Massive Parallel Hashing: Principles, Applications, and Technologies

1. Introduction

Overview

Massive Parallel Hashing represents the convergence of two fundamental computer science concepts: hashing and parallel processing. It involves the application of parallel computing architectures and algorithms to significantly accelerate the computation of hash functions, often operating on vast amounts of data or exploring extensive search spaces. This approach leverages the power of multiple processing units, frequently employing specialized hardware, to achieve throughput levels unattainable by traditional serial methods. The significance of massive parallel hashing spans numerous domains, including cybersecurity, large-scale data analytics, scientific computing, and the foundational mechanisms of various cryptocurrencies. Its ability to perform hashing operations at scale enables tasks ranging from rapid password security auditing to efficient data indexing in petabyte-scale databases and the verification of transactions in blockchain networks.

Purpose and Scope

This report aims to provide a comprehensive analysis of massive parallel hashing. It delves into the core definitions of hashing and parallel processing, explaining how their synergy forms the basis of massive parallel hashing. The report explores the primary applications and diverse use cases, identifies the specialized hardware commonly employed, and evaluates the inherent advantages and disadvantages associated with this technique. Furthermore, it offers a comparative analysis against traditional serial hashing methods, addressing the key aspects outlined in the initial query to deliver a thorough understanding of this critical computational paradigm.

2. Fundamentals: Hashing and Parallel Processing

2.1 Defining Hashing

Core Concept

Hashing is a fundamental process in computer science that transforms an input data item, often referred to as a key or message, of arbitrary size into a fixed-size output value.[1] This output, known variously as a hash value, hash code, digest, or simply hash, is generated by a mathematical algorithm called a hash function.[3] The resulting hash value is typically shorter than the original input and serves as a compact, representative "fingerprint" of the data.[1] A crucial characteristic, particularly for

cryptographic applications, is that well-designed hash functions operate as one-way functions; it is computationally infeasible to reverse the process and derive the original input data solely from its hash output.[4]

Key Properties

Effective hash functions exhibit several key properties that determine their suitability for different applications:

- **Determinism:** A given hash function must always produce the exact same hash output for the same input data.[8] Consistency is essential for verification and lookup tasks.
- **Fixed Output Size:** Regardless of the input data's size—whether a single character or an entire file—the hash function generates an output of a predetermined, fixed length (e.g., 256 bits for SHA-256).[1]
- **Efficiency:** Hash functions are designed for rapid computation, allowing hash values to be generated quickly.[5] This is vital for performance in applications like data retrieval or real-time integrity checks.
- **Pre-image Resistance (One-Way Property):** For cryptographic hash functions, it must be computationally infeasible to find any input that hashes to a specific output value.[4] This property prevents attackers from easily determining an original password or message from its hash.
- **Collision Resistance:** It should be computationally difficult to find two *different* inputs that produce the same hash output.[9] While collisions (two inputs hashing to the same output) are theoretically inevitable due to the mapping of a larger input space to a smaller, fixed-size output space (Pigeonhole Principle), good hash functions make finding them practically impossible for cryptographic purposes.[2] Collisions are a more common concern in non-cryptographic uses like hash tables, where mechanisms exist to handle them.[2]
- **Avalanche Effect:** A small change in the input data (e.g., altering a single bit) should result in a drastically different, seemingly uncorrelated hash output.[6] This ensures that similar inputs do not produce similar hashes, which is important for security and data distribution.

Types and Examples

Hash functions can be broadly categorized based on their design goals:

- **Cryptographic Hash Functions:** Prioritize security properties like pre-image resistance and collision resistance. Examples include the SHA-2 family (e.g., SHA-256, SHA-512), SHA-3, BLAKE variants, bcrypt, scrypt, and Ethash.[2] These are used for password storage, digital signatures, and data integrity verification.[6]

SHA-256 is frequently cited as a widely used standard today.[6] Older, now insecure examples include LANMAN and MD5.[7]

- **Non-Cryptographic Hash Functions:** Focus on speed and achieving a good distribution of hash values, primarily for data structures like hash tables. Security properties are secondary. Examples are often simpler algorithms used internally by programming languages or databases for indexing.[2]
- **Checksum Algorithms:** Used primarily for error detection during data transmission or storage, focusing on detecting accidental modifications rather than malicious ones. They often lack strong collision resistance.[2]

Distinction from Encryption

It is essential to distinguish hashing from encryption. Hashing is designed as a one-way process; once data is hashed, the original data cannot be recovered from the hash value (except by brute-force guessing inputs).[4] Hashing is primarily used for verifying data integrity and authenticity. Encryption, conversely, is a two-way process. Data encrypted using an algorithm and a key can be decrypted back to its original form using the corresponding key.[4] Encryption is used to ensure data confidentiality.

Basic Use Cases

Hashing serves as a foundational technique across various domains:

- **Password Storage:** Storing hash values of passwords instead of plaintext significantly enhances security. If a database is compromised, attackers obtain only the hashes, which are difficult to reverse.[4] Authentication involves hashing the user-provided password and comparing it to the stored hash.[6]
- **Data Integrity Verification:** Hashing allows verification that data has not been tampered with. By comparing the hash of received data with a previously computed or securely transmitted hash, any modification, even minor, can be detected due to the avalanche effect.[3] This is used in digital signatures and file download verification.[4]
- **Data Retrieval (Hash Tables):** In computer science data structures, hash functions map keys to indices in a hash table, enabling average-case constant time (O(1)) complexity for data insertion, deletion, and lookup operations, making data access highly efficient.[2]

The widespread use of hashing in such diverse areas—from securing passwords [7] and ensuring file integrity [6] to optimizing database lookups [5] and enabling data structures like Bloom filters [2]—demonstrates its role not merely as a single technique but as a fundamental computational primitive. Its adaptability, achieved by selecting hash functions with properties tailored to specific needs (e.g., security for cryptography,

speed and distribution for hash tables), underscores its foundational importance in modern computing.

2.2 Defining Parallel Processing

Core Concept

Parallel processing, or parallel computing, is a computational paradigm where multiple processing units—such as cores within a single CPU, multiple CPUs in a machine, or even multiple interconnected computers in a cluster or grid—are used simultaneously to execute different parts of a single large computational task or multiple independent tasks concurrently.[17] The fundamental goal is to enhance computational speed, increase overall performance and throughput, and improve efficiency by dividing the workload and executing it in parallel.[17]

Contrast with Serial Processing

This approach stands in stark contrast to traditional serial computation, where a program is executed as a sequence of instructions, one after another, on a single processor.[18] Parallel computing breaks this sequential bottleneck by enabling simultaneous execution.

Enabling Factors

The ubiquity of parallel processing in modern computing is largely driven by the proliferation of multi-core processors. As physical limitations constrained the ability to indefinitely increase the clock speed (frequency scaling) of single processor cores, manufacturers turned to integrating multiple cores onto a single chip to continue delivering performance gains.[19] Consequently, parallel execution capabilities are now standard in hardware ranging from smartphones to supercomputers.[18] This architectural shift was necessitated by the physical constraints related to power consumption and heat generation that accompany ever-increasing clock frequencies [21], making parallelism the dominant strategy for enhancing computational power.

Basic Concepts

Understanding parallel processing involves several key concepts:

- **Tasks/Threads:** A computational problem is broken down into smaller units of work, often called tasks or threads, that can be executed concurrently.[18]
- **Cores/Processors:** These are the hardware units that execute the tasks in parallel.[18]
- **Synchronization:** Mechanisms are often required to coordinate the execution of different tasks, especially if they depend on each other's results or access shared

resources.[19]
- **Communication:** Tasks running on different processors or machines may need to exchange data, introducing communication overhead.[18]
- **Granularity:** This refers to the ratio of computation to communication/synchronization within parallel tasks. Fine-grained parallelism involves frequent communication relative to computation, while coarse-grained involves less frequent communication.[20]
- **Problem Decomposition:** The effectiveness of parallel processing hinges on the ability to break the problem into discrete, often independent, parts suitable for concurrent execution.[18]

Types of Parallelism

Parallelism can manifest in several forms, including:

- **Bit-level Parallelism:** Increases processor word size to handle more bits per instruction cycle.[18]
- **Instruction-Level Parallelism (ILP):** Executes multiple instructions from a single instruction stream simultaneously using techniques like pipelining and superscalar execution within a single core.[18]
- **Task Parallelism:** Distributes different tasks or functions across multiple processors to be performed concurrently, potentially on the same data.[18]
- **Data Parallelism:** Distributes subsets of a dataset across multiple processors, with each processor performing the same operation on its assigned data subset.[21] This is highly relevant to massive parallel hashing.
- **Superword-Level Parallelism (SLP):** A vectorization technique combining multiple scalar operations into single vector instructions.[18]

Architectures

Parallel systems are typically built using different memory architectures:

- **Shared Memory:** Multiple processors share access to a common memory space, communicating implicitly by reading and writing shared variables.[18] Common in multi-core desktops and servers.
- **Distributed Memory:** Each processor has its own private local memory. Communication requires explicit message passing over a network.[18] Common in clusters and supercomputers.
- **Hybrid Memory:** Combines elements of both shared and distributed memory, such as clusters where each node is a shared-memory multi-processor machine.[18]

3. The Synergy: Understanding Massive Parallel Hashing

Combining Concepts

Massive parallel hashing emerges directly from the application of parallel processing principles to the computation of hash functions, particularly when dealing with a very large number of inputs or searching through an enormous space of potential inputs.[16] The term "massive" emphasizes the scale of parallelism involved, which can range from utilizing the multiple cores of a standard CPU to harnessing thousands of cores on Graphics Processing Units (GPUs), or even employing large clusters of machines or highly specialized hardware like Field-Programmable Gate Arrays (FPGAs) and Application-Specific Integrated Circuits (ASICs).[16] The core idea is to divide the total hashing workload into smaller, independent pieces that can be processed simultaneously, thereby drastically reducing the overall time required.

Parallelization Strategies

Hashing tasks lend themselves well to parallelization, primarily through data parallelism:

- **Data Parallelism:** This is the most common strategy. A large collection of input data items (e.g., password candidates, data blocks, sequence k-mers) is partitioned and distributed among the available parallel processing units. Each unit then independently computes the hash function for its assigned subset of inputs.[25] Since the hashing of one input typically does not depend on the result of hashing another, these computations can proceed concurrently with minimal need for interaction between units.[26] This model is fundamental to applications like brute-force password cracking, large-scale data indexing, and bioinformatics sequence analysis.[12]
- **Task Parallelism:** While the core hash computation itself is usually uniform, task parallelism might apply in broader workflows where hashing is one component among others. Different stages of a complex data processing pipeline, some involving hashing, could potentially run in parallel on different data segments or processing units.[18]
- **Pipelining:** In hardware implementations (FPGAs and ASICs), the internal steps of the hash algorithm itself can often be structured as a pipeline. As one input progresses through the initial stages of the pipeline, the next input can enter, allowing multiple inputs to be in different stages of computation simultaneously, maximizing hardware utilization and throughput.[33]

The inherent nature of hashing, where the computation for each distinct input is

generally independent of others, makes it an "embarrassingly parallel" problem.[20] This means that the task can be easily divided into numerous parallel sub-tasks (hashing individual inputs) that require little to no communication or synchronization among them during the core computation phase.[24] This high degree of independence is why hashing benefits so profoundly from parallel execution architectures like GPUs, which are designed to handle thousands of concurrent, simple operations efficiently.[35] The minimal overhead associated with coordinating these independent tasks allows for near-linear speedups in many scenarios, limited primarily by factors like data transfer and the number of available processing units.

Implementation Models

Massive parallel hashing can be implemented using various hardware and system architectures:

- **Multi-core CPUs:** Standard CPUs with multiple cores can perform parallel hashing, with the operating system or parallel programming libraries managing the distribution of tasks across cores.[19] Performance is limited by the relatively small number of cores compared to GPUs.
- **GPUs:** These are a popular choice due to their architecture featuring thousands of specialized cores designed for parallel throughput.[19] Frameworks like NVIDIA's CUDA or the open standard OpenCL are used to program GPUs for general-purpose parallel tasks, including hashing.[12] They offer a significant performance leap over CPUs for highly parallelizable workloads like hashing.[38]
- **Clusters/Distributed Systems:** Multiple computers connected via a network can work together, each potentially running parallel hashing tasks locally on its own CPU/GPU resources.[16] This allows for scaling beyond a single machine but introduces challenges related to network communication latency, bandwidth limitations, and load balancing across nodes.[16] Specific parallel hashing frameworks exist for distributed environments.[16]
- **Specialized Hardware (FPGAs/ASICs):** For maximum performance and energy efficiency on specific hashing algorithms, custom hardware is often developed. FPGAs offer reconfigurable hardware logic that can be programmed to implement hash functions directly [29], while ASICs are chips designed and manufactured for the sole purpose of executing one particular algorithm (like SHA-256 for Bitcoin mining) at the highest possible speed.[11]

4. Applications and Use Cases of Massive Parallel Hashing

The ability to compute hashes at tremendous speeds opens up a wide range of

applications, some beneficial and others potentially malicious.

4.1 Cryptography and Security Analysis

- **Password Cracking:** This is perhaps the most notorious application. Attackers use massive parallel hashing to test billions or even trillions of potential passwords against a list of stolen password hashes (e.g., NTLM, SHA1, MD5 hashes obtained from a data breach).[6] Tools like Hashcat and John the Ripper are specifically designed to leverage the parallel processing power of GPUs (and sometimes CPUs or FPGAs) to execute various attack strategies:
  - *Brute-Force Attack:* Trying every possible combination of characters up to a certain length.[13]
  - *Dictionary Attack:* Trying words from lists (dictionaries) of common passwords, names, or leaked credentials.[13]
  - *Mask Attack:* A targeted brute-force attack focusing on password patterns (e.g., capital letter + 7 lowercase + 2 digits).[13]
  - *Hybrid Attack:* Combining dictionary words with masks or rules (e.g., appending numbers or symbols to dictionary words).[13] The immense speed offered by GPUs (millions to billions of hashes per second [12]) makes cracking weakly hashed or simple passwords feasible in minutes or hours.[52] This necessitates the use of stronger, computationally intensive hashing algorithms (like bcrypt, scrypt, Argon2) with salting, and enforcing policies for long, complex passwords.[7] FPGAs and ASICs can offer even greater speed or efficiency but are less flexible and common for general password cracking.[29]
- **Cryptanalysis:** While less commonly discussed in the provided materials outside password cracking, the ability to perform rapid hashing could theoretically be applied in broader cryptanalysis efforts, such as searching for collisions in hash functions or analyzing patterns in cryptographic outputs, although practical successes against strong modern primitives are rare and require immense computational resources.[51]

4.2 Large-Scale Data Processing and Databases

- **Database Indexing:** Hash tables are a cornerstone of efficient database indexing, providing average O(1) lookup times.[2] For extremely large datasets, the process of *building* these hash indexes can be accelerated using parallel hashing. Furthermore, in distributed database systems, parallel hashing enables concurrent index lookups and updates across multiple nodes.[16]
- **Hash Joins:** A common database query optimization technique. When joining two tables, the database can build a hash table in memory based on the join key from the smaller table. It then scans the larger table, hashing each row's join key and

probing the hash table for matches. Parallelism can be applied to both the build phase (hashing rows of the first table concurrently) and the probe phase (scanning and probing the second table concurrently).[14]

- **Data Partitioning/Sharding:** In distributed databases and data warehouses, large tables are often partitioned or sharded across multiple physical nodes or storage units. Hashing is a common method for determining which partition a given row belongs to (e.g., based on the hash of a primary key or distribution key).[14] Calculating these hashes in parallel facilitates efficient data loading and distribution, enabling parallel query processing across partitions.[55]
- **Data Integrity and Deduplication:** For massive datasets, verifying integrity or finding duplicate entries can be computationally expensive. Parallel hashing allows for the rapid calculation of hash values for large files or data chunks across multiple processors. Comparing these hashes is much faster than comparing the full data content, enabling efficient integrity checks and deduplication at scale.[6]
- **Change Data Capture (CDC):** Identifying changes between versions of large datasets is crucial for replication and synchronization. Parallel hashing can be used to quickly compute hashes for data blocks or rows in both versions; differing hash values indicate modified data, which can then be captured and processed.[14]

4.3 Bioinformatics

- **Parallel Sequence Alignment:** The explosion of genomic data necessitates high-performance computing techniques. Algorithms like PSALR leverage parallel hashing for rapid sequence alignment.[30] They work by breaking down long DNA or protein sequences into smaller overlapping subsequences called k-mers. These k-mers are then hashed and stored in distributed hash tables, allowing for fast indexing and searching of vast reference genomes. Parallelism is applied both to the hashing/indexing process and the distribution of search tasks across multiple compute nodes (often using MPI), addressing the computational intensity and massive data volumes involved.[30] Techniques may also include data compression to reduce memory and transfer overhead.[30]

4.4 Cryptocurrency Mining

- **Proof-of-Work (PoW):** Many cryptocurrencies, most famously Bitcoin, use PoW consensus mechanisms that rely heavily on hashing.[7] Miners compete to find a valid block by repeatedly hashing the block's header data along with a variable number (a 'nonce'). The goal is to find a hash value that falls below a dynamically adjusted difficulty target.[11] This process requires immense computational power, directly translating to trillions of hash computations per second across the network.

- **Hardware Race:** The economic incentives of PoW mining led to a rapid evolution of hardware specifically designed for parallel hashing. Mining initially performed on CPUs quickly migrated to GPUs due to their superior parallel processing capabilities.[37] For algorithms like Bitcoin's SHA-256, this progressed further to FPGAs and ultimately to highly specialized ASICs.[11] These ASICs implement only the specific hashing algorithm (e.g., SHA-256) but do so with orders of magnitude higher speed (hash rate, measured in TH/s - terahashes per second) and better power efficiency than GPUs or FPGAs.[11] Optimizations like AsicBoost further enhance efficiency for SHA-256 ASICs.[49] This hardware race is a prime example of massive parallel hashing driving specialized technology development.

4.5 Scientific Computing

- **State Space Exploration:** In fields like formal methods and model checking, systems can have astronomically large state spaces. Parallel hashing, particularly on GPUs using specialized hash table structures and techniques (e.g., compact-cuckoo hashing storing compressed tree representations of states), enables the efficient storage and exploration of these vast spaces, making verification feasible for more complex models.[28]
- **Geometric Hashing:** Used in computer graphics, computational geometry, and pattern recognition. It involves hashing geometric features (e.g., based on coordinate pairs or triplets) into a hash table. By probing this table with features from a scene, potential matches to stored models can be identified efficiently, even under transformations like rotation, translation, and scaling.[2] Parallelism can accelerate both the hash table construction (for many models) and the probing process (for complex scenes).[59]

The diverse applications of massive parallel hashing clearly illustrate its power as an accelerator. However, this power presents a duality: the same techniques that enable faster scientific discovery [28], efficient data management [14], and decentralized consensus [11] also empower malicious activities like accelerated password cracking.[12] The technology's impact is therefore highly dependent on its context of use.

Furthermore, the choice of hardware platform—ranging from general-purpose multi-core CPUs to highly specialized ASICs—is strongly dictated by the specific application's demands. Cryptocurrency mining, prioritizing raw speed and efficiency for a single algorithm, inevitably led to ASICs.[11] Password cracking often favors GPUs for their balance of high throughput and flexibility across different algorithms.[12] Large-scale data processing in databases might rely on distributed systems with multi-core CPUs [16], while specialized scientific computing or low-power embedded

systems might utilize FPGAs.[29] This highlights that there is no single "best" hardware for parallel hashing; the optimal choice depends on a trade-off between performance, power efficiency, cost, and the need for programmability or flexibility.

5. Enabling Hardware: Architectures for Parallel Hashing

The implementation of massive parallel hashing relies heavily on the underlying hardware architecture. Different platforms offer distinct advantages and disadvantages in terms of performance, efficiency, flexibility, and cost.

5.1 Graphics Processing Units (GPUs)

- **Architecture:** GPUs feature a massively parallel architecture comprising thousands of relatively simple processing cores, optimized for Single Instruction, Multiple Data (SIMD) or Single Instruction, Multiple Threads (SIMT) execution.[19] This design allows them to perform the same operation on large amounts of data concurrently, contrasting sharply with CPUs, which typically have a smaller number of more powerful, complex cores designed for general-purpose, often serial or less parallel, tasks.[36]
- **Suitability:** Their architecture makes GPUs exceptionally well-suited for the repetitive, independent computations inherent in many hashing tasks. They can achieve very high throughput (hashes per second) by executing numerous hash calculations in parallel.[12] Parallel programming models like NVIDIA's CUDA and the open standard OpenCL provide frameworks for developers to harness this power for general-purpose computing (GPGPU), including hashing.[25]
- **Use Cases:** GPUs are widely used for password cracking, were instrumental in the early-to-mid stages of cryptocurrency mining, and are employed in various scientific computing and machine learning applications where parallel processing, potentially including hashing operations, is beneficial.[12]
- **Pros:** Offer significantly higher throughput for parallel tasks compared to CPUs.[38] They are relatively accessible and affordable compared to developing ASICs.[35] GPUs are programmable and can be used for various hashing algorithms and other parallel tasks.[31] A mature ecosystem of tools (like Hashcat [12]) and libraries exists.
- **Cons:** Generally consume more power than FPGAs or ASICs when performing the *same* highly specialized task.[31] While throughput is high, the latency for completing a single, isolated hash computation might be higher than a CPU due to data transfer overhead and architectural design differences.[41] Achieving optimal performance requires careful programming to manage parallelism, memory access patterns (coalescing), and data transfers between the host CPU

and the GPU.[32]

5.2 Field-Programmable Gate Arrays (FPGAs)

- **Architecture:** FPGAs consist of an array of configurable logic blocks (CLBs), memory blocks, and programmable interconnects.[29] Unlike fixed-architecture processors (CPUs, GPUs), FPGAs can be programmed at the hardware level using Hardware Description Languages (HDLs) like VHDL or Verilog to implement custom digital circuits tailored precisely to a specific algorithm, such as a hash function.[29]
- **Suitability:** This reconfigurability allows for the creation of highly optimized, deeply pipelined hardware implementations of hash algorithms. This can lead to significant advantages in performance-per-watt (energy efficiency) compared to GPUs executing the same algorithm in software on their general-purpose cores.[29] FPGAs enable true hardware parallelism, where different parts of the chip perform different tasks simultaneously without contention for shared execution units.[40]
- **Use Cases:** FPGAs find use in specialized cryptographic hardware, network processing (where line-rate hashing might be needed), high-frequency trading, accelerating specific scientific computations, and prototyping designs before committing to expensive ASIC manufacturing.[29] Some high-end commercial password cracking systems also utilize FPGAs.[34]
- **Pros:** Excellent energy efficiency (high hashes per watt).[29] Low latency due to direct hardware implementation.[40] Reconfigurable, allowing updates or switching between different algorithms.[40] Can achieve high throughput with optimized, parallel, and pipelined designs.[29]
- **Cons:** Development is significantly more complex and time-consuming than software development for CPUs/GPUs, requiring specialized hardware design skills (HDL expertise).[29] The initial hardware cost can be higher than GPUs, although potentially lower than custom ASICs.[31] Peak performance for a single, heavily optimized algorithm might still be lower than a dedicated ASIC.[50] The ecosystem of pre-built tools and libraries for general hashing tasks is less mature than for GPUs.[50]

5.3 Application-Specific Integrated Circuits (ASICs)

- **Architecture:** ASICs are integrated circuits designed and manufactured for one single, specific purpose.[11] All the transistors and logic gates on the chip are laid out and optimized exclusively for executing that one task—for example, computing SHA-256 hashes as quickly and efficiently as possible.[11] The design is fixed during manufacturing and cannot be changed later.

- **Suitability:** For the specific algorithm they implement, ASICs offer the absolute maximum performance (highest hash rate) and the best possible power efficiency (lowest energy consumption per hash) achievable with current technology.[11]
- **Use Cases:** ASICs dominate fields where extreme performance and efficiency for a single, stable algorithm are paramount, and the scale of operation justifies the high development costs. The most prominent example is Bitcoin mining (SHA-256) [11], but they could also be used in dedicated, high-volume cryptographic hardware appliances.
- **Pros:** Unmatched speed and throughput for their designated task.[31] Highest possible energy efficiency (performance per watt).[29]
- **Cons:** Extremely high non-recurring engineering (NRE) costs associated with design, verification, and manufacturing.[47] Long development and fabrication lead times.[47] Complete inflexibility – an ASIC designed for SHA-256 cannot be used for any other algorithm.[31] Susceptible to rapid obsolescence if the target algorithm changes, is found to be insecure, or if competitors release faster ASICs.[31]

5.4 Comparative Analysis of Hardware

The following table summarizes the key characteristics of CPUs, GPUs, FPGAs, and ASICs in the context of massive parallel hashing:

Feature	CPU (Multi-core)	GPU	FPGA	ASIC
Peak Throughput	Low	High	Medium to High (Design Dependent)	Very High (Algorithm Specific)
Power Efficiency	Low	Medium	High	Very High (Algorithm Specific)
Flexibility	Very High (General Purpose)	High (Programmable, Multi-Algorithm)	Medium (Reconfigurable, Multi-Algorithm)	None (Fixed Algorithm)
Development Complexity	Low (Standard Software Dev)	Medium (Parallel Programming - CUDA/OpenCL)	High (Hardware Description Language - HDL)	Very High (IC Design & Fabrication)

Initial Hardware Cost	Low to Medium	Medium to High	High	Very High (incl. NRE costs)
Latency (Single Op)	Lowest	Higher than CPU	Low (Hardware Implementation)	Lowest (Hardware Implementation)
Typical Hashing Use Cases	Low-volume hashing, General tasks	Password Cracking, Mining (early), SciComp	Specialized Crypto, Network Proc., Prototyping	High-Volume Mining (e.g., Bitcoin)

6. Benefits and Advantages

Employing massive parallel hashing offers several significant advantages over traditional serial approaches, primarily centered around performance and scale.

Performance Gains

- **Speed and Throughput:** The most prominent benefit is a dramatic increase in the rate at which hash computations can be performed.[17] Instead of processing inputs one by one, parallel systems process hundreds, thousands, or even millions simultaneously. This translates to vastly higher throughput, measured in hashes per second. For instance, password cracking tools leveraging GPUs can test millions or billions of candidates per second [12], and specialized GPU implementations for scientific problems like state space exploration can process hundreds of millions of states per second.[28] This raw speed is the primary driver for adopting parallel hashing.[27]
- **Reduced Computation Time:** For tasks involving a fixed, large number of hash computations (like indexing a massive dataset or completing a PoW mining cycle), parallel execution significantly reduces the total time required to finish the job.[19]

Efficiency Considerations

- **Performance-per-Watt:** While high-performance parallel systems can consume substantial power overall, specialized hardware can offer superior energy efficiency for specific tasks. FPGAs and particularly ASICs are designed to maximize the number of hashes computed per unit of energy (joule or watt) consumed, often outperforming general-purpose hardware like CPUs or even GPUs in this metric when focused on a single, optimized algorithm.[29] For parallelizable tasks, GPUs themselves are generally more energy-efficient per unit

of work done than CPUs.[35] However, the notion of "efficiency" in parallel hashing is multifaceted. While ASICs excel in hashes-per-watt for their specific algorithm [31], their inflexibility and high development cost impact overall economic efficiency. GPUs might offer better cost-efficiency for tasks requiring flexibility across multiple algorithms [35], despite potentially lower power efficiency than an ASIC. FPGAs represent a compromise, often providing better power efficiency than GPUs with more flexibility than ASICs.[31] CPUs remain the least power-efficient option for massively parallel hashing workloads.[36] Thus, efficiency must be evaluated contextually, considering speed, power, cost, and flexibility requirements.

Scalability

- **Handling Large Problems:** Massive parallel hashing makes it feasible to tackle computational problems involving enormous datasets or search spaces that would be computationally intractable using serial methods.[14] This includes processing petabytes of data, analyzing entire genomes, or searching vast password spaces.
- **Resource Scaling:** Performance can often be scaled by adding more parallel processing resources—more cores, more GPUs, more nodes in a cluster—allowing systems to handle growing problem sizes or achieve faster results.[16]

Concurrency

- **Simultaneous Operations:** Parallel architectures inherently support concurrency, allowing multiple hashing operations or related tasks to execute simultaneously.[24] This can improve the responsiveness of systems that handle many concurrent requests involving hashing (e.g., a server authenticating multiple users) or increase the overall throughput of complex workflows.[32]

7. Drawbacks and Implementation Challenges

Despite the significant performance advantages, implementing and deploying massive parallel hashing systems presents considerable challenges and drawbacks.

Economic Factors

- **Hardware Cost:** Acquiring the necessary hardware can be expensive. High-end GPUs, clusters of machines, FPGAs, and especially the design and fabrication of ASICs represent substantial capital investments.[22] Even renting cloud-based GPU or FPGA instances for large-scale tasks can incur significant costs.[52]
- **Development Cost/Effort:** Creating efficient parallel hashing solutions is often

complex and costly. Programming for parallel architectures requires specialized skills and tools.[21] Development for FPGAs using HDLs is particularly demanding [29], and designing ASICs is an even more resource-intensive process requiring deep expertise in integrated circuit design.[50] This increases development time and personnel costs.

Technical Complexity

- **Parallel Programming:** Writing, debugging, and tuning parallel software is inherently more difficult than developing sequential programs.[19] Developers must manage issues like task decomposition, data distribution, inter-process communication, synchronization (e.g., using locks or atomic operations to prevent race conditions), and potential deadlocks.[19]
- **Optimization:** Achieving near-theoretical speedups requires meticulous optimization tailored to the specific hardware architecture. This involves managing memory access patterns (e.g., ensuring coalesced memory access on GPUs to maximize bandwidth), minimizing data transfer overheads (e.g., between CPU and GPU), efficiently utilizing cache hierarchies, and navigating the complexities of parallel programming APIs (like CUDA, OpenCL, MPI, or HDL toolchains).[32]
- **Algorithm Suitability:** While basic hashing is highly parallelizable, Amdahl's Law dictates that the overall speedup of any application is limited by the portion of the code that must run sequentially.[21] Complex workflows that incorporate hashing might have inherent serial bottlenecks that limit the benefits of parallelization.

Operational Factors

- **Power Consumption:** High-performance parallel systems, especially those using numerous GPUs or ASICs, can consume vast amounts of electrical power.[21] This leads to high operational costs (electricity bills) and contributes to environmental concerns, particularly in large-scale operations like cryptocurrency mining farms.
- **Heat Dissipation:** The significant power consumed by parallel hardware is dissipated as heat. Managing this thermal load is crucial for system stability, reliability, and longevity.[21] Effective cooling solutions (e.g., high-performance fans, liquid cooling) are often necessary, adding to system cost and complexity.[60]

Inherent Limitations

- **Load Balancing:** Distributing the workload evenly across all available processing units is critical for efficiency. If some units are idle while others are overloaded (e.g., due to skewed data distributions or poor task assignment), the overall performance suffers.[16] In network contexts, naive use of hashing for load

balancing across multiple hops can even lead to "polarization," where certain paths become consistently overloaded.[44]

- **Communication Overhead:** In distributed memory systems (clusters), the time spent communicating data between nodes over the network can become a significant bottleneck, especially for fine-grained tasks or if network bandwidth/latency is limited.[16] This overhead can potentially negate the benefits of parallel computation.
- **Memory Limitations:** The amount of local memory available on accelerators like GPUs can be a limiting factor for problems involving very large datasets or complex state representations that don't fit entirely within the device's memory.[42] Techniques like data compression or specialized hashing schemes may be needed to mitigate this.[28]

Security Implications

The power of massive parallel hashing is a double-edged sword. The same speed that benefits legitimate applications also significantly empowers attackers, enabling them to crack passwords or potentially brute-force other cryptographic challenges much faster than before.[13] This necessitates continuous improvement in cryptographic practices, such as adopting slower, memory-hard hash functions for passwords and increasing key lengths.

Ultimately, the adoption of parallel hashing does not eliminate complexity but rather transforms it. The challenge of slow serial execution is replaced by a new set of engineering hurdles related to managing parallel execution, optimizing for specific hardware, handling increased power and thermal loads, mitigating high costs, and addressing the intricate details of synchronization, communication, and load balancing.[16] Achieving the potential performance gains requires overcoming these significant implementation and operational complexities.

8. Comparative Analysis: Parallel vs. Serial Hashing

Understanding the fundamental differences between serial and massive parallel hashing is crucial for selecting the appropriate approach based on application requirements and available resources.

Execution Model

- **Serial Hashing:** Computes one hash value at a time, processing inputs sequentially on a single processing unit, typically a single core of a CPU.[18] The control flow is simple and linear.
- **Parallel Hashing:** Computes multiple hash values concurrently by utilizing

multiple processing units simultaneously. This can involve multiple CPU cores, thousands of GPU cores, dedicated logic blocks on an FPGA, specialized circuits on an ASIC, or multiple nodes in a distributed system.[17] Requires mechanisms for distributing work and potentially coordinating results.

Performance Profile

- **Serial Hashing:** Performance is limited by the clock speed and instruction execution capabilities of a single core. It is suitable for applications with low hashing volume or where the latency of a single hash computation is the primary concern.[42] Throughput is inherently low.
- **Parallel Hashing:** Designed for high throughput, capable of processing vastly larger numbers of hashes per unit time.[21] Performance generally scales with the number of parallel units, although limitations like communication overhead or serial bottlenecks (Amdahl's Law [21]) exist. While aggregate throughput is high, the latency for any single hash computation might be slightly higher than serial execution due to overheads like data transfer and task scheduling, especially when using accelerators like GPUs.[41]

Resource Requirements

- **Serial Hashing:** Minimal hardware requirements—a single CPU core is sufficient. Power consumption per operation is relatively low.
- **Parallel Hashing:** Requires hardware with multiple processing elements (multi-core CPUs, GPUs, FPGAs, ASICs) or networked systems.[21] Can lead to significantly higher overall power consumption and necessitates adequate cooling solutions.[21] Depending on the implementation (especially GPU-based), it may require substantial amounts of memory for holding data or intermediate results.[42]

Complexity and Cost

- **Serial Hashing:** Implementation is straightforward using standard programming techniques. Hardware and development costs are minimal.
- **Parallel Hashing:** Implementation is significantly more complex, requiring expertise in parallel programming paradigms, synchronization techniques, and potentially specialized hardware languages (HDL).[21] Hardware acquisition, development effort, and potentially higher operational (power) costs contribute to a higher overall cost.[22]

Use Case Fit

- **Serial Hashing:** Appropriate for applications where only occasional hashes are

needed, where computational resources are severely limited, or where minimizing the latency of individual hash computations is critical. Examples include hashing configuration files on startup or verifying single user inputs.
- **Parallel Hashing:** Essential for high-throughput, computationally intensive applications such as cracking large password lists, indexing or processing massive datasets, performing Proof-of-Work calculations in cryptocurrency mining, and running large-scale scientific simulations that rely on hashing.[11]

Key Differences Summary

The table below provides a concise comparison highlighting the core distinctions:

Feature	Serial Hashing	Massive Parallel Hashing
Execution Method	Sequential (one at a time)	Concurrent (many at once)
Primary Performance Metric	Latency (single operation)	Throughput (hashes per second)
Typical Hardware	Single CPU Core	Multi-Core CPU, GPU, FPGA, ASIC, Cluster
Scalability	Limited by single-core speed	Scales with number of processing units (up to limits)
Implementation Complexity	Low	High
Cost (Hardware & Dev.)	Low	Medium to Very High
Power Consumption Profile	Low	Medium to Very High
Ideal Use Cases	Low volume, Latency-sensitive	High volume, Throughput-intensive (Cracking, Mining, Big Data)

9. Conclusion

Massive parallel hashing stands as a potent technique for accelerating hash computations by orders of magnitude compared to traditional serial methods. By harnessing the power of parallel processing architectures—ranging from multi-core CPUs to highly specialized ASICs, with GPUs and FPGAs occupying crucial

intermediate roles—it enables the tackling of computationally intensive tasks across diverse fields. Its applications are widespread, underpinning critical functions in cybersecurity analysis (like password cracking), large-scale data processing and database operations (indexing, joins, partitioning), bioinformatics (sequence alignment), cryptocurrency mining (Proof-of-Work), and various scientific computing domains.

The primary advantage driving its adoption is the dramatic increase in speed and throughput, allowing problems previously considered intractable due to sheer computational volume to become feasible. However, this performance comes at a cost. The implementation of massive parallel hashing introduces significant complexities related to parallel programming, system optimization, hardware expenditure, power consumption, and thermal management. The choice of hardware platform involves a critical trade-off analysis, balancing peak performance against energy efficiency, cost, development effort, and the need for algorithmic flexibility. As observed, the optimal hardware choice is highly dependent on the specific application's constraints and goals, with ASICs offering peak efficiency for fixed tasks like Bitcoin mining, GPUs providing a flexible high-throughput option for tasks like password cracking, FPGAs serving niches requiring power efficiency and reconfigurability, and multi-core CPUs/clusters handling general-purpose parallel data processing.

Furthermore, the capabilities enabled by massive parallel hashing present a duality: the same acceleration that benefits scientific research and data analysis also enhances the effectiveness of malicious activities, demanding continuous vigilance and evolution in security practices. The complexity inherent in computation does not vanish with parallelism; rather, it shifts from raw execution time to the engineering challenges of managing concurrency, resources, and costs effectively.

Looking ahead, the field will likely continue to evolve rapidly. Advances in hardware architectures (e.g., next-generation GPUs, more accessible FPGAs, novel accelerator designs) will push performance boundaries further. Concurrently, research into new parallel-friendly hashing algorithms and techniques, alongside more sophisticated parallel programming models and tools, will aim to improve efficiency and ease implementation challenges. As datasets continue to grow exponentially and the demand for secure, high-performance computation persists, massive parallel hashing will remain a critical and dynamic area of computer science and engineering.

Works cited

1. www.techtarget.com, accessed April 21, 2025, https://www.techtarget.com/searchdatamanagement/definition/hashing#:~:text=Hashing%20is%20the%20process%20of,or%20employ%20the%20original%20string.
2. Hash function - Wikipedia, accessed April 21, 2025, https://en.wikipedia.org/wiki/Hash_function
3. hashing - Glossary | CSRC - NIST Computer Security Resource Center, accessed April 21, 2025, https://csrc.nist.gov/glossary/term/hashing
4. Hashing: What Is It and How is It Used? - Codecademy, accessed April 21, 2025, https://www.codecademy.com/resources/blog/what-is-hashing/
5. Introduction to Hashing | GeeksforGeeks, accessed April 21, 2025, https://www.geeksforgeeks.org/introduction-to-hashing-2/
6. What Is Hashing and How Does It Work? - Digital Warroom, accessed April 21, 2025, https://www.digitalwarroom.com/blog/what-is-hashing
7. What Is Hashing in Cybersecurity? - CrowdStrike, accessed April 21, 2025, https://www.crowdstrike.com/en-us/cybersecurity-101/data-protection/data-hashing/
8. Hashing (OCR A Level Computer Science) : Revision Note - Save My Exams, accessed April 21, 2025, https://www.savemyexams.com/a-level/computer-science/ocr/17/revision-notes/3-exchanging-data/3-1-compression-encryption-and-hashing/hashing/
9. The Basics of Hashing: A Key Concept in Computer Science - Prasams, accessed April 21, 2025, https://www.prasams.com/the-basics-of-hashing-a-key-concept-in-computer-science/
10. What is Hashing? - GeeksforGeeks, accessed April 21, 2025, https://www.geeksforgeeks.org/what-is-hashing/
11. SHA-256 Cryptographic Hash Algorithm - Komodo Platform, accessed April 21, 2025, https://komodoplatform.com/en/academy/sha-256-algorithm/
12. Password Cracking - Brown University Department of Computer Science, accessed April 21, 2025, https://cs.brown.edu/courses/csci1660/old/2020/demos/password-cracking/
13. How Long Does It Take Hackers to Crack Modern Hashing Algorithms?, accessed April 21, 2025, https://thehackernews.com/2025/01/how-long-does-it-take-hackers-to-crack.html
14. Hash In Data Engineering: Key Concepts — DataForge, accessed April 21, 2025, https://www.dataforgelabs.com/advanced-sql-concepts/hash-in-sql
15. How do databases manage large-scale transactions? - TutorChase, accessed April 21, 2025, https://www.tutorchase.com/answers/ib/computer-science/how-do-databases-manage-large-scale-transactions
16. Design and evaluation of parallel hashing over large-scale data - SciSpace, accessed April 21, 2025, https://scispace.com/pdf/design-and-evaluation-of-parallel-hashing-over-large-s

cale-210cnm1aau.pdf
17. www.lenovo.com, accessed April 21, 2025, https://www.lenovo.com/us/en/glossary/parallel-processing/#:~:text=Parallel%20processing%20is%20a%20method,complex%20calculations%2C%20and%20computational%20simulations.
18. What is parallel computing? - IBM, accessed April 21, 2025, https://www.ibm.com/think/topics/parallel-computing
19. What Is Parallel Processing, or Parallelization, in Computing? - BizTech Magazine, accessed April 21, 2025, https://biztechmagazine.com/article/2024/09/what-parallel-processing-or-parallelization-perfcon
20. What is Parallel Computing? Definition and FAQs - HEAVY.AI, accessed April 21, 2025, https://www.heavy.ai/technical-glossary/parallel-computing
21. Parallel computing - Wikipedia, accessed April 21, 2025, https://en.wikipedia.org/wiki/Parallel_computing
22. What is Parallel Processing ? | GeeksforGeeks, accessed April 21, 2025, https://www.geeksforgeeks.org/what-is-parallel-processing/
23. Is parallel computing the same as parallel processing? - Reddit, accessed April 21, 2025, https://www.reddit.com/r/computing/comments/frmzsh/is_parallel_computing_the_same_as_parallel/
24. Introduction to Parallel Computing Tutorial - | HPC @ LLNL, accessed April 21, 2025, https://hpc.llnl.gov/documentation/tutorials/introduction-parallel-computing-tutorial
25. What is Parallelization & the Different Types | Lenovo US, accessed April 21, 2025, https://www.lenovo.com/us/en/glossary/parallelization/
26. Parallel computing | AP CSP (article) - Khan Academy, accessed April 21, 2025, https://www.khanacademy.org/computing/ap-computer-science-principles/algorithms-101/x2d2f703b37b450a3:parallel-and-distributed-computing/a/parallel-computing
27. Implementation of a Data-Parallel Approach on a Lightweight Hash Function for IoT Devices, accessed April 21, 2025, https://www.mdpi.com/2227-7390/13/5/734
28. The fast and the capacious: memory-efficient multi-GPU accelerated ..., accessed April 21, 2025, https://www.frontiersin.org/journals/high-performance-computing/articles/10.3389/fhpcp.2024.1285349/full
29. FPGA-Accelerated Password Cracking, accessed April 21, 2025, https://www.american-cse.org/csce2023-ieee/pdfs/CSCE2023-5LlpKs7cpb4k2UysbLCuOx/275900c541/275900c541.pdf
30. PSALR: Parallel Sequence Alignment for long Sequence Read with Hash model, accessed April 21, 2025, https://jad.shahroodut.ac.ir/article_3342_d2200f0fd46f157f7a80860cd44e4edb.pdf
31. Comparing ASIC Crypto Mining: Evaluating GPU, FPGA, and Emerging Hardware

Solutions, accessed April 21, 2025, https://minerset.com/comparing-asic-crypto-mining-evaluating-gpu-fpga-and-emerging-hardware-solutions/
32. Parallel Implementation of Lightweight Secure Hash Algorithm on CPU and GPU Environments - MDPI, accessed April 21, 2025, https://www.mdpi.com/2079-9292/13/5/896
33. Experience using a low-cost FPGA design to crack des keys | Request PDF - ResearchGate, accessed April 21, 2025, https://www.researchgate.net/publication/2535735_Experience_using_a_low-cost_FPGA_design_to_crack_des_keys
34. Efficient High-Speed WPA2 Brute Force Attacks using Scalable Low-Cost FPGA Clustering - IACR, accessed April 21, 2025, https://www.iacr.org/archive/ches2016/98130108/98130108.pdf
35. Top 10 Best GPUs for Mining in 2025 - Cherry Servers, accessed April 21, 2025, https://www.cherryservers.com/blog/best-gpus-for-mining
36. CPU vs. GPU: What's the Difference? - CDW, accessed April 21, 2025, https://www.cdw.com/content/cdw/en/articles/hardware/cpu-vs-gpu.html
37. GPU Usage in Cryptocurrency Mining - Investopedia, accessed April 21, 2025, https://www.investopedia.com/tech/gpu-cryptocurrency-mining/
38. CPU vs GPU - Jawad's Notes, accessed April 21, 2025, https://notes.jawad.ca/infosec/tools/password-cracking/cpu-vs-gpu
39. GPU vs CPU Performance Comparison: What are the Key Differences? - Cherry Servers, accessed April 21, 2025, https://www.cherryservers.com/blog/gpu-vs-cpu-what-are-the-key-differences
40. FPGA vs GPU: 5 Parallel Processing Considerations - New Wave Design, accessed April 21, 2025, https://newwavedesign.com/blog/fpga-vs-gpu-parallel-processing/
41. Comparing Performance Characteristics of NVIDIA GPUs and CPUs for Cryptographic Hash Functions - Massed Compute, accessed April 21, 2025, https://massedcompute.com/faq-answers/?question=How+do+the+performance+characteristics+of+NVIDIA+GPUs+compare+to+CPUs+for+cryptographic+hash+functions+in+terms+of+throughput+and+latency%3F
42. CPU hashes faster than GPU? - c++ - Stack Overflow, accessed April 21, 2025, https://stackoverflow.com/questions/46530852/cpu-hashes-faster-than-gpu
43. CPU vs GPU - Hashcat, accessed April 21, 2025, https://hashcat.net/forum/thread-6336.html
44. Hashing Design in Modern Networks: Challenges and Mitigation Techniques - Minlan Yu, accessed April 21, 2025, https://minlanyu.seas.harvard.edu/writeup/atc22.pdf
45. PSALR: Parallel Sequence Alignment for long Sequence Read with Hash model, accessed April 21, 2025, https://www.researchgate.net/publication/378066441_PSALR_Parallel_Sequence_Alignment_for_long_Sequence_Read_with_Hash_model
46. PSALR: Parallel Sequence Alignment for long Sequence Read with Hash model - OUCI, accessed April 21, 2025, https://ouci.dntb.gov.ua/en/works/4bg2xmdl/

47. What is an FPGA? | Uses, Applications & Advantages – Digilent Blog, accessed April 21, 2025, https://digilent.com/blog/what-is-an-fpga/
48. Shop SHA 256 Algorithm ASIC Miners - CryptoMinerBros, accessed April 21, 2025, https://www.cryptominerbros.com/product-category/sha-256-miner/
49. What is the difference between SHA256 and SHA256AsicBoost? - NiceHash, accessed April 21, 2025, https://www.nicehash.com/support/mining-help/asic-mining/what-is-the-difference-between-sha256-and-sha256asicboost
50. How do Field Programmable Gate Arrays (FPGAs) compare to Graphics Processing Units (GPUs); for cracking hashes? - Information Security Stack Exchange, accessed April 21, 2025, https://security.stackexchange.com/questions/196007/how-do-field-programmable-gate-arrays-fpgas-compare-to-graphics-processing-uni
51. Understanding ASICs in Cryptography: A Comparative Study with CPUs, GPUs, and FPGAs, accessed April 21, 2025, https://www.onlinehashcrack.com/how-asics-hardware-works.php
52. Hashcat Password Cracking & Password Policy | Part 1 | ProSec GmbH, accessed April 21, 2025, https://www.prosec-networks.com/en/blog/hashcat-password-cracking-password-policy/
53. Password Series: 8 Practical First Steps to Crack Difficult Passwords - Raxis, accessed April 21, 2025, https://raxis.com/blog/password-series-8-practical-first-steps-to-crack-difficult-passwords/
54. Introduction to Hashcat - YouTube, accessed April 21, 2025, https://m.youtube.com/watch?v=EfqJCKWtGiU&t=0s
55. Other Applications of Hashing for Parallel Processing - Teradata Vantage, accessed April 21, 2025, https://docs.teradata.com/r/Teradata-VantageCloud-Lake/Database-Reference/Database-Design/Database-Design-for-Vantage/Data-Placement-to-Support-Parallel-Processing/Other-Applications-of-Hashing-for-Parallel-Processing
56. (PDF) A Parallel Hash based method for local sequence alignment - ResearchGate, accessed April 21, 2025, https://www.researchgate.net/publication/353346287_A_Parallel_Hash_based_method_for_local_sequence_alignment
57. Bitcoins Miner Solo Lottery Miner 2.4G WiFi SHA-256 BM1366 Latest Upgrade Model V7 1TH/S Crypto BTC Miner Asic Chip Home Use Machine - Amazon.com, accessed April 21, 2025, https://www.amazon.com/Bitcoins-Miner-Lottery-SHA-256-Upgrade/dp/B0D92P7M7R
58. SHA256 - NiceHash - Leading Cryptocurrency Platform for Mining | NiceHash, accessed April 21, 2025, https://www.nicehash.com/algorithm/sha256
59. On a Parallel Implementation of Geometric Hashing on the Connection Machine - DTIC, accessed April 21, 2025, https://apps.dtic.mil/sti/trecms/pdf/AD1020187.pdf
60. 7 advantages of GPU overclocking - DIGITIMES Asia, accessed April 21, 2025,

https://www.digitimes.com/news/a20250415PR200.html?chid=9
61. FPGA vs Microcontroller: A Comprehensive Comparison - Arshon Inc. Blog, accessed April 21, 2025, https://arshon.com/blog/fpga-vs-microcontroller-a-comprehensive-comparison/