Specialized Evolution of the General-Purpose CPU

Ravi Rajwar Martin Dixon Ronak Singhal

Intel Corporation
Hillsboro, OR, USA

ABSTRACT

Commodity general-purpose CPUs remain the predominant computing platform for servers. However, these CPUs continuously evolve, incorporating increasingly specialized primitives to keep up with the evolving need of critical workloads. Specialization includes support for floating-point and vectors, compression, encryption, and synchronization and threading. These CPUs now have sufficient specialized support that the term general-purpose can often be misleading. Recent announcements such as a server product with an FPGA integrated with a CPU make the possibilities even more intriguing.

This paper discusses the evolution of specialization in commodity general-purpose CPUs, trade-offs from an industrial perspective, and the opportunities for software going forward.

1. INTRODUCTION

In 1965, Gordon Moore predicted that the number of transistors on a chip would double roughly every year [1]. Moore subsequently revised the prediction in 1975 to the number approximately doubling every two years, a prediction known as Moore's Law. David House subsequently observed that the increased transistor counts and faster transistors would translate to microprocessor performance doubling every 18 months.

In 1974, Robert Dennard and others described the traditional MOSFET scaling rules governing improvements in transistor density, switching speed, and power dissipation [2]. They observed that even though transistors became smaller, their power density remained constant. As a result, these transistors could switch faster while using less power.

While Moore's law provided the transistor counts, Dennard scaling provided a simple model to make these transistors effective. The two provided a roadmap for the semiconductor industry. In recent years, Dennard scaling has become less influential because the addition of innovative methods (involving material and structure innovation) have allowed for a continued density scaling (e.g., e-SiGe and strained Si for 90 nm and 65 nm, high-k metal-gates for 32nm and 45nm, and tri-gate for 22 nm and 14 nm).

This article is published under a Creative Commons Attribution License (http://creativecommons.org/licenses/by/3.0/), which permits distribution and reproduction in any medium as well as allowing derivative works, provided that you attribute the original work to the author(s) and CIDR 2015.

7th *Biennial Conference on Innovative Data Systems Research (CIDR '15)* January 4-7, 2015, Asilomar, California, USA.

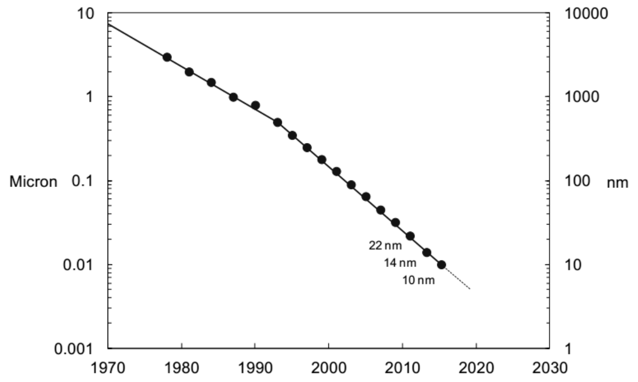

Figure 1 Intel Process Scaling Trends

Figure 1 shows the scaling trend for Intel going back four decades. Scaling feature size every generation results in smaller transistors and thus higher performance, lower power, and lower cost per transistor. These performance and scaling trends have enabled the commodity general-purpose CPU[1] to run a broad range of workloads efficiently and with high performance. These trends have fuelled growth in software size, features and functionality, and programmer abstractions [3]. These have led to improved programmability and manageability of large software and in turn have enabled innovation in software development and usage.

Increasingly, the industry faces technical challenges to sustain the historic rates of performance and power improvements. This has led to growing concerns around software's ability to continue to innovate if the CPUs cannot sustain software-transparent performance growth rates. This has spurred a debate on the future of the CPU and a growing interest in specialized custom hardware solutions, such as custom processors, offload engines, and accelerators tailored to specific problem domains [4]. The argument for these tailored solutions includes better performance and efficiency as compared to the CPU.

However, over the years, these CPUs have already been incorporating specialized hardware capabilities in response to the changing software landscape. These specializations allow the CPU to provide significant domain-specific performance gains while remaining general-purpose. As such, the dichotomy between general-purpose and specialized is misleading.

[1] Referred to as general-purpose because of their ability to run a wide range of workloads.

This paper discusses the evolution of specialized hardware support in the commodity general-purpose CPU. Section 2 discusses recent CPU technology trends that continue to enable improved power and performance. These represent the continuation of software-transparent improvements. Section 3 goes into examples of domain-specific hardware specializations in the CPU. Software can make use of these capabilities to improve performance.

The commodity CPU benefits from an economy of scale, a rich software eco-system, wide availability, and inherent adaptability. This has contributed to numerous specializations going mainstream and broadly used beyond their original domains. Section 4 discusses the practical considerations when deciding to use a custom hardware solution as compared to one using a CPU.

The growth of cloud services and the data center have accelerated software development cycles. This provides an opportunity to re-think critical aspects of software architectures. For example, specialized CPU capabilities such as virtualization support can help split the control and data planes of the network infrastructure. The resulting specialized software stack running on a general-purpose CPUs increases the infrastructure's flexibility and adaptability. We discuss such an example in Section 5.1.

In spite of the flexibility of general-purpose CPUs, key specialized tasks exist where fixed-function engines with limited flexibility provide compelling gains while freeing the CPU to process other more complex tasks. We discuss this in Section 5.2

On the other hand, customized logic such as FPGAs and ASICs can help accelerate the hardware development cycle. Their tighter integration with the CPU raises interesting possibilities. However, much work remains as to how easily and widely software can benefit from these capabilities and their impact on software efficiency and cost. We discuss this in Section 5.3.

As new usages and services evolve, so does the software developed to provide them. Early implementations may be limited in performance but specialized techniques develop to improve it. Experimentation for hardware acceleration may occur using fixed-function hardware or configurable logic such as FPGAs. As these techniques become widely applicable, they find their way into commodity CPUs. This integration lowers the bar for their adoption by making such solutions ubiquitous and cost-effective and improves their efficiency and performance. However, to aid software development and experimentation, the programming interfaces must abstract the hardware mechanisms from the programmer. This enables a spectrum for hardware acceleration options. We discuss this in Section 5.4.

While the paper uses examples from the Intel processor families, the observations apply to other general-purpose CPUs as well.

2. CPU TECHNOLOGY TRENDS

Process and microarchitecture optimizations continue to provide commodity CPUs with steady improvement in performance and efficiency every generation.

Figure 2 plots the leakage and transistor performance curves for various Intel process nodes. Every generation provides a range of transistor designs to balance leakage and performance. This helps pick the right transistors for the appropriate segment. Transistor designs strive to push these curves down and to the right [5].

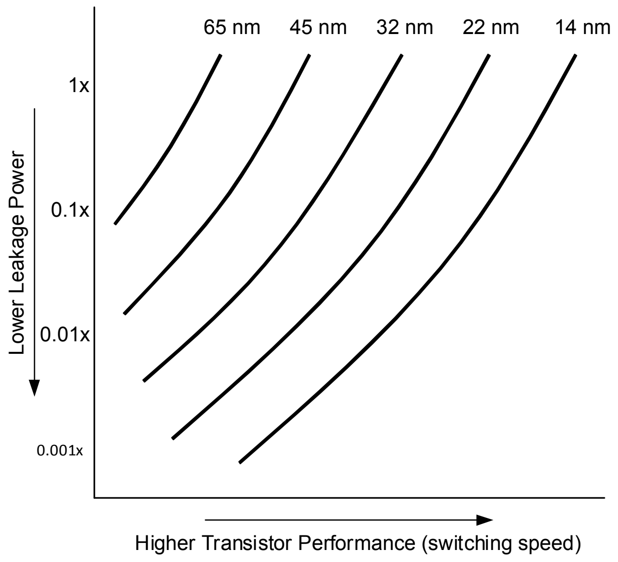

Figure 2 Transistor Performance vs. Leakage

The 65 nm transistor had a much smaller range. The goal at the time was to make the transistor as fast as it could be. Recent generations have significantly expanded the range. One can pick transistor designs optimized for low leakage circuits or high-performance circuits or a combination along the curve. This type of specialization allows the CPU to scale appropriately for the given market segment. For example, a product targeted at the highest frequency may pick the top end of the curve to emphasize performance.

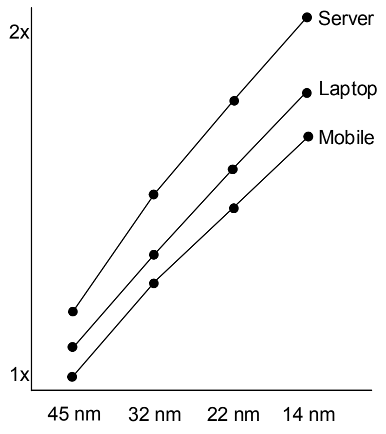

Figure 3 Performance Trends

Figure 3 shows performance scaling over different transistor generations for three key market segments and Figure 4 shows the reduction in active power for the same. Both figures show a steady improvement over the generations for both performance improvements and active power reductions.

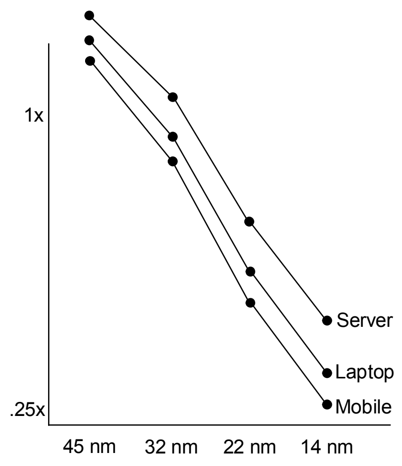

Figure 4 Active Power Trends

Figure 5 plots the performance-per-watt improvements. One can see from this graph that the performance-per-watt has consistently improved over the generations, whether the optimization point picked power, performance, or some intermediate point. Process technology has provided a 1.6x improvement in performance-per-watt every generation. The most recent process (14 nm) provides a 2x improvement. This is due to multiple factors including advances in transistor technology (with improved low voltage performance and lower leakage), improved area scaling, a co-optimized approach between process and design, and numerous microarchitecture optimizations to improve active power [5].

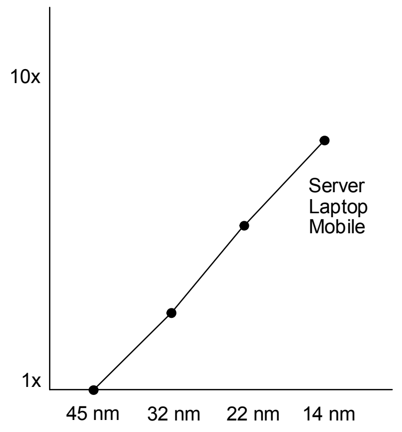

Figure 5 Performance per Watt Trends

In addition to process improvements, microarchitecture enhancements play an important role in improved performance and efficiency. These enhancements transparently improve the performance of existing software with no need for software changes. A common measure of such an improvement is the instruction per cycle (IPC) metric for a single thread application. Increasing IPC requires hardware to expose and exploit instruction-level parallelism in the application and/or to reduce the latency of critical instructions. Methods to improve performance include improved out-of-order execution, better branch prediction, larger instruction windows, increased memory level parallelism, faster and higher bandwidth cache and memory systems, and improvements in paging and TLB, among others. Some optimizations target characteristics of workloads from specific segments. CPUs continually incorporate numerous techniques to improve performance [6]. The focus however remains on increasing efficiency at every operating point. Figure 6 plots the cumulative growth in single thread IPC for the Core™ family over the past decade for a broad mixture of workloads.

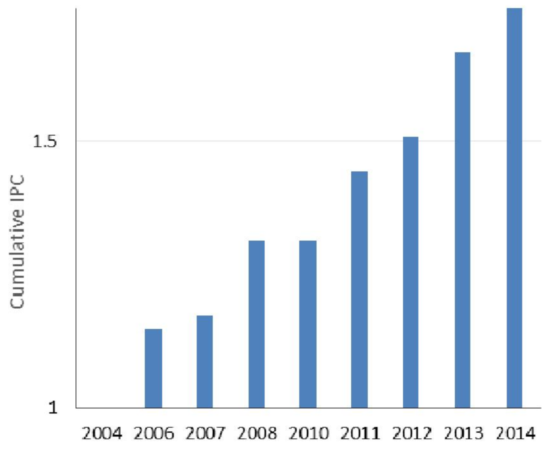

Figure 6 Cumulative Single Thread IPC Growth

Area scaling and power improvements have enabled integration of multiple such CPUs onto a single package, thus providing a means to scale throughput in a power efficient manner. Figure 7 plots the maximum single socket thread counts for Intel's highest volume server products. Additionally, these products can be linked together into multi-processor systems to build even larger scalable systems.

Even with the advent of multi-core processors and the focus on throughput improvements, single thread performance remains critical. For example, CPUs with high throughput but reduced single thread performance could result in the system not meeting latency targets [7].

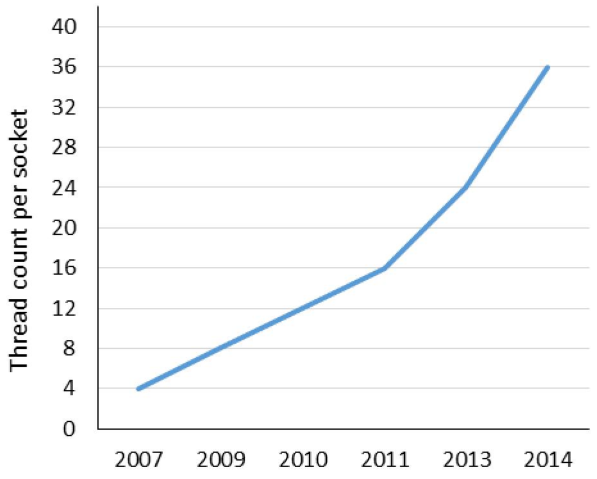

Figure 7 Thread Count Growth (Volume Server Segment)

3. SPECIALIZED GENERALIZATION

While the CPU continues to benefit from process and microarchitecture advances, it must keep pace with evolving usages. CPUs increasingly incorporate numerous software-visible hardware specializations to provide domain specific improvements. These specializations provide composable capabilities to software but typically do not implement solutions. Software uses these hardware primitives to implement solutions.

This section explores examples of such specialization. These examples have had a major impact on domain-specific software development.

3.1 Floating-Point Arithmetic

Support for floating-point operations is an early example of a commodity general-purpose CPU incorporating specialized functionality. Floating-point is a format to represent real numbers in a finite number of bits with varying precision. Specialized hardware for floating-point operations was initially a domain of expensive supercomputers such as the Control Data® 6600 (1964, cost of around $7 million at the time) and Cray 1 Computer System® (1976, cost between $5 million and $8 million at the time).

On the other hand, commodity CPUs of the time relied on software emulation for floating-point operations. Customers requiring high-performance floating-point used a dedicated floating-point unit (FPU) co-processor. For example, in 1980 Intel introduced its first FPU coprocessor, the 8087, for the 8086, Intel's general-purpose CPU at the time [8]. The 8087 provided 32-bit, 64-bit, and 80-bit precision. Even with the co-processor, there was desire to make it as general purpose as possible. From Palmer's 1980 paper, "The 8087 is intended to be general purpose and satisfy a very wide range of needs for mathematical computation…". The co-processor significantly improved performance of floating-point applications but at increased power consumption and platform cost.

In 1985, the IEEE Standard for Floating-Point Arithmetic (IEEE 754) provided a standardized method to perform floating-point computation thus enabling portability of floating-point computations across different machines. In 1989, the Intel486™ CPU integrated the floating-point unit into its design (the extensions since referred to as x87). This made fast floating-point operations broadly available without the need to purchase a co-processor. It also simplified the programming interface as the programmer could now assume a floating-point "co-processor" was present. The incorporation of floating-point support also simplified algorithms originally expressed in fixed-point arithmetic. The shared address space made the conversions between different types easier.

The floating-point domain itself has expanded; today, floating-point is virtually everywhere from computing the physics for games on handheld devices to forming the basis for advanced science in high-performance computing (HPC). Almost all languages recognize the number format and runtime systems handle floating-point exceptions.

3.2 SIMD Extensions

Support for Single Instruction Multiple Data (SIMD) operations is another example of the commodity general-purpose CPU adding specialized hardware support for domain-specific applications. SIMD represents a class of computation where multiple processing engines perform the same operation on multiple data elements simultaneously. Similar to floating-point operations, SIMD operations used to be the domain of highly specialized vector supercomputers (e.g., Control Data STAR 100, Cray 1 Computer System, and the Connection Machine® Model CM-1 from Thinking Machines Corporation).

Over time, the CPU increased in performance, thanks to Moore's law and Denard scaling. In 1997, SIMD processing was introduced to commodity CPUs with the inclusion of MMX™ technology extensions to a model of the Pentium® processor. The MMX extensions enabled the calculation of 8 bytes or 4 words in parallel using 64-bit registers that supported vector integer operations. These enabled improvements in multimedia and communications workloads, in addition to improving graphics realism and full-screen, full-motion video.

As CPUs became more powerful, software running on them increased in complexity. The Streaming SIMD Extensions (SSE) in 1999 expanded the SIMD capabilities in the architecture to include 128-bit wide architectural registers and packed single-precision calculations. To take advantage of this new architectural capability, implementations introduced dedicated hardware to support SSE such that performance gains could be capitalized on. These extensions also addressed some of the MMX limitations by supporting the floating-point data type. This enabled mixed integer and floating-point SIMD. In 2001, the SSE2 extensions added support for 8-bit, 16-bit, and 32-bit integer vectors in addition to double-precision data types. These extensions also provided programmers greater control to access and cache data. Since streaming data types do not always have cache locality, the extensions also provided software direct control over cacheability to minimize cache pollution, as well as support for software-directed prefetching of data. With the increasing sophistication of the SSE extensions, they supplanted x87 for most floating-point operations (except 80-bit extended precision) and MMX for media applications.

The instruction set architectures remain fluid to match the needs of applications, so it is not atypical for every CPU generation to add new instruction capabilities. The SSE3 extensions in 2004 provided additional instructions for format conversion, and supported unaligned address loads while the SSSE3 extensions in 2006 accelerated operations on packed integers. The SSE4.1 and SSE4.2 extensions in 2007 and 2008 provided further enhancements to improve compiler vectorization, support for packed double word computation and for string and text processing.

In 2010, the Intel® Advanced Vector Extensions (AVX) widened the vector registers to 256 bits and introduced the first set of instructions to utilize these registers, focused on floating point. Implementations could take advantage of this widening to instantly double the floating-point operations (FLOP) provided by the CPU with the same number of functional units as previously implemented. In 2012, the Intel AVX2 extensions widened the integer data type to 256 bits. This made wide SIMD available to a broader set of applications utilizing wide integer vector computation and full-width element compute. The addition of FMA and vector gather operations targeted high-performance computing, audio and video processing, and games. The domain for SIMD itself is also expanding. For example, recent work utilizes 256-bit SIMD instructions (AVX2) to vectorize database column scans, with both range and complex predicates [9].

SIMD widths continue to grow. For example, the Intel® Xeon® Phi™ processor supports a 512-bit wide vector register, and these wider widths will be available in future mainstream Xeon servers.

Figure 8 plots the peak memory bytes per clock that can be loaded via the L1 cache and single-precision floating-point operations (FLOPs) per clock over the years for a single CPU. The FLOPs have steadily increased and have driven changes through the rest of the microarchitecture to ensure that the compute capability is matched by the ability to feed data into the compute elements. Increased core count on the die would result in a corresponding increased in FLOPs for the die. The improvements are expected to keep pace with growing software demands with future processors.

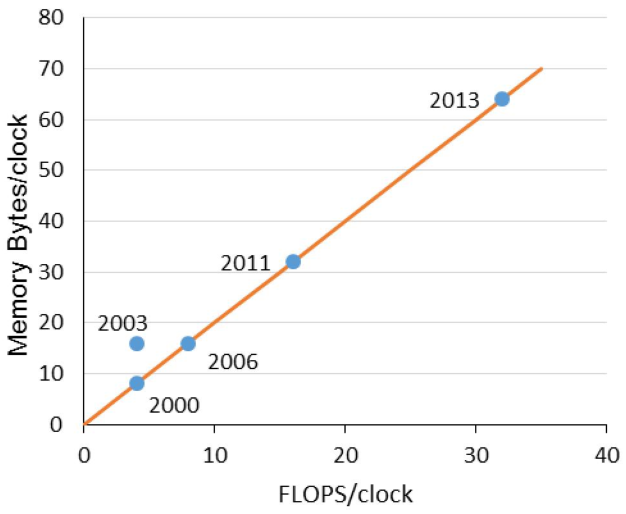

Figure 8 Growth in FLOPs per CPU core

The wide spread availability of wide-vector SIMD support in the commodity CPU lowers the barrier to entry for software. This enables extensive software infrastructure and libraries, and creates the framework for innovation in algorithms and usages.

3.3 Parallelism

In a paper in 1971, Schorr explored the design principles for a high-performance mainframe system capable of executing large scientific applications [10]. He identified three approaches to improve performance. The first involved extending the look-ahead capabilities on a single instruction counter using hardware-controlled parallelism (now known as instruction-level parallelism). The second involved using vector or array processing (now known as data-level parallelism). The third involved multiple instruction counters (now known as thread-level parallelism).

Parallelism for a while was the domain of high-end super-computers. We have discussed how commodity CPUs already incorporate elements of instruction-level parallelism (See Section 2) and data-level parallelism (See Section 3.2). These CPUs also incorporated specialized support for thread-level parallelism. Starting with the 80286 in 1982, software running on Intel's CPU could perform mutually exclusive access to shared resources using a LOCK prefix on a set of read-modify-write instructions. This prefix ensures the read and write phases of the instruction execute atomically. Even though the compare and swap operation had been in mainframe systems since the 1970s (starting with the IBM System/370™), it made its way into the commodity CPU in 1989 starting with Intel486 (with the mnemonic CMPXCHG). Subsequently in 1993, the Pentium family introduced an 8-byte version of the instruction. This enabled newer lock-free data structure construction. Today, the compare and swap primitive is the most popular synchronization primitive to implement lock-based and lock-free algorithms and exists in some form in nearly all general-purpose CPUs.

Similarly, the CPU added inter-thread communication primitives such as MONITOR/MWAIT to improve the coordination of multiple threads.

The early 2000s saw an inflection point where multi-threading went mainstream with commodity general-purpose CPUs. This led to an increase in software eco-system investment and runtime frameworks and thread-safe libraries. As a result, software increased the use of synchronization primitives and the CPU adapted by improving the performance of these primitives. For example, Figure 9 plots the latency to perform a cached lock operation for the Core™ processor family over the years.

The past decade has seen an unprecedented software movement to incorporate multi-threading and parallelism. However, coordinating and synchronizing threads remains a significant challenge. Recently, CPUs added specialized support to simplify synchronization. For example, the Intel® Transactional Synchronization Extensions (Intel TSX) provides support for transactional execution to simplify the development of multi-threaded software. These extensions allow software to identify critical sections. Hardware attempts to execute these transactionally without acquiring the lock protecting the critical section. If the hardware succeeds, then the execution completes without the threads acquiring a lock, thus exposing concurrency and removing serialization. Recent work has explored how database implementations can benefit from such capability to improve performance [11, 12]. Such capabilities provide new opportunities for software innovation, especially in areas such as in-memory databases with different cost metrics than traditional disk-optimized databases. In traditional databases, disk access used to dominate latency. However, the advent of low latency disks and low cost memory have enabled a growth in in-memory databases where access to memory dominates. Similarly, Yoo et al. apply the extensions to improve performance of high-performance computing applications [13].

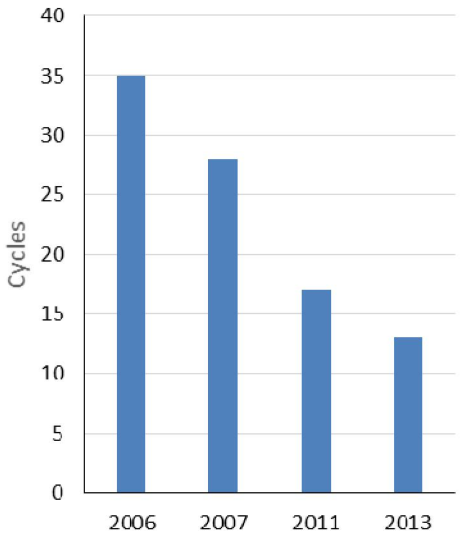

Figure 9 Reduction in Latency of Lock Instructions

Specialized support for multi-threading in commodity general-purpose CPUs have led them to become the compute engine of choice in servers. These CPUs can satisfy the functionality and performance demands of the wide range and variety of workloads in data centers and the cloud. These CPUs lowered the bar for specialized parallel programming by making the hardware capabilities widely available.

3.4 Virtualization

Virtualization is another example where the commodity general-purpose CPU has incorporated capability that was once specific to specialized high-end server and mainframe systems (e.g., the IBM VM/370 announced in 1972).

In a non-virtualized system, a single operating system controls the platform resources. With a virtualized system, a new layer of software, called the Virtual Machine Monitor (VMM), allows multiple operating systems to share the same hardware resources. The VMM arbitrates software accesses from multiple operating systems (called guests) running on the hardware system (called host).

However, implementing the VMM required specialized and complex software systems. The commodity CPUs added hardware support for processor virtualization thus enabling simplifications of virtual machine monitor software. The resulting VMMs can support a wider range of legacy and future operating systems while maintaining high performance.

Virtualization support has also followed an evolutionary trajectory. Once again, we use the Intel example here. In 2005, the Intel processors introduced Intel® Virtualization Technology (Intel® VT) to enable x86 instruction set virtualization. This allowed guest software to run unmodified on the CPU. Memory access virtualization was subsequently added to allow guest software direct access to paging hardware (EPT extensions). Following this, VT-d enabled virtualization of memory accesses by I/O (DMA) allowing guest software direct access to I/O devices that access memory. Then in 2013, interrupt virtualization was introduced to allow guest software to access the legacy interrupt architecture directly.

CPU specialization progressively reduced the need for involvement by a VMM layer by either allowing guest software direct access to legacy hardware or by giving guest software CPU-maintained shadows of that hardware. Virtualization improves system utilization, manageability, and reliability. In addition, it also enables workload isolation, workload consolidation, and migration. These benefits have led to widespread use of virtualized platforms in data centers.

As virtualization has increased in its usage, there has been a focus to reduce the overhead of running software when virtualized. Figure 10 plots the latency in cycles for a round trip VT transition since first introduction and shows the significant progression that has been made in reducing this overhead.

Evolutionary extensions continue with support for cache quality-of-service to allow cache partitioning. Recent extensions such as VMFUNC allow hyper-calls without requiring a VMEXIT.

Virtualization support significantly reduced the barrier to entry for software virtualization. The widespread availability of the general-purpose CPU with virtualization support has led to alternate usages beyond classic virtualization thus enabling innovative software architectures. For example, to reduce operating systems kernel overheads, researchers have been recently investigating decoupling the control and data plane and using virtualization hardware to implement protection.

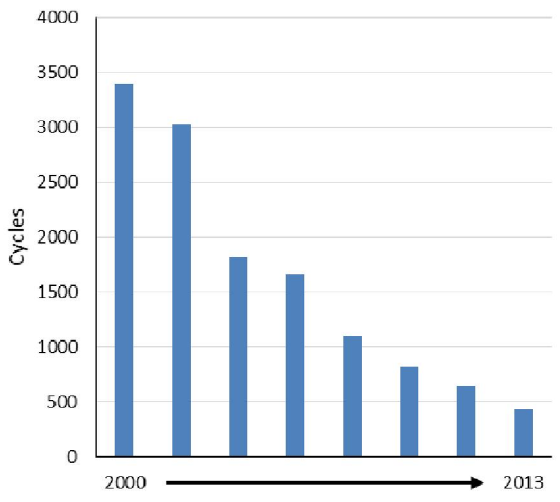

Figure 10 Latency Reduction of VT-x Transitions

3.5 Cryptography

Cryptography is an example of a usage model that has seen an explosion in the past decade due to the rise of e-commerce and the increase in privacy concerns around data communications. Today, many mainstream websites utilize the https protocol to encrypt all of their web traffic (e.g., Google web searches, Facebook, etc.). The CPU, in response, has incorporated specialized hardware support to improve the performance of cryptography algorithms. In general, this is a good example of the phenomenon that leads to specialized extensions being integrated into the CPU. As algorithms/usages become pervasive and take a greater percentage of cycles on the CPU, adding dedicated support for such algorithms can provide significant performance and efficiency improvements.

Cryptographic algorithms are fundamental to the security of modern platforms and communication channels. Numerous security algorithms exist. In 2001, the US National Institute of Standards and Technology established the Advanced Encryption Standard (AES). As cryptographic algorithms became widely used, the CPU incorporated hardware support to accelerate such functions. For example, in 2010 the CPUs added instructions to implement some of the complex and performance intensive steps of the AES algorithm in hardware[2]. These instructions provided significant speedups over completely software approaches [14].

The focus since has been to continue to improve solutions for consumers through faster encryption/decryption and key operations. Lowering the cost of implementations makes these solutions more widely available.

Figure 11 plots the performance improvements of various specialized cryptography algorithms normalized to the initial hardware support in 2010.

[2] Standardization is helpful. Similar to the IEEE Floating-Point standards, the AES standard provides a portable definition for hardware to optimize.

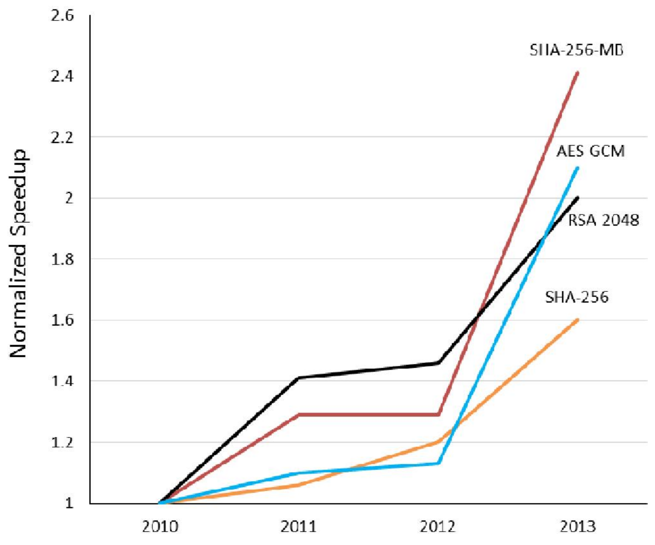

Figure 11 Impact of Cryptography Support

AES-GCM is a version using the Galois Counter Mode (GCM) of operation implemented with the AES instruction extensions. RSA-2048 is a version of the RSA algorithm using a 2048-bit key implemented with the RORX instruction specifically added to allow non-destructive fast rotates by a constant. SHA-256 is a secure hash algorithm. The algorithm takes as input a 512-bit data block and a 256-bit state vector. It outputs a modified vector. The implementation uses the AVX2 SIMD extensions and the RORX instruction. SHA-256-MB is a version of the SHA algorithm using multiple independent data buffers and implemented with the AVX2 SIMD extensions.

As can be seen from the figure, the throughput of these cryptographic operations has been steadily improving over the years thus lowering the bar for their usage. Intel has not been alone in adding cryptographic operations; for example, the ARMv8-architecture adds a cryptographic extension as an optional feature [15].

Other instructions in this realm have become important. In response to the need for more ephemeral keys, the CPU introduced a direct access to a digital random number generator instruction (RDRAND). Previously entropy on commodity systems was incredibly difficult to obtain. OS libraries used hashes of pages of memory or timing of inter-arrivals of network packages to approximate true randomness. Real entropy sources were available as stand-alone boxes or add-in cards such as SSL accelerators.

4. PRACTICAL CONSIDERATIONS

The increasing technical challenges faced by the industry to sustain the historic rates of performance improvements have raised questions about software's ability to continue to innovate if the CPU is unable to provide such growth. This has spurred a debate on the future of the general-purpose CPU and an increased interest in custom hardware solutions, such as accelerators, appliances, and offload engines, tailored to specific domains.

While custom domain-specific hardware solutions improve efficiency and performance for the problem at hand, deciding whether to adopt a custom hardware solution or one based on a general-purpose CPU depends on numerous factors.

Development cost: The cost for a custom hardware solution includes both hardware and software development. Typically, custom hardware solutions have low volumes as they are targeting a given problem domain. Thus, the hardware development cost needs to be amortized over a small volume. In addition, depending on the solution definition, developers may need to write new customized software stacks. The value proposition needs to be significant for the development cost to be justified.

Problem domain applicability: A question to ask is whether the custom solution addresses a broad problem domain or just one of many with each requiring its own custom solution. For example, a data center that runs a broad range of workloads may not see a benefit from only accelerating a small fraction of those workloads, especially if the opportunity cost of that solution impacts the performance for the remaining set of workloads.

Adaptability with changing requirements: Workloads continuously evolve and their requirements rapidly change. This is especially true in the new software landscape of cloud services and data centers. A key attribute is whether the custom hardware solution can keep up with changing workload requirements. As Barroso et al. state in discussing implications for a data center [16] – *"...there is substantial risk that by the time a hardware solution is implemented, it is no longer a good fit even for the problem area for which it was designed."*

Commodity general-purpose CPUs incorporate specialized primitives once the techniques the primitives improve gain wider acceptance and applicability or the power and die area cost to implement is low. The large volume of the commodity CPU amortizes the incremental cost to implement specialized hardware capabilities due to the economy of scale. Further, the general-purpose CPU enjoys a rich software eco-system consisting of numerous operating systems, runtimes, frameworks, libraries, and tools. It also broadens the user base by making the capabilities available ubiquitously to the general customer base without requiring special investments to be made.

As the general-purpose CPU has evolved, so has the software landscape. In such a fast moving landscape, building special purpose hardware solutions for specific problems comes with its own costs and challenges. In an effort to justify the investment cost in such tailored solutions, significant work goes into making them more general-purpose[3].

5. OPPORTUNITIES AHEAD

Increased integration of platform components coupled with CPU specialization provide unique opportunities for software going forward. This may include optimized domain-specific software taking advantage of the CPU's capabilities, software offloading key algorithms and functions to configurable and fixed-function accelerators, or software taking advantage of specialized support in the CPU. The software requirements influence each point in this spectrum (shown in Figure 12) [17].

The computing landscape has seen a significant change with an increasing movement of software and services to the cloud and

[3] The ongoing work in GP-GPUs is a similar example with an effort to make a graphics processor look more like a general-purpose processor.

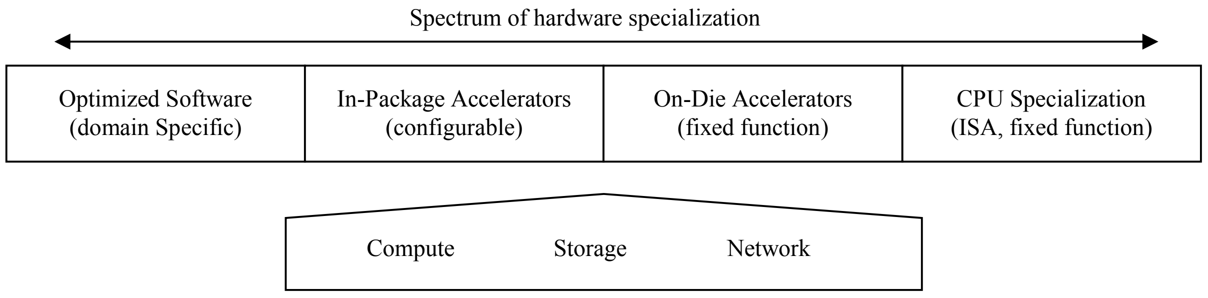
Spectrum of hardware specialization
Optimized Software (domain Specific)
In-Package Accelerators (configurable)
On-Die Accelerators (fixed function)
CPU Specialization (ISA, fixed function)
Compute
Storage
Network

Figure 12 Specialization Opportunities to Improve Workload Performance

large data centers providing the compute horsepower behind these services. This has accelerated the software development cycle and creates an opportunity to revisit historical design choices. Section 5.1 discusses one example where the performance efficiency and specialization of the CPU helps redefine the architecture of network and storage services to enable flexibility and adaptability. This is an example where hardware enables software to define the system architecture in a manner that is flexible and scalable, yet provides high performance.

In spite of the flexibility of CPUs, key specialized tasks exist where fixed-function engines with limited flexibility provide compelling gains while freeing the CPU to process other more complex tasks. We discuss this in Section 5.2

While the CPU provides flexibility for software and fast software development cycles, customized logic using FPGAs and ASICs provide an opportunity for faster hardware development cycles where customers can tailor hardware solutions for specific workloads. We discuss this in Section 5.3.

Section 5.4 explores the continuing role of software to guide CPU specializations and the role software abstractions play in bridging the disparities in acceleration support.

5.1 Revisiting Software Architectures

The infrastructure behind the cloud services and enterprise data centers highlights the tension between custom hardware vs. generalized solutions. This is an example where the general-purpose CPU enables flexibility and thus allows adaptation as workloads and services evolve.

The growth in devices has placed significant pressure on infrastructure, including computing, networks, and storage. Such infrastructure must be capable of deploying new services quickly and efficiently. However, the installed custom hardware often directly implements policies and algorithms. This makes it difficult for the hardware to keep up with the evolution of services. This adds to capital and operational expenditures and slows deployment.

This has led to the push for inter-operable solutions and standards for software-defined networks and storage. These efforts decouple the actual network and storage hardware from their functions through abstraction and virtualization. For example, certain hardware-based network appliances can be replaced by software-based functions running in virtual machines on general-purpose CPU-based commodity servers. This is possible because the performance of the general-purpose CPU has reached levels where software can implement most of the custom hardware functions in a flexible manner. This enables significantly improved adaptability as software innovates with newer usage models. The virtualization support in the general-purpose CPU plays a key role (See Section 3.4). Standardization also helps. The PCI-SIG Single Root I/O Virtualization and Sharing (SR-IOV) specification defines a standard method to implement natively shared devices to provide fast I/O without involving the VMM. It does so by providing independent memory space, interrupts, and DMA streams for the various virtual machines. These efforts improve user-level IO for communication.

Such examples where specialized capability in the CPUs enable new and improved ways to restructure the solution have the potential for significant benefit as they address the fundamental problem at the software architecture level.

5.2 Exploiting Offload Engines

The general-purpose CPU runs a wide range of software effectively and with high performance. This enables software to implement complex functions while retaining flexibility. However, ubiquitous tasks, such as bulk compression and cryptographic algorithms, are well suited for offload to specialized fixed-function engines. For example, compressing data for transfer between multiple nodes in a distributed system would reduce disk and network bandwidth. Since compression is well understood and standardized, offloading this action to an engine would be more efficient and allow the CPU to execute complex tasks. Similarly, offload of bulk encryption using standardized algorithms can also provide benefit.

When using such engines, the APIs employed must hide the implementation details from the application software. Except for performance differences, the application should not know whether the called function is a software-only implementation or a fixed-function accelerated implementation or one that uses specialized CPU support for acceleration. Specialized chipsets already provide such capabilities.

5.3 Exploring Silicon Customization

Custom solutions can be attractive to customers for whom the benefits outweigh any associated costs of customization. These often represent examples either where the ingredients of the custom solution are not yet broadly available or where the solution inherently benefits from a discrete engine.

The general-purpose CPUs coupled with customized silicon and configurability allows customers to tailor their solutions for targeted workloads. For example, they may elect to run the compute-heavy and complex tasks to the CPU while using customized silicon to perform specific functions. This customized silicon may be in the form of on-package configurable logic such as FPGAs. For example, Intel recently announced that certain future server chips will have an FPGA integrated on package.

FPGAs have been around for a long time and remain popular due to their configurability. They have also found use in accelerating search engines [18]. However, integration onto the same package as the CPU raises interesting opportunities for co-optimization. What type of workloads would most benefit from such an integration? What bandwidth and latency characteristics would make such a model compelling? Such configurability also provides an experimental platform for capabilities that may eventually find their way into the general-purpose CPU. While we use the FPGA example, one can do the same with custom ASICs integrated close to the CPU.

5.4 Influencing Hardware Specialization

Software plays a key role in determining the future of hardware. As new usages and services evolve, so does the software developed to provide them. Early implementations may be limited in performance but specialized techniques develop to improve it. Experimentation for hardware acceleration may occur using fixed function hardware or configurable logic such as FPGAs. The CPU instruction set evolves to match the needs of software. As algorithms reach a tipping point of ubiquity (e.g., floating-point, AES, SHA, RNG, Virtualization) the CPU adds instructions to support them. This integration improves their efficiency and performance.

However, given the development cycle of mainstream processors, support for important emergent algorithms is not immediately available. To aid software development and experimentation, the programming interfaces must abstract the hardware mechanisms from the programmer. This enables a spectrum for hardware acceleration options as shown in Figure 12.

Software development time is a precious resource. Languages and enhancements that enable developers to maximize their efficiency without having to worry about the underlying hardware mechanics provide a productivity win. The community has reacted by abstracting away the underlying hardware mechanism via run-time environments such as Java, OpenCL, etc.

Simple mechanism to use accelerators such as the x87 floating-point ISA allowed emulation of the floating-point mechanics with integer-only code. Today's runtime environments have expanded to run SIMD code on the CPU, GPU or other vector processor. Further hardware enhancements to allow cache coherence and easy page-table sharing between CPU and GPU ease the software burden of moving computes to the optimal spot. We have also seen standards play an important role (IEEE Floating-Point, AES) and help provide consistency in expectations and behaviors.

6. CONCLUDING REMARKS

The challenge for the software industry remains how to continue the cycle of innovation and improved functionality and performance.

In this paper, we discussed the evolution of specialized hardware support in the commodity general-purpose CPU. We are increasingly seeing the incorporation of fixed-function specialization into the CPU as it evolves with the market place. In addition, we are seeing the integration of more platform components with the CPU. This tight integration influences latency and bandwidth and opens up interesting possibilities for software

We expect the commodity general-purpose CPU to continue to provide multidimensional capabilities ranging from improving performance efficiency to increased specialization and increased integration. The performance and specializations in the CPU enable new software architectures. Further, the new software usage models can guide the evolution of specialized hardware support, be it in the CPU or tightly integrated with it. These are exciting times for software with a wide range of options and capabilities from which to choose.

7. ACKNOWLEDGMENTS

We would like to thank Konrad Lai, Gil Neiger, Vinodh Gopal, and Bhushan Chitlur for discussions.

8. REFERENCES

[1] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics Magazine* (April 1965).

[2] Robert H. Dennard, Fritz Gaensslen, Hwa-Nien Yu, Leo Rideout, Ernest Bassous, and Andre Leblanc. Design of ion-implanted MOSFETs with very small physical dimensions. *IEEE Journal of Solid State Circuits* (October 1974).

[3] James Larus. Spending Moore's Dividend. *Communications of the ACM* (May 2009).

[4] David A. Wood. Resolved: Specialized Architectures, Languages, and System Software Should Supplant General-purpose Alternatives Within a Decade. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems* (March 2014).

[5] Mark Bohr. 14 nm Process Technology: Opening New Horizons. *Intel Developer Forum, San Francisco* (September 2014).

[6] Per Hammarlund, Alberto J. Martinez, Atiq A. Bajwa, David L. Hill, Erik G. Hallnor, Hong Jiang, Martin Dixon, Michael Derr, Mikal Hunsaker, Rajesh Kumar, Randy B. Osborne, Ravi Rajwar, Ronak Singhal, Reynold D'Sa, Robert Chappell, Shiv Kaushik, Srinivas Chennupaty, Stéphan Jourdan, Steve Gunther, Thomas Piazza, and Ted Burton. Haswell: The Fourth-Generation Intel Core Processor. *IEEE Micro* (2014).

[7] Urs Hölzle. Brawny cores still beat wimpy cores, most of the time. *IEEE MICRO* (July-Aug 2010).

[8] John F. Palmer. The INTEL® 8087 Numeric Data Processor. In *Proceedings of the May 19022, 1980, National Computer Conference* (May 1980)

[9] Thomas Willhalm, Ismail Oukid, Ingo Müller, and Franz Faerber. Vectorizing Database Column Scans with Complex Predicates. In *Fourth International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures* (Aug 2013).

[10] Herb Schorr. Design principles for a high performance system. In *Proceedings of the Symposium on Computers and Automata* (April 1971).

[11] Tomas Karnagel, Roman Dementiev, Ravi Rajwar, Konrad Lai, Thomas Legler, Benjamin Schlegel, and Wolfgang Lehner. Improving in-memory database index performance

with Intel® Transactional Synchronization Extensions. In *Proceedings of the 20th IEEE International Symposium on High Performance Computer Architecture* (February 2014).

[12] Viktor Leis, Alfons Kemper, and Thomas Neumann. Exploiting hardware transactional memory in main-memory databases. *30th International Conference on Data Engineering* (March 2014).

[13] Richard H. Yoo, Christopher J. Hughes, Konrad Lai, and Ravi Rajwar. Performance evaluation of Intel® transactional synchronization extensions for high-performance computing. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC'13* (November 2013).

[14] Kahraman Akdemir, Martin Dixon, Wajdi Feghali, Patrick Fay, Vinodh Gopal, Jim Guilford, Erdinc Ozturk, Gil Wolrich, and Ronen Zohar. Breakthrough AES Performance with Intel® AES New Instructions. *Whitepaper* (https://software.intel.com/sites/default/files/m/d/4/1/d/8/10TB24_Breakthrough_AES_Performance_with_Intel_AES_New_Instructions.final.secure.pdf).

[15] ARM Limited. ARMv8 Instruction Set Overview (November 2011).

[16] Luiz A. Barroso and Urs Hölzle. The Data Center as a Computer: An Introduction to the Design of Warehouse-Scale Machines. Morgan & Claypool Publishers.

[17] Diane Bryant. The Data Center Opportunity. *Intel Developer Forum, San* Francisco(September *2014*).

[18] Andrew Putnam, Adrian M. Caulfield, Eric S. Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, Michael Haselman, Scott Hauck, Stephen Heil, Amir Hormati, Joo-Young Kim, Sitaram Lanka, James R. Larus, Eric Peterson, Simon Pope, Aaron Smith, Jason Thong, Phillip Yi Xiao, and Doug Burger. A reconfigurable fabric for accelerating large-scale datacenter services. In *Proceedings of the 41st International Symposium on Computer Architecture* (June 2014).