Multi-Threading

Hyper-, Multi-, and Simultaneous Thread Execution
Apple
Now with Dual Core Technology
More Performance

- Increasing processor performance
  - Pipelining
  - Branch prediction
  - Super-scalar execution
  - Out-of-order execution
  - Caches
  - Hyper-Threading
    - Intel’s implementation of SMT (simultaneous multithreading technology)
    - Introduced in the Foster MP-based Xeon and the 3.06 GHz Northwood-based Pentium 4 in 2002
Threading

- A thread is the smallest sequence of programmed instructions that can be managed independently.
- A thread is composed of multiple instructions. Threads exist within the same process and could share the same resources while different processes do not share the same resources.
- Multi-threading is a type of execution model that allows multiple threads to exist within the context of a process such that they execute independently but share their process resources. A thread maintains a list of information relevant to its execution including the priority schedule, exception handlers, a set of CPU registers, and stack state in the address space of its hosting process.
Overview

- Hyper-threading
  - Uses processors resources in a highly efficient way
  - Consists of two logical processors for every physical core
  - Separate threads can be executed on each logical processor simultaneously
    - Here is a dumb video
- Multi-threading is enhanced by Multi-core technology which allows the parallel execution of software threads across multiple processor cores.
Advantages and Disadvantages

Advantages of Hyper-threading:
- Performance boost by as much as 30% when compared to identical processing core without hyper-threading

Disadvantages of Hyper-threading:
- Possible increase in core size of about 5 percent caused by the duplication of certain sections of the CPU core (Intel's claim)
- Overall power consumption is higher
- Increases cache thrashing (the repeated displacing and loading of cache lines), which ARM claims could be as much as 42%.
Simultaneous Multithreading (SMT)

Key Idea: Exploit ILP across multiple threads

- Convert thread-level parallelism into more ILP
- Exploit the following features of modern processors:
  - Multiple functional units
    - Modern processors typically have more functional units available than a single thread can utilize
  - Register renaming and dynamic scheduling
    - Multiple instructions from independent threads can co-exist and co-execute
- Multi-threading trades latency for throughput
  - Improves combined latency of threads
  - Sharing the processor degrades latency of individual threads
Hyper-Threading Architecture

- Makes a single physical processor appear as multiple logical processors.
- Each logical processor has a copy of architecture state.
- Logical processors share a single set of physical execution resources.
- From an architecture perspective we have to worry about the logical processors using shared resources.
  - Caches, execution units, branch predictors, control logic, and buses.
HT vs. Dual Processors

- Dual Processor
  - System resources are duplicated

- Hyper-Threading
  - Architectural state is duplicated
    - Data, segment, control, and debug registers
  - Advanced Programmable Interrupt Controller (APIC)
  - Resources shared by the logical CPUs
    - Execution engine, caches, system-bus interface and firmware
    - This can increases the performance of the processor up too 30%
Implementing Multithreading

- Two main questions when Multithreading:
  - Thread scheduling policy
    - *When to switch threads?*
  - Pipeline partitioning
    - *How do threads share the pipeline?*

- The choices depend on:
  - How much sacrifice you are willing to take in single thread performance
  - What kind of latencies you are willing to tolerate
OS Support

- The OS must support HT since it manages the threads
Types of Multi-Threading

- Coarse-grain multithreading
  - Only switch on long stall (e.g., L2-cache miss)
  - Simplifies hardware, but doesn’t hide short stalls (e.g., data hazards)

- Fine-grain multithreading
  - Switch threads after each cycle
  - Interleave instruction execution
  - If one thread stalls, others are executed
Types of Multithreading

- Coarse-grain multithreading (CGMT)
- Fine-grain multithreading (FGMT)
- Simultaneous multithreading (SMT)
## Another Look

<table>
<thead>
<tr>
<th></th>
<th>Superscalar</th>
<th>Fine-Grained</th>
<th>Coarse-Grained</th>
<th>Multiprocessing</th>
<th>Simultaneous Multithreading</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thread 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thread 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thread 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thread 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Idle slot</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Superscalar**: High performance with high clock speed.
- **Fine-Grained**: Uses multiple processors with fine-precision timing.
- **Coarse-Grained**: Uses multiple processors with coarse-precision timing.
- **Multiprocessing**: Uses multiple processors simultaneously.
- **Simultaneous Multithreading**: Uses multiple threads simultaneously.

*Time (processor cycle)*
Fine-Grain (FGMT)

- Sacrifices significant single thread performance
- Tolerates latencies (L2 misses, mis-predicted branches)
- Thread scheduling
  - Round-robin thread switching after each cycle
- Pipeline partitioning
  - No flushing, dynamic
  - Multiple threads in pipeline at once
- Lots of threads needed
- Finding a home in GPU’s
Software Implementation

- From a software aspect - synchronization of objects is often required when implementing software apps with multi-threading.

- These objects are used to protect memory from being modified by multiple threads at the same time.

- A mutex is one such object. It is a lock which can be locked by a thread, and any successive attempt to lock it by another thread or by the same thread, will be blocked until the mutex is unlocked, thus keeping an item from being accessed more than once at the same time.

- Computer Scientists like to refer to pieces of code protected by objects like mutexes as Critical Sections.
Software Implementation

- It is best to keep Critical Sections as short as possible to allow the application to be as parallel as possible, because the larger the Critical Section the greater the chance that multiple threads will access it at the same time, thus causing stalls.

- Intel provides several helpful templates to make implementation easier.
Conclusions

- Hardware support for Multi-, hyper-, and simultaneous threading exists in all major processors
- Operating system must support multi-threading
- Types of multi-threading
  - Coarse-grain
  - Fine-grain
  - Simultaneous
- Individual latencies are sacrificed for the good of the entire program (or process)