# **AI Accelerator**

## **About the Author**

Andrea Bruno born in Sondrio in 1972, is an independent inventor and researcher with a lifelong passion for mathematics, physics, and artificial intelligence. Although he did not pursue formal academic qualifications, his intellectual aptitude has been recognized since an early age. During a physics examination in technical school, his professor publicly referred to him as a genius for solving a complex problem in an unconventional and elegant way. Later, Andrea achieved a perfect score of 160 on an IQ test, further affirming his natural analytical capabilities.

In the 1980s, he began developing early artificial intelligence algorithms, including a chess engine that was published in the magazine CHIP. Unlike traditional brute-force engines, his software was powered by a self-designed AI logic, demonstrating a pioneering approach to intelligent computation.

The mathematical and algorithmic work presented in this repository did not require extensive effort or formal tools. In fact, the core innovation—the fast multiplication algorithm—was conceived intuitively during sleep, without the need for pen, paper, or prolonged analysis. This intuitive process reflects Andrea’s unique cognitive style and his ability to internalize and synthesize complex systems with minimal external input.

Through this project, Andrea Bruno aims not only to contribute to the advancement of AI computation, but also to establish a clear and enduring association between his name and the intellectual property he has created.

## **Project Vision**

The Fast Multiplication Algorithm presented here is only one component of a broader initiative: the AI Accelerator project. This effort aims to develop a suite of computational tools specifically optimized for artificial intelligence workloads and scientific computing.

Several additional algorithms have already been conceived and are in development, each designed to enhance performance, efficiency, and scalability. The overarching goal is to create a modular framework of innovations that, when integrated, offer substantial improvements over general-purpose solutions.

To illustrate this concept, consider the analogy of automotive engineering. While a Formula 1 engine and a standard car engine may share similar materials and basic mechanical principles, the arrangement, precision, and advanced technologies employed in the former result in vastly superior performance. Likewise, the AI Accelerator project seeks to embed cutting-edge computational techniques into the "engine" of AI systems.

The Fast Multiplication Algorithm can be viewed as one such enhancement—akin to the transition from carburetors to fuel injectors in engine design. When paired with dedicated hardware, this algorithm represents a leap in computational efficiency, enabling faster and more energy-conscious processing.

Historically, companies like NVIDIA have leveraged GPU architectures originally designed for gaming, which—by fortunate coincidence—proved highly effective for AI computation. However, as both science and engineering demonstrate, general-purpose solutions are often outperformed by purpose-built technologies. This project embraces that principle, aiming to deliver targeted innovations that surpass conventional architectures in both raw performance and energy efficiency.

While the Fast Multiplication Algorithm is a key innovation, it is not intended to be presented as a standalone solution capable of accelerating AI computation on its own. The true impact emerges when multiple aspects of AI processing are optimized through targeted, purpose-built technologies. When each computational layer is enhanced with dedicated solutions, the cumulative effect yields significant improvements in both energy efficiency and computational throughput.

The AI Accelerator project already includes the design of additional algorithms and custom chip architectures aimed at accelerating AI workloads in other domains. These components will be published and disclosed progressively, forming part of a broader technical vision focused on high-performance, domain-specific computing for artificial intelligence.

## **Fast Multiplication Algorithm – README**

### **Legal Notice and Copyright Protection**

All content described, referenced, or included in this document—including algorithmic concepts, implementation details, and the underlying intellectual framework—is protected under the exclusive copyright license authored by Andrea Bruno.

This license strictly prohibits any form of reproduction, modification, redistribution, or commercial use of the material without prior written consent. The publication of this project serves as demonstrative evidence of authorship and establishes a verifiable creation date, while preserving the author's right to pursue patent protection in relevant jurisdictions.

For the complete and legally binding version of the license, please refer to the full text available in the LICENSE.md file within this repository.

### **Legal Enforcement and Consequences of Violation**

The intellectual property contained in this repository—including algorithmic concepts, technical methods, and original implementations—is protected under exclusive copyright and is subject to legal safeguards.

Any use of this content in external projects, academic publications, or patent filings without a formal transfer of intellectual property rights constitutes a serious violation. Such actions go beyond plagiarism and may be classified as misappropriation or infringement under applicable laws.

Under ****United States law****, violators may be prosecuted pursuant to:

* ****17 U.S. Code § 501**** – Copyright infringement
* ****18 U.S. Code § 1832**** – Theft of trade secrets (Economic Espionage Act)
* ****35 U.S. Code § 271**** – Patent infringement, if derivative patents are filed unlawfully

Under ****European Union law****, infringement may be pursued under:

* ****Directive 2004/48/EC**** – Enforcement of intellectual property rights
* ****Directive 2001/29/EC**** – Copyright in the information society
* ****Regulation (EU) 2019/790**** – Digital Single Market and protection of non-authorized uses

Violators may be held liable for:

* Economic damages, including loss of commercial opportunity
* Statutory and punitive damages
* Injunctions, cease-and-desist orders, and criminal prosecution in cases of willful misappropriation

The author reserves all rights to initiate legal proceedings in any competent jurisdiction to protect the integrity, exclusivity, and economic value of the work.

## **Presentation of the Algorithm Underlying Technological Innovation**

The code presented below serves as a clear and academically rigorous demonstration of the logic and functioning of the fast multiplication algorithm. It is designed to be transparent and accessible, allowing researchers, engineers, and practitioners to understand the mathematical principles that underpin the innovation.

The implementation is written in C#, a widely adopted programming language chosen for its readability and accessibility. However, it is important to emphasize that the algorithm itself is language-independent. The computational logic and mathematical structure are universal and can be expressed in any programming environment or hardware architecture.

As evident from the example, the algorithm is currently implemented for execution on a CPU. Due to the byte-oriented architecture of conventional CPUs—organized into segments such as 8-bit, 16-bit, 32-bit, or 64-bit units—the algorithm must accommodate certain structural complexities. These constraints influence how calculations are performed and how data is processed internally.

In contrast, when deployed on custom-designed hardware that is not segmented by byte boundaries, the computation becomes significantly more linear and efficient. The mathematical operations can be executed without the overhead imposed by traditional CPU architectures, resulting in simplified logic, higher performance, and substantial energy savings.

Therefore, the current CPU-based code should be interpreted as a mathematical demonstration of the algorithm’s core logic. It is intended to provide the academic and technical community with a reference model, while the ultimate goal remains the deployment of this algorithm within dedicated hardware environments optimized for AI acceleration.

### **Demonstrative Purpose and Academic Validity**

The algorithm implementation provided below is intended to serve as a demonstrative model of the underlying mathematical logic. In academic and technical contexts, a well-structured algorithm—expressed in code—can effectively substitute for a textual description of its functioning, offering clarity, precision, and reproducibility.

This example is written in C#, a widely adopted programming language chosen for its readability and accessibility. However, the algorithm itself is not bound to any specific language. Its core logic is language-agnostic and can be implemented in any computational environment, including hardware-level architectures.

It is important to note that the current implementation targets execution on conventional CPUs. Due to the segmented nature of CPU architectures—organized into byte-based units such as 8-bit, 16-bit, 32-bit, or 64-bit—the algorithm must accommodate structural constraints that affect how data is processed and how operations are performed.

In contrast, when deployed on custom-designed hardware free from byte segmentation, the computation becomes significantly more linear and efficient. The absence of architectural fragmentation allows for simplified execution paths, reduced overhead, and enhanced performance, with corresponding gains in energy efficiency.

Therefore, the code presented here should be interpreted as a mathematical demonstration of the algorithm’s logic, intended to provide the academic and technical community with a transparent and verifiable reference. It does not represent the final optimized implementation, which will be adapted for dedicated hardware environments as part of the broader AI Accelerator initiative.

/\*

\* Bruno Multiplication Algorithm

\* Invented and authored by Andrea Bruno

\* This algorithm represents a novel approach to fast integer multiplication,

\* optimized for future deployment on dedicated hardware architectures.

\* All rights reserved under exclusive copyright.

\*/

public static byte[] Multiplication(byte[] multiplicand, byte[] multiplier)

{

//============ initialization

const int word = 8;

if (multiplicand.Length < multiplier.Length)

{

var tmp = multiplicand;

multiplicand = multiplier;

multiplier = tmp;

}

var result = new byte[multiplicand.Length + multiplier.Length];

var o = new int[multiplier.Length \* 8];

var ad = 0;

var r = 0;

//============ end initialization

for (var p = 0; p < result.Length \* word; p++)

{

var by = p >> 3;

var bt = p & 7;

var s = 0;

if (p < multiplier.Length \* word)

if ((multiplier[by] & 1 << bt) != 0)

{

o[ad] = p;

ad++;

}

if (p < multiplicand.Length \* 8)

for (var i = 0; i < ad; i++)

s += (multiplicand[(p - o[i]) >> 3] & (1 << ((p - o[i]) & 7))) != 0 ? 1 : 0;

else if (r == 0)

return result;

r += s;

result[by] |= (byte)((r & 1) << bt);

r >>= 1;

}

return result;

}

### **Algorithmic Efficiency and Hardware Implications**

It is important to clarify that the effectiveness of an algorithm is not necessarily measured by its computational complexity or the difficulty of its implementation. Instead, true efficiency lies in how well the algorithm aligns with the architecture on which it is executed, and how it minimizes resource consumption while maximizing performance.

The fast multiplication algorithm presented here, when transposed onto dedicated hardware, can be executed with minimal computational effort. In its optimized form, the algorithm does not require iterative or force-based computation. Instead, it operates by placing the input arrays (byte[] multiplicand, byte[] multiplier) into memory in a specific configuration, which will be described in detail in the following sections of this document. The result can then be retrieved through a single linear scan of the memory as a bitstream, effectively bypassing traditional arithmetic operations.

This approach leverages the principle that, on custom hardware free from byte-segmented constraints, computation can be redefined as structured memory access. Unlike conventional CPUs—where processing is segmented into 8, 16, 32, or 64-bit units, introducing architectural overhead—the dedicated hardware envisioned for this algorithm allows for seamless bit-level manipulation. This results in a dramatically simplified execution path, enhanced performance, and significant energy savings.

Therefore, the current CPU-based implementation serves as a mathematical demonstration of the algorithm’s logic. The ultimate goal is to deploy this logic on specialized hardware where its full potential can be realized, transforming multiplication from a computational task into a deterministic memory operation.

As may already be evident, when transposed onto dedicated hardware, this algorithm can be executed without the use of traditional computational operators or arithmetic calculations. This architectural simplification significantly reduces the required circuitry and minimizes clock cycles, resulting in maximum performance with optimal energy efficiency.

By eliminating the need for complex processing units and iterative logic, the algorithm enables a streamlined execution model that relies solely on structured memory placement and deterministic bit-level access. This not only enhances speed and scalability, but also aligns with the principles of low-power, high-throughput computing—making it particularly suitable for next-generation AI acceleration platforms.

## **Scientific Relevance and Commercial Value of Primitive Algorithms**

Multiplication is one of the most fundamental operations in mathematics—classified as a primitive function due to its foundational role in arithmetic and computation. As such, multiplication algorithms are among the most widely studied and frequently applied across virtually every domain of science and engineering.

Their applications include, but are not limited to:

* ****Cryptography****: Modular multiplications are central to RSA, ECC, and other public-key systems.
* ****Artificial Intelligence****: Neural networks rely heavily on matrix multiplications and tensor operations.
* ****Scientific Computing****: Simulations in physics, biology, and chemistry require high-precision arithmetic.
* ****Graphics and Signal Processing****: Transformations and filters often involve large-scale multiplications.

Given this ubiquity, any innovation in multiplication algorithms has the potential to impact a wide range of industries. The fast multiplication algorithm presented in this project, when executed on dedicated hardware—as originally conceived—represents a generational leap in computational design. By eliminating traditional arithmetic operations and relying instead on structured memory placement and bitstream scanning, the algorithm achieves unprecedented efficiency in both speed and energy consumption.

Primitive algorithms are the subject of intense academic scrutiny precisely because they underpin so many systems. Countless researchers have attempted to improve upon them, and breakthroughs are rare. When such breakthroughs do occur, their commercial value can be substantial—often reaching into the millions of dollars.

### **Notable Examples of High-Value Algorithms**

* ****RSA Algorithm****: While not a multiplication algorithm per se, RSA relies on modular exponentiation, which is built on repeated multiplications. The original patent (U.S. Patent No. 4,405,829) was licensed commercially and generated millions in revenue before expiring.
* ****Schönhage–Strassen Algorithm****: Used for multiplying extremely large integers, this algorithm was implemented in software like GMP (GNU Multiple Precision Arithmetic Library), which powers many cryptographic and scientific applications. Its efficiency has enabled breakthroughs in prime number discovery and π computation.
* ****Fast Fourier Transform (FFT)****: Though not a multiplication algorithm directly, FFT reduces the complexity of convolution operations, which involve multiplication. The Cooley–Tukey FFT algorithm has been foundational in signal processing and has enabled entire industries—from telecommunications to audio engineering.

These examples illustrate that technical algorithms, especially those with broad applicability and performance advantages, can become invaluable assets. Their value lies not only in their intellectual merit but in their ability to unlock new capabilities, reduce costs, and enable technologies that were previously impractical.

The fast multiplication algorithm presented here, if adopted in scientific or industrial contexts, holds similar potential. Its simplicity, scalability, and compatibility with custom hardware make it a candidate for integration into next-generation AI accelerators, cryptographic processors, and high-performance computing platforms.

### **Potential Cryptanalytic Implications**

While the fast multiplication algorithm presented in this project is not designed to break RSA encryption, it is built upon a mathematical intuition that may hold deeper implications. The core idea—currently undisclosed—suggests a novel way of approaching multiplication that, if reversed or reinterpreted by experts in number theory and cryptography, could inspire alternative strategies for tackling problems such as large prime factorization.

RSA encryption relies on the computational difficulty of factoring the product of two large prime numbers. The absence of an efficient algorithm for this task is what ensures the security of RSA-based systems. However, the logic underpinning this algorithm introduces a new computational paradigm that bypasses traditional arithmetic operations, and instead leverages structured memory access and bit-level manipulation.

It is conceivable that a team of highly qualified mathematicians could study this paradigm and, through reverse engineering or inversion of its principles, explore new avenues in cryptanalysis. Such exploration might challenge long-standing assumptions about the hardness of certain problems, including those on which RSA is based.

The author of this algorithm openly declares that the mathematical foundations and logical principles underlying the algorithm—not the algorithm itself—may serve as a scientific basis for developing techniques capable of compromising RSA encryption. Given that RSA’s security relies on the absence of an efficient method for factoring large prime products, any breakthrough in this domain could have serious implications for global cybersecurity.

For this reason, and to avoid potential harm to the public and critical infrastructure, the author has chosen not to disclose the core theoretical constructs that inspired the algorithm. The decision reflects a commitment to responsible innovation and the ethical handling of sensitive mathematical insights that may, if misused, undermine the integrity of encrypted communication systems worldwide.

## **Structural Design of the Multi-Precision Unit (MPU)**

The foundational architecture of the MPU is based on a square bit matrix embedded directly onto the chip. This matrix serves as the computational substrate for executing large-number multiplications with exceptional efficiency.

The matrix is strictly square, meaning the X and Y axes must be of equal length. Its side length, measured in bits, must be ****twice the bit-width of the largest operand**** the unit is intended to support. For example, if the maximum operand size is 1024 bits, the matrix must be 2048 bits per side (i.e., 2048 × 2048 bits). This ensures sufficient space for representing all intermediate states and results without overflow or segmentation.

For production-grade implementations, it is recommended to support operand sizes up to 4096 bits. Accordingly, the bit matrix should be at least ****8192 × 8192 bits**** in dimension. This configuration allows the MPU to handle high-precision multiplications required in cryptographic, scientific, and AI workloads, while maintaining linearity and minimizing computational overhead.

The square matrix design is central to the algorithm’s ability to bypass traditional arithmetic operations. It enables deterministic placement of input values and facilitates direct bitstream scanning for result retrieval, forming the basis of a radically simplified and energy-efficient multiplication process.

## **Offset Matrix and Horizontal Brush Logic in the MPU Architecture**

In the proposed architecture of the Multi-Precision Unit (MPU), the two input operands—referred to as the multiplicand and the multiplier—are first analyzed to determine which of the two contains fewer bits set to 1. The operand with the lower bit density is designated as the ****Offset Matrix****, due to its intrinsic role in defining the activation pattern across the computational grid.

To determine the starting position of the computational brush on each horizontal line, the system may employ a diagonal scan mechanism. This scan is conceptualized as a 45-degree traversal along the main diagonal of the square bit matrix, beginning at the top-left vertex. For each row intersected by the diagonal, the presence of a bit set to 1 indicates the ****offset position**** where the horizontal brush will initiate its operation. Conversely, rows intersected by a 0 bit remain inactive, contributing to thermal efficiency by reducing unnecessary switching activity.

It is important to note that the diagonal scan is not a mandatory physical implementation. An alternative approach involves directly analyzing the positions of bits set to 1 within the Offset Matrix and applying those positions as row-specific offsets. This method simplifies the logic and allows for more flexible hardware design.

Once the offset positions are established, the system utilizes ****Shift Register technology****, such as ****Linear Feedback Shift Registers (LFSR)****, to propagate a consistent bit pattern across each active row. The pattern remains uniform across rows, but each row applies its unique offset, resulting in a staggered and efficient multiplication process. This mechanism forms the basis of the horizontal brush logic, enabling parallel and deterministic computation without traditional arithmetic operations.

The combination of offset-based activation and LFSR-driven propagation allows the MPU to execute large-number multiplications with minimal energy consumption and maximal throughput, leveraging spatial memory placement rather than iterative computation.

The following is a purely minimalist example to simplify the operation as much as possible. Suppose we want to handle multiplications with numbers of up to 3 bits, so we need a 6-bit matrix (3 x 2 = 6).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ****0**** | 0 | 0 | 0 | 0 | 0 |
| 0 | ****0**** | 0 | 0 | 0 | 0 |
| 0 | 0 | ****0**** | 0 | 0 | 0 |
| 0 | 0 | 0 | ****0**** | 0 | 0 |
| 0 | 0 | 0 | 0 | ****0**** | 0 |
| 0 | 0 | 0 | 0 | 0 | ****0**** |

In the table above, the diagonal representing the "offset matrix" is highlighted in bold. This indicates the rows where the horizontal brush will begin drawing the pattern—specifically at the positions marked with a one.

It is important to note that the binary representation adopted in this project places the most significant bit (MSB)—the bit representing the highest unit—on the ****left****, which is visually opposite to the standard decimal notation where the least significant digit appears on the right. This left-to-right binary layout is used consistently throughout the documentation and matrix visualizations.

In our example, we aim to multiply 3 and 5, whose binary representations—following this convention—are:

* ****3 → 110****
* ****5 → 101****

NOTE:

While this project adopts a left-to-right binary layout—placing the most significant bit (MSB) on the left for visual and structural clarity—it is important to note that the system is fully compatible with the conventional right-to-left binary format, where the least significant bit (LSB) appears on the right.

The computational model and matrix logic are inherently symmetrical, meaning that the algorithm can operate correctly regardless of the chosen bit orientation. Whether the binary representation places the units on the left or the right, the offset detection and horizontal brush propagation mechanisms remain valid, provided that the interpretation of bit positions is consistent throughout the system.

This flexibility allows for seamless integration with existing standards and hardware architectures, and ensures that the algorithm can be adapted to various implementation environments without loss of functionality or precision.

### **Offset Matrix Initialization**

In our example, both 3 and 5 have the same number of bits set to 1 (two bits each). Therefore, it is irrelevant which operand is selected as the "offset matrix." For simplicity, we choose the first operand—****3****—and map its binary representation onto the diagonal of the matrix.

Using the left-to-right binary convention adopted in this project, the number ****3**** is represented as 110. We place these bits along the diagonal of a 6×6 matrix, corresponding to the base model designed to support 3-bit operands (matrix size = 3 × 2 = 6).

The updated matrix is shown below, with the diagonal cells reflecting the binary pattern of the offset matrix:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ****1**** | 0 | 0 | 0 | 0 | 0 |
| 0 | ****1**** | 0 | 0 | 0 | 0 |
| 0 | 0 | ****0**** | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |

### **Pattern Propagation Using LFSR Technology (Horizontal Activation)**

In our example, both operands—3 and 5—have two bits set to 1, so either can be selected as the "offset matrix." We choose ****3**** (binary 110) and place its bits along the diagonal of the 6×6 matrix. The remaining operand, ****5**** (binary 101), is treated as the ****pattern****.

Each row in the matrix corresponds to a bit position in the offset matrix. Where a diagonal cell contains a 1, it signals that the row is active and the horizontal brush must initiate at that offset. The brush then propagates the ****entire pattern**** horizontally from that position, using ****Shift Register technology****, such as ****Linear-Feedback Shift Registers (LFSR)****.

This means:

* The pattern 101 is applied ****horizontally**** on each active row.
* The starting column (offset) is determined by the position of the 1 bit in the offset matrix.
* Inactive rows (where the offset bit is 0) remain untouched, contributing to thermal efficiency.

The updated matrix below reflects this logic:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ****1**** | 0 | 1 | 0 | 0 | 0 |
| 0 | ****1**** | 0 | 1 | 0 | 0 |
| 0 | 0 | ****0**** | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |

Note: The pattern 101 is applied from left to right on each active row, starting at the offset position indicated by the diagonal. The LFSR mechanism ensures efficient bit propagation without traditional arithmetic operations.

This horizontal propagation model is central to the MPU’s architecture, enabling fast and energy-efficient multiplication through spatial logic rather than iterative computation.

In the computation, lines without offset placeholders for the pattern are completely irrelevant and can be excluded. This way, we can simplify the logic and reduce the matrix to:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ****1**** | ****0**** | ****1**** | 0 | 0 | 0 |
| 0 | ****1**** | ****0**** | ****1**** | 0 | 0 |

Vertical scan result: 1 1 1 1 0 0 (i.e. 15 : [1] + [2] + [4] + [8] + [0] + [0] )

### **Result Composition via Vertical Scan and Binary Addition**

Once the pattern has been propagated horizontally across the active rows—starting at the offset positions defined by the offset matrix—the final step is to ****compose the result**** by scanning the matrix ****vertically****, column by column.

Each column represents a bit position in the final binary result. During the vertical scan:

* If a column contains a single 1, that bit is directly included in the result.
* If a column contains ****multiple**1**s****, they are ****combined using binary addition****, and the excess is ****carried over**** to the next column on the right.

This behavior reflects the fundamental principle of binary arithmetic:

Two 1s in the same column produce a 0 in that column and a 1 in the column to the right (carry bit), since each position in binary represents a power of two.

For example:

* Column 2 has two 1s → binary sum is 10 → 0 in column 2, 1 carried to column 3.
* Column 3 now receives the carry and is processed similarly.

This process continues across all columns, resulting in a final binary number that represents the product of the two original operands.

The vertical scan and carry logic can be implemented using simple ****adder circuits**** or ****bitwise accumulation logic****, depending on the hardware architecture. This final step completes the multiplication without requiring traditional arithmetic operations, relying instead on spatial logic and binary principles.

### **1. Executive Summary**

****Technology Brief: Parallel Pattern Propagation Engine (P3E)**** ****A Novel Hardware Architecture for High-Performance Cryptographic and Mathematical Acceleration****

This document outlines a groundbreaking hardware architecture, the ****Parallel Pattern Propagation Engine (P3E)****, designed to perform ultra-fast large-number multiplication—the foundational operation for modern cryptography (RSA, ECC), AI model inference, and scientific computing. Unlike traditional sequential processors or even modern GPUs, the P3E leverages a massively parallel array of shift-register-like structures, enabling deterministic, low-power computation with a single, efficiently broadcast data pattern. This technology represents a paradigm shift in computational efficiency and is the subject of a pending patent portfolio.

The Parallel Pattern Propagation Engine (P3E) is a leap forward in computational architecture. By leveraging the simple, yet powerful, properties of shift registers and LFSRs in a massively parallel and deterministic design, it offers a compelling advantage in performance, power efficiency, and security over all existing solutions. The pending patent secures a strategic position in the future of computing, from securing digital infrastructure to powering the AI revolution. This technology is not merely an improvement; it is a re-imagination of how computation can be done.

### **2. Core Technological Innovation**

The P3E architecture is based on a key insight: ****large-number multiplication can be decomposed into the parallel application of a single binary pattern across a computational grid, where each row processes the pattern with a unique, predetermined offset.****

* ****The "Brush" and "Canvas" Metaphor:**** A single master data pattern (the "Brush") is broadcast to all rows of a computational matrix (the "Canvas"). Each row applies this pattern but starts processing it at a different, predefined offset. This offset is not loaded from memory but is generated deterministically in situ based on the row's index.
* ****Shift-Register/LFSR-Based Execution:**** The core processing element of each row is not a traditional arithmetic logic unit (ALU) but a configurable shift register or Linear-Feedback Shift Register (LFSR). This design allows the entire grid to be updated and advanced with a single, global clock cycle, propagating the calculation in perfect lockstep with minimal control overhead.

### **3. Key Advantages and Benefits**

#### **A. Unprecedented Performance per Watt**

* ****Massive Parallelism:**** The architecture performs the core computational step for all rows simultaneously on every clock cycle.
* ****Elimination of Data Movement Bottleneck:**** By generating offsets deterministically instead of transferring them, the P3E avoids the immense power and time cost of moving terabytes of data between memory and processors, which is the primary limiter in modern computing.
* ****Constant-Time Execution:**** The computation time is deterministic and dependent only on the operand size, not on their values, which is critical for mitigating timing-based side-channel attacks in cryptography.

#### **B. Significant Reduction in Hardware Complexity & Cost**

* ****Minimal Input Data Requirement:**** Instead of loading gigabytes of data, the P3E is initialized by loading only one pattern (N bits) and starting the clock. This simplifies memory hierarchy and bus design.
* ****Repeated Use of Simple Logic:**** The grid is composed of simple, identical processing elements (shift registers) rather than a mix of complex multipliers, adders, and carry-propagators. This reduces design verification time, silicon area, and manufacturing costs.
* ****Native Scalability:**** Supporting larger operands (e.g., moving from 4096-bit to 8192-bit math) primarily involves adding more identical rows, making the design future-proof and easily scalable across product lines.

#### **C. Enhanced Security Profile**

* ****Natural Resistance to Side-Channel Attacks:**** The power consumption and electromagnetic footprint of the P3E are consistent on every cycle, regardless of the data being processed. This makes it inherently resistant to Differential Power Analysis (DPA) and similar attacks that plague traditional software and hardware implementations.
* ****LFSR Integration for Cryptographic Strength:**** Using LFSRs adds a layer of cryptographic confusion. The feedback taps can be made configurable, effectively allowing the hardware's "personality" to be altered for different algorithms or security domains, making reverse engineering exceptionally difficult.

### **4. The Critical Role of Shift Registers and LFSRs**

The choice of shift registers and LFSRs is not arbitrary; it is the engine of this innovation.

* ****Shift Registers: The Workhorses of Parallelism:**** They provide the perfect mechanism for applying the patterned transformation with an offset. A simple barrel shifter at the input of each row can create the initial offset instantly upon load.
* ****LFSRs: Adding Algorithmic Power:**** LFSRs transform the design from a fixed-function multiplier into a more general-purpose cryptographic engine. LFSRs are mathematically defined and can be used to generate pseudo-random sequences, perform Galois Field arithmetic crucial for ECC, and add a non-linear element to the computation, greatly enhancing utility and security over a simple shift register.

### **5. Utility and Patent Strategy**

The patent application protects not just a specific circuit but a ****novel method of computation****.

* ****Broad Claims:**** The claims cover the fundamental concept of parallel pattern propagation with deterministic offsets for mathematical acceleration. This grants protection that is independent of the specific transistor-level implementation.
* ****Foundational Technology:**** The P3E is not just another accelerator; it is a new way of organizing computation. This makes it a foundational technology. Licensees could implement it in ASICs, FPGAs, or integrated into future CPU/GPU designs.
* ****Diverse Application Markets:**** The patent's value is multiplied by its applicability in numerous high-value markets:
  + ****Cryptography:**** Next-generation secure elements, Hardware Security Modules (HSMs), blockchain accelerators, and post-quantum cryptography research.
  + ****Artificial Intelligence:**** Acceleration of core operations in Large Language Models (LLMs) and other transformers that rely on matrix multiplication and attention mechanisms.
  + ****High-Performance Computing (HPC):**** Scientific simulation, climate modeling, and astrophysics calculations requiring precise, high-precision arithmetic.

## **Integrating METIS into AIPU Architectures: A Scalable Engine for Integer Acceleration in AI Systems**

The Multi-Precision Unit, referred to as METIS—short for Multiplicator Engine for Tensor-Integrated Systems—is designed to integrate seamlessly into modern AI acceleration platforms, especially those built around AIPU (AI Processing Unit) architectures. Its modular structure, low power consumption, and deterministic execution model make it a natural fit for AIPU-based systems, regardless of the deployment format.

METIS can be embedded into a variety of hardware configurations commonly used in AI environments. In compact M.2 AI Edge Accelerator Cards, which are optimized for edge inference tasks, METIS enhances capabilities by enabling high-speed integer multiplication for cryptographic preprocessing and control logic, all while maintaining energy efficiency. In high-throughput PCIe AI Accelerator Cards, METIS can function as a co-processor or be integrated directly into the card’s logic fabric, allowing mathematical operations to run in parallel with floating-point AI workloads and improving overall system responsiveness. Embedded compute boards, including those based on ARM or RISC-V architectures, also benefit from METIS’s simplicity and scalability, making it a powerful addition to industrial, automotive, and IoT applications that require secure and efficient mathematical acceleration.

The integration of METIS into AIPU architectures offers several strategic advantages. Its minimal resource footprint allows it to occupy only a small portion of the silicon area compared to traditional ALUs, which is especially valuable in systems where space and power are limited. While AIPUs are optimized for tensor and floating-point operations, METIS complements them by delivering deterministic, high-speed integer multiplication—an essential capability for cryptographic routines, hashing, and low-level AI control logic. Its architecture is inherently scalable, allowing multiple instances to be deployed across the AIPU fabric to meet the performance demands of modern AI systems.

By embedding METIS into AIPU-based platforms—such as those powered by Axelera AI’s Metis® chips—developers can significantly expand the mathematical capabilities of their systems without sacrificing efficiency or security. METIS offloads bitwise and integer-heavy operations, freeing AIPU cores to focus on neural inference and tensor manipulation. This positions METIS not only as a versatile and forward-compatible component in the evolving landscape of edge and embedded AI computing, but also as a strategic enabler for the next generation of AIPU deployments across industries.

### **In-Memory Execution of the Bruno Algorithm via METIS Technology**

The Bruno algorithm represents a groundbreaking approach to large-number multiplication, distinguished by its mechanical simplicity and its compatibility with in-memory computing paradigms. Unlike conventional multiplication methods that rely on arithmetic logic units, carry propagation, and complex operator chains, Bruno performs its computation through deterministic bit placement and offset alignment. This makes it an ideal candidate for execution within memory itself—without invoking traditional logical or mathematical operations.

This architectural elegance aligns perfectly with ****Dataflow In-Memory Computing (DIMC)****, the memory-centric paradigm used in platforms such as Axelera AI’s Metis® AIPU. DIMC enables computation to occur directly where data resides, eliminating the latency and energy overhead of data movement. The Bruno algorithm, when deployed via the ****METIS**** execution engine (****Multiplicator Engine for Tensor-Integrated Systems****), can be computed entirely within memory arrays using simple write logic and offset control.

One of Bruno’s most compelling features is that it does not require a two-dimensional matrix to operate. While the parallel brush model offers performance advantages, the algorithm can be executed linearly by collapsing the matrix into a single array of bits. This array has a length equal to the sum of the bit-widths of the multiplicand and multiplier. The computation then becomes a matter of writing bits into this array at the correct offset positions, following a fixed pattern. No arithmetic units, no logical gates—just a controlled bitstream propagation.

The process is so intuitive that it can be described as ****mechanical computation****. A human, using strips of paper and binary annotations, could perform the multiplication manually by aligning the patterns and applying the offset rules. This analog analogy underscores the algorithm’s accessibility and scalability: Bruno is not bound by the limitations of digital logic and scales naturally with operand size.

When executed via METIS, Bruno becomes a memory-native algorithm. The METIS engine does not require sophisticated logic circuitry; it simply orchestrates bit placement according to Bruno’s rules. This results in a dramatically simplified execution path, enhanced performance, and significant energy savings. The absence of conditional branching, arithmetic evaluation, or logical comparison makes Bruno ideal for low-power, high-throughput systems.

In essence, Bruno transforms multiplication from a computational task into a deterministic memory operation. Its compatibility with DIMC and AIPU platforms positions it as a foundational algorithm for next-generation AI, cryptographic, and embedded systems. By leveraging the METIS execution engine, Bruno offers a new paradigm in algorithmic design—one that is elegant, scalable, and profoundly efficient.  
  
What sets this computational method apart is not merely its efficiency, but the way it reimagines the act of multiplication itself. Rather than relying on traditional arithmetic logic or complex operator chains, the algorithm executes through a spatial and mechanical process—one that can be carried out entirely within memory, without invoking any logical or mathematical circuitry. This shift from computation to structured bit placement marks a fundamental innovation in how large-number multiplication can be approached.  
  
The algorithm’s execution model is so straightforward that it transcends the need for conventional processing units. It operates by writing bits into a linear array according to fixed offset rules, requiring no branching, no evaluation, and no iterative logic. This simplicity makes it inherently compatible with Dataflow In-Memory Computing architectures, where computation occurs directly where data resides. When deployed through a memory-native engine like METIS (Multiplicator Engine for Tensor-Integrated Systems), the algorithm becomes not just efficient—it becomes elemental.  
  
This mechanical nature has profound implications for energy consumption. In traditional systems, every logical operation contributes to heat, requiring cooling infrastructure and increasing power draw. Here, the absence of active logic means that thermal dissipation is drastically reduced. The algorithm’s deterministic behavior allows it to run with minimal switching activity, translating into lower energy usage and a smaller environmental footprint. It is not just a faster way to multiply—it is a cleaner one.  
  
In a world increasingly concerned with sustainability, this approach offers a compelling alternative. It enables high-precision arithmetic without the overhead, allowing systems to scale computationally without scaling their energy demands. The result is a form of green technology that doesn’t compromise performance, but rather enhances it through architectural elegance.  
  
This is not merely a technical refinement—it is a philosophical one. It suggests that the future of computation may lie not in ever more complex logic, but in simpler, more deliberate design. By reducing the act of multiplication to a memory operation governed by spatial rules, this method opens the door to a new class of algorithms that are efficient, scalable, and environmentally responsible. It is a quiet revolution—one that begins not with heat and noise, but with silence and precision.