Chip Level Multithreading

Jens Peter Lindemann

AG Neurobiologie
Universität Bielefeld

10. Januar 2006

Warum CMT?

Intel: Hyperthreading

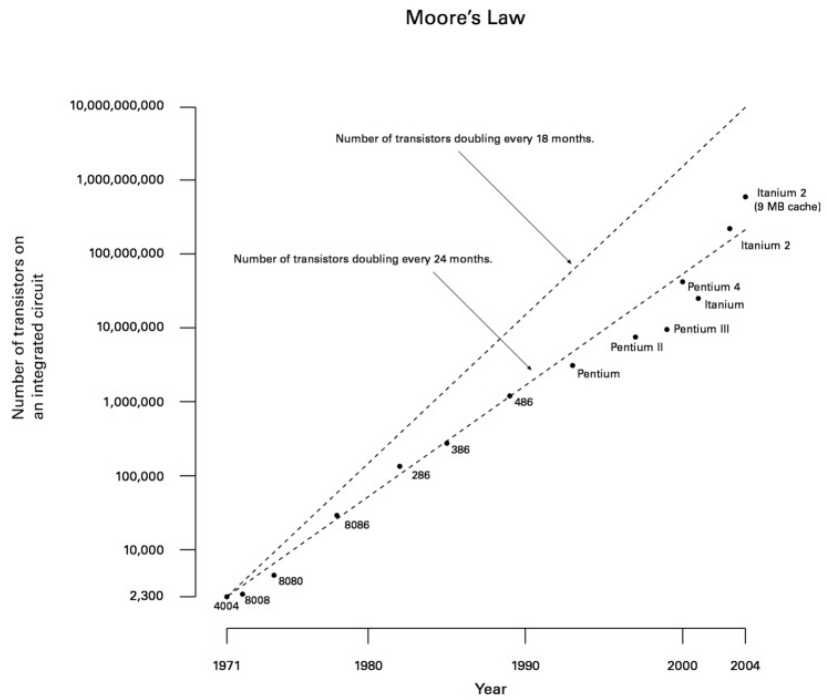
IBM: Power5

Sun: Niagara

Software

Moores Gesetz

The complexity for minimum component costs has increased at a rate of roughly a factor of two per year ... [Electronics Magazine 19 April 1965]



Was tun mit den vielen schnellen Transistoren?

Taktfrequenz und Pipeline-Länge

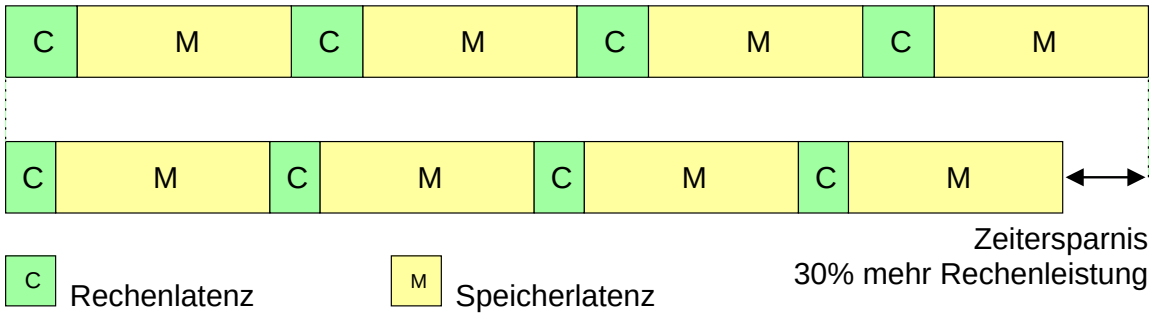
Verlängerung der Pipeline (feinere Unterteilung):

- $\Rightarrow$ Weniger Zeitaufwand je Stufe
- $\Rightarrow$ Höhere Taktrate
- aber auch höherer Aufwand
  - Out-of-Order execution
  - Branch prediciton
- und “Verluste” bei Pipeline-Flush
  - Datenabhängigkeiten
  - Branch mispredicitons

$\Rightarrow$ Mehr als 20-30 Stufen machen keinen Sinn.

Memory Gap

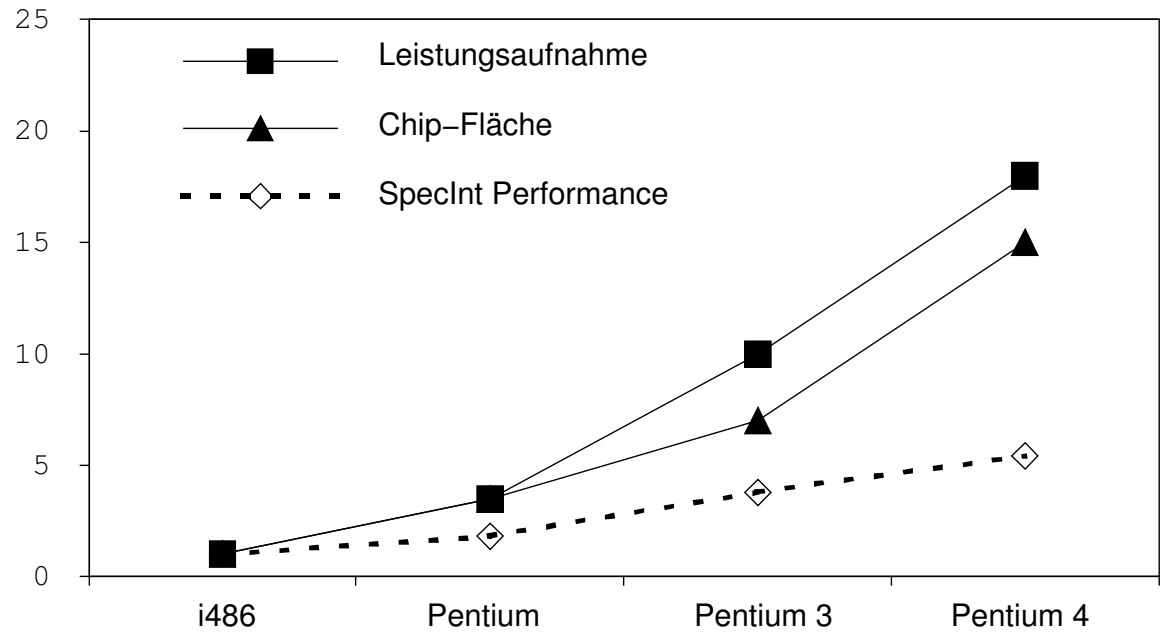
- Hohe Taktraten intern (L1-Cache), aber
- Hohe Latenz (CPU-Zyklen) bei Speicherzugriff



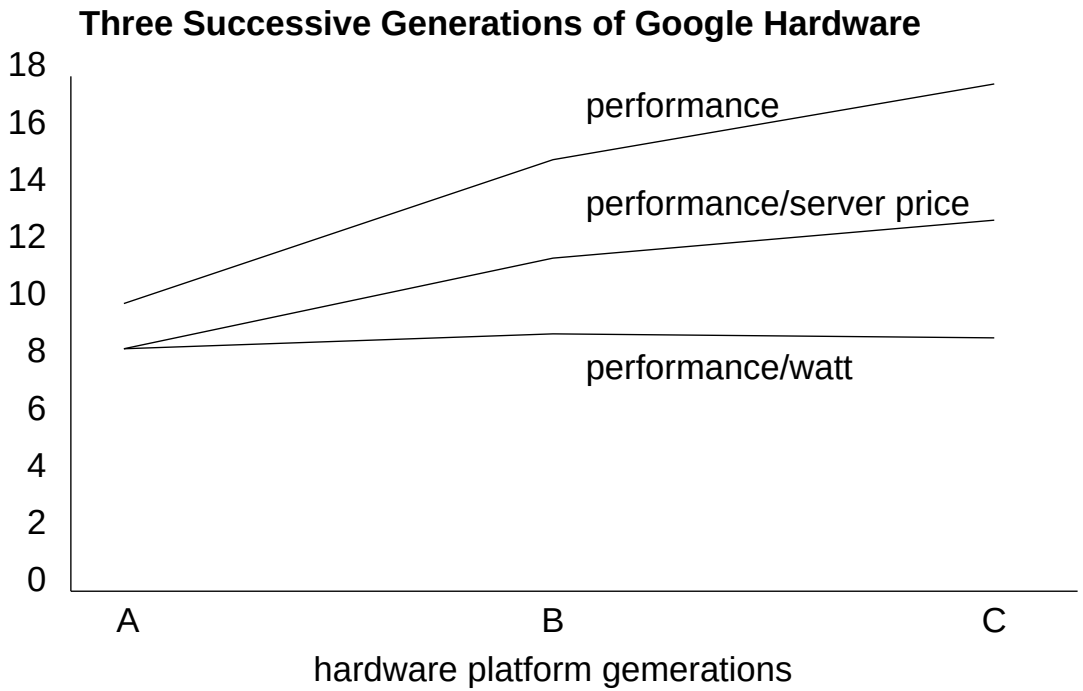
Moderne Prozessoren warten oft auf den Speicher - aber das sehr schnell...

(Verlust-)Leistungssteigerung

- Verdopplung der Taktfrequenz: 8facher Stromverbrauch
- Typischer Highend-Prozessor: >100 Watt elektr. Leistung



Beispiel: Google



⇒ Extrem steigender Kühlaufwand

⇒ Stromrechnung *echter* Kostenfaktor für Serverfarmen

- 4 Jahre Laufzeit:
  Stromkosten = 40% der Hardwarekosten

Warum also CMT?

- Komplexitätssteigerung: Grosser Aufwand, kleine Wirkung
  - Superskalarität: begrenzte Instruktionsparallelität
  - Pipeline-Länge ausgereizt
  - noch grössere Caches?
- Erhöhung der Taktraten: Kühlprobleme

Neuere Prozessoren werden kaum noch schneller im Single-Thread.
“The free lunch is over”

Vertical Multithreading

- Software Multithreading
  - Thread-Wechsel als Software-Entscheidung
  - Sicherung der Registerinhalte

  $\Rightarrow$ Hohe Latenz bei Threadwechsel
- Coarse Grained Hardware Multithreading
  - Event-gesteuerted Thread-Wechsel: z.B. L2 Cache-Miss
  - Kosten entsprechend Context-Switch
- Fine Grained Hardware Multithreading
  - Thread-Wechsel im Prozessortakt
  - Bindung aller Prozessor-Resourcen an aktiven Thread

Simultaneous Multithreading

Multithreading in Hardware

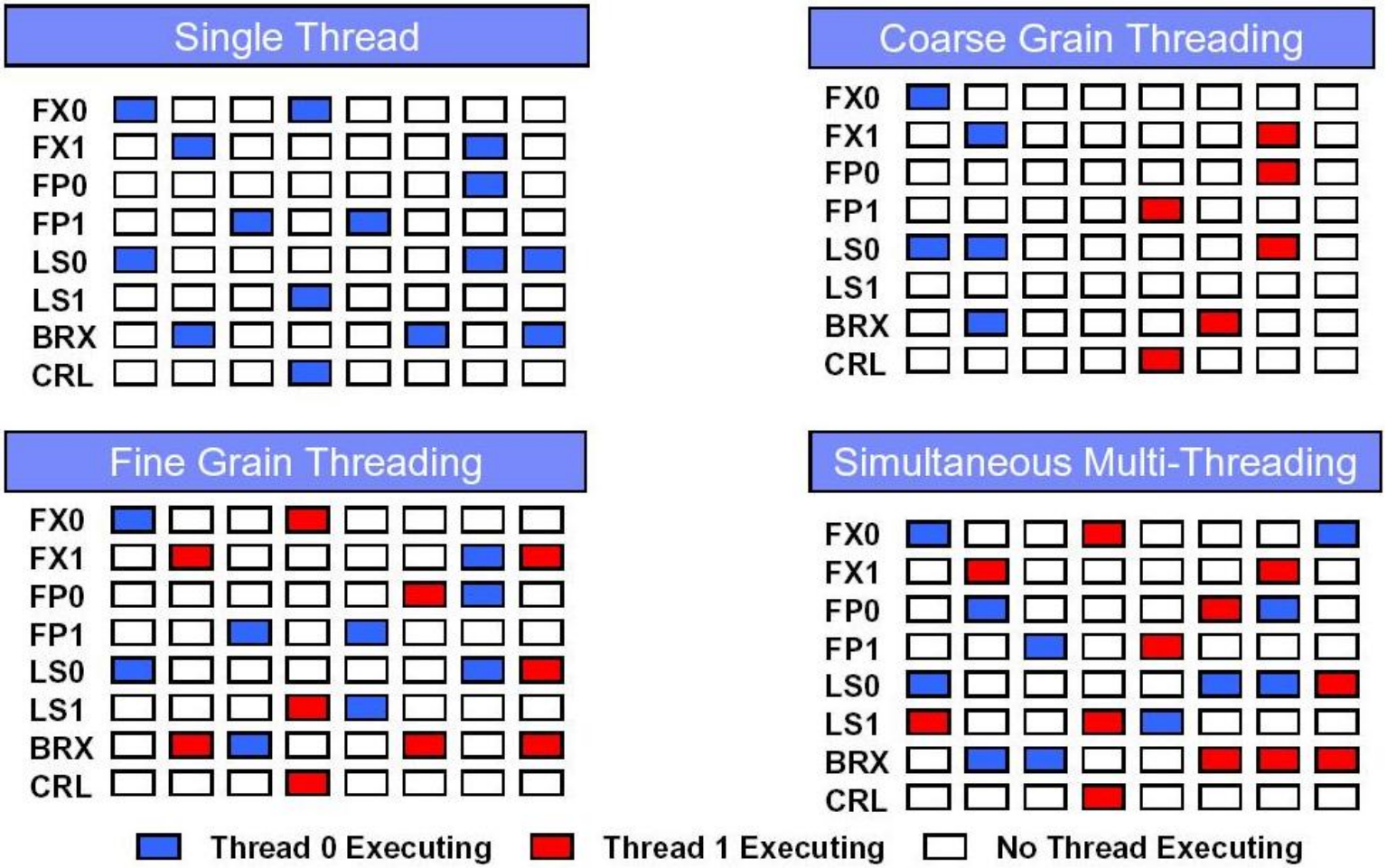
- Ein eigenes Registerfile je Thread

$\Rightarrow$ Keine Latenz bei Threadwechsel

- Thread-Scheduling: Round-Robin und Hardware Events
- Parallelisierung durch Verteilung auf Execution Units

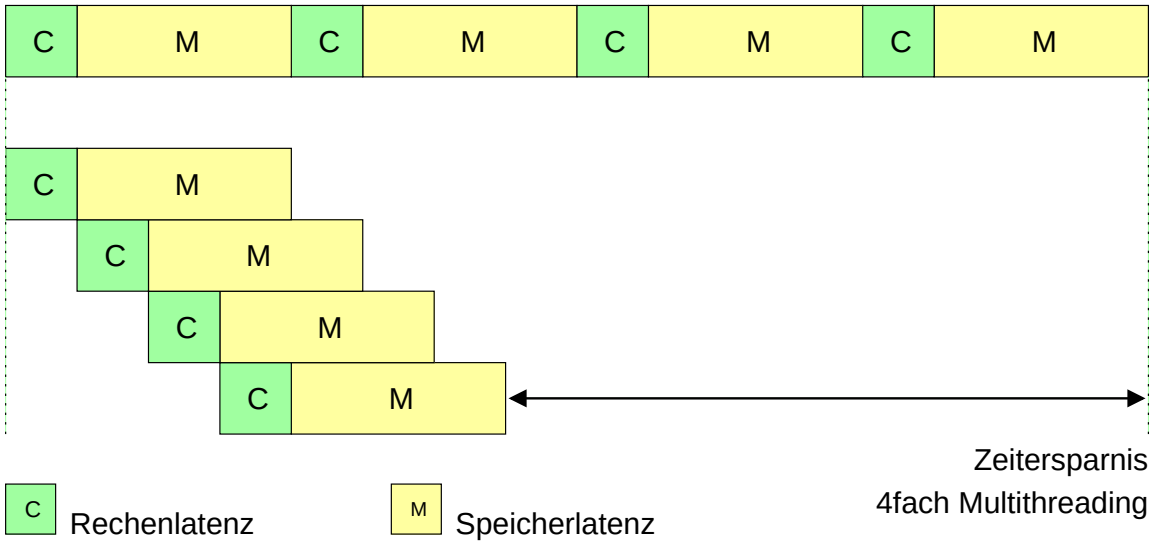
$\Rightarrow$ Teilweise gleichzeitige Bearbeitung der Threads

Multithreading Evolution



Performancegewinn furch SMT

Vorraussetzung: Multithreaded (Server-)Applikation



- Zeitersparnis durch bessere Auslastung der Chip-Resourcen
- Performance-Gewinn bei wenig Mehraufwand

Hyperthreading (Pentium 4, Xeon)

Implementation von SMT: 2 logische Prozessoren

- Verdopplung des “architectural state”
- shared Execution Units
- shared L1/L2 Cache
- 5% mehr Chipfläche
- 15-30% mehr Performance (laut Intel)

Replay System

Replay System als Teil des Singlethread P4:

- P4 Pipline: 31 Stages (Prescott)
- Optimistische Instruktions-Scheduler
- Detektion von erfloglosen Instruktionen (Datenabhängigkeiten!)
- Stop des Schedulers und Replay bis zum Erfolg

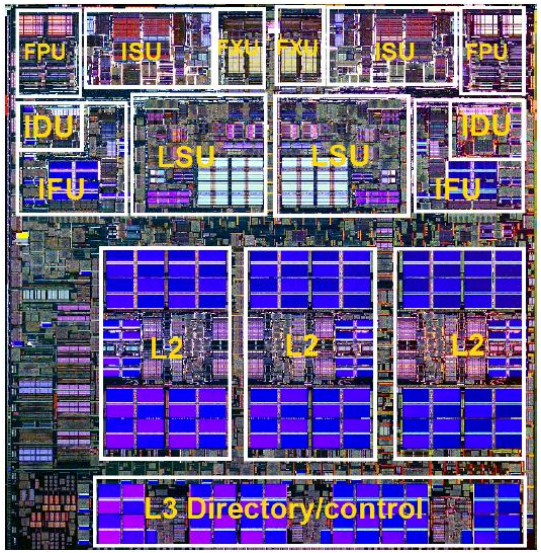
Replay und Hyperthreading:

- Replay System belegt Instruction Units zu 100%

$\Rightarrow$ Ausbremsen des anderen Threads

Power5

- Dual-Core Prozessor
- Superscalar
- Simultaneous Multithreading
  - 2 virtuelle Prozessoren je Core
  - 24% Chipfläche pro Core für SMT



SMT in Superskalarem Prozessor

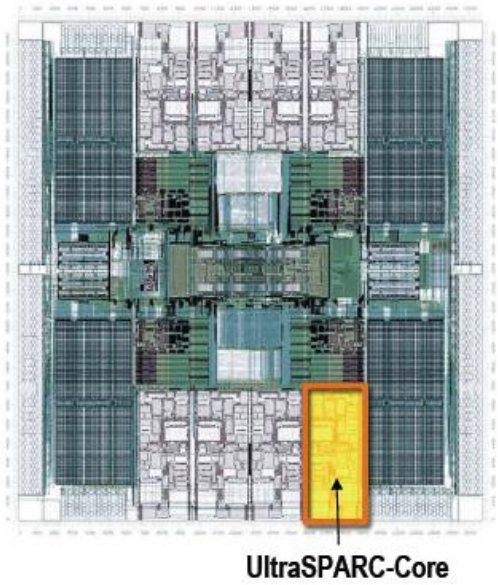
- Zweiter Program Counter (PC)
- Erweiterung des Register-Renamings für zweites Registerfile: Higher-Oder Bit “adressiert” Thread
- Überwachungslogik verdoppelt für zweitem Thread
- Thread Bit als Erweiterung in internen Buses

Single Thread Operation

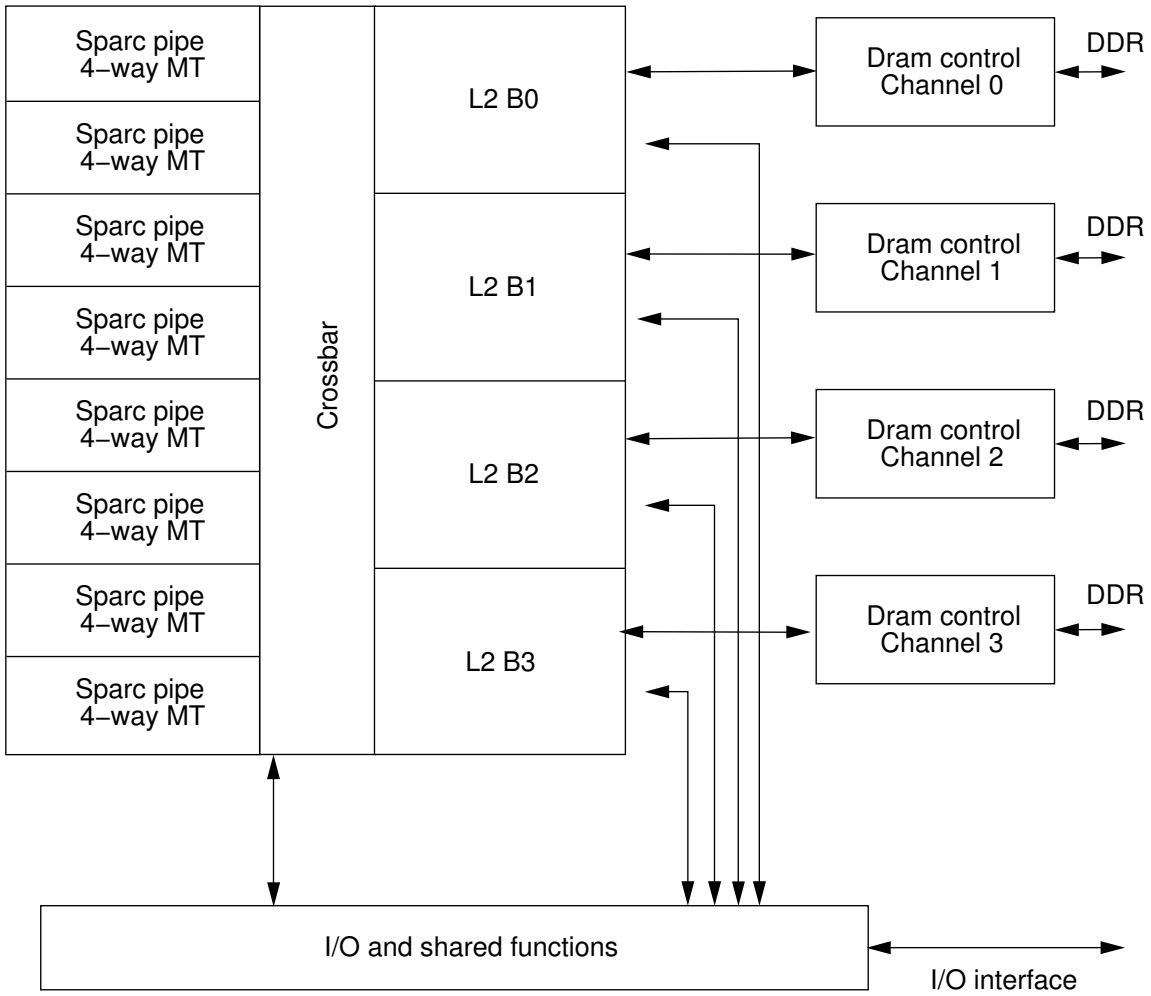
- Vorteilhaft bei Limitierung durch Execution Units
  - Rechenintensive Anwendung: Floating/Fixed Point
- Limitierung durch Execution Units bedeutet wenig Gewinn durch SMT
  - Zusätzliche Resourcen für SMT bringen zusätzliche Performance im Single-Thread
- Dynamische Detektion je Prozessor

Niaraga: UltraSparc T1

- 8 Kerne (UltraSparc IIi, 1GHz)
- 4 Threads je Kern
- Shared Level 2 Cache (4 Bänke, 3MByte)
- Crossbar mit 200GByte/s zur Inter-Thread Kommunikation
- 20GByte/s DDR2 Speicherinterface



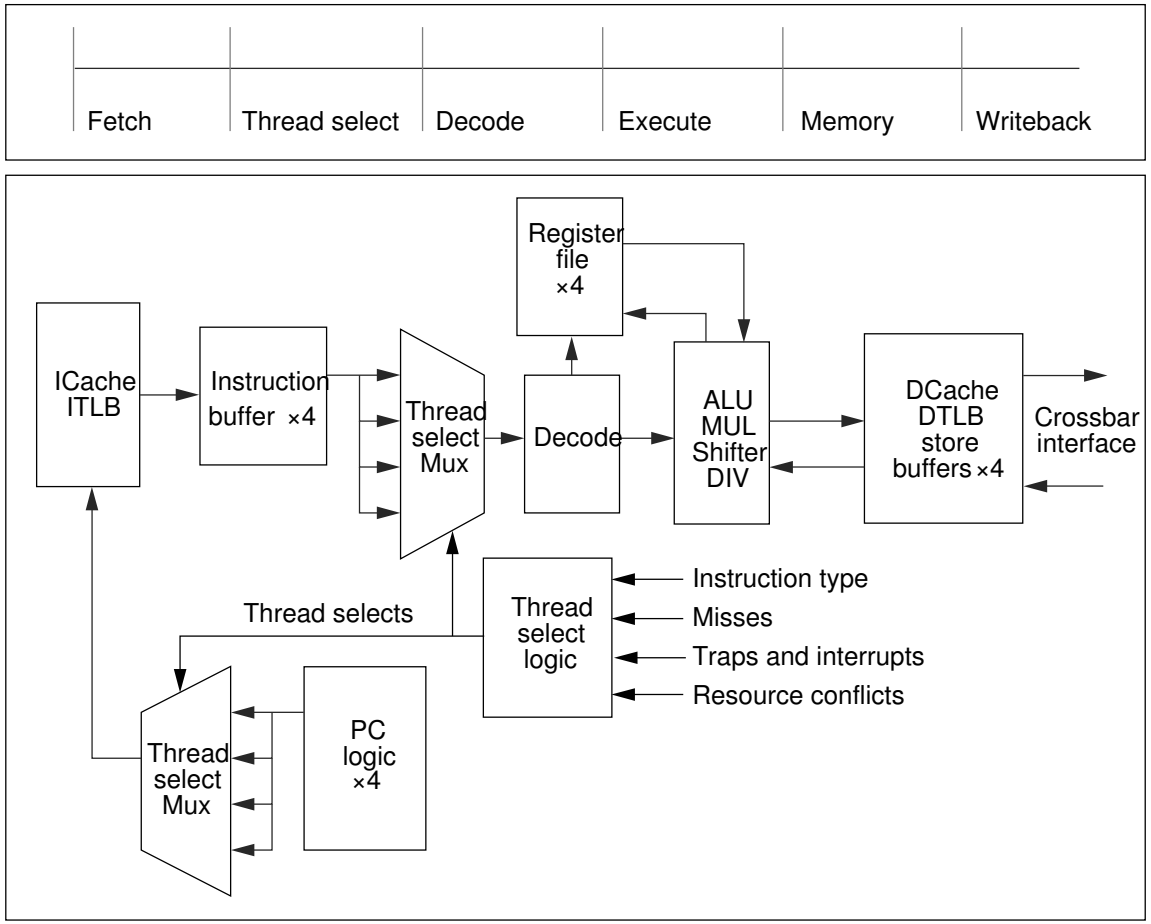
Niagara: Blockschema



Sparc Pipeline mit 4 Threads

- Threads teilen sich L1 Cache, TLBs, Execution Units
- Thread-Select im Prozessor-Takt
  - Round Robin: Wechsel je Takt
  - Thread wird übergangen bei Memory Stall, Verzweigungen, MUL/DIV

Skizze Sparc-Pipeline



Software-Sicht auf SMT-Prozessoren

Vereinfacht: SMT-Systeme sind SMP Systeme

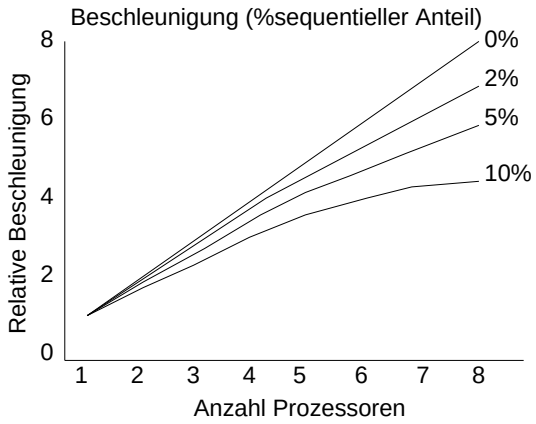
- Jeder Thread wird ein logischer Prozessor
  - Pentium mit Hyperthreading: 2 Prozessoren
  - Power5: 4 Prozessoren
  - Niagara: 32 Prozessoren
- zwischen den Threads geteilte Resourcen
  - Konflikte
  - Synergien

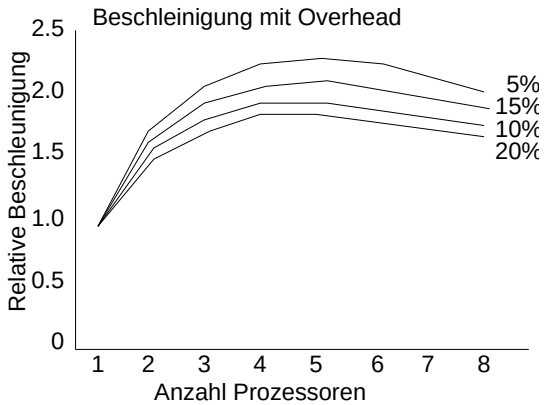
Amdahls Gesetz

Erreichbare Beschleunigung durch Parallelisierung

$$\frac{1}{F + (1 - F)N}$$

- F Anteil des sequentiellen Codes
- N Anzahl Prozessoren





Concurrency Revolution

Performancegewinn durch CMP/CMT setzt vorraus:

- Multithreaded Applikation
- Gut skalierende Locking-Strategien
- Gut skalierendes Betriebssystem

Aus Sicht der Software: CMP/CMT sind SMP-Systeme
...aber jetzt auch auf dem Desktop...

Locking

- Per-Objekt Locks
- Deadlocks
- Zentrale Objekte erzwingen Sequentialisierung
- Writer-Locks, Read-Zugriff ohne Locking
- Vorsicht: Nicht zu viele Locks!

Cache

Gemeinsam genutzter Cache durch mehrere Threads:

- konstruktive Zugriffe
- gegenseitige Behinderung

Niagara (Multithreaded Multicore)

- Writeback macht eventuell L1-Caches auf anderen Cores ungültig

$\Rightarrow$ Threads mit ähnlichen Zugriffsmustern auf einen Core

Falle Speicherverwaltung

- zentrale Speicherverwaltung
  - Sequentialisierung bei alloc/dealloc
- wenige Speicherverwaltungen skalieren gut
  - Hoard (www.hoard.org)
  - libumem (Solaris 10)
  - SmartHeap (MicroQuill)
- Unterschiedliche Eigenschaften je nach Zugriffsmuster

Software Lizenzen

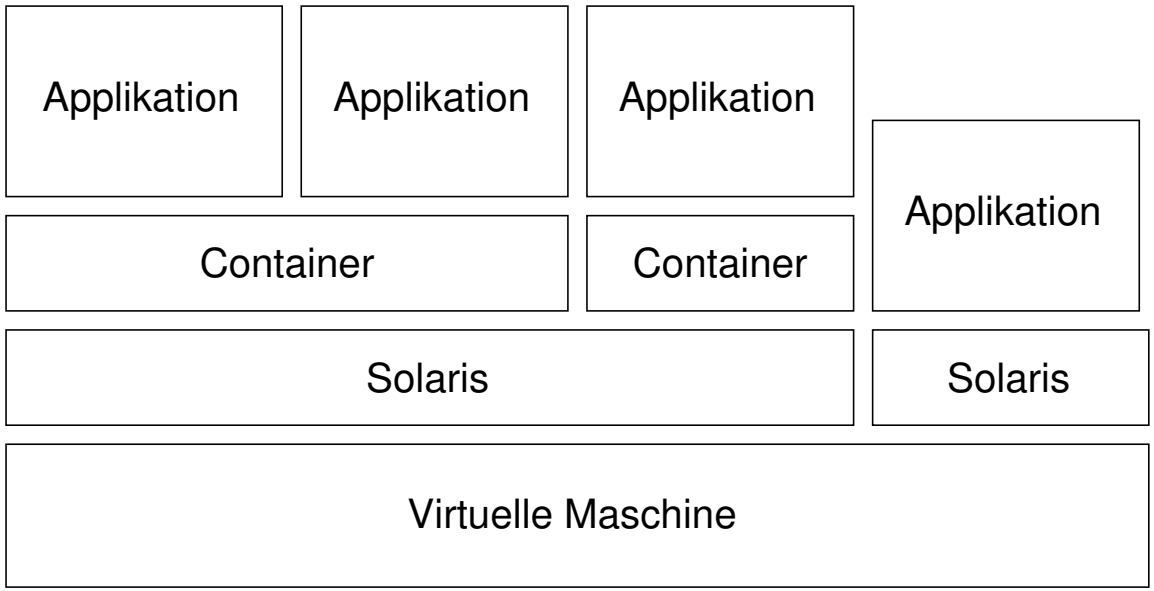
- Bisher: Ein Prozessor, eine Lizenz
- Zukunft?
  - Ein Thread, eine Lizenz?
  - Ein Kern, eine Lizenz?
  - Ein Chip, eine Lizenz?
    - Ankündigungen: Microsoft, IBM
  - Neue Leistungs-“Metrik” im Betriebssystem?

Betriebssystem-Optimierung

- “Busy wait loops”: Idle thread bremst eventuell andere aus
- Scheduler: Optimierte Zuweisung der Threads (Cache!)
- Kernel Skalierbarbkeit
  - Locking-Strategie
  - Parallele I/O-Zugriffe
  - Netzwerk
  - Speicherverwaltung
- Kernel-Statistiken
  - Update zentraler Staistiken: Sequentieller Code!
  - Realitätsferne Werte: HT Pentium mit einem Thread 50% Load?

Virtualisierung

- Mehrere nicht skalierbare Applikationen
- Mehrere nicht skalierbare Kernel-Instanzen



Sicherheit: Cache footprints

Gemeinsam genutzter Cache:

- Messung von Speicherlatenzen zwischen zwei Threads
  - gemeinsam zugängliche Daten (konstruktive Cache-Nutzung)
  - getrennte Daten (page-out messbar)

  $\Rightarrow$ Spy Prozess kann Speichererhalten des anderen Threads ausspähen
- Theoretisch nutzbare Informatonsübertragung zwischen Threads

Cache-footprints 2

- Exploit: OpenSSL Schlüssel-Klau
  - Latenzmessung im L1 Cache
  - Unterschiedliche Bearbeitungsdauer BN_mul und BN_sqr

  $\Rightarrow$ Ausreichend um den Schlüssel zu knacken
- Auswege
  - Hardware: Split der Data-Caches
  - Betriebssystem: Nur “gleichberechtigte” Threads auf einen Kern
  - Applikation: Datenabhängige Zugriffsmuster vermeiden

Zum Weiterlesen...

- Wikipedia (EN)
- ACM Queue

  http://www.acmqueue.com/
- Herb Sutter: The Free Lunch is Over

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

  http://ogun.stanford.edu/~kunle/publications/niagra_micro.pdf