Hardware Trends Impacting Floating-Point Computations In Scientific Applications

Jack Dongarra
University of Tennessee
Oak Ridge National Laboratory
University of Manchester
dongarra@icl.utk.edu

John Gunnels
NVIDIA Corporation
Santa Clara, CA, USA
jgunnels@nvidia.com

Harun Bayraktar
NVIDIA Corporation
Santa Clara, CA, USA
hbayraktar@nvidia.com

Azzam Haidar
NVIDIA Corporation
Santa Clara, CA, USA
ahaidarahmad@nvidia.com

Dan Ernst
NVIDIA Corporation
Santa Clara, CA, USA
dane@nvidia.com

***Abstract*—The evolution of floating-point computation has been shaped by algorithmic advancements, architectural innovations, and the increasing computational demands of modern technologies, such as artificial intelligence (AI) and high-performance computing (HPC). This paper examines the historical progression of floating-point computation in scientific applications and contextualizes recent trends driven by AI, particularly the adoption of reduced-precision floating-point types. The challenges posed by these trends, including the trade-offs between performance, efficiency, and precision, are discussed, as are innovations in mixed-precision computing and emulation algorithms that offer solutions to these challenges. This paper also explores architectural shifts, including the role of specialized and general-purpose hardware, and how these trends will influence future advancements in scientific computing, energy efficiency, and system design.**

***Index Terms*—floating-point, computer architecture, GPU, CPU, emulation, mixed-precision**

I. Introduction

Floating-point computation is foundational to modern scientific applications, enabling the representation of real numbers across a wide range of magnitudes and providing the precision necessary for calculations in fields like physics, chemistry, and engineering. Over the decades, the evolution of floating-point computation has been influenced by the increasing complexity of scientific problems, technological advancements, and the rise of new computational paradigms, such as deep neural network- (DNN-) based AI algorithms [1].

This paper explores the history of floating-point computation, focusing on architectural innovations that have shaped the current landscape. It examines key developments, from early emulation to dedicated hardware, and highlights recent trends, including mixed-precision computing and reduced-precision floating-point types (Figure 1). The impact of these trends on scientific computing and AI is analyzed, along with the challenges they present regarding system design, energy efficiency, and performance.

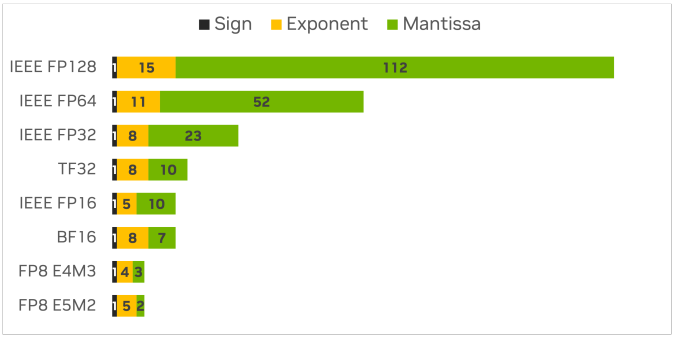

Fig. 1. Various floating-point (FP) representations used today in scientific computing and AI. The exponent bits determine the dynamic range of the FP number, while the mantissa bits determine the precision. Four IEEE FP types are shown: half (FP16), single (FP32), double (FP64), and quad (FP128). TensorFloat-32 (TF32), available on NVIDIA GPUs starting with the Ampere architecture, is a Tensor Core matrix multiply compute mode where input and output are FP32, but input operands are truncated. Bfloat16 (BF16), which was introduced by Google [2], has the same range as FP32 at the expense of mantissa bits. Two variants of FP8, with different splits of exponent and mantissa bits [3] are shown.

II. Benchmarks

Throughout this paper, we will refer to several community benchmarks that have emerged over time, each serving a critical role in evaluating and exposing performance characteristics and limitations of the underlying hardware, as well as serving as a readily conveyed, widely understood record of progress. These benchmarks have become standard tools in the high-performance computing (HPC) community for assessing the efficiency and effectiveness of various computing systems. While in this paper, we focus on floating-point operation-focused benchmarks, other benchmarks exist. One such example is the Graph 500 [4], which measures a system's performance on graph-based problems important for large dataset analysis.

The most well-known benchmark is HPL [5] (High-Performance Linpack). Traditionally used to rank systems in the TOP500 list [6], [7], which focuses on a system's ability to solve dense linear equations, it measures the floating-point computing power of supercomputers, highlighting their raw computational capability. As valuable as it is, HPL emphasizes peak performance of double-precision dense matrix multiplications, which may not always correlate with real-world application performance.

In contrast, the Green500 [8] list focuses on the energy efficiency of the HPL benchmark, measuring the FLOPS per watt delivered by a system. As power consumption becomes an increasingly critical factor in supercomputing, with recent systems approaching 40 MegaWatts of power consumption [7], Green500 plays a pivotal role in pushing the development of energy-efficient architectures and balancing performance with power usage.

HPCG [9] (High-Performance Conjugate Gradient) was introduced to provide a more comprehensive measure of real-world application performance, especially for systems that perform well in terms of memory bandwidth, network latency, and irregular memory access patterns. HPCG aims to capture a broader range of system performance characteristics, some of which HPL might overlook, providing insight into a machine's ability to handle more complex, memory-bound workloads.

A more recent addition, HPL-MxP (formerly called HPL-AI) [10], or HPL for mixed precision, is designed to benchmark systems optimized for AI and machine learning workloads. By focusing on mixed-precision operations, HPL-MxP reflects the growing need for systems capable of efficiently handling lower-precision calculations typical in AI models, exposing the performance capabilities of modern hardware in these emerging domains without abandoning the needs of scientific computing.

Each of these benchmarks exposes different performance aspects of a computer system, from raw computational power and energy efficiency to memory and network bandwidth and latency to flexibility concerning floating-point precision. They give a holistic view of how well a machine will likely perform across various real-world tasks. Viewed against the landscape of history, as seen in Figures 2 and 3, these benchmarks can provide not only a comparator to larger hardware trends (such as processor efficiency or transistor density), but an indication of the factors that will constrain progress in future systems. Taken together, they are essential tools for guiding hardware design and system optimizations in the quest for faster, more efficient, and versatile computing platforms.

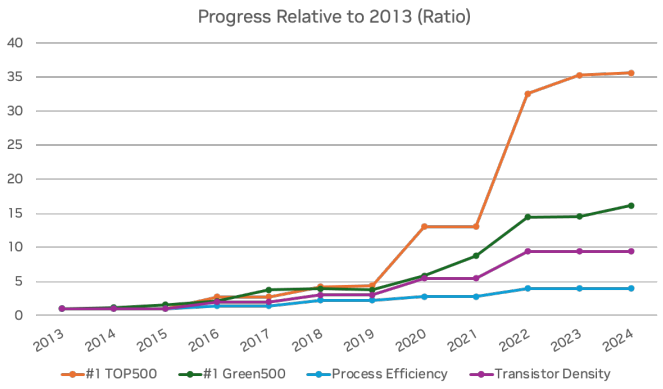

Fig. 2. Historical record of the advances made in transistor density and process efficiency [11], contrasted with the increases seen in the TOP500 and Green500 lists since 2013. All series are normalized to 1.0 at the outset of the chart in 2013. It is notable that both the TOP500 and Green500 entries have improved at a far greater rate than process technology and, further, that the TOP500 (performance) has increased at a greater rate than the Green500 (efficiency).

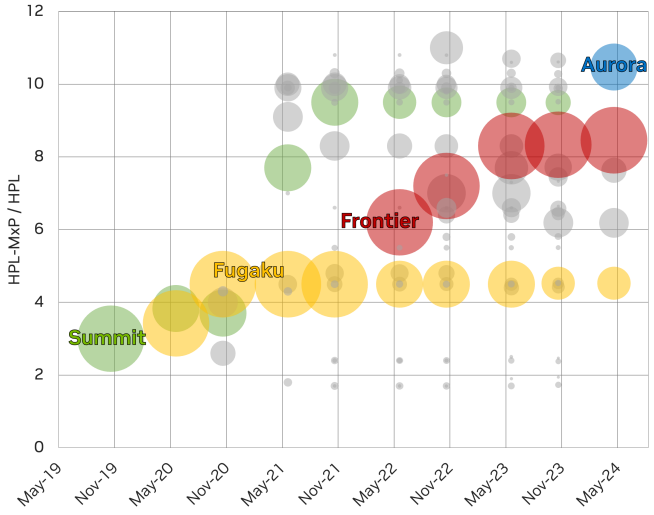

Fig. 3. Ratio of HPL-MxP (formerly HPL-AI) R_{max} to HPL R_{max} over time, since the inception of the HPL-MxP benchmark. Some top-ranked Supercomputers are highlighted with colors and labels. The bubble size is inversely proportional to the Top500 HPL ranking of that particular supercomputer, which explains why Summit or Fugaku bubbles shrink over time. In addition to the general trend of the ratio increasing over time for top-ranked systems, implementation optimizations also increase the speed-up over time, as illustrated by the Summit and Fontier systems.

III. The Evolution of Floating-Point Computation

A. Early Emulation

In the early days of computing, into the mid-1950s, floating-point operations were typically performed via software emulation, where general-purpose processors simulated floating-point arithmetic utilizing fixed-point representations and operations [12]. This approach, while functional, was slow and resource-intensive. Emulation required many CPU cycles for each floating-point operation, making these computations much slower than integer

arithmetic.

Despite its limitations, emulation allowed early computers to perform scientific calculations and helped lay the foundation for future advancements in floating-point hardware throughout the 1960s and 1970s.

B. The Co-Processor Era

The introduction of dedicated floating-point co-processors in the 1980s marked a significant leap forward in floating-point computation. These co-processors, such as Intel's 8087 [13], were separate hardware components designed to handle floating-point operations independently of the CPU. This separation significantly improved performance and allowed computers to tackle more complex scientific problems.

While the 8087 might be the best-known example of this technology, co-processors were in widespread use at the time. For example, the Motorola 68020, relied on external FPUs, such as the 68881 [14], similar to the way in which Intel's pre-x486 and AMD's pre-K5 architectures, used external units like the 8087. External FPUs were also seen in early SPARC systems and MIPS processors, such as the SPARCstation 1 with the Weitek 3170 FPU [15] and the MIPS R4000.

However, using co-processors introduced additional system complexity, requiring specialized hardware and coordination between the CPU and the floating-point unit (FPU). Despite this, co-processors became a staple in high-performance systems, enabling faster scientific computing and simulations.

C. Integration of Floating-Point Units

The release of Intel's x486 [16] processor in 1989 marked a turning point in floating-point computation. The x486 integrated the FPU directly into the CPU, eliminating the need for separate hardware. This integration simplified system design, improved performance, and made floating-point operations a standard feature of general-purpose computing.

As computing requirements expanded, especially in fields such as scientific computing and multimedia, FPUs became standard across many architectures. With the x486, floating-point computation became more accessible and widely used in applications such as computer graphics, simulations, and scientific calculations. This development set the stage for the widespread adoption of floating-point hardware in both consumer and professional computing environments.

Beyond Intel's x86 line, many CPU architectures incorporated FPUs, either as external coprocessors or integrated directly into the CPU, to enhance floating-point performance. Notable examples include the Motorola 68040 [17], which was the first in the 68000 series to integrate the FPU, and the PowerPC 601 [18], which became popular in Apple's early Macintosh systems for its integrated floating-point capabilities.

D. The GPU Revolution

GPUs were initially developed in the 1980s and 1990s to meet the growing demand for 2D and 3D graphics rendering. Early GPUs, produced by companies such as SGI (Silicon Graphics), 3dfx, NVIDIA, and ATI (later acquired by AMD), were primarily focused on enhancing the real-time rendering of images, textures, and geometry, especially in gaming and graphical user interfaces (GUIs). A significant milestone came in 1999 with the release of NVIDIA's GeForce 256 [19], marketed as the first "GPU" capable of processing graphics independently from the CPU. This allowed the CPU to focus on other tasks while the GPU specialized in real-time rendering. GPUs handled critical graphics tasks such as vertex transformations, lighting calculations, and texture mapping, all essential for 3D graphics in gaming and multimedia.

The early 2000s saw another major shift with the introduction of programmable graphics processing units (GPUs). NVIDIA's GeForce3, launched in 2001, was a key milestone in this revolution, offering programmable shaders that allowed developers to directly program the GPU for custom operations, including floating-point computations [20].

The mid-2000s marked a turning point for GPUs, as their parallel processing capabilities were broadly recognized as useful in scientific and engineering computing. Unlike CPUs, which are optimized for serial processing and excel at handling a few threads quickly, GPUs have hundreds or even thousands of cores optimized for parallel workloads, making them ideal for tasks that can be divided into many smaller operations that thousands of threads can execute. This made GPUs especially useful in HPC, where they could accelerate simulations and large-scale mathematical computations, as well as in machine learning and AI, where they became indispensable for training deep learning models.

TABLE I
GPU GENERATIONS: COMPUTE THROUGHPUT VS MEMORY BANDWIDTH

Figure of Merit	Volta (V100)	Ampere (A100)	Hopper (H200)	Blackwell (B200)
FP64 FMA (TFLOP/s)	7.8	9.75	33.5	40
FP64 Tensor (TFLOP/s)	N/A	19.5	67	40
FP16 FMA (TFLOP/s)	31.4	78	134	80
FP16 Tensor (TFLOP/s)	125	312	989	2250
Memory BW (TB/s)	0.9	2.0	4.8	8
FP64 FMA (B/FLOP)	0.124	0.225	0.158	0.220
FP64 Tensor (B/FLOP)	N/A	0.112	0.079	0.220
FP16 FMA (B/FLOP)	0.031	0.028	0.039	0.110
FP16 Tensor (B/FLOP)	0.008	0.007	0.005	0.004

NVIDIA's CUDA [21] (Compute Unified Device Architecture) platform, first introduced in 2006, enabled a broad developer base to harness GPU power for non-graphics workloads. As a result, GPUs, initially designed for rendering graphics, soon became indispensable for general-purpose computing tasks, particularly in scientific computing [22]. Breakthroughs in AI algorithms in the same timeframe [23], [24] rapidly led to leveraging GPUs outside of scientific computing. The

GPU's ability to perform parallel floating-point operations at high speeds, complemented by equally impressive bandwidth capabilities (see Table I), firmly established them as ideal for diverse tasks requiring massive computational power, from training deep learning models to running large-scale simulations in physics and biology.

Today, GPUs are widely used in fields as varied as engineering and scientific simulations, medical research, cryptocurrency mining, big data processing, finance, and countless AI applications, cementing their role as a critical tool for a wide range of computational tasks beyond graphics rendering.

IV. Recent Trends in Floating-Point Computation

A. The Rise of AI and Reduced Precision

Artificial intelligence, particularly deep learning, has profoundly impacted floating-point computation. AI workloads are dominated by matrix multiplications and tensor operations which can tolerate lower precision without a significant loss in accuracy. As a result, reduced-precision floating-point types, such as FP16 (half-precision), BF16 (brain floating-point, also referred to as bfloat16), and, most recently, FP8 [3] have become increasingly popular in AI applications (see Figure 1).

Reduced precision offers several advantages, including faster computations and lower energy consumption. In AI, especially in training and inference tasks, models can maintain high accuracy using lower precision, which leads to improved throughput and efficiency. This shift has driven the development of specialized hardware, such as NVIDIA's Tensor Cores [25], initially optimized for matrix operations at a reduced precision.

It should be noted that the move toward reduced precision capabilities poses challenges for applications that require higher accuracy, such as scientific simulations and financial modeling. While AI applications can often tolerate lower precision, many scientific fields depend on high-precision calculations to ensure the validity of their results.

B. Mixed-Precision Computing

Mixed-precision computing has emerged as a promising solution to address the limitations of reduced precision. Mixed-precision algorithms [26], distinct from the mixed-precision operations they sometimes leverage, dynamically adjust the precision of floating-point calculations based on the accuracy required for each specific task. This approach allows systems to use lower precision for less critical calculations while reserving higher precision for tasks that require greater accuracy (see [27]), requiring careful algorithmic design to ensure that performance gains from using lower precision do not come at the expense of accuracy.

In our discussion of mixed-precision computation, we will explore an approach that involves using different levels of precision at various stages of the computation process for solving a dense system of equations. Specifically, one phase of the computation, where the highest level of accuracy is not critical, may be performed using a lower (or even fixed) precision, such as half-precision (FP16), to optimize performance and reduce resource consumption [28].

As the computation progresses, we will switch to a higher precision, such as single-precision (FP32) or double-precision (FP64), for parts of the process where increased accuracy is essential. This transition is necessary in phases where errors accumulated during earlier steps must be corrected or refined, ensuring that the final result meets the desired accuracy thresholds. The dynamic adjustment between lower and higher precision enables a balance between computational speed, energy efficiency, and numerical precision. By strategically employing mixed precision, we can achieve significant performance gains without compromising the overall accuracy of the computation.

Mixed-precision computing has proven especially useful in AI [29] and HPC [30]. For example, in deep learning, models can be trained using a combination of FP8, BF16, FP16, and FP32 (single precision) calculations, reducing the computational load without sacrificing accuracy. The same domain affords opportunities for finer-grained application of this technique. For example, forward propagation through a neural network typically requires higher precision for weights and activations. In contrast, gradients in the backward propagation (used for updating weights) require a higher dynamic range. This has led to the introduction of two different FP8 variants, E4M3 and E5M2 [3]. Mixed-precision algorithms must manage these precision transitions efficiently to maximize performance without sacrificing the quality of results.

TABLE II
Mixed-precision iterative refinement solver (from cuSOLVER) performance and efficiency for solution of a 32k double-precision complex system of equations

		Volta (V100)	**Ampere (A100)**	**Hopper (H200)**
Performance TFLOP/s	FP64	6.6	16.9	42.6
	FP16+FP64 MxP	34.6	74.6	124.2
Efficiency GFLOP/s/Watt	FP64	28	45	78
	FP16+FP64 MxP	173	262	529

This technique has also been applied in scientific computing, in cases where certain phases of simulations can tolerate lower precision, allowing for faster computations (see Table II and Figure 4). Using error-correction techniques, such as stochastic rounding [31] and iterative refinement to prevent the propagation of errors, allows mixed-precision algorithms to maintain high accuracy even when using reduced precision for certain operations, making mixed-precision computing suitable

for various scientific and AI applications. By optimizing pre-

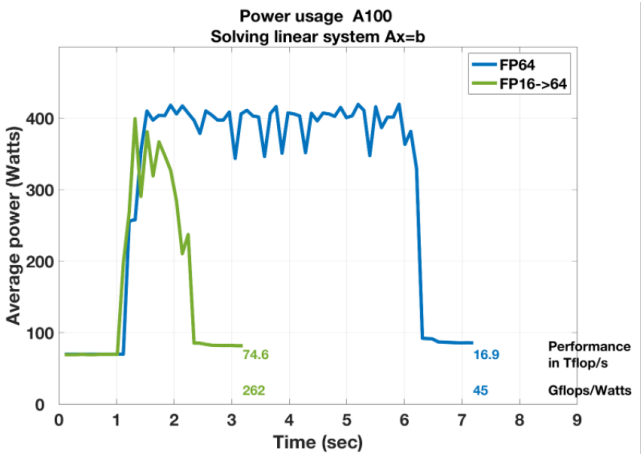

Fig. 4. Representative power consumption curves measured on an NVIDIA Ampere A100 GPU during the execution of two different equations solvers is shown. The blue line shows the FP64 LU solver (corresponds to ZGETRF & ZGETRS in LAPACK) while the green line shows the Tensor Core accelerated mixed-precision iterative refinement solver available in the cuSOLVER library, which relies on cuBLAS for Level 3 BLAS operations, both for a matrix size of 32000. The mixed-precision solver is 4.4 times faster and 5.8 times more power-efficient. Comparison with other GPU architectures can be found in Table II.

cision based on task requirements, mixed-precision computing improves both performance and energy efficiency, making it an essential technique for modern floating-point computation.

C. Emulation in Modern Systems

Emulation, once a necessity in the absence of dedicated floating-point hardware, has come back in modern systems. Emulation techniques have evolved in terms of their underlying methods, implementations, and the flexibility with which they leverage (plentiful) hardware resources [32]–[34]. This adaptability is particularly important in domains such as AI and HPC, where computational needs often exceed the capabilities of the more established, higher-precision hardware resources.

TABLE III
PRELIMINARY HPL PERFORMANCE AND EFFICIENCY MEASUREMENTS ON BLACKWELL B200 GPU COMPARING EMULATION WITH S=7 [33] TO NATIVE FP64 [DATA SUBJECT TO CHANGE]

		Native FP64	**Emulation (s=7)**	**Ratio**
At Maximum Performance	TFLOP/s	34.5	68.4	2.0
	GFLOP/s/Watt	41.7	71.3	1.7
At Maximum Efficiency	TFLOP/s	23.1	53.4	2.3
	GFLOP/s/Watt	51.4	82.1	1.6

For example, high-precision floating-point operations can be emulated on hardware designed for lower, or even fixed-, precision, allowing systems to balance performance and accuracy. Emulation also provides a novel means to support evolving computational demands without requiring a complete hardware overhaul and something of an added degree of freedom in designing new systems, as it allows some components to serve a dual purpose. This is demonstrated in Table III where the performance of HPL employing native FP64 and emulation using the techniques described in [33] are compared. When configured for maximum performance, emulation yields a two-fold improvement in performance and 70% improvement in power efficiency. In contrast, when maximum efficiency is the target, emulation yields a 2.3x speed-up and a 60% improvement in power efficiency.

V. CHALLENGES IN FLOATING-POINT COMPUTATION

A. Balancing Precision and Efficiency

One of the key challenges in floating-point computation is finding the right balance between precision and efficiency. High-precision formats like FP64 (double-precision) are necessary for certain scientific applications, but they come at a cost regarding speed and energy consumption, as seen in Figure 3. In contrast, reduced-precision formats like FP16 are much faster and more efficient but may not provide the accuracy needed for all tasks (See Figure 1).

This trade-off is particularly pronounced when considering fields like AI, where reduced precision is sufficient for many tasks, and scientific computing, where precision cannot be sacrificed. Today, the same systems often run applications of both stripes and the integration of AI methods and scientific computing is a burgeoning field of study [35]. As a result, system designers must carefully consider the needs of each application when selecting the appropriate precision format.

B. Specialized vs. General-Purpose Hardware

Another challenge in floating-point computation is the tension between specialized hardware (such as custom FPUs with the ability to execute compound instructions) and general-purpose processors (CPUs). On the one hand, specialized hardware is optimized for specific tasks, such as matrix multiplications in AI, but may lack the flexibility needed for broader applications. General-purpose processors, on the other hand, can handle a wide range of tasks but may not be as efficient for specialized computations.

VI. ARCHITECTURAL INNOVATIONS AND THEIR IMPACT

A. Heterogeneous Computing

Heterogeneous computing has become a cornerstone of modern floating-point computation, combining CPUs, GPUs, and other accelerators to optimize performance. By accelerating floating-point operations via different types of processors, systems can achieve higher performance and energy efficiency.

Unsurprisingly, the tolerance that programmers once had for (relatively) distant accelerator engines, in terms of both programmability and access latency, has decreased over the years and concerted efforts have been undertaken to enable practitioners to “eat their cake and have it too.”

Almost a decade ago, the Sierra and Summit supercomputing systems, at Lawrence Livermore (LLNL) and Oak Ridge (ORNL) National Laboratories, respectively, leveraged the first generation of NVLINK to tightly couple IBM's POWER9 processor to NVIDIA's Volta (V100) GPU [36]. Today, NVIDIA's Grace Hopper Superchip architecture [37] integrates tightly coupled CPU and GPU components, enabling seamless transitions between general-purpose and specialized processing. This integration allows systems to handle highly serial sparse calculations, high-throughput scientific calculations, and reduced-precision AI workloads efficiently. AMD's Instinct MI300A and Apple's M3 [38]–[40] are additional realizations of the goal to tightly couple distinct compute resources in a way that allows them to be viewed less as separate entities and more as a potent, unified resource.

Entwined with the drive for tighter integration, the trend of designing custom silicon tailored to specific software needs has grown significantly. Companies like Google and Apple have invested in designing chips, such as Google's TPUs for the acceleration of AI workloads and Apple's A-series chips for sophisticated mobile devices. These chips are designed in tandem with the software they will run, allowing for optimizations that general-purpose hardware cannot achieve. This hardware-software co-design allows for significant improvements in performance and efficiency, as the hardware is fine-tuned to the software's needs.

Addressing the needs of the developer community, heterogeneous computing environments [41]–[44] offer several advantages, including the ability to optimize each task for the most suitable processor. This approach not only improves performance, but also reduces energy consumption for a broad range of applications by offloading computationally intensive tasks to specialized hardware.

B. Energy Efficiency in Floating-Point Hardware

Energy efficiency has become a critical consideration in the design of floating-point hardware, particularly in large-scale computing environments like data centers and supercomputers. As computational workloads continue to grow, the energy required to power these systems has become a significant constraint, approaching 40 MegaWatts. The centrality of this is reflected in the growing attention given to the Green500 list, described in Section II.

Introduced by NVIDIA with the Volta [45] architecture in 2017, Tensor Cores are specialized hardware designed to accelerate matrix multiply-accumulate (MMA) operations, which are critical for deep learning tasks such as matrix multiplication and convolution in neural networks. These cores are particularly efficient due to the use of complex instruction types optimized for high-throughput operations (Table I). Tensor Cores are available across many different precisions, but are highly utilized in AI workloads that make extensive use of dense matrix operations. This complex operation approach significantly speeds up computations and reduces energy requirements without compromising requisite accuracy (see Figure 4 and Table II).

Tensor Cores [46] excel at accelerating matrix multiplications ($A[m \times k] \cdot B[k \times n]$), a fundamental operation in neural networks. They process these operations in blocks (e.g., (m, n, k) = (16, 16, 8)) [47], which boosts throughput for critical tasks like forward and backpropagation. This allows neural network operations that typically require hundreds of regular GPU instructions to be executed with relatively few tensor operations executed in a fraction of the time. Furthermore, Tensor Cores enable massive parallelism by executing multiple floating-point operations simultaneously and in a systolic manner, resulting in highly efficient throughput for deep learning models such as convolutional neural networks (CNNs) [24] or transformers [48]. Additionally, their power efficiency is notable, as the ability of a single Tensor Core to perform operations that would require multiple steps on a traditional GPU reduces energy consumption—a key consideration for both chip-level performance limits and the capabilities of large-scale data centers.

Energy-efficient hardware is especially important in AI and HPC environments, where systems must process massive amounts of data while minimizing their environmental impact. The development of low-power FPUs and specialized processors has enabled these systems to meet the growing demand for computational power without exceeding energy constraints.

VII. The Role of Floating-Point in AI and HPC

A. AI-Driven Workloads

Artificial intelligence, particularly deep learning, has revolutionized floating-point computation. AI workloads are characterized by large-scale matrix multiplications requiring massive computational power. Reduced-precision floating-point types like FP16 have become the standard for these workloads, offering the best balance between performance and accuracy [49].

Tensor operations, which are the foundation of most AI models, benefit from the parallel processing capabilities of GPUs and Tensor Cores. These specialized processors are optimized for matrix operations, allowing AI models to be trained and deployed more quickly and efficiently.

As AI grows in importance, the demand for floating-point hardware that can handle AI-specific workloads will increase. This trend will drive further innovations in reduced-precision computing and specialized hardware for AI.

B. Scientific Computing and High Precision

In contrast to AI, scientific computing often requires high-precision floating-point calculations. Most simulations in fields like molecular dynamics, computational mechanics,

and fluid dynamics depend on FP64 precision to ensure the accuracy of their results. These calculations are typically run on HPC systems designed to handle large-scale simulations that require high precision and significant computational power.

While reduced-precision techniques are becoming more common in scientific computing [50], [51], they are often used in conjunction with high-precision calculations. This approach improves the speed and efficiency of scientific simulations, enabling researchers to run more complex models in less time without sacrificing accuracy.

VIII. Hardware's Impact on Software

The interaction between hardware and software is deeply interdependent, with hardware setting the constraints within which software must operate. As hardware evolves, it directly influences how software is designed, written, and optimized. This co-evolution drives both fields forward, as hardware improvements open new opportunities for software innovation while software demands push hardware development to new heights.

A. Performance and Capabilities

The performance of hardware, particularly in terms of floating-point computation, dictates the upper limits of what software can achieve. For example, the availability of GPUs and Tensor Cores with specialized floating-point capabilities has enabled popular AI frameworks like PyTorch [52], TensorFlow [53], and JAX [54] to handle large-scale matrix operations more efficiently. As a result, software developers can design more complex models and algorithms that rely on the enhanced performance provided by these hardware features without undue consideration of architectural details.

B. Instruction Sets and Architectures

On CPUs, the hardware architecture, such as x86, ARM, or RISC-V, determines the instruction sets (ISAs) available to software. Software must be compatible with the hardware's architecture, which affects how low-level operations are performed and how efficiently the software can execute. The introduction of SIMD (Single Instruction, Multiple Data) and specialized floating-point instructions has allowed scientific and AI software to accelerate matrix operations and other floating-point-intensive tasks.

Similarly, on GPUs, the Parallel Thread Execution (PTX) [47] ISA is available to enable it as a computing device. PTX is a part of CUDA, the parallel computing platform developed by NVIDIA that enables GPUs to perform general-purpose computing tasks beyond graphics rendering. It allows developers to leverage the massive parallel processing power of GPUs by offloading compute-intensive tasks from the CPU. CUDA provides a unified architecture that manages thousands of threads in a SIMT model (Single Instruction, Multiple Threads), supports various precision modes (such as FP16 and FP64), and optimizes memory access, making it highly efficient for parallel workloads like scientific computing, AI, and machine learning.

As NVIDIA's GPUs have evolved, CUDA has remained central to utilizing them to maximum advantage, particularly with the introduction of Tensor Cores for mixed-precision operations, beginning with the Volta architecture. These innovations have led to dramatic increases in compute throughput, especially in AI/ML applications (Table I). Memory bandwidth has also significantly improved, offering higher Bytes/FLOP for non-Tensor Core operations. CUDA's ability to manage parallel execution, memory hierarchy, and precision makes it essential for extracting maximum performance from modern GPUs.

C. Specialized Hardware and Software Paradigms

Developing specialized hardware, such as GPU Tensor Cores and Tensor Processing Units (TPUs) [55], has created new software paradigms. For instance, deep learning frameworks are optimized to leverage the parallelism of GPUs, which has drastically improved the training times for neural networks. Without these hardware advances, modern AI software could not scale to the levels required for training models like GPT-4 [56] or other large neural networks. Conversely, this is an example of the inseparability of hardware and software. Because software has been able to leverage hardware capabilities to great advantage, technology has been pushed to deliver ever greater resources to supply the needs of applications.

IX. Software's Impact on Hardware

Software has increasingly become a driving force in hardware design, as complex and demanding applications push the limits of existing hardware capabilities. As software grows more intricate, hardware must evolve to meet the demands for greater performance, efficiency, and flexibility.

A. Energy Efficiency and Power Constraints

As software applications become more resource-intensive, hardware must prioritize energy efficiency to maintain performance without consuming unsustainable amounts of power. Big data, real-time analytics, and AI-driven applications have all contributed to a demand for hardware that can deliver high performance per watt. In response, hardware manufacturers have developed energy-efficient architectures, such as Arm processors for data centers, GPUs with dynamic voltage and frequency scaling (DVFS) [57], as well as applications of this technology to commodity desktop and server processors, for example Intel's SpeedStep [58] and AMD's Cool'n'Quiet [59].

X. Non-Standard Data Types

While floating-point representations, from FP8 to FP64, dominate AI and scientific applications, several non-standard data types, and their corresponding computational mechanisms, have emerged in recent years. These data types offer

marked potential advantages in terms of precision, energy efficiency, and computational speed, particularly in specialized applications. Exotic formats and technologies such as posits [60], Spiking Neural Networks (SNNs) [61], and analog computing [62] offer tantalizing potential advantages along several axes of interest, but they face significant hurdles in terms of hardware and software support, scalability, and noise management. As computing demands evolve, these data types could find greater adoption in specialized fields where their advantages are most beneficial and their shortcomings are acceptable or less keenly felt, for example, in environments requiring extreme levels of energy efficiency or domains with specific accuracy requirements.

XI. Algorithmic Complexity and Memory Bandwidth

A. Impact of Algorithmic Complexity on Floating-Point Performance

The complexity of algorithms significantly influences the efficiency of floating-point operations. For example, dense linear algebra algorithms, such as matrix multiplications, exhibit high floating-point operation intensity (*a.k.a.* arithmetic intensity) and are well-suited to GPUs with high floating-point throughput. In contrast, sparse matrix operations often require irregular memory accesses, leading to memory bandwidth bottlenecks that reduce floating-point efficiency. This impact is so broadly and acutely felt, especially in contexts that require real-time data processing, that multiple high-profile benchmarks, most notably HPCG (see section II), provide a figure of merit for systems, largely based on this characteristic. As a result, optimizing for memory access patterns becomes a critical aspect of hardware-software co-design in HPC and AI. Table I shows pertinent GPU specifications for FP16 and FP64 computations and the corresponding Bytes/FLOP values which are important for application performance.

The introduction of high-bandwidth memory (HBM) [63] and on-chip memory hierarchies has helped mitigate some of these issues by providing faster data access and reducing the latency associated with memory operations. Figure 5 plots the data from Table I, illustrating improvements over time in the data access per floating-point operation over generations, which is important for a very broad class of applications that cannot leverage high-throughout matrix multiply operations.

Even with these hardware innovations, one of the remaining key challenges in HPC and AI applications is performance optimization across high and low arithmetic intensity components of complex algorithms. Overcoming these challenges often involves a technique called kernel fusion, where large amounts of repeated data load and store operations are avoided by combining multiple kernels, sometimes at the expense of performing some extra floating-point operations. A good example of this, from Transformers for LLM applications, is Flash Attention and its variants [64], [65].

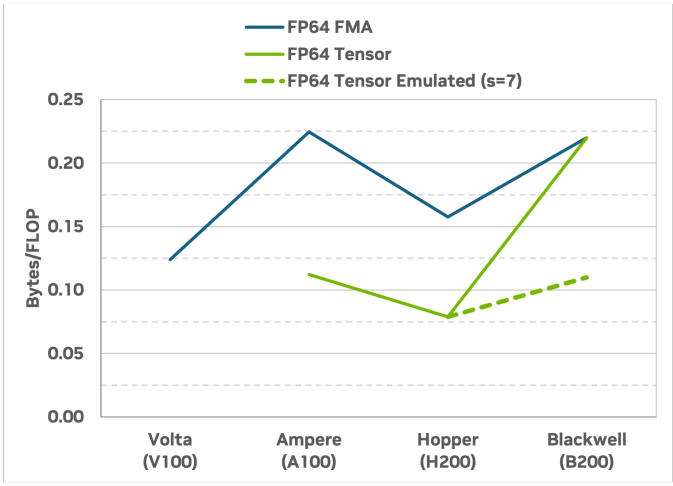

Fig. 5. Comparison of Bytes/FLOP across four generations of GPUs for FMA and Tensor Core throughput from Table I. The dashed line shows Tensor Core accelerated DGEMM performance using integer based emulation with 7 slices.

XII. Future Directions in Floating-Point Computation

A. Advances in Mixed-Precision Techniques

As AI and scientific computing evolve, mixed-precision computing will become increasingly important. Future systems will likely incorporate more sophisticated algorithms that dynamically adjust precision levels to optimize performance and energy efficiency. These systems will be able to switch between FP8, BF16, FP16, FP32, and FP64 as needed, ensuring that each task is handled with the appropriate level of precision.

This flexibility will be particularly valuable in environments where both AI and scientific computing tasks are performed, as it will allow systems to optimize for both speed and accuracy without compromise.

B. Emulation for Flexibility

Emulation will continue to play a key role in floating-point computation, particularly as new applications require higher precision or more specialized calculations. Emulation provides a flexible solution for extending the capabilities of existing hardware, allowing systems to perform floating-point operations that are not natively supported by the hardware, and offering power efficiency gains (see Table III).

As computational demands grow, emulation will become an increasingly valuable tool for maintaining flexibility and extending the lifespan of hardware systems.

C. Energy-Efficient Designs

The need for energy-efficient floating-point hardware will only increase as computational workloads grow. Future innovations in floating-point design will focus on reducing power consumption while maintaining high performance. This will

involve the development of more energy-efficient FPUs and new architectures that minimize the energy cost of floating-point operations. These innovations will be significant in data centers and HPC environments, where energy consumption is a major constraint on system performance.

XIII. Conclusion

The evolution of floating-point computation has been driven by advancements in both hardware and software, shaped by the demands of scientific research, artificial intelligence, and high-performance computing. As the field continues to evolve, innovations in mixed-precision computing, emulation, energy-efficient design, and non-standard data types will play a critical role in meeting the growing demands of modern applications. By balancing the competing needs for precision, performance, and energy efficiency, floating-point hardware will remain a key component of scientific and AI-driven computation, enabling future breakthroughs in research and technology.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[2] D. Kalamkar, D. Mudigere, N. Mellempudi, D. Das, K. Banerjee, S. Avancha, D. T. Vooturi, N. Jammalamadaka, J. Huang, H. Yuen, J. Yang, J. Park, A. Heinecke, E. Georganas, S. Srinivasan, A. Kundu, M. Smelyanskiy, B. Kaul, and P. Dubey, "A study of bfloat16 for deep learning training," 2019.

[3] P. Micikevicius, D. Stosic, N. Burgess, M. Cornea, P. Dubey, R. Grisenthwaite, S. Ha, A. Heinecke, P. Judd, J. Kamalu, N. Mellempudi, S. Oberman, M. Shoeybi, M. Siu, and H. Wu, "Fp8 formats for deep learning," 2022.

[4] R. C. Murphy, K. Pingali, J. D. Feo, and D. A. Bader, "Introducing the graph 500," *Cray User Group (CUG)*, 2010.

[5] J. J. Dongarra, "The linpack benchmark: Past, present, and future," *Concurrency and Computation: Practice and Experience*, vol. 15, no. 9, pp. 803–820, 2003.

[6] J. Dongarra and P. Luszczek, *TOP500*, pp. 2055–2057. Boston, MA: Springer US, 2011.

[7] Erich Strohmaier, Jack Dongarra, Horst Simon, Martin Meuer, "Top 500. The List.," June 2024. Website and database.

[8] W. Feng and K. Cameron, "The green500 list: Encouraging sustainable supercomputing," *Computer*, vol. 40, no. 12, pp. 50–55, 2007.

[9] J. J. Dongarra, M. A. Heroux, and P. Luszczek, "A new benchmark for ranking high performance computing systems," Tech. Rep. UT-EECS-13-736, University of Tennessee, 2013.

[10] J. J. Dongarra, P. Luszczek, S. Tomov, and M. A. Heroux, "Hpl-ai mixed-precision benchmark: The next frontier of supercomputing," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, 2021.

[11] [Accessed 16-11-2024].

[12] J. W. Backus, "The IBM 701 Speedcoding system," *j-J-ACM*, vol. 1, pp. 4–6, Jan. 1954.

[13] J. Palmer, "The intel 8087 numeric data processor," in *Proceedings of the 7th Annual Symposium on Computer Architecture, La Baule, France, May 6-8, 1980* (J. Lenfant, B. R. Borgerson, D. E. Atkins, K. B. Irani, D. Kinniment, and H. Aiso, eds.), pp. 174–181, ACM, 1980.

[14] C. Huntsman and D. Cawthron, "The MC68881 floating-point coprocessor," *IEEE Micro*, vol. 3, pp. 44–54, Nov./Dec. 1983.

[15] M. Birman, A. Samuels, G. Chu, T. Chuk, L. Hu, J. McLeod, and J. Barnes, "Developing the WTL3170/3171 Sparc floating-point coprocessors," *IEEE Micro*, vol. 10, pp. 55–64, Jan./Feb. 1990.

[16] W. A. Triebel, *The 80386, 80486, and Pentium Microprocessors: Hardware, Software, and Interfacing*. Simon & Schuster Trade, 1st ed., 1997.

[17] R. Edenfield, M. Gallup, W. Ledbetter, R. McGarity, E. Quintana, and R. Reininger, "The 68040 processor. i. design and implementation," *IEEE Micro*, vol. 10, no. 1, pp. 66–78, 1990.

[18] M. T. Vaden, L. J. Merkel, C. R. Moore, T. M. Potter, and R. J. Reese, "Design considerations for the powerpc 601 microprocessor," *IBM Journal of Research and Development*, vol. 38, no. 5, pp. 605–620, 1994.

[19] W. J. Dally, S. W. Keckler, and D. B. Kirk, "Evolution of the graphics processing unit (gpu)," *IEEE Micro*, vol. 41, no. 6, pp. 42–51, 2021.

[20] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan, "Brook for gpus: stream computing on graphics hardware," *ACM Trans. Graph.*, vol. 23, p. 777–786, Aug. 2004.

[21] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda.," in *SIGGRAPH Classes*, pp. 16:1–16:14, ACM, 2008.

[22] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "Gpu computing," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, 2012. A comprehensive overview of GPU computing and its application in various fields, including scientific simulations.

[23] R. Raina, A. Madhavan, and A. Y. Ng, "Large-scale deep unsupervised learning using graphics processors," in *Proceedings of the 26th annual international conference on machine learning*, pp. 873–880, ACM, 2009.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems* (F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.

[25] NVIDIA Corporation, "Nvidia v100 gpu architecture," 2017. White paper.

[26] M. Baboulin, A. Buttari, J. Dongarra, J. Kurzak, J. Langou, J. Langou, P. Luszczek, and S. Tomov, "Accelerating scientific computations with mixed precision algorithms," *Computer Physics Communications*, vol. 180, no. 12, pp. 2526–2533, 2009.

[27] A. Abdelfattah, H. Anzt, E. G. Boman, E. Carson, T. Cojean, J. Dongarra, A. Fox, M. Gates, N. J. Higham, X. S. Li, J. Loe, P. Luszczek, S. Pranesh, S. Rajamanickam, T. Ribizel, B. F. Smith, K. Swirydowicz, S. Thomas, S. Tomov, Y. M. Tsai, and U. M. Yang, "A survey of numerical linear algebra methods utilizing mixed-precision arithmetic," *Int. J. High Perform. Comput. Appl.*, vol. 35, p. 344–369, July 2021.

[28] A. Haidar, A. Abdelfattah, M. Zounon, P. Wu, S. Pranesh, S. Tomov, and J. Dongarra, "The design of fast and energy-efficient linear solvers: On the potential of half-precision arithmetic and iterative refinement techniques," in *International Conference on Computational Science*, pp. 586–600, Springer, 2018.

[29] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proceedings of the 32nd International Conference on Machine Learning* (F. Bach and D. Blei, eds.), vol. 37 of *Proceedings of Machine Learning Research*, (Lille, France), pp. 1737–1746, PMLR, 07–09 Jul 2015.

[30] M. Clark, R. Babich, K. Barros, R. Brower, and C. Rebbi, "Solving lattice qcd systems of equations using mixed precision solvers on gpus," *Computer Physics Communications*, vol. 181, no. 9, pp. 1517–1528, 2010.

[31] M. Croci, M. Fasi, N. J. Higham, T. Mary, and M. Mikaitis, "Stochastic rounding: implementation, error analysis and applications," *Royal Society Open Science*, vol. 9, no. 3, p. 211631, 2022.

[32] H. Ootomo and R. Yokota, "Recovering single precision accuracy from tensor cores while surpassing the FP32 theoretical peak performance," *Int. J. High Performance Computing Applications*, vol. 36, p. 475–491, June 2022.

[33] Y. Uchino, K. Ozaki, and T. Imamura, "Performance enhancement of the ozaki scheme on integer matrix multiplication unit," 2024.

[34] G. Henry, P. T. P. Tang, and A. Heinecke, "Leveraging the bfloat16 artificial intelligence datatype for higher-precision computations," 2019.

[35] A. Lavin, D. Krakauer, H. Zenil, J. Gottschlich, T. Mattson, J. Brehmer, A. Anandkumar, S. Choudry, K. Rocki, A. G. Baydin, C. Prunkl, B. Paige, O. Isayev, E. Peterson, P. L. McMahon, J. Macke, K. Cranmer, J. Zhang, H. Wainwright, A. Hanuka, M. Veloso, S. Assefa, S. Zheng, and A. Pfeffer, "Simulation intelligence: Towards a new generation of scientific methods," 2022.

[36] I. P. N. team, "Functionality and performance of nvlink with ibm power9 processors," *IBM J. Res. Dev.*, vol. 62, p. 9:1–9:10, July 2018.

[37] NVIDIA, "Nvidia gh200 grace hopper superchip architecture," 2023.

[38] S. Tandon, L. Grinberg, G.-T. Bercea, C. Bertolli, M. Olesen, S. Bnà, and N. Malaya, "Porting hpc applications to amd instinct™ mi300a using unified memory and openmp," 2024.

[39] [Accessed 14-11-2024].

[40] "Apple unveils m3, m3 pro, and m3 max, the most advanced chips for a personal computer." [Accessed 14-11-2024].

[41] N. A. Simakov, M. D. Jones, T. R. Furlani, E. Siegmann, and R. J. Harrison, "First impressions of the nvidia grace cpu superchip and nvidia grace hopper superchip for scientific workloads," in *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region Workshops*, HPCAsia '24 Workshops, (New York, NY, USA), p. 36–44, Association for Computing Machinery, 2024.

[42] S. Mittal and J. S. Vetter, "A survey of cpu-gpu heterogeneous computing techniques," *ACM Comput. Surv.*, vol. 47, July 2015.

[43] B. Saha, X. Zhou, H. Chen, Y. Gao, S. Yan, M. Rajagopalan, J. Fang, P. Zhang, R. Ronen, and A. Mendelson, "Programming model for a heterogeneous x86 platform," in *Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '09, (New York, NY, USA), p. 431–440, Association for Computing Machinery, 2009.

[44] M. Garland, M. Kudlur, and Y. Zheng, "Designing a unified programming model for heterogeneous machines," in *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pp. 1–11, 2012.

[45] Z. Jia, M. Maggioni, B. Staiger, and D. P. Scarpazza, "Dissecting the nvidia volta gpu architecture via microbenchmarking," 2018.

[46] NVIDIA Corporation, "Tensor cores," 2024. Website.

[47] *CUDA PTX ISA*. NVIDIA, May 2024. Release 8.5.

[48] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, (Red Hook, NY, USA), p. 6000–6010, Curran Associates Inc., 2017.

[49] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, *et al.*, "Mixed precision training," *International Conference on Learning Representations (ICLR)*, 2018.

[50] N. J. Higham and S. Pranesh, "Simulating low precision floating-point arithmetic," 2019.

[51] S. Hatfield, A. Subramanian, T. Palmer, and P. Düben, "Improving weather forecast skill through reduced-precision data assimilation," *Monthly Weather Review*, vol. 146, no. 1, pp. 49 – 62, 2018.

[52] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," 2019.

[53] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: a system for large-scale machine learning," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, (USA), p. 265–283, USENIX Association, 2016.

[54] James Bradbury and Roy Frostig and Peter Hawkins and Matthew James Johnson and Chris Leary and Dougal Maclaurin and George Necula and Adam Paszke and Jake VanderPlas and Skye Wanderman-Milne and Qiao Zhang, "JAX: composable transformations of Python+NumPy programs," 2018.

[55] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datacenter performance analysis of a tensor processing unit," *SIGARCH Comput. Archit. News*, vol. 45, p. 1–12, June 2017.

[56] OpenAI, "Gpt-4 technical report," *ArXiv*, vol. abs/2303.08774, 2023.

[57] E. Le Sueur and G. Heiser, "Dynamic voltage and frequency scaling: the laws of diminishing returns," in *Proceedings of the 2010 International Conference on Power Aware Computing and Systems*, HotPower'10, (USA), p. 1–8, USENIX Association, 2010.

[58] D. Genossar and N. Shamir, "Intel® Pentium® M processor power estimation, budgeting, optimization and validation," *j-INTEL-TECH-J*, vol. 7, pp. 44–49, May 2003.

[59] R. Basmadjian and H. de Meer, "Evaluating and modeling power consumption of multi-core processors," in *Proceedings of the 3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet*, e-Energy '12, (New York, NY, USA), Association for Computing Machinery, 2012.

[60] Gustafson and Yonemoto, "Beating floating point at its own game: Posit arithmetic," *Supercomput. Front. Innov.: Int. J.*, vol. 4, p. 71–86, June 2017.

[61] S. Höppner, Y. Yan, B. Vogginger, C. Liu, F. Kelber, A. Dixius, S. Scholze, J. Partzsch, M. Stolba, F. Neumärker, G. Ellguth, S. Hartmann, S. Schiefer, T. Hocker, D. Walter, G. Liu, M. Mikaitis, J. Garside, S. Furber, and C. Mayr, "The SpiNNaker 2 processing element architecture for hybrid digital neuromorphic computing." arXiv:2103.08392 [cs.AR], Aug. 2022.

[62] C. J. Maley, "Analog and digital, continuous and discrete," *Philosophical Studies*, vol. 155, pp. 117–131, 2011.

[63] H. Jun, J. Cho, K. Lee, H.-Y. Son, K. Kim, H. Jin, and K. Kim, "Hbm (high bandwidth memory) dram technology and architecture," in *2017 IEEE International Memory Workshop (IMW)*, pp. 1–4, 2017.

[64] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, "Flashattention: Fast and memory-efficient exact attention with io-awareness," 2022.

[65] T. Dao, "Flashattention-2: Faster attention with better parallelism and work partitioning," 2023.