**Case Study**

**1. Intel and AMD: Architectural Differences.**

When comparing AMD vs Intel CPUs, we must consider that two design decisions have a big impact on performance, scalability, and performanceper-dollar: Interconnects and microarchitecture.

AMD's Infinity Fabric allows the company to tie together multiple dies into one cohesive processor. Think of this as numerous piece of a puzzle that come together to form one larger picture. The approach allows the company to use many small dies instead of one large die, and this technique improves yields and reduces cost. It also grants a level of scalability that Intel might not be able to match with its new mesh interconnect inside its HEDT chips, and it undoubtedly takes the lead over Intel's aging ring bus in its desktop processors.

AMD first paired that advantage with its Zen microarchitecture, designed from the ground up for scalability, yielding an explosive 52% increase in instructions per clock (IPC) throughput over AMD's previous-gen 'Bulldozer' chips. The Zen 2 microarchitecture another 15% improvement to IPC. Paired with the 7nm process, AMD lunged forward another (up to) 31% in per-core performance (a mixture of frequency and IPC). Zen 3 brings another 19% jump in IPC, giving AMD its largest single step forward in the post-Bulldozer era.

The move to the Zen 2 architecture brought AMD's processors to nearparity with Intel's finest in terms of per-core performance. That's largely because Intel is stuck on 14nm, and its architectures are designed specifically for the nodes they are built on. That means promising new Intel microarchitectures can only ride on smaller processes, like 10nm, leaving the company woefully unprepared for its prolonged issues productizing 10nm products.

Zen 3 gave AMD a sizable lead in per-core performance, an incredibly important metric that quantifies the speed of the most important building block in a chip design. Intel's Rocket Lake chips take huge steps forward in per-core performance, leaving both companies on a relatively even playing field in terms of per-core performance.

Intel's 12th-Gen Alder Lake chips bring the company's hybrid x86 architecture, which combines a mix of larger high-performance cores paired with smaller high-efficiency cores, to desktop x86 PCs for the first time. The Golden Cove architecture powers Alder Lake's 'big' highperformance cores, while the 'little' Atom efficiency cores come with the Gracemont architecture. Intel etches the cores on its 'Intel 7' process, marking the company's first truly new node for the desktop since 14nm debuted six long years ago.

Intel's new Thread Director is the sleeper tech that enables the huge performance gains we've seen with Alder Lake. However, due to Alder's use of both faster and slower cores that are optimized for different voltage/frequency profiles, unlocking the maximum performance and efficiency requires the operating system and applications to have an awareness of the chip topology to ensure workloads (threads) land in the correct core based on the type of application.

Overall, the Alder Lake architecture has proven to be a big win for Intel, with class-leading performance in gaming, not to mention in both single- and multi-threaded workloads in standard applications. However, while the hybrid x86 architecture hails from a similar ethos as the big.LITTLE designs pioneered by Arm, it doesn't have the same tuning for power efficiency. Instead, Intel unabashedly tunes its design for performance at any cost, so AMD still holds the power efficiency crown in most types of workloads.

**Winner:** Tie. In judging AMD vs Intel CPU architecture, it's clear that the competition is now far closer than it has been over the last few years. AMD's Zen 3 architecture is a marvel that allows for enhanced scalability, and due to the efficiency-minded design paired with the TSMC 7 node, it delivers superior power consumption metrics. On the other hand, Intel's Alder Lake architecture is also a marvel in its own right, bringing the first pairing of small efficient cores with large performance cores to x86 desktop PCs for the first time. That lends it the performance advantage, but it still trails in power efficiency metrics, resulting in a tie in this category.

**2. Multicore Programming in Linux 1908.**

The Linux OS scheduler implementation follows the principle to avoid starvation and boost interactivity. Fast response to user despite high load Achieved by inferring interactive processes and dynamically increasing their priorities per the default scheduling policies and will scale well with number of processes O(1) scheduling overhead.

Per-CPU run queue – Possible for one processor to be idle while others have jobs waiting in their run queues Periodically, OS rebalance run queues and migration threads move processes from one run queue to another. The kernel always locks run queues in the same order for deadlock prevention

To simplify things a bit: the OS sets up a timer which interrupts the system at a fixed interval. A single interval is known as a time slice. Everytime this interrupt occurs, the OS runs the scheduling routine, which picks the next thread that is due to be executed. The context of the core is then switched from the currently running thread to the new thread, and execution continues.

Linux allows the user to specify which CPUs processes and interrupt handlers are bound. Each process has a bitmask saying what CPUs it can run on by default, all CPUs Processes can change the mask Inherited by child processes (and threads), thus tending to keep them on the same CPU rebalancing does not override affinity.

**3. Multicore Programming in Windows.**

One of the critical components of the operating system is called the scheduler. The scheduler consists of whatever method is used by the OS to assign work to resources, like the CPU and GPU, that then complete that work. The “unit” of work — the smallest block of work that is managed by the OS scheduler — is called a thread.

If you wanted to make an analogy, you could compare a thread to a one step on an assembly line. One step above the thread, we have the process. Processes are computer programs that are executed in one or more threads. In this simplified factory analogy, the process is the entire procedure for manufacturing the product, while the thread is each individual task.

If the computer is running quickly enough, its inability to handle more than one thread at a time becomes much less of a problem. While there are a distinct set of problems that cannot be calculated in less time than the expected lifetime of the universe on a classical computer, there are many, many, many problems that can be calculated just fine that way.

As computers got faster, developers created more sophisticated software. The simplest form of multithreading is coarse-grained multithreading, in which the operating system switches to a different thread rather than sitting around waiting for the results of a calculation.

This became important in the 1980s, when CPU and RAM clocks began to separate, with memory speed and bandwidth both increasing much more slowly than CPU clock speed. The advent of caches meant that CPUs could keep small collections of instructions nearby for immediate number crunching, while multithreading ensured the CPU always had something to do.

Modern CPUs, including the x86 chips built 20 years ago, implement what’s known as Out of Order Execution, or OoOE. All modern highperformance CPU cores, including the “big” smartphone cores in big. Little, are OoOE designs. These CPUs re-order the instructions they receive in real-time, for optimal execution.

The CPU executes the code the OS dispatches to it, but the OS doesn’t have anything to do with the actual execution of the instruction stream. This is handled internally by the CPU. Modern x86 CPUs both re-order the instructions they receive and convert those x86 instructions into smaller, RISC-like micro-ops. The invention of OoOE helped engineers guarantee certain performance levels without relying entirely on developers to write perfect code. Allowing the CPU to reorder its own instructions also helps multithreaded performance, even in a single-core context. Remember, the CPU is constantly switching between tasks, even when we aren’t aware of it.

The CPU, however, doesn’t do any of its own scheduling. That’s entirely up to the OS. The advent of multithreaded CPUs doesn’t change this. When the first consumer dual-processor board came out (the ABIT BP6), wouldbe multicore enthusiasts had to run either Windows NT or Windows 2000. The Win9X family did not support multicore processing.

There have been a handful of specific cases in which Windows needed to be updated in order to take advantage of the capabilities built into a new CPU, but this has always been something Microsoft had to perform on its own.

The exceptions to this policy are few and far between, but there are a few:

New CPUs sometimes require OS updates in order for the OS to take full advantage of the hardware’s capabilities. In this case, there’s not really a manual option, unless you mean manually installing the update.

The AMD 2990WX is something of an exception to this policy. The CPU performs quite poorly under Windows because Microsoft didn’t contemplate the existence of a CPU with more than one NUMA node, and it doesn’t utilize the 2990WX’s resources very well. In some cases, there are demonstrated ways to improve the 2990WX’s performance through manual thread assignment, though I’d frankly recommend switching to Linux if you own one, just for general peace of mind on the issue.