Computer Structure

Multi-Threading

Lihu Rappoport and Adi Yoaz
Thread-Level Parallelism

- Multiprocessor systems have been used for many years
  - There are known techniques to exploit multiprocessors
  - Chip Multi-Processing (CMP): multiple Cores on the same die

- Software trends
  - Applications consist of multiple threads or processes that can be executed in parallel on multiple processors

- Thread-level parallelism (TLP) – threads can be from
  - The same application
  - Different applications running simultaneously
  - Operating system services

- Increasing single thread performance becomes harder
  - And is less and less power efficient
Multi-Threading Schemes

- **Multi-threading**: a single processor executes multiple threads
- **Most applications’ sustained throughput << the core’s peak IPC**
- **Running ≥2 threads increases the utilization of core resources**
  - When one thread is stalled (on a cache miss, branch misprediction, or a long dependency), the other thread gets to use the free resources
  - The RS can find more independent μops

- **Switch-on-event multithreading**
  - Switch threads on long latency events such as last level cache misses
  - Works well for server applications that have many cache misses
  - Does not cover for branch mispredictions and long dependencies

- **Simultaneous multi-threading (SMT)**
  - Multiple threads execute on a single processor simultaneously w/o switching
  - When one thread is stalled / slowed down, other thread gets more resources
  - Makes the most effective use of processor resources
Hyper-Threading Technology

- Hyper-Threading Technology implements SMT

- Two logical processors within one physical core
  - Sharing most execution/cache resources using SMT
  - Look like two processors to the SW (OS and apps)

- Each logical processor executes a software thread
  - Threads execute simultaneously on one physical core

- Each logical processor maintains its own arch. state
  - Complete set of architectural registers
    - General-purpose registers, Control registers, Machine state registers, Debug registers
  - Instruction pointers

- Each logical processor has its own interrupt controller
  - Handles the interrupts sent to the specific logical processor
SMT: A High-level View of the Pipeline

- Each pipe-stage is occupied by one of the threads
- Buffers are either Replicated or partitioned
SMT: Thread Select Points

- Pipeline arbitration points select the thread gets to use a resource in a given cycle
  - If both threads are active and require the resource
    - Use a “ping-pong” scheme to switch between the two threads on a cycle-by-cycle basis
  - If only one thread has work to do and requires the resource
    - Allow that thread to get the full resource bandwidth

- The thread select scheme
  - Achieves fairness between the threads
  - Gives full bandwidth to one thread in case the other thread does not require the given resource
SMT: Thread Select Points

- The key arbitration points in the pipeline select in a given cycle which thread gets to
  - Read the I$ to Fetch instruction
  - Use the Decodes for decoding instructions into μops
  - Allocate μops to the ROB, store/load buffer, and the RS
  - Retire μops and reclaim their resources

- The RS is not a thread arbitration point
  - The RS schedules μops to the execution units based on readiness and based on age (allocation time)
  - Regardless of which thread they belong to
SMT Principals

- **When one thread is stalled, the other thread can continue to make progress**
  - Independent progress ensured by either
    - Partitioning buffering queues and limiting the number of entries each thread can use
    - Duplicating buffering queues

- **A single active thread running on a processor with HT runs at the same speed as without HT**
  - Partitioned resources are recombined when only one thread is active
Physical Resource Sharing Schemes

- **Replicated Resources**: e.g. CR3
  - Each thread has its own resource

- **Partitioned Resources**: e.g. μop-Cache
  - Resource is partitioned in MT mode, and combined in ST mode

- **Competitively-shared resource**: e.g. RS entries
  - Both threads compete for the entire resource

- **HT unaware resources**: e.g. execution unit
  - Both threads use the resource

- **Alternating Resources**
  - Alternate between active threads
  - if one thread idle
    - the active thread uses the resource continuously
Front End Resource Sharing

- The Instruction Pointer is replicated
- The small page iTLB is partitioned
- Branch prediction resources are mostly shared
- The I$ is thread unaware (thus competitively-shared)
- The pre-decode logic and the instruction decoder logic are used by one thread at a time
- The μop-Cache is partitioned
- The μop Queue is partitioned
Back End Resource Sharing

- **Out-Of-Order Execution**
  - Register renaming tables are replicated
  - The reservation stations are competitively-shared
  - Execution units are HT unaware
  - The re-order buffers are partitioned
  - Retirement logic alternates between threads

- **Memory**
  - The Load buffers and store buffers are partitioned
  - The DTLB and STLB are competitively-shared
  - The cache hierarchy and fill buffers are competitively-shared

From the Optimization Manual
Single-task And Multi-task Modes

- **MT-mode (Multi-task mode)**
  - Two active threads, with some resources partitioned as described earlier

- **ST-mode (Single-task mode)**
  - ST0 / ST1 – only thread 0 / 1 is active
  - Resources that are partitioned in MT-mode are re-combined to give the single active logical processor use of all of the resources

- **Moving the processor from between modes**

  ![Diagram showing the transition between modes](Diagram)

  - Thread 0 executes HALT
  - Thread 1 executes HALT
  - ST0, ST1, MT transitions

  *From the Optimization Manual*
The OS should implement two optimizations:

- **Use HALT if only one logical processor is active**
  - Allows the processor to transition to either the ST0 or ST1 mode
  - Otherwise the OS would execute on the idle logical processor a sequence of instructions that repeatedly checks for work to do
  - This so-called “idle loop” can consume significant execution resources that could otherwise be used by the other active logical processor

- **On a multi-processor system**
  - OS views logical processors similar to physical processors
    - But can still differentiate and prefer to schedule a thread on a new physical processor rather than on a 2nd logical processors in the same physical processor
  - Schedule threads to logical processors on different physical processors before scheduling multiple threads to the same physical processor
  - Allows SW threads to use different physical resources when possible