# Coursework 2 Multi-core Processing

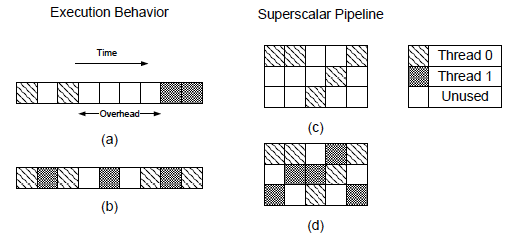
Intel came with their first single chip microprocessor, the Intel 4004, in 1971. The processor had a clock rate on 740 kHz and a 640 byte addressable memory. Since then, the hunt for speed has i.a. focused on improving the clock frequency and increasing memory performance with the use of caches. This led to the latest Intel single core multiprocessor in 2000, the Intel Pentium 4, which had a up to 512 KB L2 cache and a clock rate on 3,8 GHz[[1]](#footnote-0). The various improvements added complexity, which in addition to the incongruence between processor frequency and main memory access times, meant that designers looked for alternative ways to do processor design[[2]](#footnote-1). The development of the multi-core processors were lead by Intel and AMD who both came up with their first multi-core processor in 2006, which (as the name indicates) worked with two processors that each handled multiple threads in parallel. When looking at the modern day multi-core processors in personal computers, they have up to 6 cores on a single die, but are still running with L3 cache and the clock rate has not increased much either (relatively speaking).

One of the prime technologies behind the multi-core processor is the usage of threads. The threads within each core can work with different schemes, some of which are coarse-grained multithreading, fine-grain multithreading and simultaneous multithreading.

The coarse-grained multithreading will only execute one thread at the time, whilst other threads are on hold (see picture 1a). When a long-latency event occurs (such as a cache miss), the processor will switch to one of the threads on hold, allowing them to run instead of wasting time while waiting.

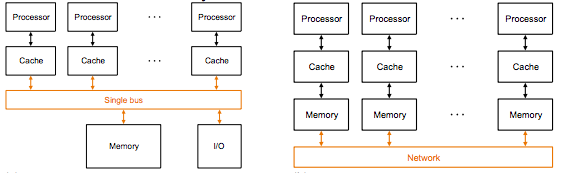
With fine-grained multithreading, the processor will equally switch thread in a long-latency event, but the former thread’s data is not wholly out of the processor, which allows the threads to be swapped more frequently and faster than is the case with coarse-grained multithreading (see picture 1b).

Unlike the previous types of multithreading (both time-sliced), simultaneous multithreading allows the processor to execute more than one thread at the time – this means that different instructions from several threads can be scheduled for different functional units at the same time (see picture 1c and d). The simultaneous multithreading is used by the IBM Power5 and by Intel in their Hyperthreading scheme[[3]](#footnote-2).



*Picture 1: Ma, 2007: 15*

When multiple, parallel threads run processes on multiple cores, the cores need a way to communicate each other. The main ways to do this is by the shared- or distributed memory model. The shared memory model makes use of a single bus approach whereas the distributed memory model is an interconnected network[[4]](#footnote-3).



*Picture 2: Schauer, 2008: 5*

A multi-core can be either homogeneous or heterogeneous. A homogeneous multi-core system will have all cores looking the same. This means that all cores can process the exact same functions, which makes it easier to produce. In contrast, a heterogeneous multi-core system will be able to have different functions; cache, frequency and memory model on each core. This has an advantage in the usage of the multi-core processors, since they may be more efficient. However, this does demand that the program running has been coded for the purpose of being run on a heterogeneous core system (parallel programming)[[5]](#footnote-4).

Linux and Windows now support CPU chips with up to 8 cores. The Intel i7 chip has 4 cores and 8 threads. This allows the operating system to run two threads on each core, which speeds up the execution of processes. The performance of multi-core processors have dramatically changed over time due to the ability to cram more transistors onto the microchip. The cost of adding more transistors to a chip has decreased and with new technology the layers in the board have been made thinner allowing more transistors onto the same physical space. This means that new processors are the exact same size however they now have more cores. The first dual core CPU’s arrived on the market in 2005. Operating systems such as Windows used to see a dual core CPU as two completely separate processors. This meant that when the processes were being scheduled on the cores that the CPU was not reaching its maximum performance due to the OS not sharing the task between cores. It would usually delegate the task to one processor or another. Some older machines would have a separate OS domain for the second core so the first core would only run the operating system and critical processes whereas the second core would handle all the other processes. Nearly all operating systems available now support multi-threading.

Multi-tasking on a computer is quite difficult because every process needs the CPU and the CPU can only run one process at a time, meaning that a single core CPU would have to swap processes constantly. This is why we have multi-cored processors, so we are able to multi-task more efficiently.

A single-user multitasking operating system such as Microsoft Windows or MAC OS is able to utilize a multi cored processor. As mentioned previously, an operating system’s main job with the CPU is process management. Making sure that each process gets the time it needs on the CPU, and also to make sure that each cycle of a core is being used for real work and is not wasted. The CPU’s job is important; it must be able to arrange the execution of processes in such a way that it appears as though they are happening at the same time. (<http://computer.howstuffworks.com/operating-system6.htm> - secontion on types of operating systems.)

The operating system controls scheduling and deals with processes and threads, depending on the operating system. Scheduling can be done in different ways and overall is usually a mixture of different methods; dependent on which processes are being run and the state of the machine. The CPU may be processing threads which all have equal priority, and therefore is running in a “round robin” mode, and then someone presses a keyboard key; an interrupt will then be sent to the operating system which will then tell the CPU that this process has priority and that it should immediately be processed. Interrupts are flags which can be sent to the CPU to tell it that this applications needs CPU time, the operating system can choose whether to ignore it or follow the interrupt.

(<http://www.cs.rutgers.edu/~pxk/416/notes/07-scheduling.html> and lecture slides on CPU scheduling).

Processes occupy space in the RAM. They can also use memory that is found in the Operating System and CPU memory space by using stacks, queues and registers. The operating system will allocate all of the programs a certain amount of a CPU cycle, then the operating system makes copies of all the registers, stacks and queues that the process used and remembers the point where the process stopped executing. It will then load all of the data structures used by another process and gives it its CPU time and repeats this for every process. On a multi-core processor, the operating system has multiple cores to work with. ( <http://my.safaribooksonline.com/book/information-technology-and-software-development/9788131733097/operating-system/ch07lev1sec5> and lecture slides on processes).

Information about the various processes are stored on a data package known as the process control block. This data package contains: the process ID (so that processes can be identified); memory pointers to the program and where the data was last processed; the contents of the memory register that the process needs; any interrupts, flags of switches and their state; pointers to the memory location – the upper and lower bounds of the memory that the process needs; a list of any files that are opened by the process; the priority the process has and any hardware devices that the process group requires. The operating system keeps track of this information using the process control block. (<http://computer.howstuffworks.com/operating-system6.htm> - process control block section paragraph 1).

According to Moore’s law the number of transistors will double every second year. (Stallings, 2012: 30) So far, Moore has been roughly right but it seems that we have now reached a point where it is no longer possible for the transistors to double. (Schauer, 2008: 11) The technology (hardware-wise) is up to date but there is a lacking on the software side (Ibid.). That means we will soon reach the limit in what we are able to do with the technology available. At the moment, it appears as the cores will “starve” for data since the amount of cores may not be fully used due to the lack of software. (Ibid.)

Taking a look at Amdahl’s law in comparison to Moore it says that when copying a system it will run the exactly same way as the original. The speed will also increase when installing more systems but even though the speed will reach a limit. (Stallings, 2012: 56) Amdahl’s law says that the limit for the increasing speed hits relatively fast and will afterwards decrease. This is evidence of the limit to the software and not hardware because the programmers are not yet familiar with utilising all of the potential hardware. (Schauer, 2008: 10) So the two laws actually conflict each other because even though the number of transistors will double every second, it will not help to increase the speed of the full use of the systems.

The problem with scalability is that even by using the super-scalar concept the software will still not be able to support all of the cores and furthermore not use the full potential of the hardware. By duplicating the systems, other problems, such as the communication between the cores and the power, will occur. As Schaur points out, the number of cores will reach their limit at 32, as this is the point where buses will be overloaded and no longer be able to communicate with each other. (Schauer, 2008: 4) Also the use of that many cores takes a lot of energy, which could result in a shut down of the computer because of a heating problem.

The near future of multi-processing could go down several paths. The simplest is to add more and more cores, but the advantage gained by adding cores diminishes as the overhead of synchronising work across them increases. This limitation led to the idea of ‘heterogeneous computing’, where several specialised processors types are used, each built to handle specific tasks efficiently. Typically, this is in the form of having a GPU (Graphics Processing Unit) to handle graphics rendering and perform calculations on large data sets whilst a general purpose CPU handles standard housekeeping and operating system tasks. There is also possibility of alongside trying to speed up processing by adding more power, making current processing methods more efficient could allow the same methods to be carried out in less time, use less power, and require less materials to create components. This could give current techniques more scope to improve before new ideas need to be developed.

As conventional computers follow Moore’s Law, we will soon reach the point where transistors are on an atomic scale and cannot be made any smaller. Entirely new techniques would then need to be explored. One such technique is quantum computing, which aims to exploit the unique behaviours of atoms to perform processing and represent memory. Whereas standard computers store information using a series of bits which can have the value 1 or 0, quantum computers store information using ‘qubits’. These exploit quantum superposition to be able to simultaneously represent a 1, 0, and all points in between. This parallelism could lay the foundation of processing where millions of calculations can be performed in the time it takes a conventional computer to perform one. Several quantum computers have been demonstrated to work and have been able to compute algorithms, such as Shor’s algorithm for integer factorisation (used in cryptography), on relatively small 7 qubit computers[[6]](#footnote-5). Research is currently in its early infancy and doubts have been voiced by academics about the scalability of several notable designs (such as D-Wave Systems’ “first commercially available quantum computer”), and whether they can be proven to be true quantum computers whilst much about the field is still unknown[[7]](#footnote-6).

Reference List

Schauer, S. (2008): *Muliticore processor - A Necessity.*

Stallings, W. (2012): *Operating Systems - Internal and Design Principles.* 7. edition

Stallings, W. (2010): *Computer Organization and Architecture - Designing for Performance.* 8. edition

**Word Count: 1,976**

1. Princeton.edu; “*List of Intel icroprocessors”,* https://www.princeton.edu/~achaney/tmve/wiki100k/docs/List\_of\_Intel\_microprocessors.html [↑](#footnote-ref-0)
2. Ma, Nicholas; “*Modelling and Evaluation of Mulit-core Multithreaded Processor Architectures in SystemC”*, Queen’s University, 2007 [↑](#footnote-ref-1)
3. Ma, Nicholas; “*Modelling and Evaluation of Mulit-core Multithreaded Processor Architectures in SystemC”*, Queen’s University, 2007 [↑](#footnote-ref-2)
4. Schauer, Brian: *“Multicore Processor – a Necissity”,* ProQuest, 2008 [↑](#footnote-ref-3)
5. Schauer, Brian: *“Multicore Processor – a Necissity”,* ProQuest, 2008 [↑](#footnote-ref-4)
6. Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance. Nature 414, 883-887 (20 December 2001). <http://www.nature.com/nature/journal/v414/n6866/full/414883a.html> [↑](#footnote-ref-5)
7. My visit to D-Wave: Beyond the roast-beef sandwich. Scott Aaronson. http://www.scottaaronson.com/blog/?p=954 [↑](#footnote-ref-6)