CURTISS-WRIGHT

Cores and Threads: Hybrid Processors for Today's Multitasking World Part-1

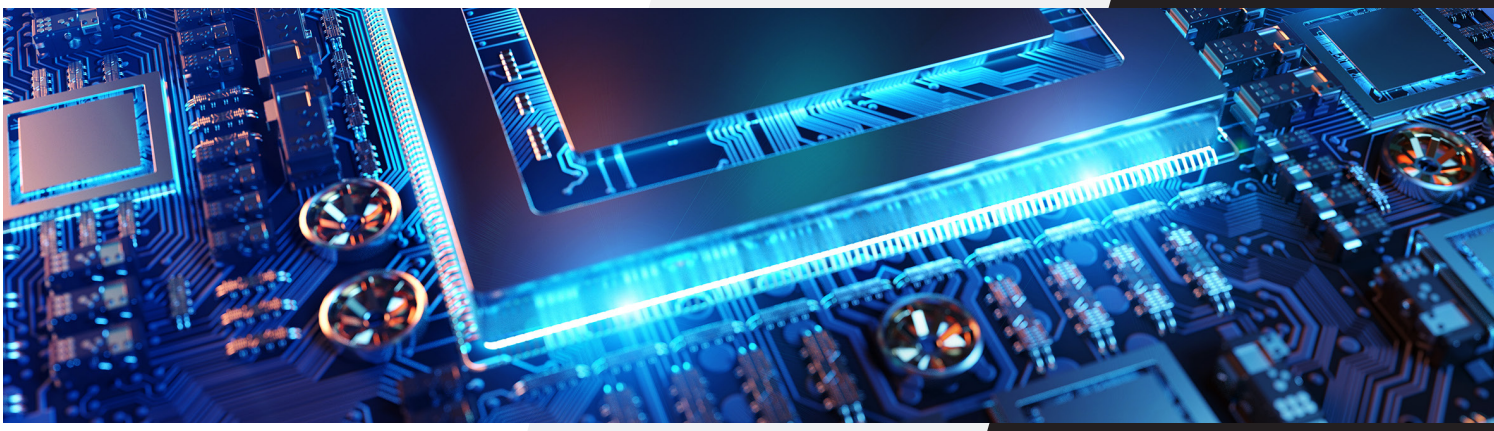
Introduction

In our fast-paced, multitasking world, parents of school-aged children often grapple with the homework dilemma. Today's adolescents are frequently seen multitasking, doing their school homework while interacting with their friends on social media and simultaneously flipping between television channels, watching hockey, soccer, and basketball games. As parents, we see their thumbs fly across smartphones while they message friends and at the same time ceremoniously display their real homework on a laptop computer.

While our children may think they are multitasking, it has been repeatedly shown[1] through research that the human mind cannot truly multi-task. Our inherent processing system of sensory inputs (eyes/ears/fingers), processing engine (mind/thinking), and outputs (hands/fingers/voice) can process only one thing at a time. Either we are watching one sports game or the other, reading and responding to our friends on Instagram, or doing our homework. The human mind cannot do all three simultaneously. Instead, we continuously shift from one task to another, and then to another, deciding which task to perform at any given time based on some inherent priority we assign to their relative importance (and as parents, we know that homework only sometimes makes it to the top of the list).

Contents

1. A good paper on human multitasking is from the National Library of Medicine, Multicosts of Multitasking, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7075496/



Computing and processing systems were initially designed to operate in a similar multitasking fashion. Early in the computing world, a CPU or processor could only perform a single operational instruction at a time. Operating Systems (OS) that managed these single thread processors evolved from simple single-tasking (aka bare metal) control programs to complex multitasking OSs that incorporate a task scheduler whose job is determining which task to run at a given time based on some priority scheme. Mirroring the human condition, these early processors still only performed a single activity at a given time. They created the illusion of multitasking because they were able to context switch very quickly – executing a given task for some number of milliseconds before switching to a different task. While a given second of computing time would appear to run dozens, perhaps hundreds, of tasks simultaneously, under the hood, the brains – the CPU or processing core - was still a single-tasking entity.

Today, finding a processor with just a single processing core is difficult. In 2000, IBM introduced the concept of a dual-core processor in their Power4 processor. AMD® followed in 2005 with the Opteron 800 and Athlon 64 X2 processors, each with two processing cores. Intel® gained commercial success with their dual-core processor in 2006 with the Pentium® Core2 processor. Today, almost two decades later, it is not uncommon to see datacenters running tens of thousands of processors, each with 64 or more cores. In addition to multiple processing cores, many architectures also support hyper-threading, which allows a processing core to execute two independent instruction threads simultaneously, mimicking a dual-core. Thus, a 64-core dual-threading processor can execute 128 different threads simultaneously. Taken to the extreme, today's high-end graphics processors (GPU) can execute thousands of simultaneous operations, which is fundamental for highly parallel 3D visualizations and complex AI processing tasks.

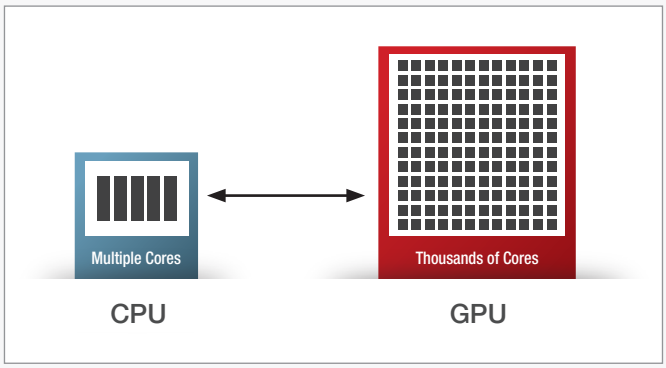


Figure 1: Processing Cores in a CPU vs. GPU

The incredible growth of processing parallelism has resulted in a corresponding explosion of performance and capabilities, but not all cores and threads are created equally. For mainstream computer users, such as the vast majority of Microsoft Windows users, the detailed usage of cores and threads is not important for the user to understand. When done editing a document, we hit the <SAVE> icon, and all the magic happens under the hood. But for designers of critical real-time processing systems, what happens under the hood matters.

This 2-part White Paper series explores the performance details of today's latest multi-core and hybrid-core processors to guide software developers and systems designers who need to understand what happens under the hood and exert tight control over their processing systems. In Part 1 of the series, we provide foundational background on the evolution of modern processors, establishing important characteristics of single-core, multi-core, and hyper-threading processor architectures. We introduce the concept of a hybrid-core processor architecture, where not all cores are created equal. Finally, we explore how today's popular operating systems use hybrid-core processors.

Part-2 of the series will dive deeper, presenting a benchmark analysis of processor performance and efficiency using different combinations of cores and threads, and it will explore Intel's new hybrid core technology. We conclude with a discussion of the benefits and drawbacks of using these processor technologies for embedded software and systems developers.

Processor Evolution and Architecture

Figure 2 illustrates a simplified view of a generic single-core processor. Important to this discussion is the data flow to and from a processing core. With few exceptions, a processor is paired with external main memory, where instructions and data are stored. Accessing even today's fastest DRAM memory subsystems is considered slow compared to the speed at which the core operates. To ensure the processing core does not sit idle waiting for memory interactions, most processors incorporate cache memory, which is a region of extremely fast local memory operating at the core speed, which mirrors regions (sometimes referred to as pages) of the external DRAM memory. If instructions and data are pre-loaded into the local cache memory, the processing core can run at full speed without waiting. Unfortunately, if the needed instructions or data are not pre-loaded in the local cache memory, the processing core will become stalled while the rest of the CPU fetches the required data from external DRAM memory into the cache. This cache miss results in a loss of performance.

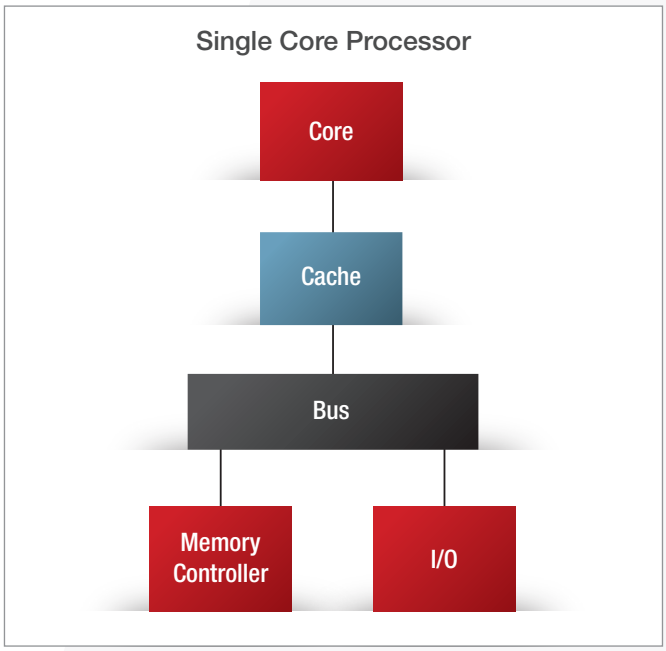


Figure 2: Simplified View of a Generic Single-Core Processor

Because cache memory is expensive in terms of silicon space, most processors have multiple levels of cache. The cache closest to the processor, called the L1 cache, is the smallest and fastest, with progressively larger size and slower access caches as you move further from the processing core (L2 cache, L3 cache, etc.). In a multi-core processor, each core typically has its own L1 cache, and often, multiple cores will share L2 or L3 cache regions. Figure 3 illustrates two generic quad-core processors. One has only two cache levels, with an L1 cache for each core and an L2 cache shared amongst the four cores. The second example has an L1 cache for each core, an L2 cache shared amongst each pair of cores, and an L3 cache shared across all four cores.

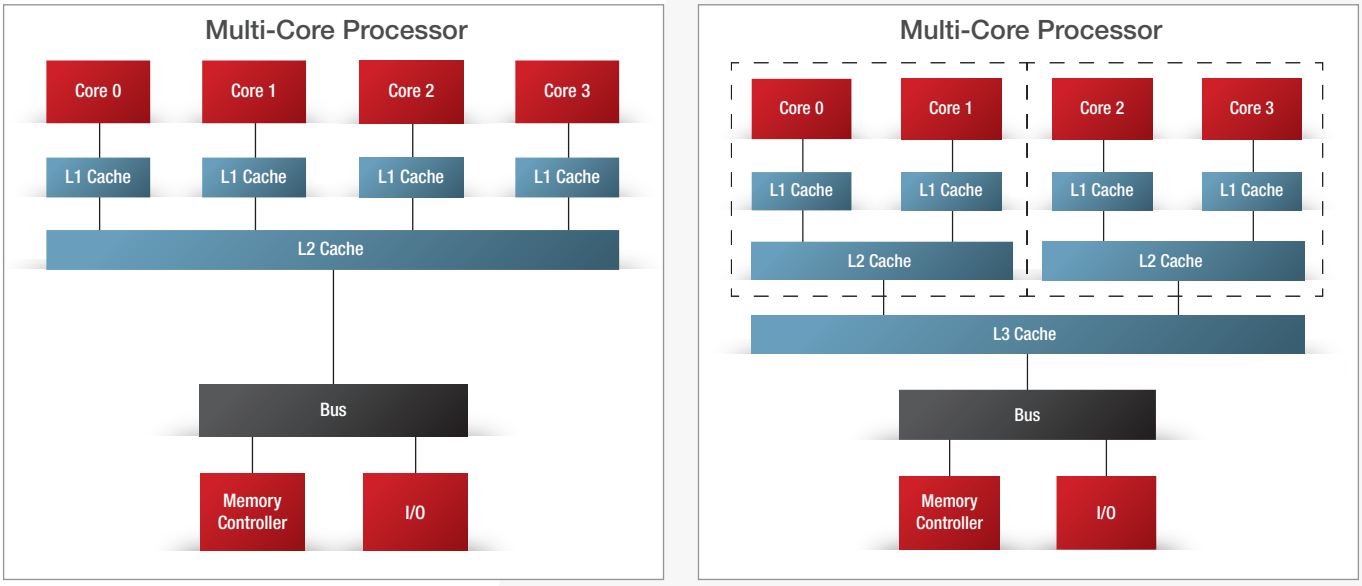


Figure 3: Multi-core Processors with Multiple and Shared Cache Levels

It is important to note that for a multi-core processor, the architecture has areas where multiple cores share common resources. It may be a shared cache memory region, a shared main interface bus, or a shared memory controller. The implication is that two cores may not be able to fully operate independently – there will be some interaction due to contention with shared resources. Thus, a dual-core architecture cannot provide a full doubling of performance compared to a single-core architecture. Similarly, a quad-core processor will not provide four times the performance. Real-world conditions reduce this performance to something less.

Multi-Core vs. Hyper-Thread

In 2002, Intel introduced the concept of hyper-threading. In a true multi-core processor, the core is duplicated, and each core has its own L1 cache, as shown on the left of Figure 4. A hyper-threading core is just a single core that appears to the OS as two logical cores, as shown on the right of Figure 4. A hyper-threading core is accomplished by using an internal superscalar architecture, in which multiple instruction streams can operate on independent instruction and data in parallel.

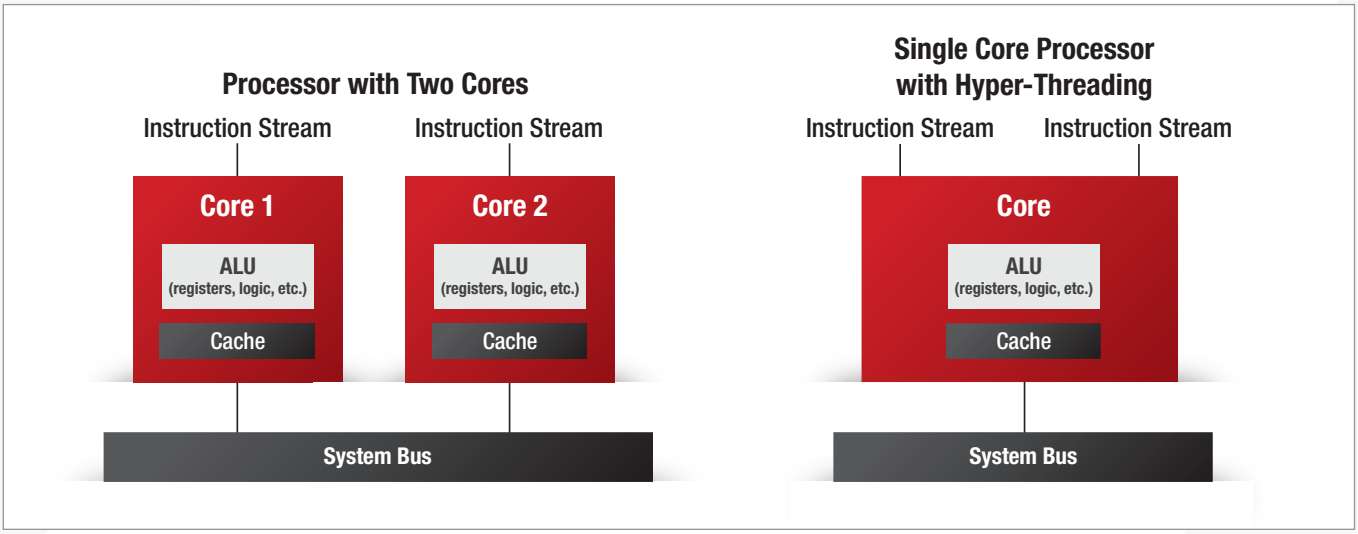


Figure 4: Multi-Core vs. Hyper-Thread

A hyper-threading core has more shared resources than two independent cores, and its overall performance in real-world applications is expected to be correspondingly lower than two separate processing cores.

Whereas Intel x86 and NXP Power Architecture provide hyper-threading cores in many of their processors, the Arm® architecture does not offer hyper-treading. An Arm core is simply a single-threaded core. A 16-core Arm processor, such as the NXP LX2160A, provides 16 fully independent cores and can execute 16 independent threads. In contrast, an Intel 8-core processor such as the Tiger Lake Xeon W-11865MRE provides 8 hyper-threading cores and presents as 16 logical processing cores to the OS.

big.LITTLE Architecture

In 2011, ARM Holdings introduced the first processor with what they called the big.LITTLE architecture. Realizing that real-world multitasking systems have a wide range of processing and performance needs, the architecture pairs some "big" cores optimized for high performance, with some "LITTLE" cores optimized for higher efficiency and sacrificing some amount of performance. Systems that make use of big.LITTLE processors will direct background and non-critical functionality to the LITTLE cores and will direct foreground and user-oriented functionality to the big cores. The goal of a big.LITTLE processor is to ultimately save power, a critical resource in battery-operated equipment such as laptops and cell phones, and to ensure responsiveness to users. For example, an Apple iPhone 13 uses the Apple-designed A15 Bionic Arm processor with 2 big cores and 4 LITTLE cores. When not actively in use, the phone will utilize only LITTLE cores, putting the big cores to sleep to reduce power consumption.

Intel Adopts a Hybrid Processor Core Architecture

While not the first to make use of a hybrid core architecture, Intel has introduced its equivalent to the Arm big. LITTLE architecture, offering hybrid core processors with what they call "Performance" cores (aka big or P-cores) and "Efficient" cores (aka LITTLE or E-cores).

Intel's rationale[2] for the introduction of hybrid core processors asserts the following:

> A recent Intel study, which examined the performance of various workloads from multiple segments by using an increasing number of CPU cores, produced the following results:
>
> - A majority of the workloads do not scale beyond 4 cores (many of these limited threading workloads closely resemble actual user experience workloads).
> - A minority of the workloads can scale to 8 cores but do not scale any further.
> - An even smaller minority of workloads can scale higher than 10 cores and continue to scale with core count.
>
> The results of this study highlighted the fact that the majority of client applications would benefit from better scalability to 8 cores or more. To better serve this market segment, Intel has designed a System on Chip (SoC) architecture where larger cores are unleashed to go after single-threaded and limited threading scenarios, while the efficient multi-threaded cores can help extend scalability performance over prior generations. The result of this effort is the development and introduction of Intel performance hybrid architecture.

The Intel 12th Gen "Alder Lake" and 13th Gen "Raptor Lake" processor families include embedded processor SKUs with up to 16 cores, consisting of 8 Performance P-cores and 8 Efficient E-cores. With Intel Performance cores supporting hyper-threading, the processor presents to the OS as a 24 logical core processor (8 hyper-threading P-cores + 8 single-thread E-cores).

Using Hybrid Processing Cores

Operating systems are now becoming aware of different application processing needs. Foreground processes, such as those interacting with users (applications in focus, visual displays, user interaction via mouse and keyboards, etc.), can be assigned to big/Performance cores to provide the best user experience, and background activities (low priority tasks, utility functions, system management, etc.) can be assigned to LITTLE/Efficient cores where high performance is not required.

2. https://www.intel.com/content/www/us/en/content-details/685861/

Core Usage in Multi-Tasking Operating Systems

Intel Thread Director

To make the best use of P-cores and E-cores, Intel provides a technology called the Thread Director to the OS. This technology allows the OS scheduler to assign tasks to P-cores and E-cores based on each task's characteristic needs for performance vs. efficiency.

Microsoft Windows 11

Under Windows 11, the Thread Director works closely with the Windows task scheduler, which has been enhanced to be aware of hybrid processor architectures. In this enhancement, the Windows 11 task scheduler considers P-cores, E-cores, and hyper-threads on P-cores when scheduling tasks. In addition, the Windows 11 task scheduler and the Intel Thread Director also monitor other processor parameters, such as clock speeds, power consumption, and thermal conditions.

Under Windows 11, workloads are monitored and classified as follows:

- Class 0: Most applications
- Class 1: Workloads using AVX/AVX2 instructions
- Class 2: Workloads using AVX-VNNI instructions
- Class 3: Bottleneck is not in the compute, e.g., I/O or busy loops that don't scale

Anything in Class 3 is recommended for E-cores. Anything in Class 1 or 2 is recommended for P-cores, with Class 2 having higher priority. Everything else fits in Class 0, with frequency adjustments to optimize for IPC and efficiency if placed on the P-cores. Even with all these conventions, the OS may still choose or be directed to assign any thread or class of workload to any core. Windows 11 also considers the computer's selected Power Plan, where a high-performance power plan will perform differently than a Balanced or Battery Saver plan.

Microsoft Windows 10

While the Thread Director also works with Microsoft Windows 10, the Windows 10 task scheduler is not designed to work optimally with the Thread Director. Under Windows 10, the scheduler assigns P-cores to the application in focus, meaning the currently highlighted application window. If an application is taken out of focus, either by minimizing the application or highlighting a different application window, the Thread Directory re-assigns the application to E-cores. Some users have reported mixed feedback with these processors under Windows 10, with the main concern being applications that are inactive or not in focus underperform when directed to E-cores.

Figure 5 shows the core usage of an Intel Alder Lake i7-12700H processor with 20 logical processor threads (6x P-cores with hyper-threading + 8x E-cores). The first 12 graphs (from top left) show the workload of P-core threads, and the last 8 graphs are E-core threads. The figure on the left shows that all 20 cores are in use when the application is in focus, driving the processor to an overall utilization of 77%. The figure on the right shows that when the application is minimized or taken out-of-focus, the application is removed from the P-cores and only executes on the lower-performance E-cores, driving them to maximum usage. The overall processor utilization is reduced to 53%, which reflects that the application task is likely underperforming with E-cores, while the P-cores mostly sit mostly idle.

Also of interest, these screen captures provide a process count and a thread count, which provides insight into the number of total application processes and threads the OS is concurrently managing. In these examples, there are 229 or 223 processes running, and 3345 or 3349 application threads running[3]. While many of these processes and threads may be sleeping or idle, most will wake up periodically to perform a task or status check.

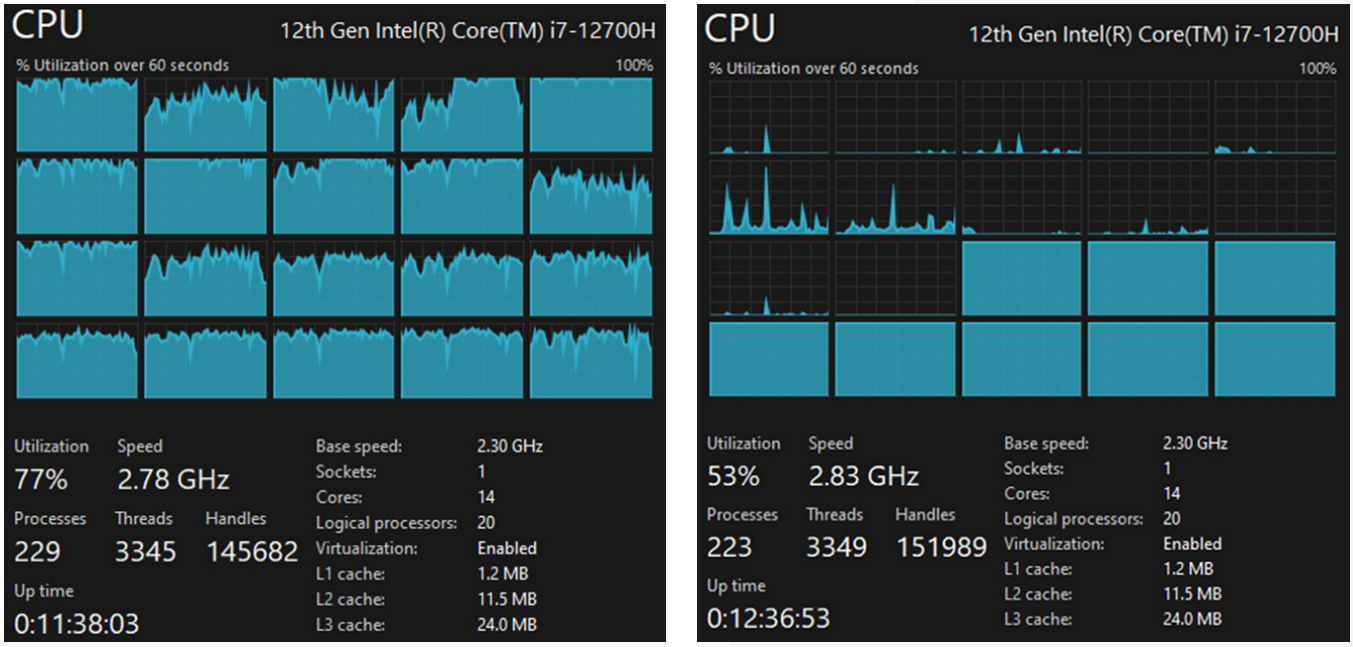


Figure 5: Alder Lake Processor under Windows 10 showing an application in-focus and out-of-focus

Linux

As of early 2023, apparently to address some reported performance bugs with the 12th Gen Alder Lake processor, Intel has added some, but not all, aspects of Linux kernel interaction with its Thread Director to Linux kernel 5.18. Officially, however, Intel has only stated publicly that Windows 11 is their priority, and they would be upstreaming a variety of features in the Linux kernel over time. More recently, it has been reported that Linux kernel 6.2 has added further support for Intel's 13th Gen Raptor Lake hybrid processors, including enhancements for the Thread Director.

Linux users have always had the ability to manually assign processes to logical processor cores using the taskset() command. With specific knowledge of which logical processors are P-cores and which are E-cores, it is not difficult to manually assign process affinity to specific cores. This can provide an embedded software developer incredible flexibility using hybrid core processors.

3. These Windows 10 Task Manager screen captures were performed on a corporate Dell computer running only three active "user" applications: Microsoft Outlook, Microsoft Word and a File Explorer window.

Summary

Approximately 70% of mobile phones are today operating with processors using the Arm big.LITTLE architecture. Intel, one of the largest processor suppliers, has adopted a hybrid (P-core/E-core) core architecture on their last two generations of consumer processors[4] and appears to be focused on extending this architecture for future generations. The mainstream commercial processing world has embraced the benefits of the hybrid core architecture.

While the aerospace and defense industry has yet to widely adopt the hybrid core processor, it is hard to ignore the potential benefits of this new technology, which promises an increase in processing efficiency. Size, weight, and power (SWaP) remain a primary consideration for all new developments, and any opportunity to increase processor efficiency will directly benefit a solution's SWaP footprint.

Part-2 of this White Paper series will explore performance testing on Intel hybrid core processors, measuring the performance and efficiency of P-cores vs. E-cores, and single-threaded cores vs. hyper-threading cores. It will discuss how these technologies can benefit the system and software developer of embedded systems and will summarize specific core and thread configurations of today's popular embedded processors.

Author

Aaron Frank
Senior Product Manager
Curtiss-Wright Defense Solutions

Aaron Frank joined Curtiss-Wright in 2010 as a Senior Product Manager for the C5ISR group. He is responsible for a wide range of COTS products utilizing advanced processing, video graphics/GPU, and network switching technologies in many industry-standard module formats. Aaron has a long history of working with and promoting open standards, participating in IEEE and SMPTE standards since 1990, and was personally awarded an Emmy from the National Academy of Television Arts and Sciences for his work to create the openGear platform (www.opengear.tv), an open equipment standard used by over 80 of the top broadcasting corporations worldwide. Aaron holds a Bachelor of Science in Electrical Engineering from the University of Waterloo.

4. 12th Generation "Alder Lake" and 13th Generation "Raptor Lake"