

Hyper-Threading on Dual Core Intel[®] Itanium[®] 2 Processors

Cameron McNairy
Itanium[®] Processor Architect
Intel[®] Corporation

Agenda

Multi-Threading Background

Dual Core Intel® Itanium® Hyper-Threading

- Sharing Resources
- Dedicated Resources
- A Targeted approach

Software Optimizations

- Idle
- Semaphores



Hyper-Threading Is/Is Not

IS

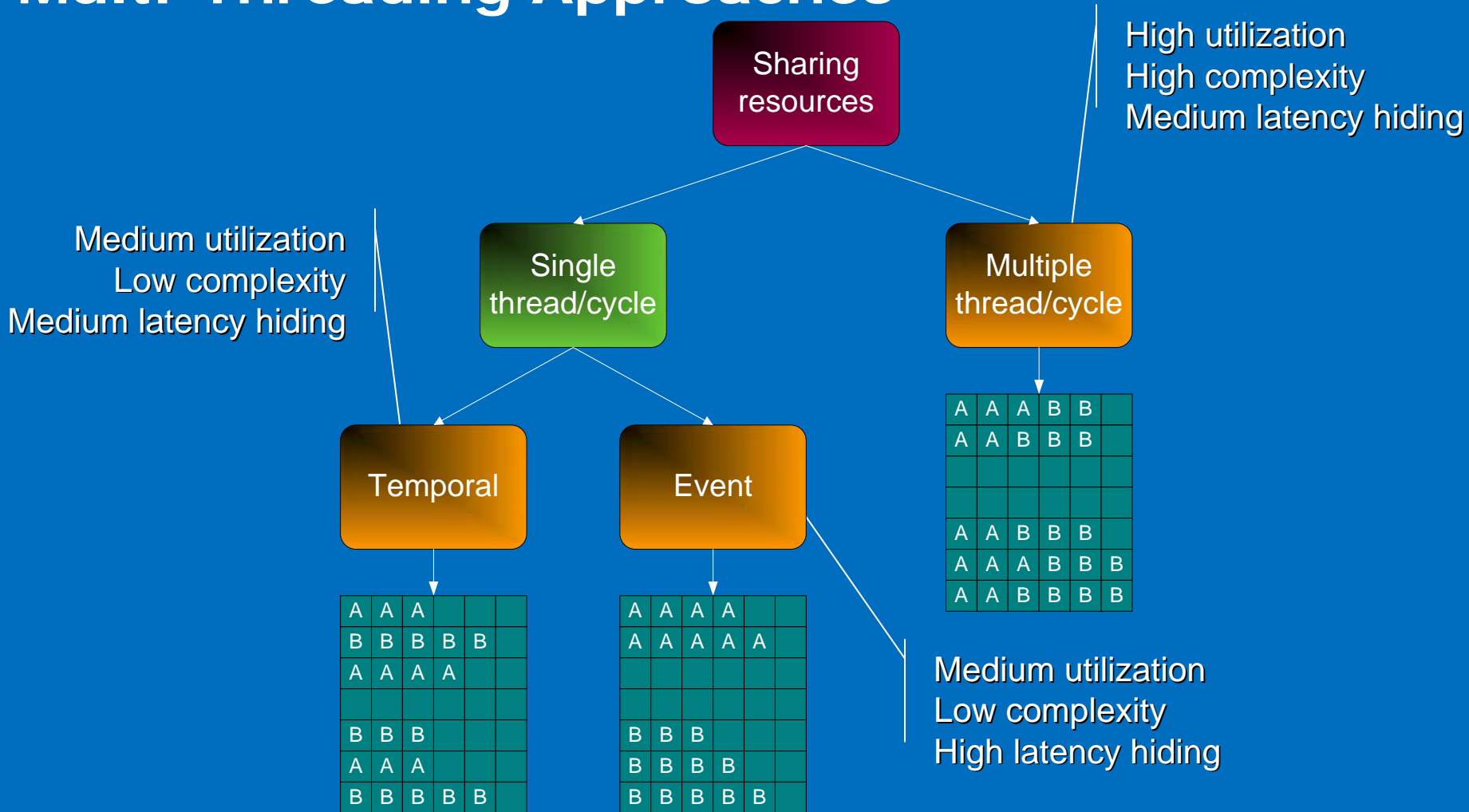
- Presenting full logical processor to OS
- All application and system state is unique to each thread
- An Intel branded name for what industry and academia typically call Multi-Threading
- Supported by any multi-processor capable OS

IS NOT

- User or application level threading known to windows or pthreads
 - An user or application level threading approach is not feasible due to various user and application level threading approaches



Multi-Threading Approaches



Pentium 4 uses SMT or Multiple thread/cycle approach
 Dual Core Itanium® 2 uses ALL approaches

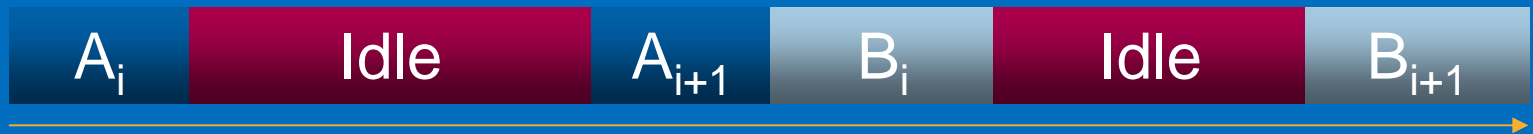
prise Group



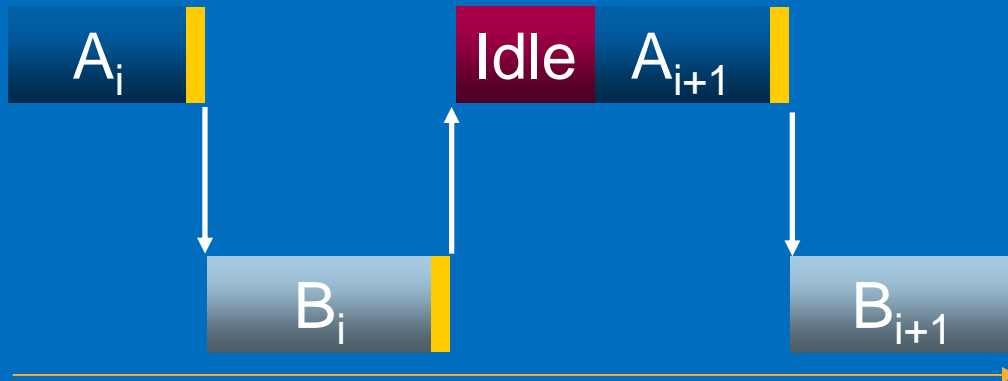
Gel

Event Multi-Threading

Serial Execution



Event Multi-Threaded Execution



Dual-Core Itanium® 2 Hyper-Threading

The core

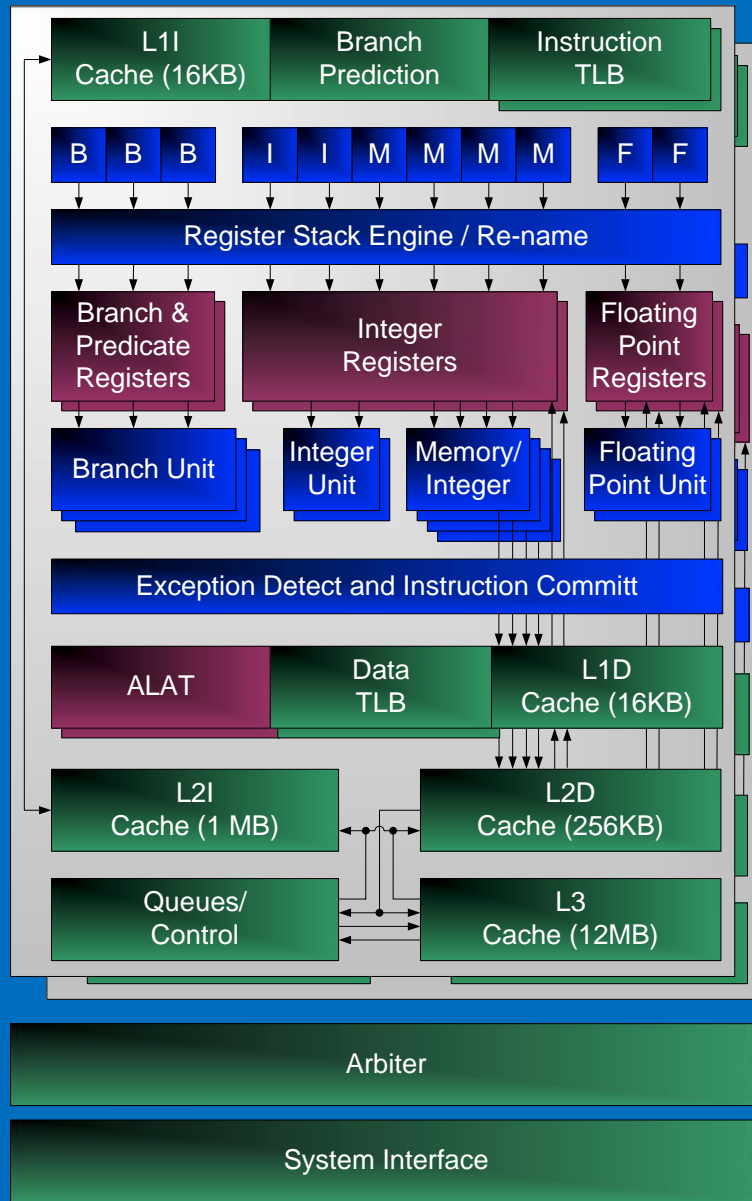
- Switch on event temporal multi-threading
- Only 1 thread owns core resources at any time
- A pipeline flush cancels all in-flight operations from a previous thread between switches
- 13 cycle cost for a thread switch

Memory hierarchy

- Simultaneous multi-threading
- Each thread compete for resource and are given equal access
- Competitively shared resources
 - I and D TLBs are tagged with thread identifier
 - Up to 32 TRs and at least 8 TCs for each thread
 - I and D Caches are tagged with thread identifier
 - Thread tracked only for register returns, ordering, and PMU
 - I Branch prediction

Dedicated resources

- Architecture state: AR, CR, GR, FR, PR, ...
- Micro-architecture structures such as ALAT, RSB, PIC, and PFS stack



Digital Enterprise Group

Gelato ICE Singapore 2006



Dynamic Thread Switching

Optimal: Determine when stalled for long latency operations

Practical: Speculate that a long latency event will stall execution and give software explicit switch control (`hint@pause`)

Switch Events

- L3 miss or UC accesses
 - Demand I or D but not prefetch I or D
 - HPW accesses that miss
- Time outs ensure fairness
- Low power requests

Hysteresis

- Urgency indicates a thread's ability to execute
- Compare urgency at miss and return events
- Latency from miss event to switch allows clustering
- Maximum switch thresholds ensure forward progress
- Switch may be delayed to ensure commit of pending state
 - i.e. long latency register writes, but NOT outstanding loads

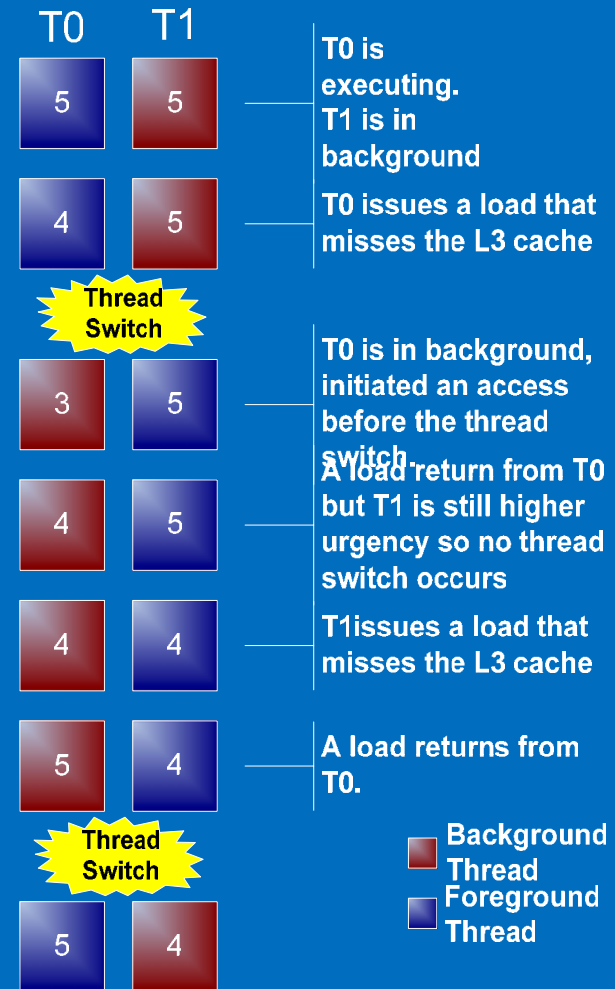
Digital Enterprise Group



Urgency Deep Dive

Each thread has an “urgency” counter that is compared at events and the thread with the greatest urgency executes

- Saturates at 0 and 5
 - L3 Miss event decrements urgency if <6 and >0
 - FSB/L3 data return increments urgency if <5
- Set to 5 at reset and >5 switch event
 - 6 = external interrupt occurred for background thread
 - 7 = thread switched out due to time slice expiration



Digital Enterprise Group

Code Examples – hint@pause

Replace:

Acquire:

```
(p1,p2)    ld.acq r3 = [z]           ;;           // try to acquire semaphore
            cmp r3 = r0           // ready?
(p2)       br Acquire           // failed, try again
```

With:

Acquire:

```
(p2)       ld.acq r3 = [z]           ;;           // try to acquire semaphore
            cmp.ne p1,p2 r3 = r0     // ready?
(p2)       hint@pause
            br Acquire
```

Replace:

Loop: br Loop

With:

```
Loop:      hint@pause
            br Loop
```

Digital Enterprise Group

Gelato ICE Singapore 2006



Code Examples – ALAT Switch Event

Replace:

Acquire:

```
(p2)    ld.acq r3 = [z]           ;;           // try to acquire semaphore
        cmp.eq p1,p2 = r3, r0    // ready?
        br Acquire              // failed, try again
```

Critical:

...

With:

Acquire:

```
(p1)    ld.acq r3 = [z]           ;;           // try to acquire semaphore
        cmp.eq p1,p2 = r3, r0    // ready?
        br Critical
```

Loop:

```
(p2)    ld.c.nc r3 = [z]         // sleep until awakened by ALAT
        cmp.eq p1, p2 = r3 , r0  // read variable and allocate ALAT
        ;;                       // ready?
        hint@pause              // no, sleep
        br Loop                  // no, branch
        br Acquire              // try to get the semaphore again
```

Critical:

...

Digital Enterprise Group



Hyper-Threading Implications

All instruction prefetching prefetches are canceled at thread switch

Branch prediction (L1I BR, PHT, L2B) are shared between threads

- L2B incorporates thread in access hashing algorithms

No high water limits on competitively shared resources except TLBs

- Each thread gets a minimum of 16 entries L2D TLB entries
- Each thread gets a minimum of 1 L2I TLB entry

TLB behavior

- The thread identifier is treated as an additional bit of VA
- L1I TLB allows PA alias between threads, but others prohibit PA alias

L1D behavior

- st.rel/semaphores on T0 may invalidate L1D line/request
 - An outstanding L1D fill to st.rel/semaphore line for T1
 - The st.rel/semaphore hit line brought in by T1
 - L1D fill buffers may also be invalidated

Dual-Core Itanium® 2 9000 PMU

12 performance counters per thread

- 48 bits of resolution
- 200+ events can be counted

Where possible, the events are thread and core aware

