# Hyperthreads in Itanium®-Prozessoren

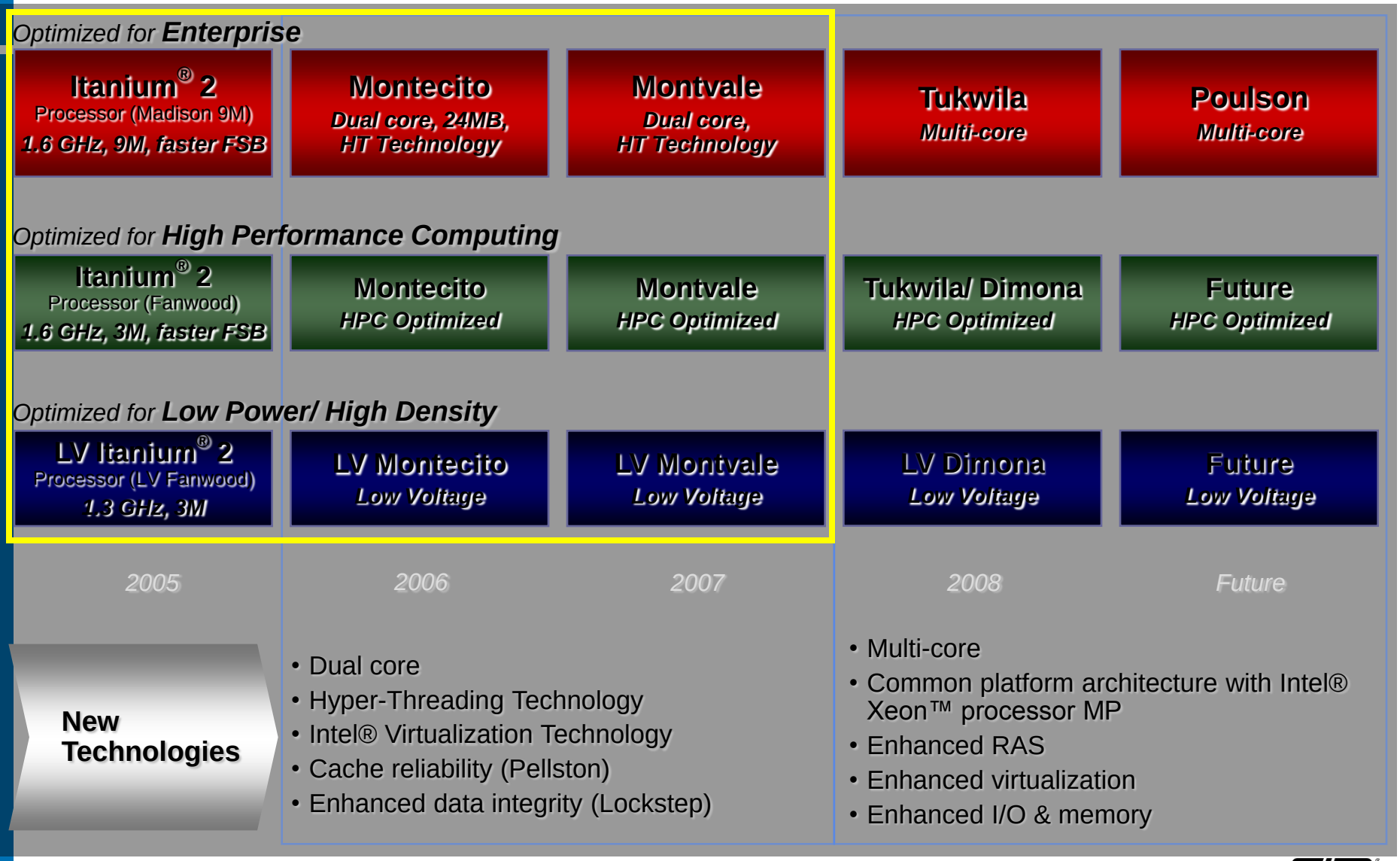
## …und wie OpenVMS damit umgeht

Thilo Lauer

Technical Consultant

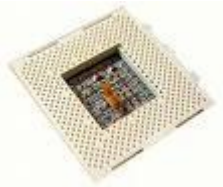
Account Support Center

# Intel® Itanium® Processor Family Roadmap



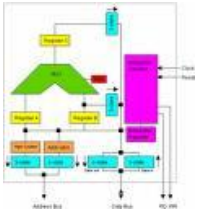

All products, dates, comparisons, and information are preliminary and subject to change without notice.

# Begriffsbestimmung: Sockets, Processors, Cores, CPUs, Threads, …

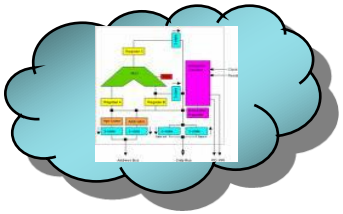


- *Socket:* physikalischer Steckplatz auf einer Platine/Motherboard.
  - ES45 hat 4 Sockets.

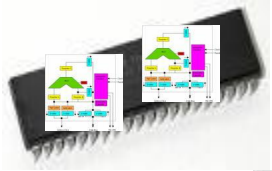
- *Processor:* wird in einen Socket gesteckt.
  - 4004, StrongARM, Alpha, Montecito



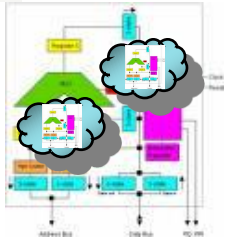
- *Core:* ein physikalisch vollständiges Set von ALUs, Registern, Caches…, führt physikalisch Programme aus
  - *Core* und *CPU* sind i.A. Synonyme



- *Thread:* ein logisches Set von ALUs, Registern, TLBs, etc.
  - Threads teilen sich physikalische Ressourcen eines Core
  - Threads führen logisch Programme aus



- ein *Processor* kann einen oder mehrere *Cores* enthalten:
  - Madison 9M enthält 1 Core pro Processor
  - Montecito enthält 2 Cores pro Processor

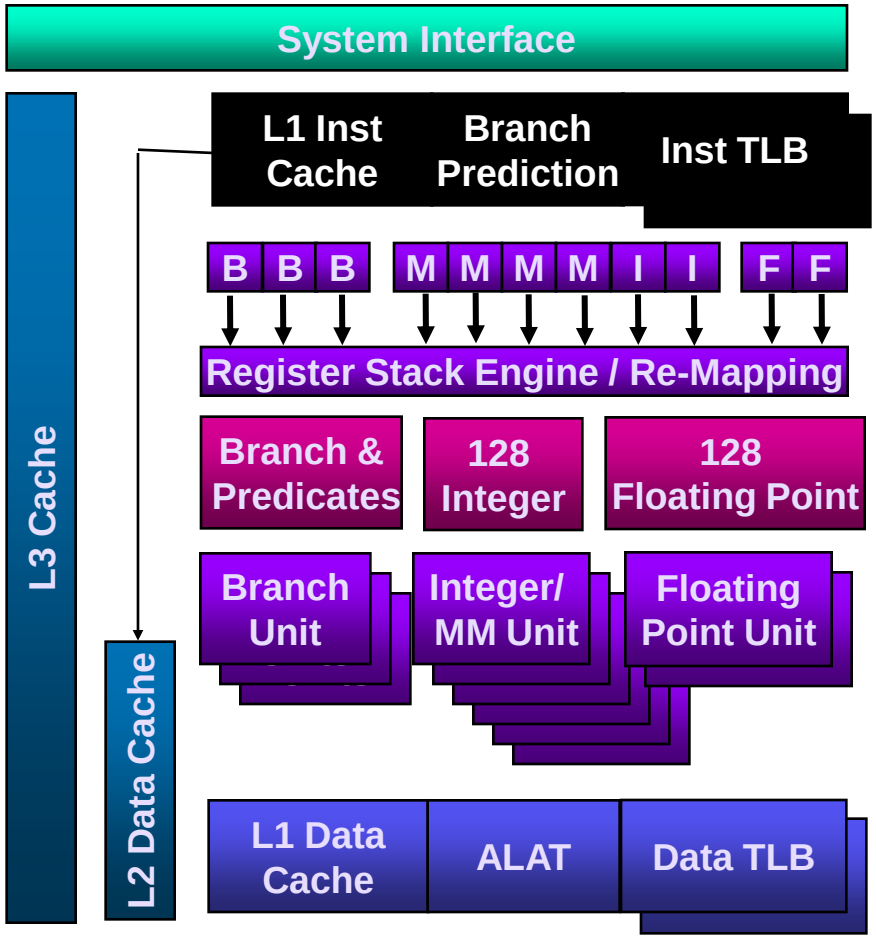


- ein physikalischer *Core* kann mehr als einen logischen *Thread* enthalten

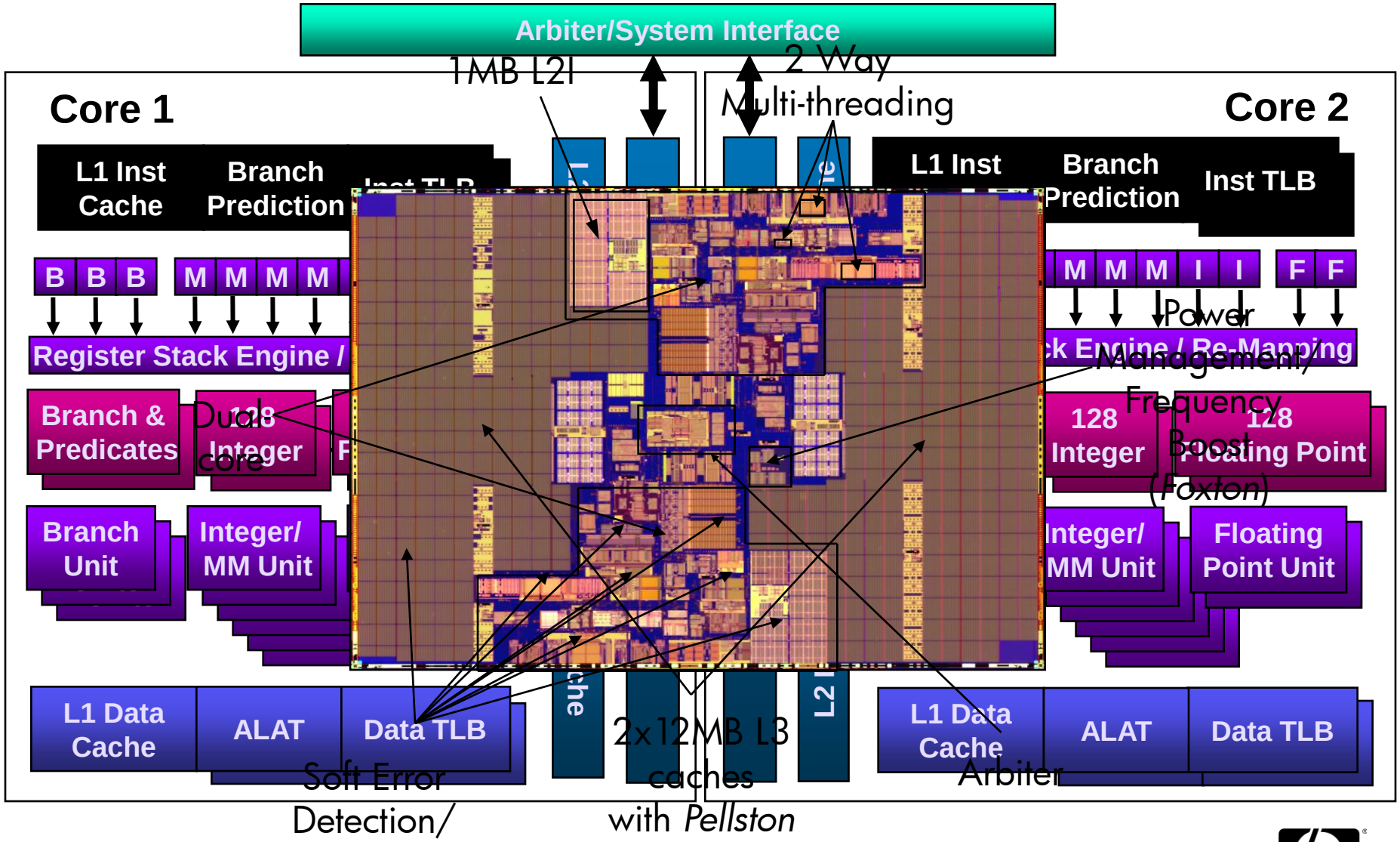
# Dual Core

- Zwei (fast) komplette CPUs auf einem Chip
- “2 klassische CPUs zusammengepappt”
- Jeder Core hat
  - eigenen Cache
  - eigene Processing Units
  - separaten Status
- Alle Cores teilen das Bus Interface
- Alle Cores arbeiten physikalisch gleichzeitig

# Die Itanium2-CPU



# Block-Diagramm des Montecito-Chips



# Hyperthreading (Montecito Multithreading)

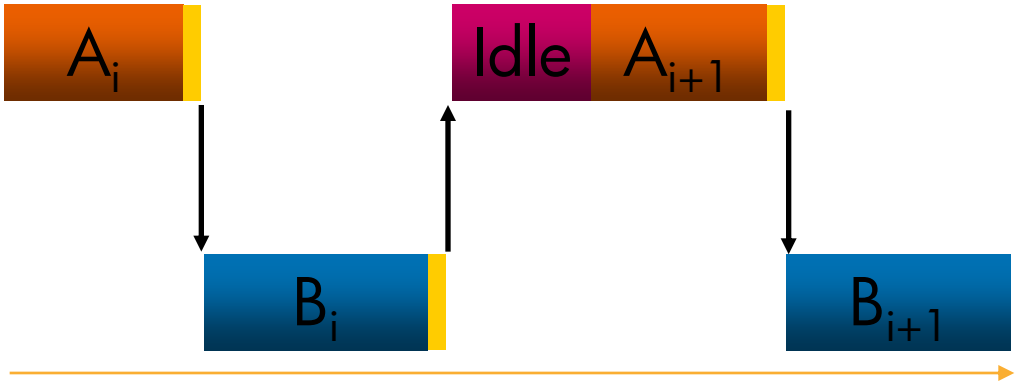
*Hyperthread:*

- Kein Software-multithreading!
- Idee: Erhöhung des Instruktions-Durchsatzes durch Nutzung von Idle Cycles während Memory Stalls
- ein Set von Informationen, welche den Status eines Cores beschreiben (User/Control Registers, IP, TLBs, ALAT, BR, etc.)
- teilt sich "core resources" mit anderen Hyperthreads
  - Zu jedem Zeitpunkt ist nur exakt ein Hyperthread eines Cores aktiv
- Core schaltet selbständig zwischen den einzelnen Hyperthreads um

# Montecito Multithreading

**Sequentielle Abarbeitung**

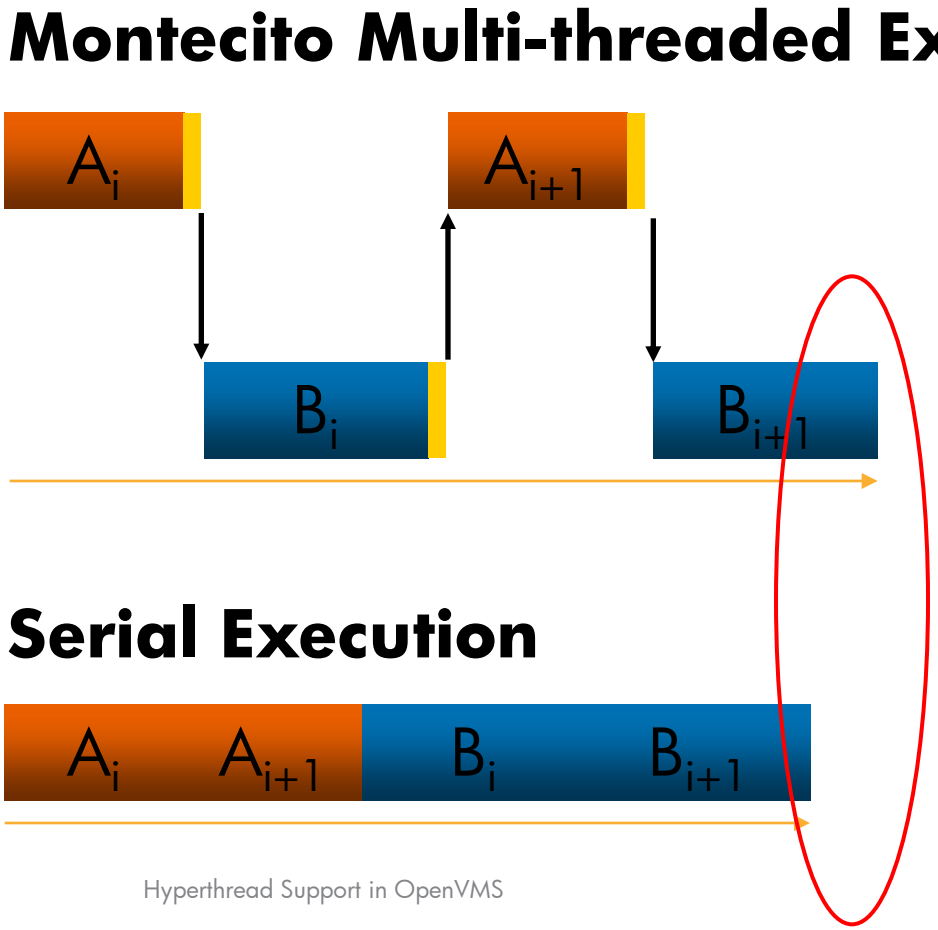

**Montecito Multithreaded Execution**




**Multithreading reduziert Stalls und erhöht den Durchsatz**

# “Hyperthreading” vs. “Dual Core”…

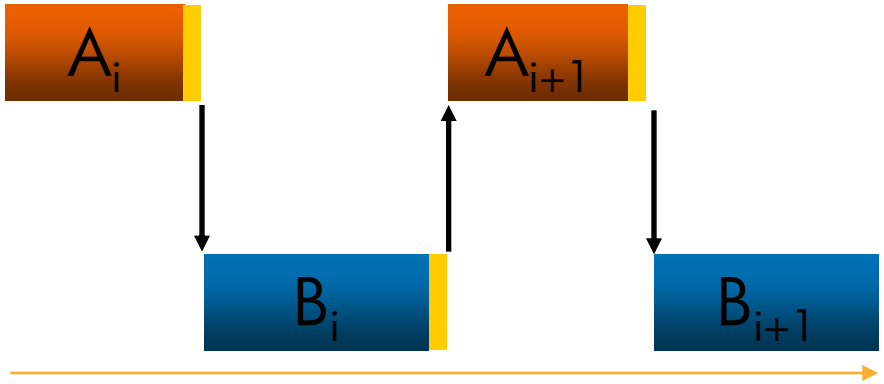
- Ein Core mit zwei Threads KANN eine bessere Performance als ein Core mit einem Thread haben.



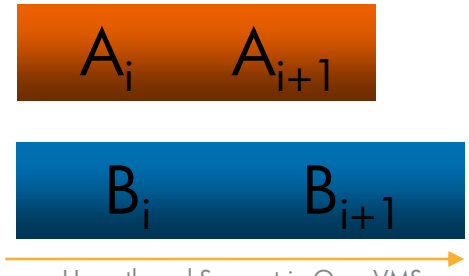
# …"Hyperthreading" vs. "Dual Core"…

- Ein Core mit zwei Threads wird NIEMALS eine bessere Performance als zwei Cores haben.

**Montecito Multi-threaded Execution**




**Execution auf zwei Cores**



# Dynamic Thread Switching

- Optimal: vorausschauendes Feststellen von langen Wartezeiten (Execution Stalls)
- Praxis: Reaktion auf Events, die in langen Wartezeiten resultieren
  - L3 Cache miss
  - Uncached accesses
- Timeout Events garantieren Fairness
- OS hat kein Wissen und keine Kontrolle über den gerade aktiven Hyperthread
- **`hint@pause`** ermöglicht Software-Einflussnahme

# Hyperthreading Support in OpenVMS…

- 3 Kategorien des Supports
  - Management/Informations-Abfrage
  - Reduzierung von “unnützen” Hyperthread Cycles
  - Scheduling

# …Hyperthreading Support in OpenVMS

- Auffrischung: 1 Prozessor/Gehäuse/Chip hat
  - 2 Cores (Montecito)
  - 4 Threads (Montecito)
- Ein Hyperthread wird in OpenVMS als CPU abgebildet
- CPUs innerhalb des gleichen Cores nennt man “CoThread CPUs”
- Cores, die sich im gleichen Prozessor/Gehäuse/Chip befinden, erfahren keine besondere Behandlung oder Namensgebung.

# Hyperthread CPUs in OpenVMS…

- Mapping von Hyperthread zu CPU
  - Erster Thread von allen Cores, dann weitere Threads
- **`SHOW CPU/BRIEF`** und **`/FULL`**
  - zeigt Mapping von CPU und CoThread CPU
- **`SET CPU/[NO]COTHREAD`**
  - Stoppt einen CoThreads des Core dieser CPU
- Accounting
  - Interval Timer Counter (ITC) tickt unabhängig von aktivem Thread
  - Process wird nur mit ½ der CPU-Zeit belastet, wenn der CoThread dieser CPUs aktiv ist

# Hyperthread CPUs in OpenVMS

```
$ SHOW CPU/BRIEF
System: DECVAX, HP rx4640   (1.40GHz/12.0MB)

CPU 0     State: RUN              CPUDB: 8202A000      Handle: 00005D70
          Owner: 000004C8         Current: 000004C8    Partition 0
          Cothd:        8

CPU 1     State: RUN              CPUDB: 820FDF80      Handle: 00005E80
          Owner: 000004C8         Current: 000004C8    Partition 0
          Cothd:        9

CPU 2     State: RUN              CPUDB: 820FFC80      Handle: 00005F90
          Owner: 000004C8         Current: 000004C8    Partition 0
          Cothd:       10

CPU 3     State: RUN              CPUDB: 82101A80      Handle: 000060A0
          Owner: 000004C8         Current: 000004C8    Partition 0
          Cothd:       11

CPU 4-7      CoThd 12-15
```

# Hyperthread Management in OpenVMS

- EFI Command: **`CPUCONFIG THREADS ON/OFF`**
  - Erfordert 2 Resets (EFI-Aufruf, Aktivierung des Kommandos)
- **`[SYSTEST]HTHREAD.EXE`**
  - Unsupported, aber nützlich (ähnlich **`RADCHECK`**)
  - Check/Modify des Firmware-Status von Hyperthreading
  - **`$HTHREAD -SHOW  $HTHREAD -ON  $HTHREAD -OFF`**
  - Änderung wirksam nach nächstem Reboot (single Reset)

# Reduzierung von “unnützen” Hyperthread Cycles

- Ein Hyperthread im Idle Mode verbraucht Cycles, die sein CoThread sinnvoll nutzen könnte
- Idle Loop
  - **`hint@pause`** zwischen Busy Checks
  - betrifft nicht Power Save Mode
- STOP/CPU
  - **`hint@pause`** während des Halt-Zustands
- Zukünftige Möglichkeiten
  - **`hint@pause`** während Spinlock Waits?
  - weitere Kompromisse, Optimierungen, Annahmen…

# Änderungen im Scheduler

- zwei Cores sind immer besser als zwei Hyperthreads auf dem gleichen Core, deshalb:
  - Prozesse auf CPUs ohne aktiven CoThread zuteilen
- Prozess erhält so automatisch alle Cycles des inaktiven Threads

# Scheduler: “Wer teilt sich einen Core?”

- Threads, die den gleichen Memory Space teilen (Kernel Threads innerhalb eines Prozesses)
  - Evtl. mehr Cache Hits, weniger Cache Fills (Stalls), dadurch bessere Performance
  - Aber weniger Stalls bedeuten auch weniger Thread Switches!
- Threads, die nichts miteinander zu tun haben
  - Mehr Cache Misses, dadurch größere Thread-Parallelität
  - Dies bedeutet aber auch: schlechtere individuelle Performance!
- Kompromiss liegt irgendwo dazwischen
  - Aber wie soll dies automatisch ermittelt werden???

# Benchmarks

| | | nothreads | 2-Hyperthreads | |
|---|---|---|---|---|
| | | ET | HET | speedup |
| | | | | |
| T1 | sieve | 46.445 | 40.703 | 2% |
| | prime | 42.703 | 46.866 | |
| | | | | |
| T2 | sieve | 46.913 | 36.046 | 7% |
| | ram | 41.899 | 43.418 | |
| | | | | |
| T3 | sieve | 50.195 | 50.221 | 0% |
| | sieve | 50.605 | 50.368 | |
| | | | | |
| T4 | ram | 42.315 | 29.775 | 30% |
| | prime | 42.207 | 27.788 | |
| | | | | |
| T5 | ram | 43.491 | 44.759 | -4% |
| | ram | 43.289 | 44.497 | |
| | | | | |
| T6 | prime | 42.831 | 44.586 | -4% |
| | prime | 42.864 | 44.763 | |

**sieve**: Integer sieve program with 5% L3 cache miss

**prime**: compute-bound program with no cache miss

**ram**: cache-friendly program with 60% L3 cache miss

**H(ET):** (Hyperthread) Execution Time in seconds

# Zusammenfassung

- Neue Features für das Hyperthread Management
  - **`SHOW CPU/BRIEF`** zeigt Thread-Info an
  - **`SET CPU/NOCOTHREAD`**
  - **`[SYSTEST]HTHREAD.EXE`**
- Neue Laufzeit-Features
  - Scheduler-Anpassung
  - Accounting
- Experimentieren mit der eigenen Applikation, um zu sehen, ob und wie Hyperthreads helfen


hp
invent