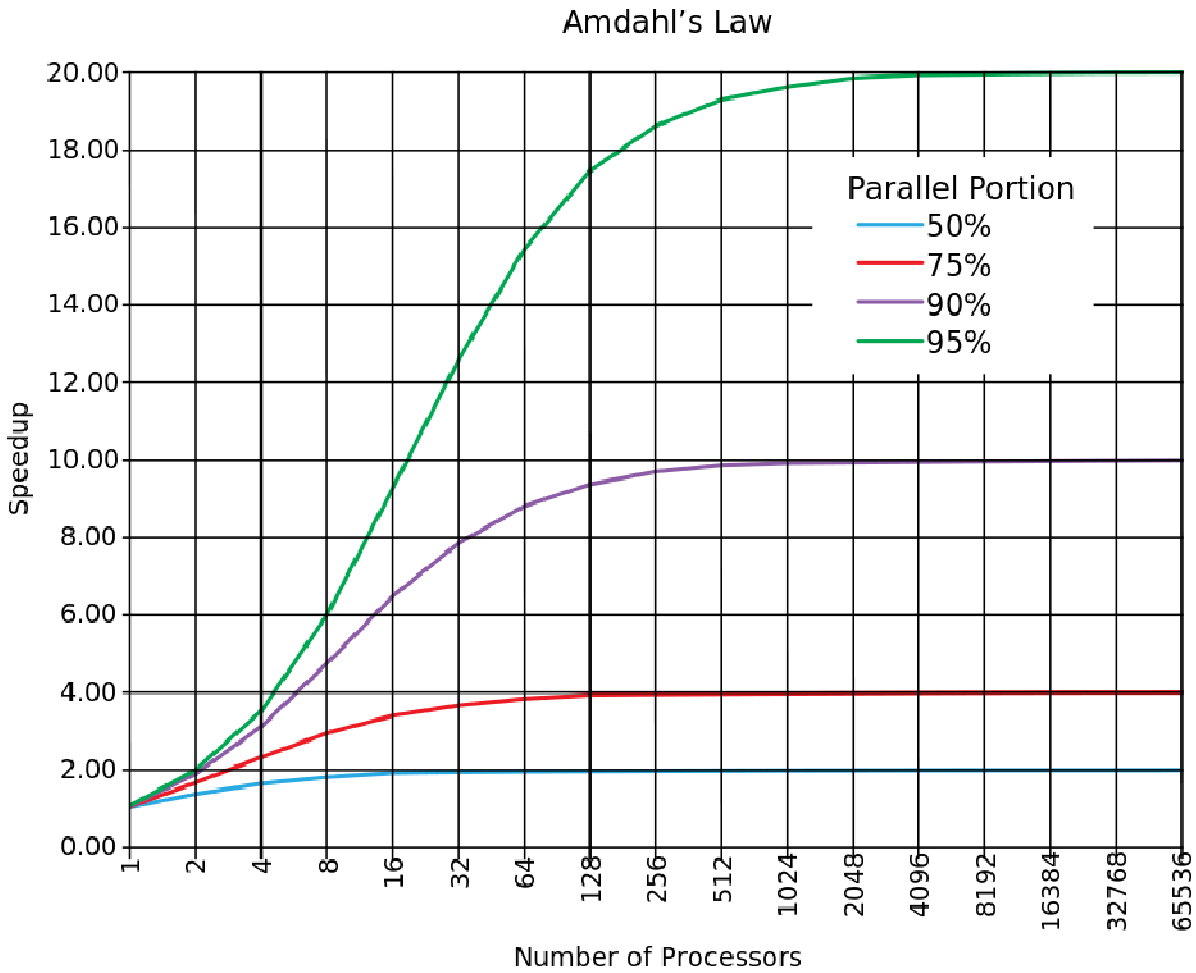
# Was heißt Parallelisieren?

Demo

# Der heilige Gral der Parallelisierung

- “Automatic parallelization of a sequential program by a compiler is the holy grail of parallel computing. Despite decades of work by compiler researchers, automatic parallelization has had only limited success.”
- Gesucht: Retter in der Not
- Optionen:
  - Hardwarehersteller
  - Compilerhersteller
  - Toolkithersteller

# Die Grenzen der Parallelisierung



# (Bisherige) Optimierungen der Chiphersteller

- Vergrößerte Registerbreite 8Bit, 16Bit …128 Bit
- Spezial-Instruktionen: FPU-Multiplikation, -Division
- (SIMD)Spezialregister/-Befehle MMX / SSE / AVX
- Mehrere Ausführungseinheiten (Hyperthreading)
- Parallele Befehls-Pipelines (superskalare Architektur)
- Instruction prefetch & branch-prediction
- Out-of-Order-Execution (Reorganisation)
- Cache, Cache, Cache (multi-level)
- Dynamisches Übertakten (Turbo Boost etc.)

# Leerlauf und einzelner Kern unter Last


# Compiler sei Dank, es wird langsam besser!

- Hier werden sie (bereits) geholfen:
  - Automatische Code-/Daten-Ausrichtung (durch JIT-Compiler)
  - Threadsichere Concurrent-Bibliotheken
  - Threadpool (Verwaltung)
  - Parallel-Loop & Co (umgebungsabhängige Optimierung)
  - Yield (Synchronisierung von Ergebnissen)
  - Async

# Viele kleine Helferlein …

- Bibliotheken, Toolkits und IDEs:
  - Task Parallel Library (Sync-Konstrukte)
  - OpenMP (**M**essage **P**assing **I**nterface)
  - Intel Threading Building Blocks TBB (job-stealing)
  - Intel Cilk Plus
  - Intel Array Building Blocks
  - CUDA & OpenCL Unterstützung für Portierung auf GPU
  - Analyse+Hinweise auf Parallelisierungsshowstopper
  - Code-Refactoring-Vorschläge

# Noch fehlende Optimierungen

- Hardware:
  - Atomic-Flag für alle Befehle
  - Transaktionaler Speicher
  - Automatische (cache-optimierte) Verteilung auf CPU-Threads
- Software:
  - Automatische Aufteilung auf CPU / GPU
  - Dynamische (JIT-code-rewriting) Anpassung an Problem

# Herausforderungen für die nächste Dekade

- Think parallel - Entwickler müssen …
  - … Aufgaben in parallelisierbare Häppchen zerlegen
  - … Nebenläufigkeitsprobleme erkennen lernen
  - … Multithread-Debugging erlernen
  - … Performance-Analyse und –Optimierung betreiben
  - … Unit-/Lasttests unter Thread-Safety-Aspekten durchführen
  - … neue Sprachen lernen (F# etc.) und mischen

# Herausforderungen für die nächste Dekade

- Parallelisierung des gewachsenen C++-Codes
  - … Modularisierung des verflochtenen Monolithen
  - … Pointer-Manipulationen durch Funktionsaufrufe ersetzen
  - … Performance durch Entflechtung von Schleifen erhöhen
  - … Thread-Sicherheit gewährleisten
  - … Portierung auf OpenCL & Co

# Unsere Lösungsansätze

- Berechnung:
  - Lokale Schleifensummen
  - Thread-lokale Zwischenobjekte
  - Parallele Iterations-Näherung mit unterschiedlichen Startwerten
  - Merken der Ergebnisse von teuren geteilten Zwischenschritten

# Unsere Lösungsansätze

- Druckstückerzeugung:
  - Parallelisierung in Teildokumenterzeugung
  - Caching von statischen bzw. unpersonalisierten Dokumenten
  - Mehrfaches Starten der 3rd-Party-Software
  - Korrektur der Seitennummerierung nach dem Zusammenfügen
  - (Verworfen: PDF-Komprimierung)

# Was man (heute leider noch) falsch machen kann

- Variablen zwischen Threads ungesichert teilen (raceconditions, deadlocks)
- Exzessive Locks verwenden (Serialisierung / deadlock) → mittelgranulare Locks / reader-writer-lock
- „Globale Variablen“ in Schleifen verwenden (Sync-Overhead) → threadlokale Variablen + „yield“
- Minitasks, innere/geschachtelte Schleifen parallelisieren (Overhead) → erst bei > 1..10 ms je Sequenz sinnvoll (bei hohem I/O-Grad x 10)

# Was man (heute leider noch) falsch machen kann

- Last ungleich verteilen (Kern des Apfelmännchens)
  → (dynamische) Analyse / „job-stealing“
- Daten falsch partitionieren (cache-invalidation)
  → Block-Aufteilung statt interlaced
  [0-31][32-63] vs. [0,2,4…][1,3,5…]
- I/O-limitierte Prozesse parallelisieren
  („Festplattenrattern“ → Analyse)
- RAM-limitierte Prozesse parallelisieren
  („paging/swapping“→ Analyse)

# Gute technische Kochrezepte

- Threadsichere Bibliotheken verwenden
  (z.B. concurrent list)
- Double locks, semaphore-Muster etc. verwenden
  (deadlocks und raceconditions verhindern)
- Möglichst nur äußere Schleifen parallelisieren
  (overhead ↓ )
- Threadpools etc. verwenden (Recycling ist billiger)
- Compilerbasiertes scheduling benutzen
  (dynamic threadcount, autopartitioning, workstealing)
  → wird i.d.R. weiter gepflegt und verbessert

# Gute fachliche Kochrezepte

- (Workflow-/Daten-)Unabhängige Teilprobleme parallelisieren (Apfelmännchen-Pixel, Schach-Halbzüge)
- Asynchrones Arbeiten mit Sync-Punkten, Teilschritte in workerqueues packen
- Fehlersignalisierung prüfen & Schleifenabbrüche testen
- Prozess genau analysieren und globale Flaschenhälse vermeiden (Konkurrenz um rare „Ressourcen“)
- Performance-Monitoring (CPU,RAM,I/O) in der Testphase – wo bleibt meine Zeit?
- Serielle Fallback-Version bei hoher Last im Serverbetrieb

# Und wie immer gilt …

HW-/SW-Unterstützung hinkt immer um einige Jahre hinterher

Gegen Spaghetti-Code hilft kein „wizard“ dieser Welt

Der Ideenreichtum des menschlichen Gehirns

ist durch keine Maschine zu ersetzen

Programmierer lernen lebenslang

Go ahead and think parallel

3.– 6. September 2012
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

# Vielen Dank!


## Dr. Uli Hilburger

NÜRNBERGER Versicherungsgruppe