

Wolfgang E. Nagel  
Center for Information Services and High Performance Computing (ZIH)

# Rechnerarchitektur II

Vorlesung 4: Aufbau und Entwicklung von Prozessoren



# Prozessoren

Vorige Vorlesung:

- Pipelining
- Superskalarität
- SIMD
- MIMD (Multicore)
- x86 (CISC)

Diese Vorlesung:

- Very Long Instruction Word
- Multi Threading
- Cores, Module, Dies und Packages
- Beispielhafte Prozessoren (RISC)

# Performancekennzahlen von Prozessoren

## Anmerkung

- Komplexes Thema
- Kann hier nicht kurz UND ausführlich behandelt werden
  
- Bei Interesse
- Leistungsanalyse von Rechnersystemen im WS19

# Typische Performance-Kenngrößen

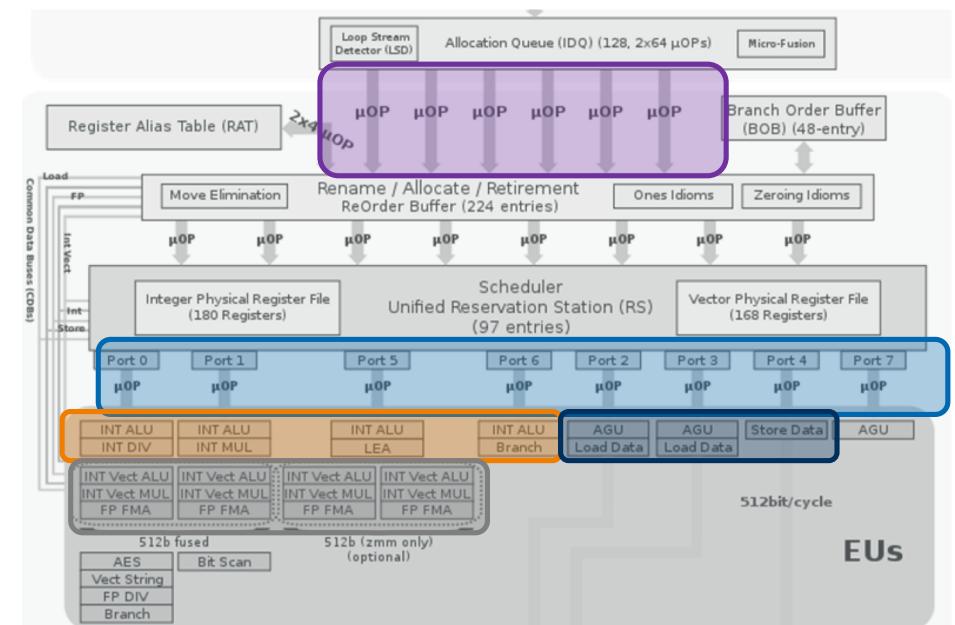
- Instructions Per Cycle (IPC)
  - Gibt an wieviel Instruktionen pro Takt ausgeführt werden können
  - Hängt ab von Anzahl der Verarbeitungseinheiten, Instruction Fetch und Decode Umsetzung
  - Genutzt für Betrachtung von Bottlenecks in der Mikroarchitektur
- Cycles Per Instruction (CPI)
  - Invertiertes IPC
- Instructions Per Second (IPS)
  - Skaliert IPC mit Taktfrequenz
  - Genutzt bei Betrachtung von allgemeiner Performance eines Programms
  - Aufteilung in
    - Integer Operations Per Second (IOPS)
    - Floating Point Operations Per Second (FLOPS)
    - I/O Operations Per Second (IOOPS)

# Sustained vs. Peak-Performance

- Peak-Performance ist abhängig von Hardware, z.B.
  - 2 SIMD Einheiten (AVX, FMA), double precision, 2.5 GHz
  - $\frac{256 \text{ bit (vector length)}}{64 \text{ bit (operand length)}} = 4$
  - $2 \frac{\text{Instructions}}{\text{cycle}} * 4 * 2 \frac{\text{Operations}}{\text{Instruction}} * 2.5 * 10^9 \frac{\text{cycle}}{\text{second}} = 40 \text{ GFLOPS}$
- Sustained Performance ist abhängig von Hardware und Software, z.B.
  - Daten liegen im Speicher → Limitiert durch Speicherbandbreite
  - Nur Additionsbefehle → FMA kann nicht genutzt werden
  - Viele Abhängigkeiten → Pipelining funktioniert nicht

# Beispiel Peakperformance

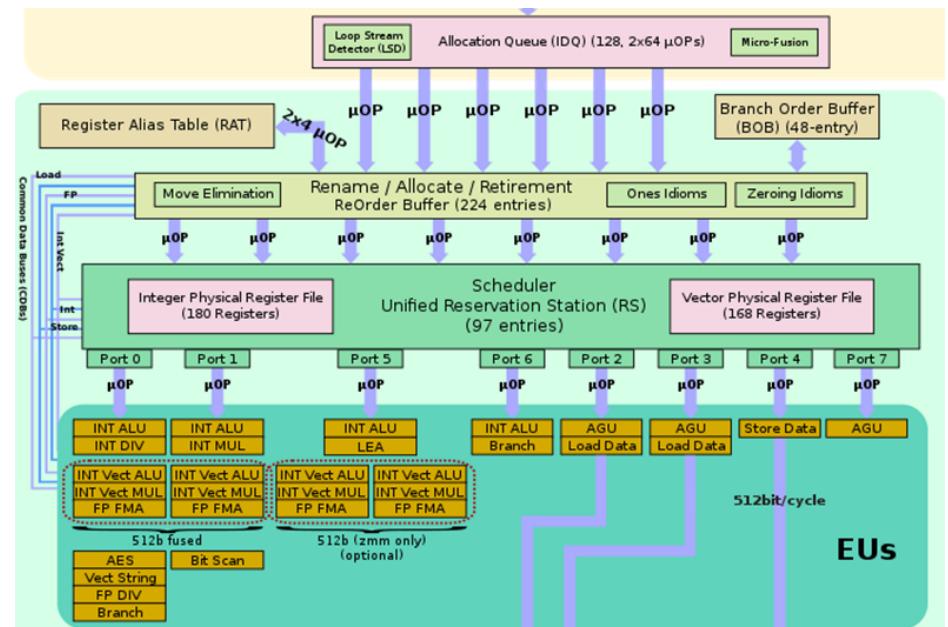
- Es können maximal 6 dekodierte µops an die OoO engine gegeben werden (~6 IPC)
- Es können 8 verschiedene Ports für Operationen genutzt werden
  - 4 Ports können nicht SIMD-parallele Integer Berechnungen machen  
➤ Peak: 4 IOPC
  - 3 Ports können Fließkommaberechnungen machen (FMA, 2\*256 Bit, 1\*512 Bit)  
➤ 32 FLOPC (double precision 64 Bit Werte)
  - Nicht gleichzeitig mit IOPC
  - 3 Ports können Laden/Speichern:  
➤ 3 IOOPC



# Very Long Instruction Word (VLIW)

# Rückblick

- Es gibt mehrere Ausführungseinheiten (Execution Units)
- Diese sind teilweise spezialisiert (ALU, FPU, AGU, ...)
- Diese werden parallel genutzt
  - Pipelining
  - Superskalarität
- Hardware überwacht Abhängigkeiten zwischen Befehlen



# Interaktion mit anderen Disziplinen (Wiederholung)

Compilerbau:

- Kennt Befehle des Prozessors
- Kennt Latenzen der Mikroarchitektur für Optimierung
- Kennt Software und Abhängigkeiten
- Könnte entscheiden welche Instruktionen parallel ausgeführt werden

# Compiler statt Hardware

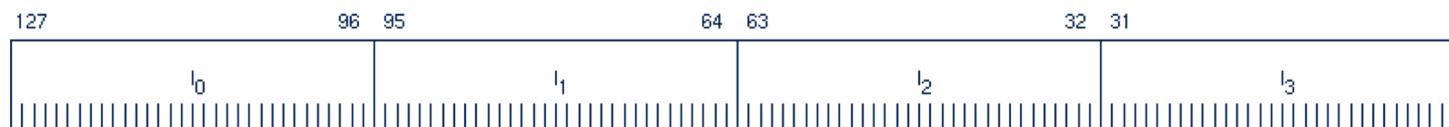
- Viele Transistoren werden für Abhängigkeitsanalyse und Out-of-Order verwendet
- Prozessor entscheidet zur Laufzeit, welche Befehle unabhängig von einander ausgeführt werden können
- Kann ggf. zur Compilezeit entschieden werden

Vorteile:

- Muss nicht zur jeder Ausführung neu gemacht werden
- Spart Transistoren (und Leistungsaufnahme), welche für andere Dinge (z.B. Caches genutzt werden können)

# Grundkonzept

- Ein (langes) Befehlsswort mit mehreren parallel zu bearbeitenden Befehlen
- Z.B. Vier Befehle à 32 Bit Byte → Ein 128 Bit Wort



- z.B. 1983, ELI-512 computer system [Fis83], >500 bit Befehlswort → 10-30 RISC Befehle pro Takt  
[Fis83] Fisher, Very Long Instruction Word architectures and the ELI-512, 1983

# Einige Prozessoren, wenig dauerhafter Erfolg

- Wenige Implementierungen in 80ern
- Multiflow Trace Prozessoren
  - Zunächst 256 bit breite Befehlswörter
  - Bis zu 1024 bit breite Befehlswörter
- Cydrome
- Intel i860
- Elbrus 3



Elbrus 3 Computer

By Koshmarov at Russian Wikipedia, CC BY-SA 3.0,  
<https://commons.wikimedia.org/w/index.php?curid=18414958>

# Gründe des Misserfolgs und neuer Versuch

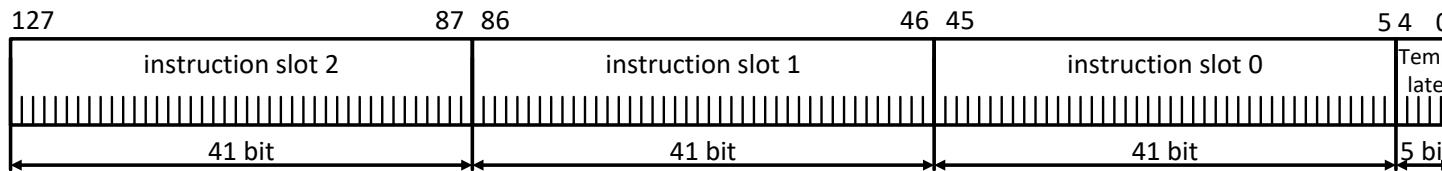
- Compiler nicht schlau genug
  - Zu wenig Parallelismus, viele nops (No OPerations)
- Zur Compilezeit ist nicht alles bekannt
  - Ob Speicherbereiche überlappen weiß man ggf. erst zur Laufzeit
- Muss ggf. mit nop s aufgefüllt werden → Lange Befehlsworte mit wenig Inhalt
- Mehrere Jahre Compilerentwicklung später versucht es Intel 1994 erneut

# Intel IA64 (EPIC)

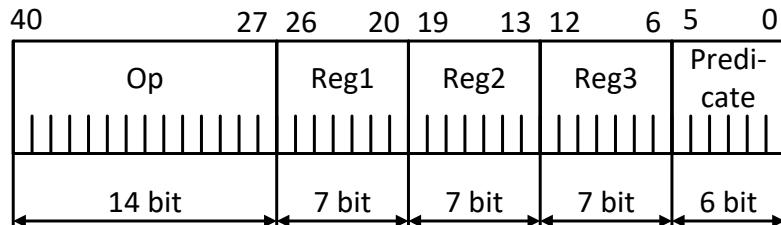
- Von Intel und Hewlett-Packard seit 1994 in Kooperation entwickelt
  - Neue 64-Bit ISA
  - Kann auch Intel IA-32 und Hewlett-Packard Precision Architecture (HP-PA) Programme abarbeiten
- Verlagerung der Komplexität von der Hardware zur Software
  - Keine out-of-order Execution in Hardware
  - Compiler/Programmierer steuert Nebenläufigkeit
- Im Gegensatz zu out-of-order Architekturen soll die wachsende Anzahl zur Verfügung stehender Transistoren nicht für immer komplexere Steuerungen eingesetzt werden, sondern für:
  - Größere Anzahl unabhängiger Funktionseinheiten
  - Wachsende Anzahl zur Verfügung stehender Register und
  - Größere on-chip Caches

# Befehlsaufbau

- 128 Register
- 3 Einzelbefehle in einem Befehlswort
- Template enthält Angaben über Abhängigkeiten



IA-64-Befehlswort (aus /Märt 01a/, S. 162, Bild 6.3)



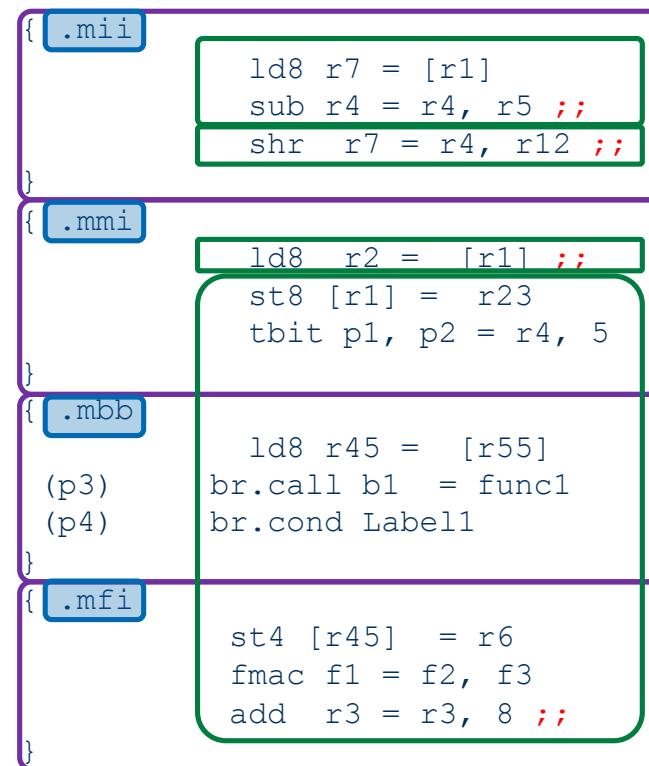
Einzelbefehl im IA-64-Befehlswort (aus /Inte 01a/, S. 31)

# Befehlsaufbau

- Unabhängige Befehle in Befehlsgruppen zusammengefasst, durch STOPS voneinander getrennt
  - Gruppen können mehrere Befehlsworte überspannen
  - Befehlsgruppen können voneinander abhängen
- Template-Feld charakterisiert:
  - Typ der Operation für jeden Befehl innerhalb des IA-64-Befehlswortes (MFI, MMI, MII, MLI, MIB, MMF, MFB, MMB, MBB, BBB),
    - I: Integer
    - M: Memory
    - F: Floating Point
    - B: Branch
    - L+X: Extended
  - Abhängigkeiten zwischen Befehlsgruppen innerhalb eines Befehlswortes
  - Abhängigkeiten zwischen den Befehlsworten

# Beispiel

- Befehlsworte
- Werden parallel gelesen
- Operationstypen
- Geben interne Struktur vor
- Befehlsgruppen
- Können parallel ausgeführt werden



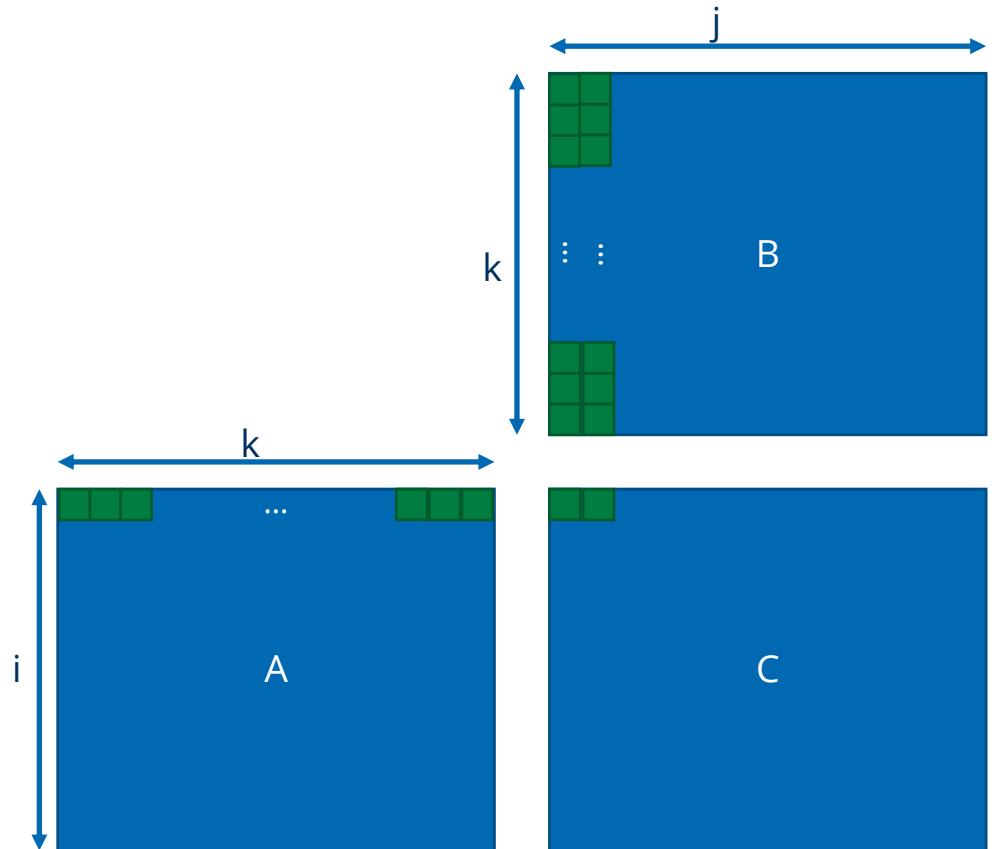
# Sind die Compiler besser geworden? Beispiel Matrix-Multiplikation

# Hintergrund Matrix-Multiplikation

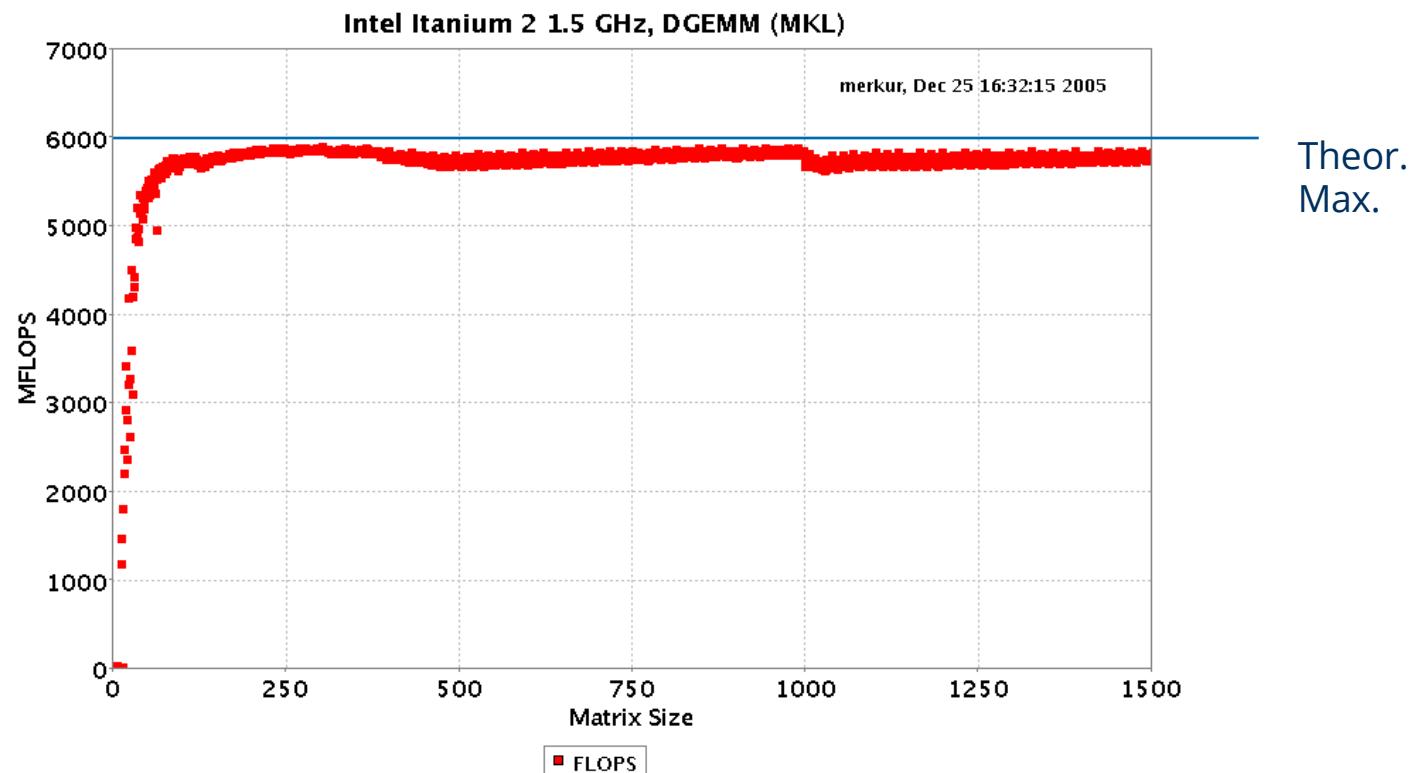
- Algorithmus (siehe Animation):

```
for (i=0;i<N;i++)  
  for (j=0;j<N;j++)  
    for (k=0;k<N;k++)  
      c[i][j]=c[i][j]+a[i][k]*b[k][j]
```

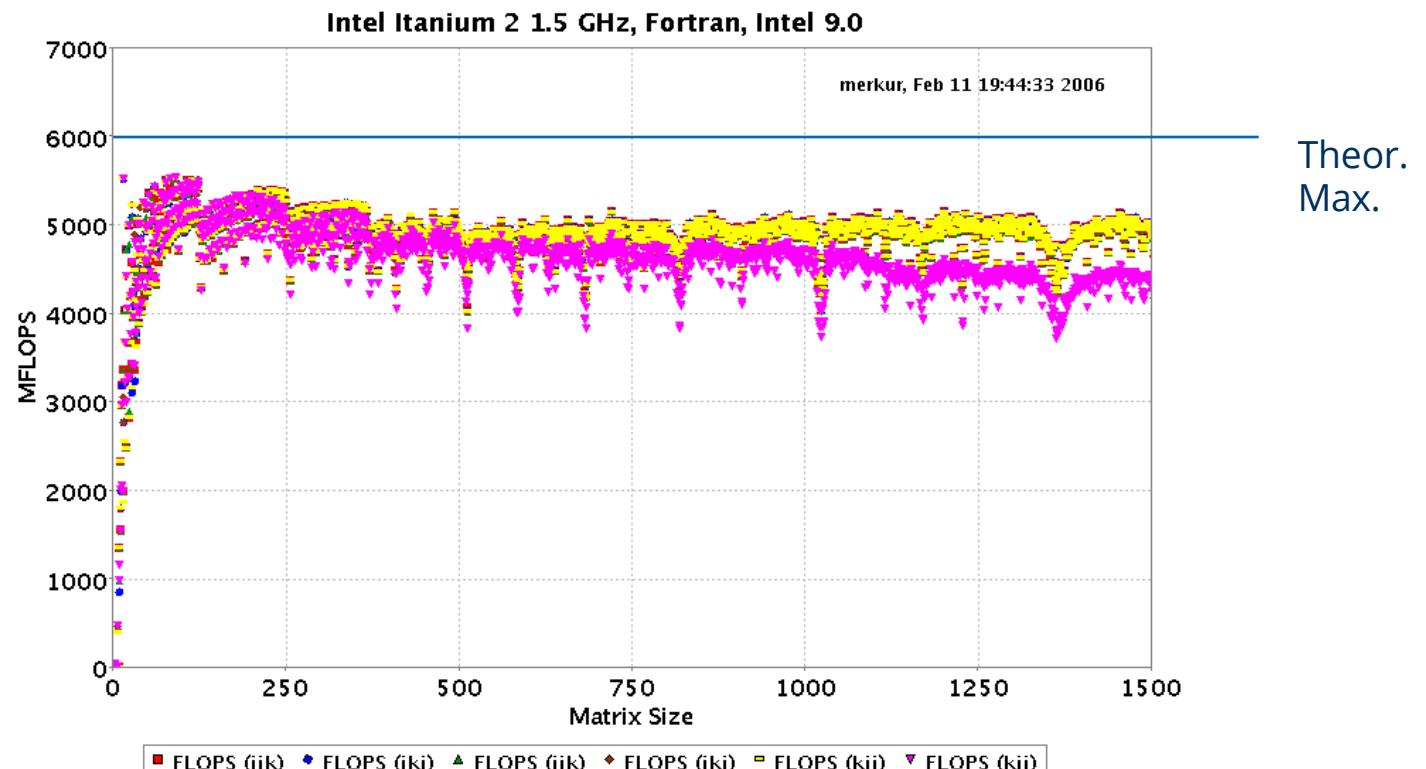
- Gut optimierbar
  - Z.B. Schleifenentrollen
- Gut parallelisierbar
  - Elemente von C unabhängig zu berechnen
- Bewertung von HPC Systemen (Linpack)



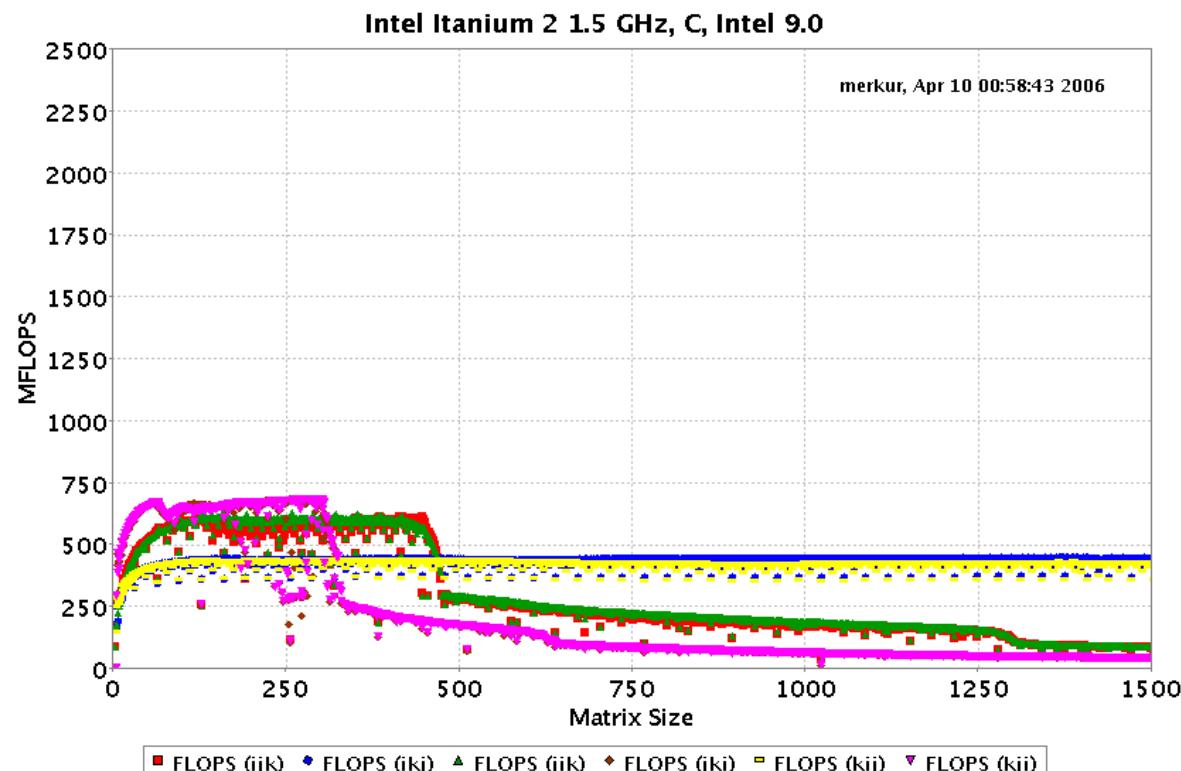
# Matrix Multiplikation: Optimierte Bibliothek



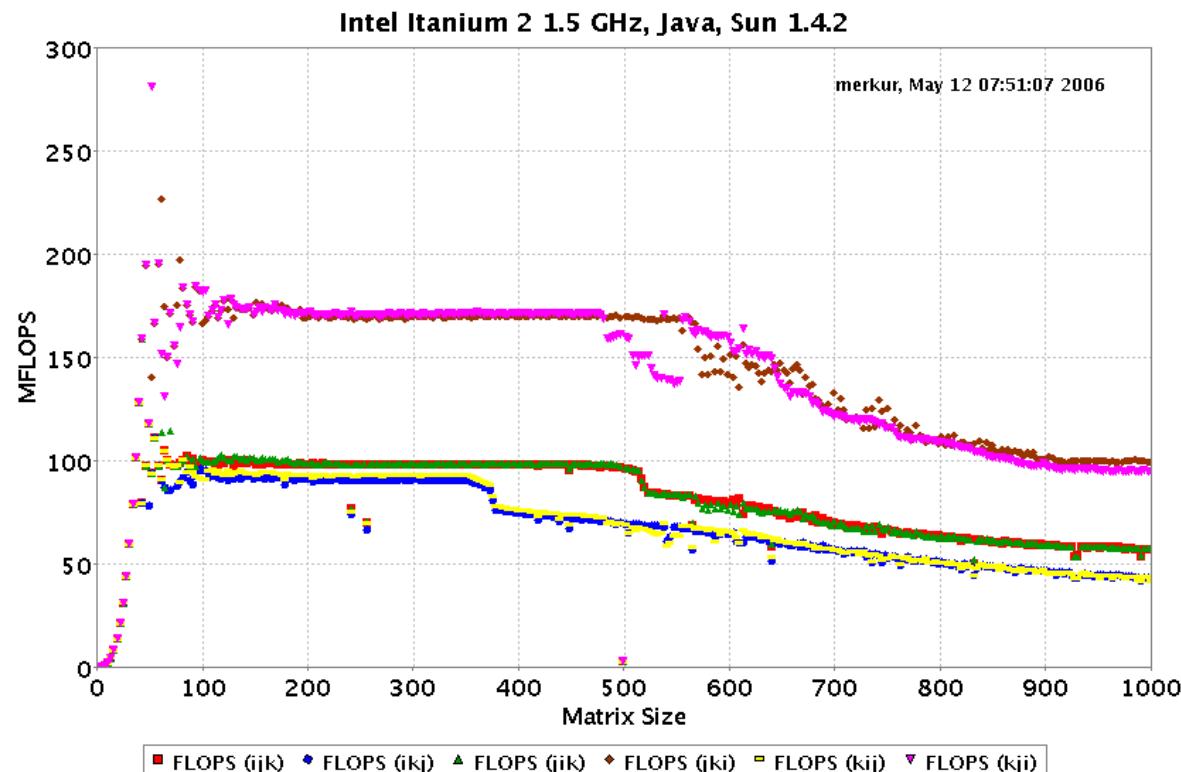
# Matrix Multiplikation: Fortran



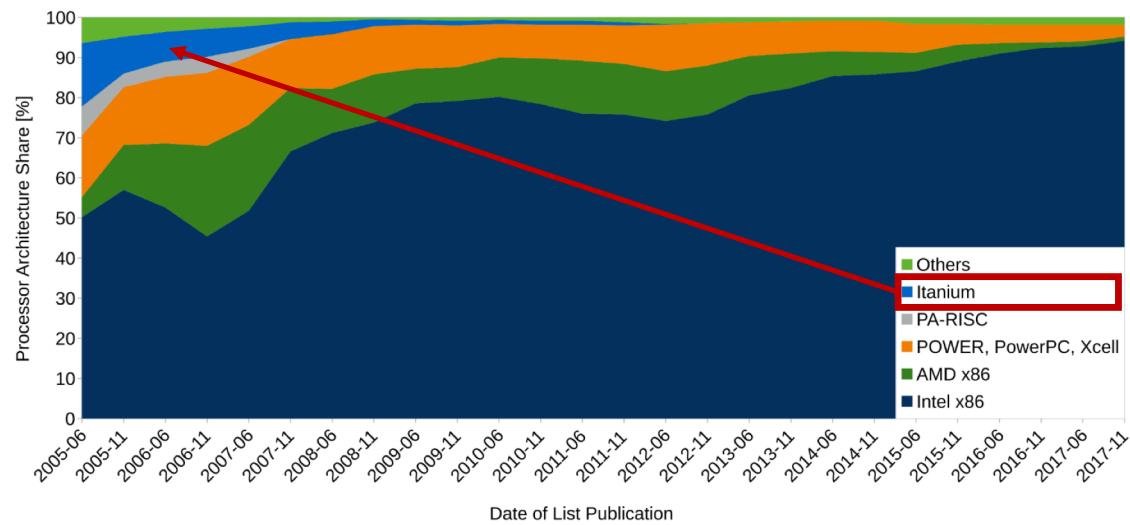
# Matrix Multiplikation: C



# Matrix Multiplikation: JAVA



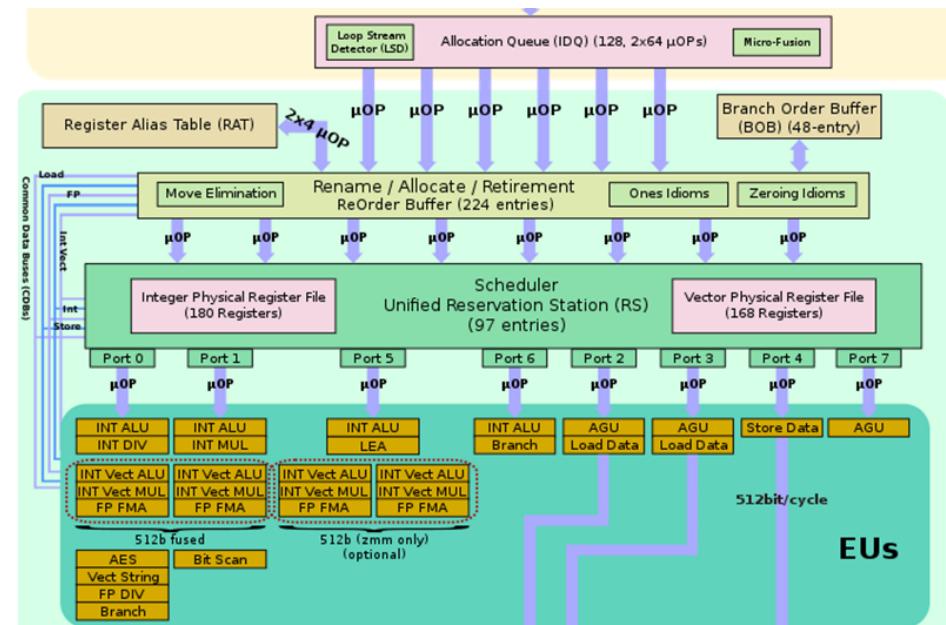
## Aktueller Stand IA64 (HPC Top500)



# Hardware Multi Threading

# Rückblick

- Mehrere Ausführungseinheiten
- Durch Superskalarität werden mehrere Befehle gleichzeitig ausgeführt
- Limitiert durch Abhängigkeiten und Latenzen (z.B. zum Speicher)
- Out-of-Order kann das verbessern
- Trotzdem Limitierung
  - Meist werden nur wenige Ausführungseinheiten genutzt

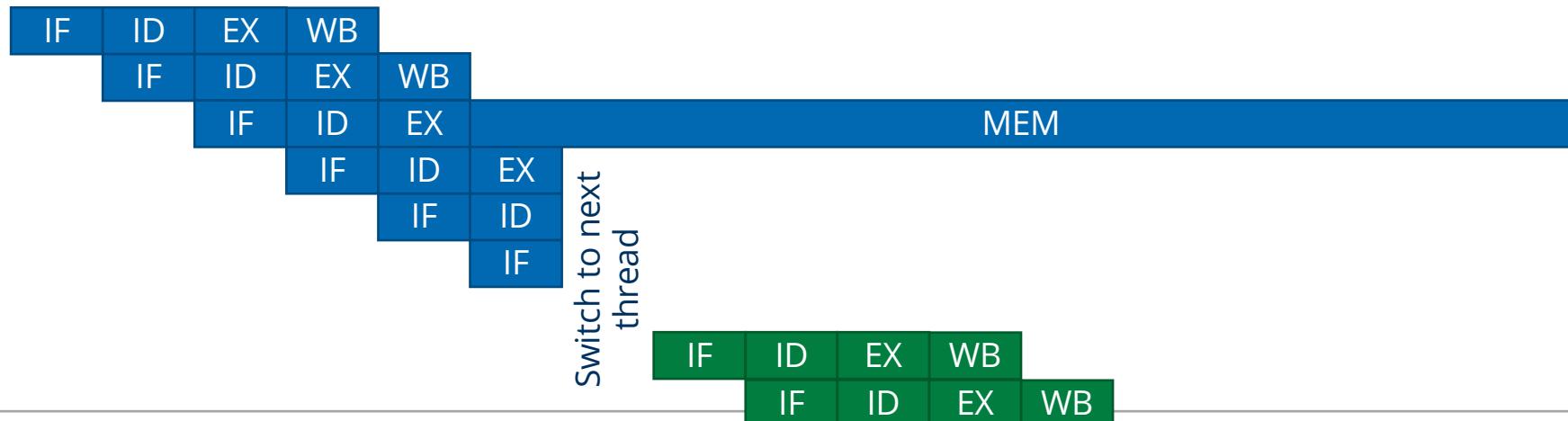


# Lösung: Mehrere Threads auf einem Kern ausführen

z.B. Zugriff auf Speicher: 200 Takte

- Sobald ein Thread zu lange wartet, anderen Thread auswählen
  - Sobald dieser wartet nächsten Thread auswählen
- Coarse grained multithreading

Beispiel (Pipelining, keine Superskalarität, *stall* durch Speicherzugriff)



# Lösung: Mehrere Threads auf einem Kern ausführen

z.B. Zugriff zum L1 Cache: 4 Takte

- Selbst wenn alle Daten im L1 liegen, muss 4 Takte auf Datum gewartet werden
  - Alternativ: 4 Threads, die abwechselnd dran kommen
- Cycle-by-cycle multithreading / fine grained multithreading

Beispiel (kein Pipelining, keine Superskalarität, Zugriff auf L1)

— Vorher:



— Nachher:



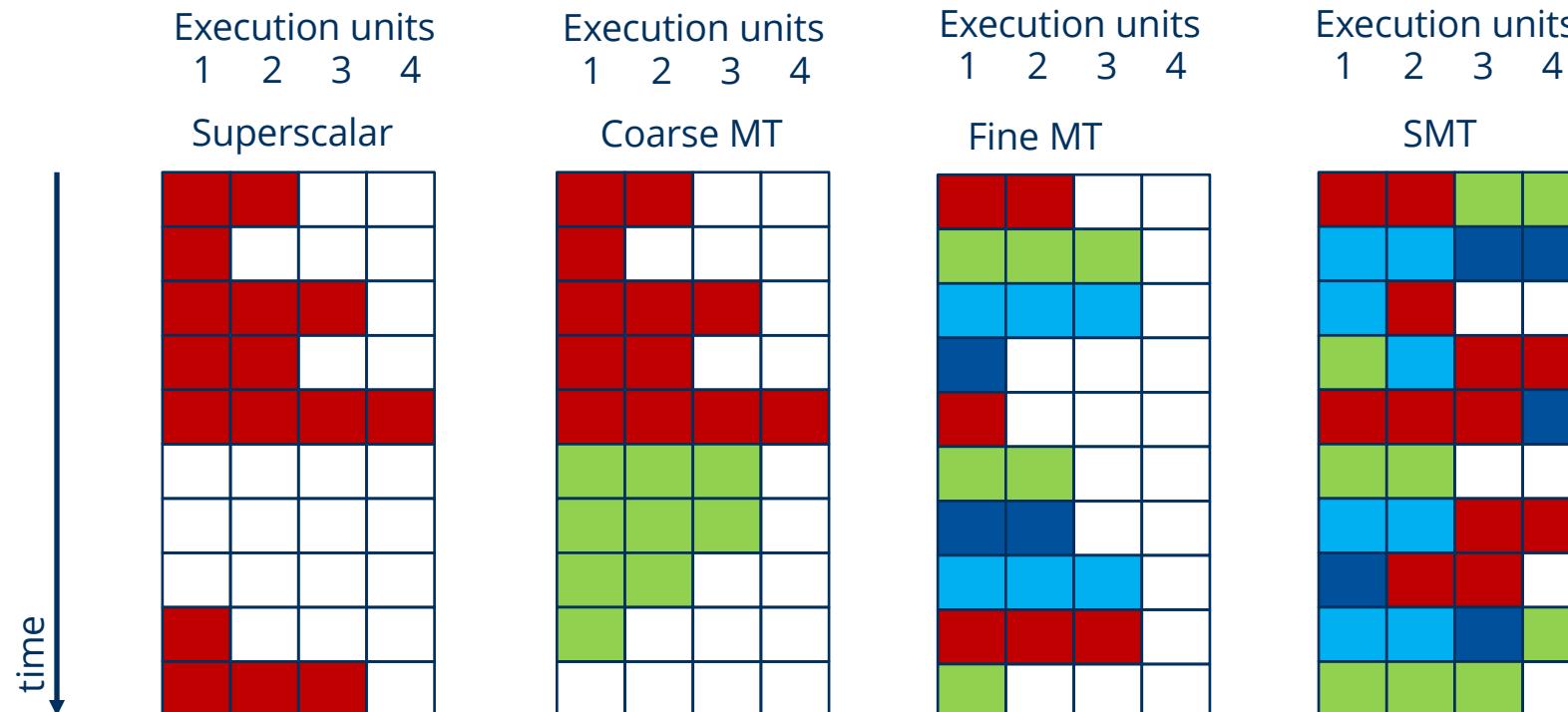
# Lösung: Mehrere Threads auf einem Kern ausführen

Superskalare Prozessoren können mehrere Befehle pro Takt starten

- Out-of-Order Engine schaut, welche Befehle ausgeführt werden können
  - Alle Befehle, die keine Abhängigkeiten haben können ausgeführt werden
  - Je mehr unabhängige Threads, desto höhere Wahrscheinlichkeit, dass irgendein Befehl ausgeführt werden kann
- **Simultaneous multithreading**
- In einem Takt können Befehle unterschiedlicher Threads gestartet/abgeschlossen werden
  - (Beispiel in Übersicht folgende Folie)

# Zusammenfassung

Better utilization of functional units:



# Zusätzliche Hardware

- Ein Satz an Registern pro Thread
- Ein Program Counter pro Thread
- Ggf. mehr IF/ID Einheiten
- Jeder abgearbeitete Befehl muss dazugehörigen Thread annotiert haben
- Jeder Eintrag im Cache/TLB muss dazugehörigen Thread annotiert haben

# Probleme

- Volle Prozessor Performance kann nur mit allen Threads erreicht werden
  - Insb. bei cycle-by-cycle Multithreading
- Threads nutzen gemeinsame Ressourcen
  - Können einander behindern (z.B. Daten von 2 Threads passen nicht in L1 Cache)
  - Können einander „ausspähen“ (Auswirkungen des anderen Threads auf geteilte Ressourcen)

# Threads, Cores, Modules, Dies und Packages

# MIMD ist nicht gleich SIMD

Rechnersysteme, die mehrere Prozesse/Threads gleichzeitig ausführen können:

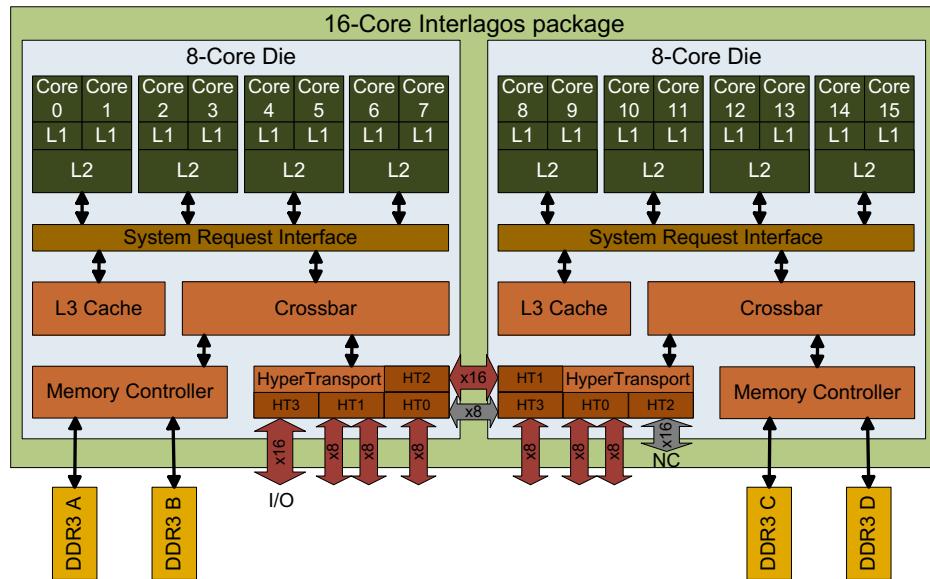
- Single Core mit Multithreading
- Mehrere Kerne in einem Prozessor
- Mehrere Prozessoren auf einem Board

- Brauchen gemeinsame Sprache

# Typische Begriffe

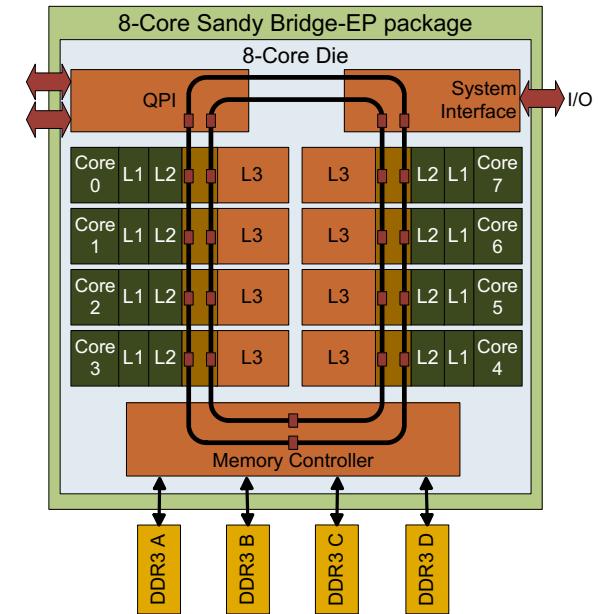
- Package:
  - Physischer "Chip", den man auf das Mainboard setzt, besteht aus einem oder mehreren Dies
- Die:
  - In einem Siliziumstück gefertigt
  - Mehrere Dies in einem Package müssen über Schnittstellen verbunden werden
  - Beinhaltet mehrere Kerne
- Kerne:
  - Ein Kern beinhaltet Rechen- und Steuerwerk, sowie möglicherweise eigene weitere Ressourcen, typischerweise heute: L1 Caches, L2 Cache, ...
- Module:
  - Eine Anzahl von Kernen, die Ressourcen miteinander teilen (diese sind auch Teil des Moduls) aber nicht mit anderen Kernen.
- Uncore
  - Ressourcen auf einem Die, die keinem Kern zugeordnet werden können (z.B. geteilter L3 Cache)
- Offcore
  - Ressourcen, die keinem Kern zugeordnet werden können (z.B. geteilter L3 Cache, aber auch off-chip)
- Thread
  - Siehe Hardware Multithreading

# Beispiele



Molka et al. Main Memory and Cache Performance of Intel Sandy Bridge and AMD Bulldozer, 2014

16 Core AMD Bulldozer



8 Core Sandy Bridge-EP processor

# Informationen unter Linux

- Kommunikation in Software
  - Wie weit sind die kommunizierenden Threads auf der Hardware voneinander entfernt?
- Gemeinsam genutzte Ressourcen
  - Welche Ressourcen werden geteilt?
  - Wie werden die Ressourcen geteilt?

Informationen unter Linux: Unter Linux ist jeder Thread eine CPU

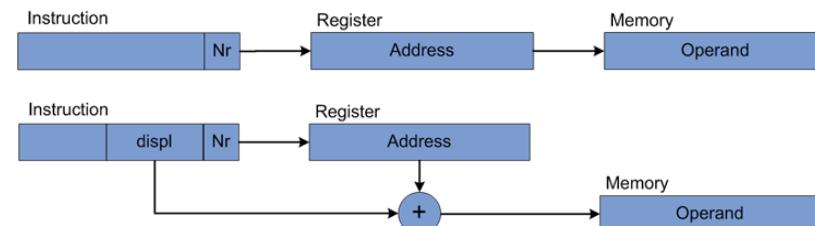
- Welche anderen CPUs gehören zu CPU <x>:
  - `/sys/devices/system/cpu/cpu<x>/topology/thread_sibling_list`
- Welche anderen CPUs gehören zum Die von CPU <x>:
  - `/sys/devices/system/cpu/cpu<x>/topology/core_siblings_list`

# Bekannte Prozessorfamilien

# Wiederholung und Vorwort

- Letzte Vorlesung: x86 als CISC Prozessor
- Alle folgenden Prozessorfamilien sind RISC Prozessoren
- RISC hat auch Vorteile!
  - Einfache Befehle:
    - Braucht kein Mikroprogrammsteuerwerk, sondern kann festverdrahtet Befehle umsetzen
      - Weniger Transistoren für Instruction Decode
  - Mächtige Befehle:
    - Abbildung fehlender Adressierungsmodi auf vorhandene
    - z.B. *Register indirekt* (nicht vorhanden) auf

*Register indirekt mit displacement* (vorhanden)  
mit displacement == 0



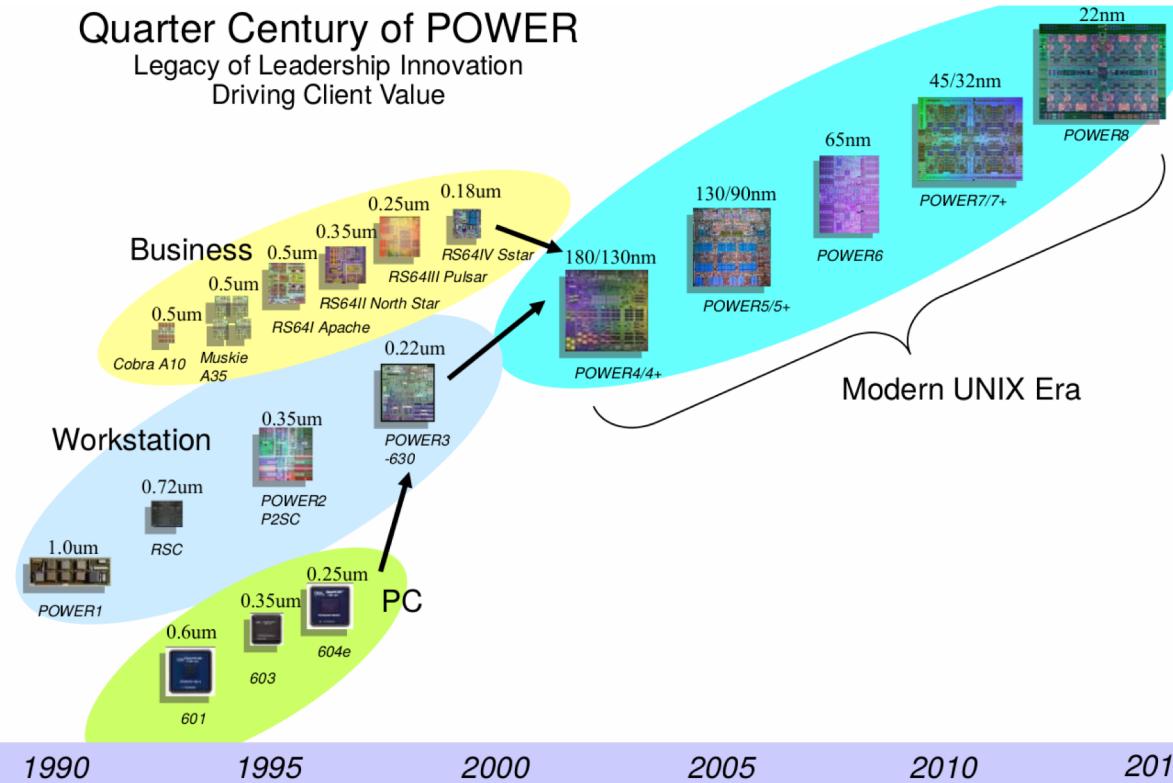
# POWER und PowerPC

- RISC Architektur
- Hauptsächlich Business Server und HPC
- PowerPC Abspaltung für Heimmarkt  
(Frühere Mac, XBox360, Wii)
- Aktuell schnellstes HPC System mit POWER9 Prozessoren und NVIDIA GPGPUs



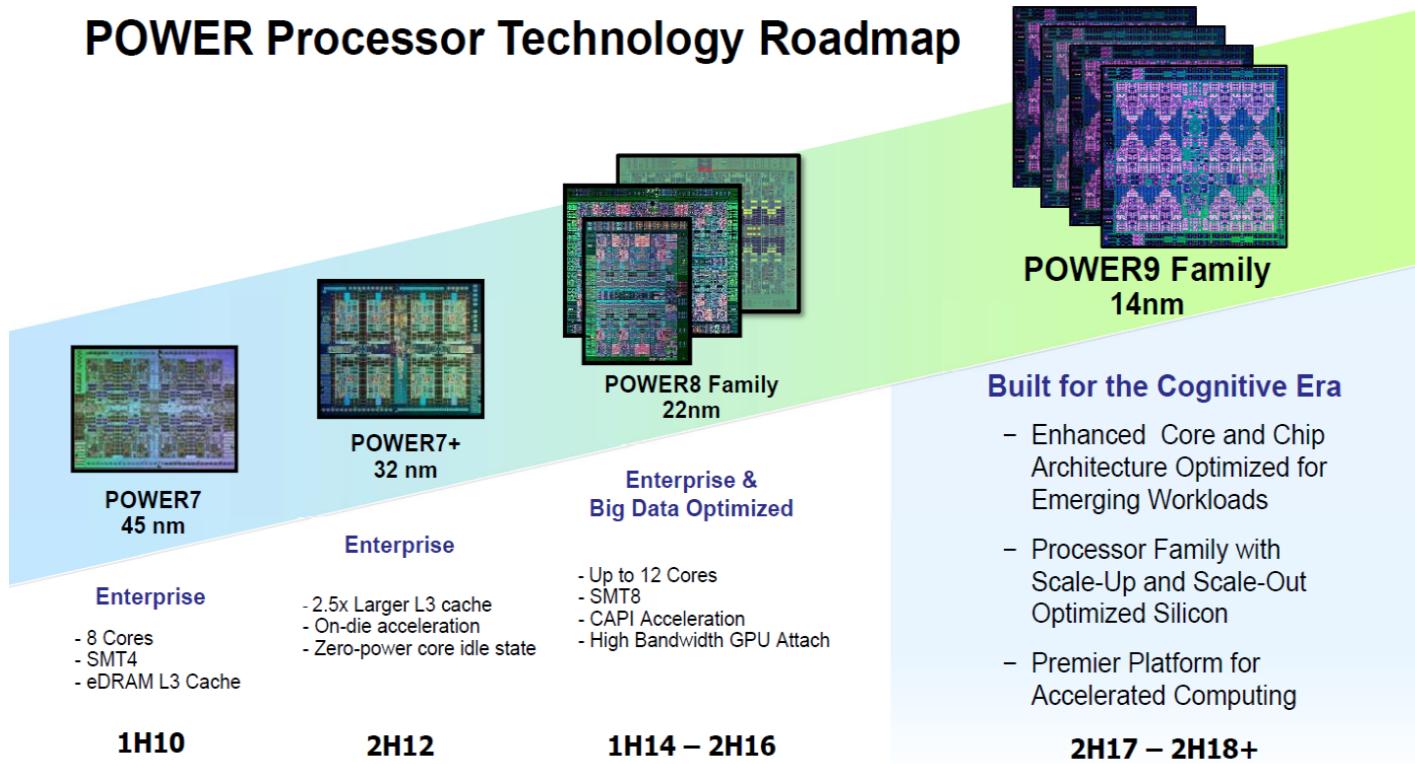
Carlos Jones/ORNL • [CC BY 2.0](#)

# POWER Prozessoren – Rückblick



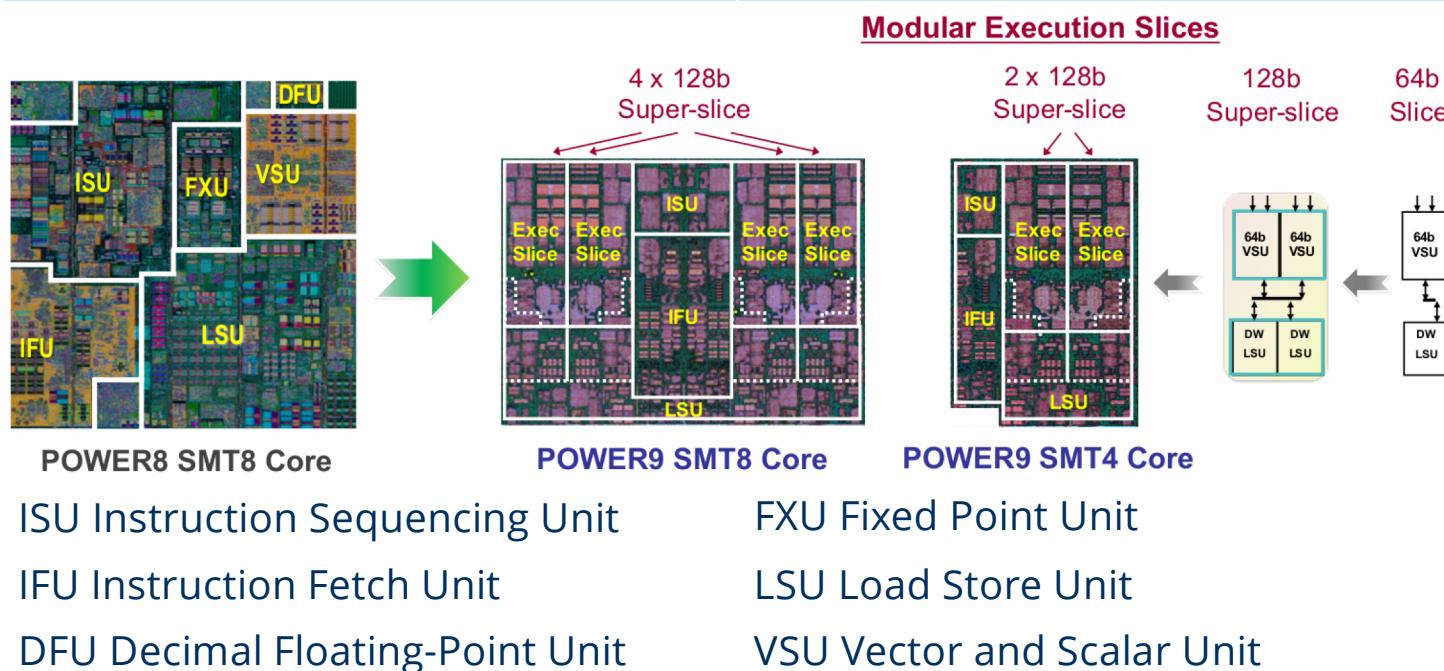
- Major POWER® Innovation**
- 1990 RISC Architecture
  - 1994 SMP
  - 1995 Out of Order Execution
  - 1996 64 Bit Enterprise Architecture
  - 1997 Hardware Multi-Threading
  - 2001 Dual Core Processors
  - 2001 Large System Scaling
  - 2001 Shared Caches
  - 2003 On Chip Memory Control
  - 2003 SMT
  - 2006 Ultra High Frequency
  - 2006 Dual Scope Coherence Mgmt
  - 2006 Decimal Float/VSX
  - 2006 Processor Recovery/Sparing
  - 2009 Balanced Multi-core Processor
  - 2009 On Chip EDRAM

# POWER Prozessoren – Rückblick



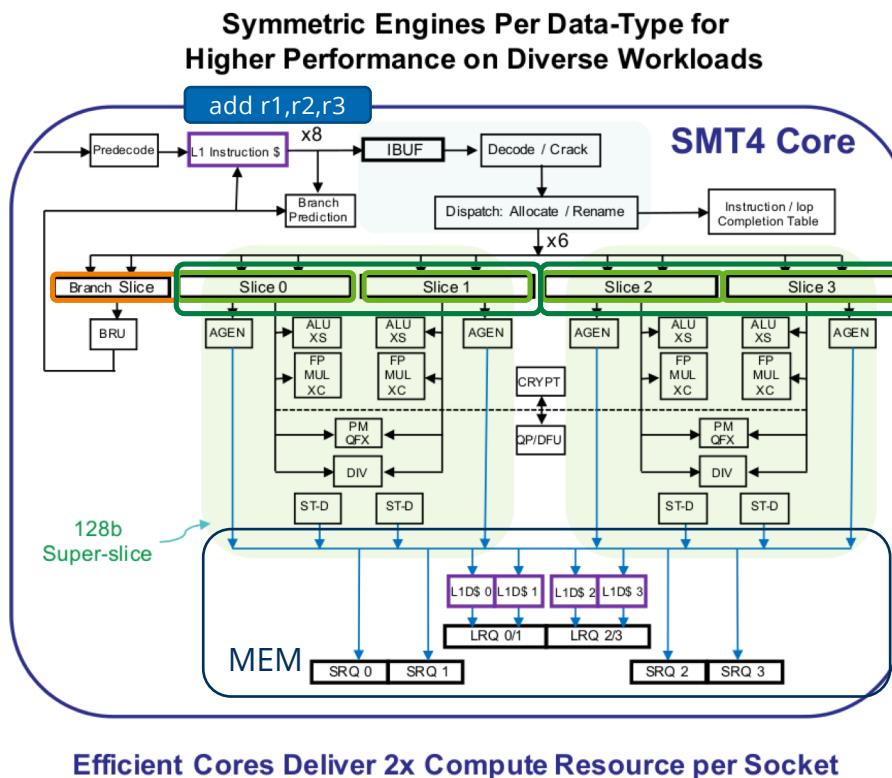
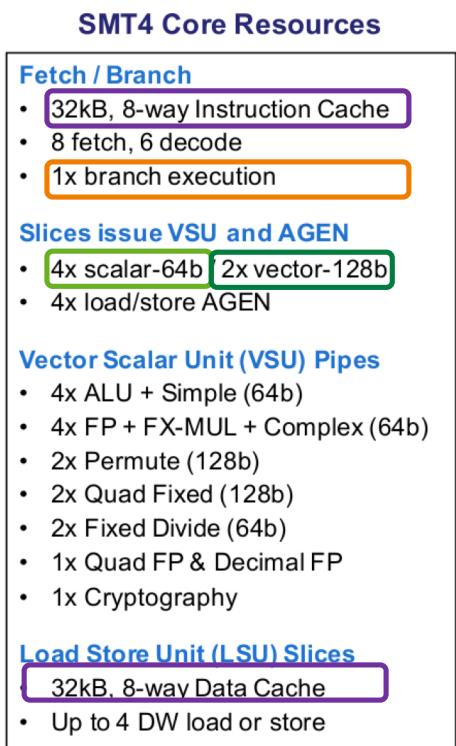
## POWER9 Kerne 2 Varianten

SMT8 Core	SMT4 Core
12 Kerne à 8 Threads	24 Kerne à 4 Threads
Ausgelegt für PowerVM Ecosystem (AIX)	Ausgelegt für Linux Ecosystem



Quelle: POWER8/9 Deep Dive, Jeff Stuecheli, POWER Systems, IBM Systems

# POWER9 Kern im Detail



- IF
- ID
- Danach in OoO-Engine
- EX
  - Zuweisung an eine Pipe
  - Alle Pipes sind gleich aufgebaut (außer Branch)
  - Für komplexe Befehle (z.B. DIV, Permute, Decimal FP, Crypt) werden mehrere Pipes gebraucht

# POWER9 Kern Performance

## SMT4 Core Resources

### Fetch / Branch

- 32kB, 8-way Instruction Cache
- 8 fetch, 6 decode
- 1x branch execution

### Slices issue VSU and AGEN

- 4x scalar-64b / 2x vector-128b
- 4x load/store AGEN

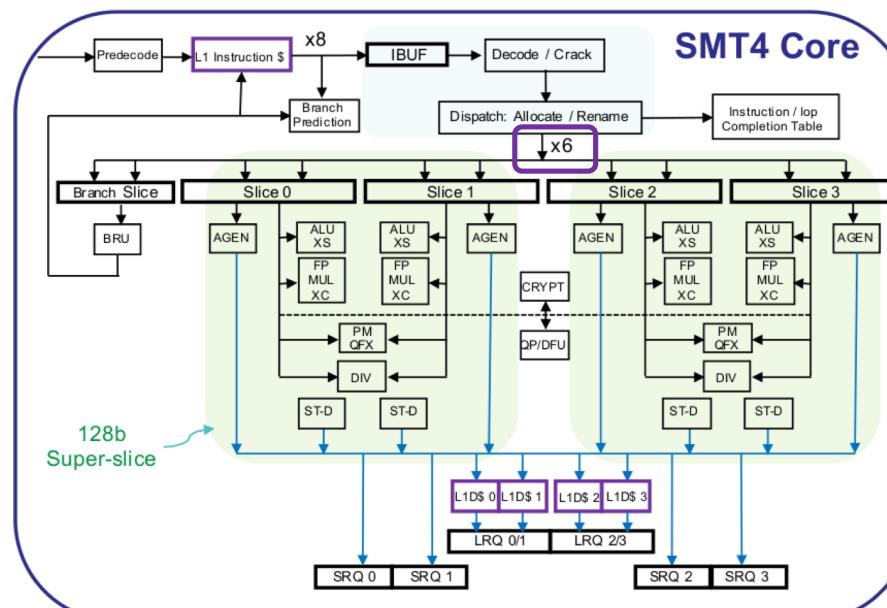
### Vector Scalar Unit (VSU) Pipes

- 4x ALU + Simple (64b)
- 4x FP + FX-MUL + Complex (64b)
- 2x Permute (128b)
- 2x Quad Fixed (128b)
- 2x Fixed Divide (64b)
- 1x Quad FP & Decimal FP
- 1x Cryptography

### Load Store Unit (LSU) Slices

- 32kB, 8-way Data Cache
- Up to 4 DW load or store

## Symmetric Engines Per Data-Type for Higher Performance on Diverse Workloads

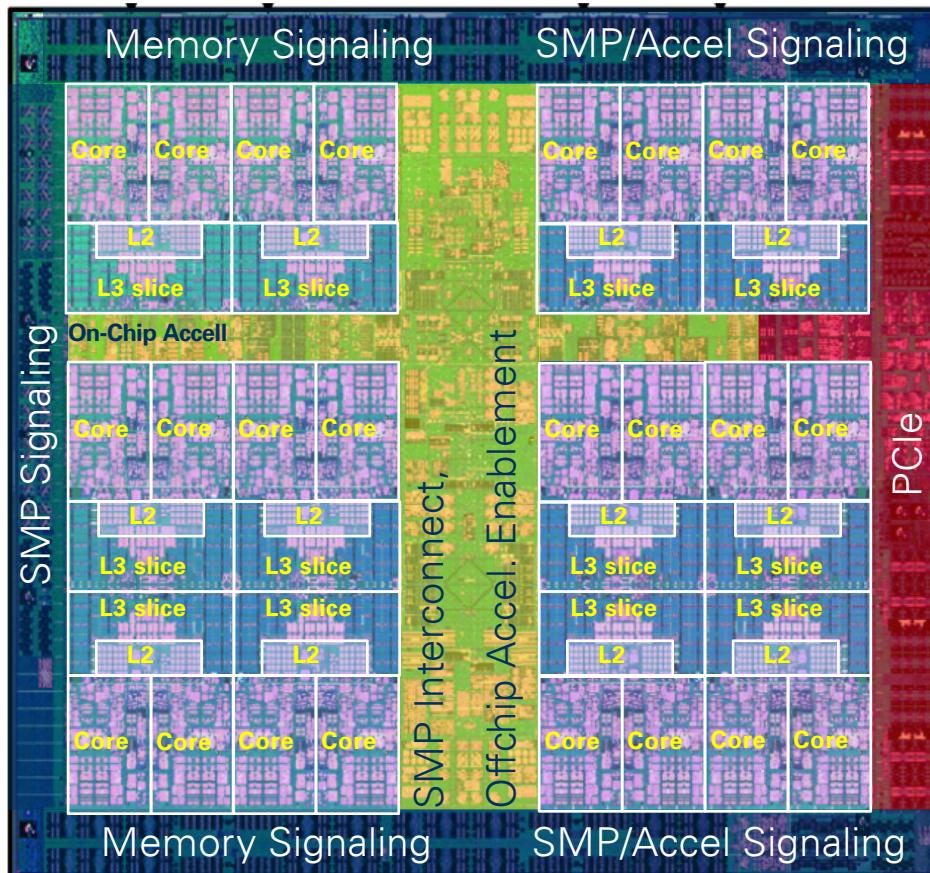


Efficient Cores Deliver 2x Compute Resource per Socket

- SIMD does not increase performance  
2\*128 Bit vs. 4\*64 Bit
- Max. 6 IPC
- Max. 4 IOPC
- Max. 4 IOPPC

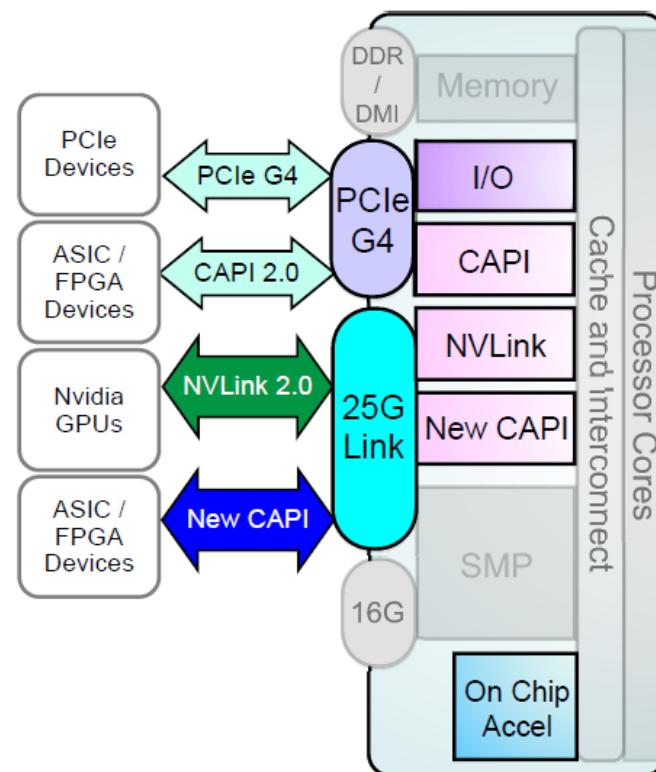
Aber laut IBM werden bis zu 64 Befehle pro Takt und Kern fertiggestellt

# POWER9 Die/Package



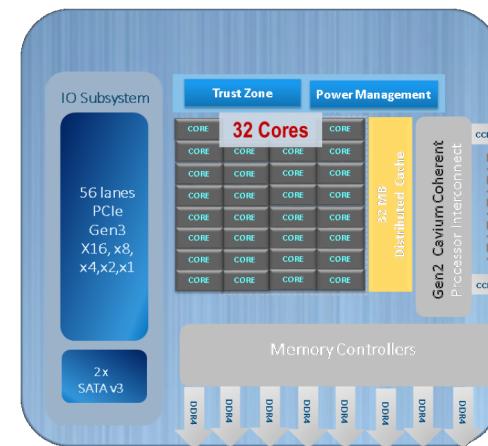
# POWER9 – Off-Chip Connection

- I/O hosts
  - PCIe Gen 4 x 48 lanes – 192 GB/s duplex
  - 25G Link x 48 lanes – 300 GB/s duplex
- I/O Schnittstellen
  - CAPI 2.0 – Offener Standard zur Anbindung von I/O Devices (über PCIe)
  - New CAPI – Offener Standard zur Anbindung von I/O Devices (über 25G Link)
  - NVLink 2.0 – Anbindung von NVIDIA Grafikkarten (über 25G Link, mehr dazu auf folgenden Folien)



# ARM

- RISC Architektur
- Hauptsächlich Embedded Systems und mobile Devices
  - Smartphones, Tablets, Handhelds
- Seit 2017 auch im HPC angekommen
  - Thunder X2



Rafael Fernandez

ThunderX2 ® CN99XX Product Brief

# ARM Big.LITTLE

- Designkriterien bei mobile Devices: Energieeffizienz und Performance
  - Leistungsaufnahme steigt mit Anzahl der Transistoren → Kleine Kerne
  - Performance steigt mit Anzahl der sinnvoll eingesetzten Transistoren → Große Kerne
  - Widerspruch
- Lösung:
  - Kleine Kerne (z.B. weniger Out-of-order und spekulative Ausführung) *und* große Kerne auf Chip
  - Betriebssystem schaltet je nach ausführter Anwendung oder Auslastung um
- z.B. Samsung Galaxy S8
- *Heterogener Multicore Prozessor*

# MIPS

- 1996&2002 HPC System an der TU Dresden
- 1991 erster kommerzieller 64-Bit-Mikroprozessor
- Architektur seit 2018 unter Open Source
- MIPS = erweitertes DLX (Übung)



SPARC

- RISC Architektur
  - Serverprozessoren (SUN, Oracle) und HPC (Fujitsu)
  - Bis zu 8 Threads pro Kern (Hardware Multithreading)
  - Register Windows (zwischen 3 und 32 je nach Implementierung)
    - Jeweils 8 Register für
      - Argumente an aktuelle Funktion (**input**)
      - Gültig innerhalb der Funktion (**local**)
      - Argumente für die nächste Funktion (**output**)
      - Global (nicht im Beispiel)
    - Bei Funktionsaufruf werden aktive Register um 16 geshifted. Beispiel Funktionsaufruf:

i 0 i 1 i 2 i 3 i 4 i 5 i 6 i 7 i 0 i 2 i 3 i 4 i 5 i 6 i 7 i 0 i 1 i 2 i 3 i 4 i 5 i 6 i 7 i 0 i 2 i 3 i 4 i 5 i 6 i 7 i 0 i 1 o 0 o 0 o 4 o 5 o 6 o 7

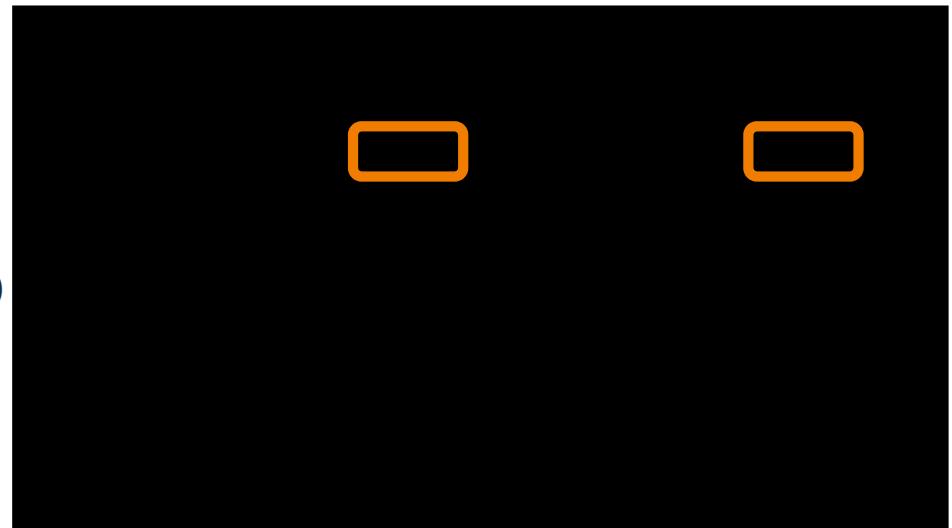
# Zusammenfassung

# Very Long Instruction Word

- Idee: Parallelismus von Befehlen innerhalb eines Threads wird durch Compiler gefunden
- ISA beinhaltet Beschreibung zum Parallelismus
- Mehrere parallele Befehle in einem Befehlswort
- Hat sich nicht durchgesetzt

# Hardware Multithreading

- Erhöht nicht die Peakperformance, sondern die Auslastung der Rechenressourcen
- Verdecken von Speicherlatenzen
- Möglichst unterschiedliche Workloads
- Aber: Threads konkurrieren um Ressourcen, mehr Parallelismus heißt mehr Synchronisierung  
→ Performance *kann* sinken
- Beispiel:  
Spiel *Hitman* auf Intel Core i7 8700k  
Quelle: Youtube, Testing Games,  
Hyper Threading ON vs OFF Test in 9 Games (i7 8700k)  
Dec. 2017



# Threads, Cores, Modules, ...

- CPUs werden vom Betriebssystem genutzt um Threads/Prozesse auszuführen
- Es spielt eine Rolle, welche CPUs genutzt werden
  - Hardware-Threads konkurrieren um Ressourcen im Kern und im Offcore
  - Kerne konkurrieren um Ressourcen im Offcore
- Platzierung der Threads wichtig für hohe Performance

# Prozessorfamilien

- Unterschiedliche ISAs dominant in unterschiedlichen Märkten
  - x86 dominant in PC, Server und HPC Bereich
  - ARM dominant im embedded/mobile Bereich
  
  - Alle Prozessorfamilien nutzen jetzt Pipelining, Superskalarität und implementieren SIMD
  - MIMD (Multicore) ist Mainstream
  - Die meisten nutzen out-of-order Ausführung
  - Einige nutzen Multithreading
- Prinzipien verstehen um die Ressourcen eines Prozessors gut zu nutzen