Week 4: Out of order execution, symmetric multiprocessing and simultaneous multithreading-2

Out of order execution (http://en.wikipedia.org/wiki/Out-of-order_execution) is a technique where the processor will attempt to re-arrange the instructions queued up for it, such that it can complete the batch of instructions in the shortest possible time, with the fewest instances of pipelines stalling.

Symmetric multiprocessing (and multi-core processors)

Symmetric Multiprocessing (http://en.wikipedia.org/wiki/Symmetric_multiprocessing) (SMP) is a technique where there is more than one processing unit (of identical type), so that more than one instruction can be executed at the same time. Virtually all consumer multi-processor systems are designed in this way.

When transistors on silicon were somewhat large, this involved having more than one CPU socket on the motherboard, with each CPU holding one processor core. Nowadays, CPU packages can store many cores - these are referred to as multi-core processors.

However, there are issues with symmetric multiprocessing:

- The processor units need to communicate with each other, so that they can coordinate how processing is to be performed between them.
- The question of memory access needs to be answered: will they share the same pool of memory? (In conventional systems, the answer to this question is "yes".) How will memory fetching transactions be rationed so that the two processors don't trample over each other's requests?
- Each processor has its own cache, and this complicates memory access further. How will the processors 'know' what's in each other's cache - and how will it know if the other processor has since changed that data, based on an instruction it performed recently? This is known as **cache coherency** (http://en.wikipedia.org/wiki/Cache_coherency).
- The computational workload must be easily parallelisable; that is, able to be broken up so that two separate processing units can work on it at the same time.

  You can think of parallelisable workloads with the analogy of trying to cook a meal: if a meal takes 60 minutes to cook in the oven, it doesn't mean you can split it in half, and cook each half in a separate oven for 30 minutes. *Cooking is not parallelisable.*

  However, the more kitchen hands available for chopping onions, the faster onions can be chopped. This workload is very parallelisable.

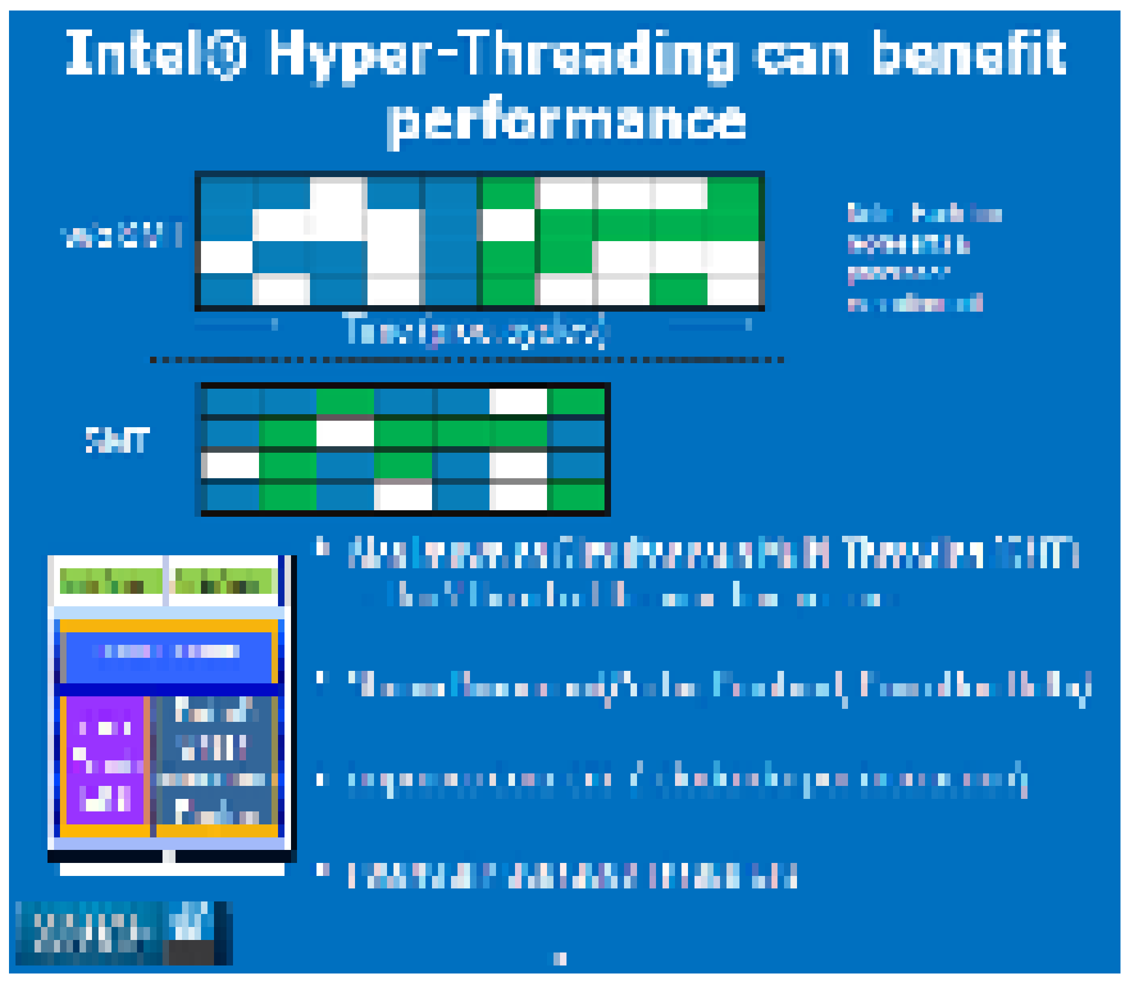
For these reasons, adding multiple processors doesn't necessarily lead to a doubling (or quadrupling) of performance for a particular task. However, nowdays it's fairly uncommon for mainstream computers to concentrate on performing single tasks - instead, these systems are constantly *multitasking*; running more than one process at once.

In this use case, multi-core processors can provide significant benefits; multiple (active) tasks can be run at once with little or no performance penalty, as for example, in a 4-core configuration, up to four processes can be active and each have their own execution core.

Simultaneous multithreading

Simultaneous multithreading (http://en.wikipedia.org/wiki/Simultaneous_multithreading) refers to the process of exploiting pipeline stalls in the processor, to feed in another thread of execution. The most well-known implementation of this is Intel's **Hyper-Threading Technology** (http://en.wikipedia.org/wiki/Hyper-Threading).

This technique seeks to fill up these bubbles in the pipeline by scheduling another execution thread in them. What this means is that a single physical SMT-capable CPU core will make itself appear to the operating system as two logical processors, and it will interleave the processes scheduled for each of these two processors such that any potential bubbles from one one logical processor can be used to execute the instructions of the other.



A visual representation of Simultaneous Multithreading. Diagram : Intel

Although this isn't as good as having two dedicated processor cores, it does make each individual core more efficient.