Scaling up prime factorization with self-organizing gates: A memcomputing approach

Tristan Sharp, Rishabh Khare, Erick Pederson, and Fabio Lorenzo Traversa*
MemComputing, Inc. San Diego, California
(Dated: September 18, 2023)

We report preliminary results on using the MEMCPU™ Platform to compute the prime factorization of large biprimes. The first approach, the direct model, directly returns the factors of a given biprime. The second approach, the congruence model, returns smooth congruences to address the bottleneck of standard sieve methods. The models have size-dependent structure, and the MEMCPU Platform requires structure-dependent tuning for optimal performance. Therefore, for both models, we tuned the platform on sample problems up to a given size according to available resources. Then we generated RSA-like benchmark biprimes to perform rigorous scaling analysis. The MEMCPU timings over the tuned range followed low degree polynomials in the number of bits, markedly different than other tested methods including general number field sieve. MEMCPU's congruence model was the most promising, which was scaled up to 300-bit factorization problems while following a 2^{nd} degree polynomial fit. We also discuss the approach to tuning the MEMCPU Platform for problems beyond the reach of today's most advanced methods. Finally, basic analysis of the acceleration expected from an ASIC implementation is provided and suggests the possibility of real time factorization of large biprimes.

I. INTRODUCTION

Electronic computing machines have played a crucial role in shaping the contemporary technological landscape [1, 2], and for almost a century, the von Neumann architecture [3] has served as the standard reference. Introduced in the mid-20th century, this architecture comprises essential components, including input and output modules, a central processing unit, and a memory bank. It has facilitated the development of general-purpose computing machines, where sets of instructions (programs) are stored in memory and executed by the CPU, with data continuously exchanged between memory and the processing unit [4]. Although the von Neumann architecture has proven versatile [5], it is not without its inherent limitations, most notably the well-known "von Neumann bottleneck" [4, 6], which constrains system throughput due to data transfer between the CPU and memory, thereby resulting in energy inefficiency [4, 7, 8]. Additionally, as certain computational problems increase in size, the efficiency of von Neumann architectures becomes limited [5], rendering certain problems intractable. Consequently, the demand for enhanced performance, energy efficiency, and reliability in computing devices has spurred the investigation of novel paradigms and alternative computing solutions.

In recent times, both academia and industry have put forth diverse innovative methodologies to overcome the limitations of traditional von Neumann architectures and meet the escalating demands for computational capabilities. These solutions encompass specialized hardware and architectures tailored for specific applications, such as graphical processing units (GPUs) [9, 10] that were initially devised for graphical tasks but are now extensively employed in various domains, including simulations and machine learning [11–14]. Moreover, there has been a revived interest in neuromorphic computing, where asynchronous digital or analog circuits [15, 16] emulate brain neural networks [17–20]. In the pursuit of mitigating the von Neumann bottleneck, the concept of near-memory computing [21] has emerged as a promising approach, closely related to in-memory computing [22–27]. However, pushing the boundaries even further, the notion of memcomputing has been introduced as a non-Turing paradigm [28], representing the most idealized model of computational memory [28–30].

The diverse array of emerging computing approaches presents captivating possibilities for meeting the impending computational demands, albeit lacking a definitive front runner at this stage. The exploration of alternative computing paradigms, such as p-bits [31] and coupled oscillators [32–35], has exhibited promise, while several specialized hardware solutions are actively under development [36–44]. As the exponential growth in computing power demand continues, it becomes increasingly apparent that the evolution of computing devices and architectures will exert a pivotal influence on the future technological landscape. This manuscript delves into the critical role that memcomputing can play in surmounting these challenges, offering wide-ranging industrial applications.

While ordinary logic gates constitute the foundation of the present electronic industry, which has made remarkable advancements, current architectures still face exponential time constraints when tackling NP-hard optimization problems. In contrast, memcomputing based solutions have shown potential on mitigating the exponential complexity of these problems [45–50]. Memcomputing can be realized in analog [51] or digital form [52]. Its most effective embodiment is represented by self-organizing gates (SOGs) [52–54]. These gates are engineered to achieve equilibrium based on prescribed relations at their terminals [52, 53]. Networking SOGs creates self-organizing circuits (SOCs) [29, 52, 53] adept

* ftraversa@memcpu.com

at solving complex combinatorial optimization problems [45–49, 55–57]. Remarkably, these gates can be realized in silicon emplying the same electronic components as ordinary electronic circuits. We formulate our problems as Integer Linear Programming (ILP) equations, which can be directly mapped to a Self-Organizing circuit. Our MEMCPU™ Platform [58] automates all of the aforementioned processes.

The MEMCPU Platform has exhibited remarkable success in solving challenging logistics problems pertinent to the oil & gas industry [55], aircraft scheduling [57, 59], and swarm optimization for drones [56]. These are examples of combinatorial optimization problems encountered in industry that are often intractable for state-of-the-art solvers [60, 61]. This renders them impractical for industry partners aiming to apply them at a larger scale. In contrast, our MEMCPU Platform demonstrates a polynomial-like scaling for most of these problems, resulting in substantial savings in labor and allocated resources for these specific tasks [55–57, 59].

Among intractable combinatorial problems, large-scale prime factorization is a well-known challenge, the study of which is critical to guaranteeing secure encryption systems in numerous technologies. Any algorithm capable of factoring large biprimes (products of two prime numbers) within a reasonable time frame poses a significant threat to several current encryption methods, particularly those relying on RSA-based encryption [62, 63]. Quantum Computers [64–66], leveraging Shor's algorithm [67], are considered a potential risk to RSA encryption [5]. There have been some approaches based on variational algorithms [68], however, the realization of a fault-tolerant and scalable quantum computer remains a distant aspiration [69], contingent on considerable experimental breakthroughs. Presently, sieve methods represent the state-of-the-art algorithms showing promise, with the general number field sieve method [70] being the most effective. Nevertheless, even these methods struggle to factor a 2048-bit RSA key within a sensible timeframe [71, 72], and past instances have taken almost 2700-CPU-years to factor an 829-bit number using computer clusters [72].

Our approach converts prime factorization into an ILP problem which is then conveniently mapped into a network of SOGs, which are the core of the MEMCPU Platform. This is prepared according to problem structures by tuning SOG design parameters. Finally, the circuit dynamics are initialized for a given input problem and run until MEMCPU converges to an equilibrium and hence the ILP solution is reached. If the structure of the problem changes with size, as for the factorization, the design parameter tuning process is repeated at each size. Details are discussed throughout this manuscript.

In [73] preliminary results of the MEMCPU Platform were reported. In that case, no size-dependent design parameter tuning was performed and no different structures of the ILP problem were considered during the preparation phase, and an older and less performant design parameter tuner was employed. By employing this whole and more advanced approach, the MEMCPU Platform is now seen to converge more efficiently to solutions of instances of factorization problems using a similar ILP formulation. The time-to-solution is observed to grow slowly with problem size. Empirically, the scaling follows a low-degree polynomial, a 5th-order polynomial fitting, while an example state-of-the-art ILP solver exhibits scaling following a 20th-order polynomial. The MEMCPU hardware, as estimated from the simulation results, appears to follow the scaling of a 3rd-degree polynomial, which would correspond to breaking 2048-bit RSA in just a few minutes. However this analysis was performed only up to 60-bit biprimes mainly due to budget constraints.

On the other hand, leveraging congruence methods for factorization further enhances the MEMCPU Platform results. We could push our analysis up to 300 bit biprimes before running into budget constraints. The scaling trend fitted by a 2nd-degree polynomial in this case, suggests reaching RSA-relevant sizes at the time scale of weeks of simulation or real time in hardware implementation. Although these initial results show great promise, constructing and scaling to higher numbers of bits present their own challenges, as discussed in this article. However, as far as our understanding goes at this point, we do not see any physical or mathematical roadblock that would prevent continuing this trend to larger sizes.

The outline of the manuscript is as follows: In section II, we introduce the MemComputing technology. In section III, we describe the problem, our formulation, and the timing results. In section IV we describe effective methods of tuning.

II. MEMCOMPUTING TECHNOLOGY

A. Brief Overview of Self-Organizing Gates

The MEMCPU™ Platform finds solutions to problems cast in Integer Linear Programming (ILP) format. However, before discussing the ultimate version of the technology used for this project to handle ILPs, let us briefly introduce a simplified version of the MEMCPU Platform for solving logic functions. This platform emulates networked Self-Organizing Logic Gates (SOLGs) [52, 74]. Each SOLG is an electronic circuit whose voltages at the terminals encode variables of the problem, and the SOLG drives these voltages to satisfy a Boolean relation among the variables[52, 74].

To give a better idea, let us first consider a standard logic gate as in Figure 1a. This is a traditional electronic circuit composed of transistors that sets the voltage v_o of the output terminal depending on how input terminals $v_{a'}$ and $v_{a''}$ are set. It does not work in reverse, which means that we cannot set the voltage at v_o and expect to find something meaningful at $v_{a'}$ and $v_{a''}$. The wavy arrows in Fig. 1a indicate the flow of meaningful signals

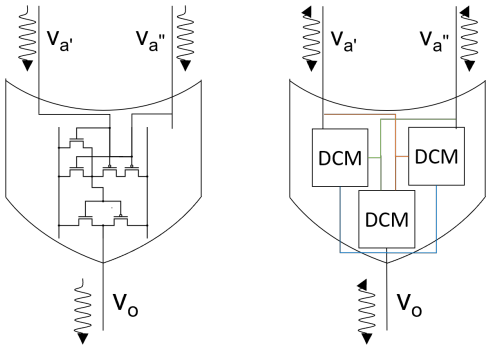


FIG. 1. The structures of a standard logic gate (a) and a self-organizing logic gate (SOLG) (b). (a) A standard logic gate composed of transistors that sets the voltage of the output terminal depending on the input terminals. The wavy arrows indicate the flow of meaningful signals through the gate. (b) A SOLG with a Dynamic Correction Module (DCM) connected to each terminal that generates feedback according to the signals it receives from the other terminals. The feedback ceases when the gate's terminals satisfy the logic relation, leading to the term "self-organizing". The building blocks of both gates are standard electronic devices and both encode information through thresholds. SOLGs, however, support asynchronous operation, superposition of input-output signals, and cooperative parallel computation.

through the standard logic gate.

SOLGs (Figure 1b) have similarities and major differences with standard gates. The most prominent similarities are:

1. The building blocks of both gates are standard electronic devices (transistors, resistors, etc.).

2. Both gates encode information through thresholds: if the terminal voltage is above a threshold, it encodes a logical 1 otherwise it encodes a logical 0

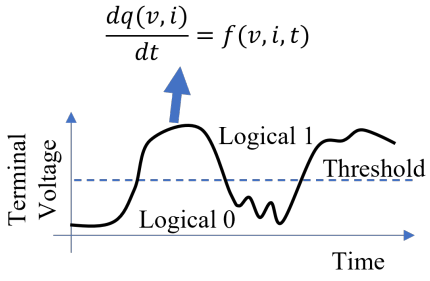


FIG. 2. Information encoding through thresholds. As in a standard logic gate, SOLGs encode information through thresholds, i.e., when a terminal voltage is above a defined threshold, it is in the logical state 1, otherwise in the logical state 0. The full voltage dynamics is determined by a system of differential equations among voltages and currents as in any electronic circuit.

(Figure 2).

These are important shared traits. The first implies that the emulation can be performed quite efficiently using standard modern computing hardware. The second shows that we can use the MEMCPU Platform to encode problems in a digital form. This avoids well-known precision issues associated with analog computers.

For clarity, what is meant by an efficient emulation should be explained. In this case, "efficient" means that it has a polynomial overhead with respect to the actual physical system, i.e., the circuit as realized in silicon. In contrast, for example, simulating a fault-tolerant quantum computer requires exponential overhead on modern hardware, which makes large scale simulations impractical. Given the polynomial overhead, the silicon version of the MEMCPU Circuit is expected to be several orders of magnitude faster than the MEMCPU Emulation that we have today. And, considering that an SOG circuit does not require access to an external memory bank during computation (the computation is within the SOG network), the energy reduction would be significant, since this eliminates the von Neumann bottleneck. The result would be an ultra-high performance and ultra-low power computational device.

Significant differences between SOLGs and standard gates are:

1. SOLGs are asynchronous and do not require a clock.

2. SOLGs support the superposition of input-output signals at each terminal. Standard gates have dedicated input and output terminals.

3. SOLGs compute cooperatively in parallel with other SOLGs, while standard gates compute sequentially.

These differences are profound and define the way SOLGs work. Figure 3 illustrates how these features impact the working principle of SOLGs.

Consider the logic formula represented in Conjunctive Normal Form (CNF) in Fig. 3. Each variable in the formula is a binary variable, and the goal is to find an assignment that makes the whole formula true. To find this assignment, we can use Self-Organizing (SO) NOT and multi-terminal OR gates connected as depicted in Fig. 3. By setting the output of the SO multi-terminal ORs to 1, we require that the circuit finds the assignments at the SO OR gate terminals that satisfy all gates simultaneously and therefore the CNF formula [75].

This example helps us understand the implications of the differences listed above. First, we notice that the SOLGs in Fig. 3 need to work cooperatively to find the assignment that satisfies all SOLGs. (Note that standard logic gates certainly could not be used in this setting, due to the differences listed above.) For a circuit like the one in Fig. 3 to work properly, i.e., to have gates cooperate effectively, the gates should exchange information about

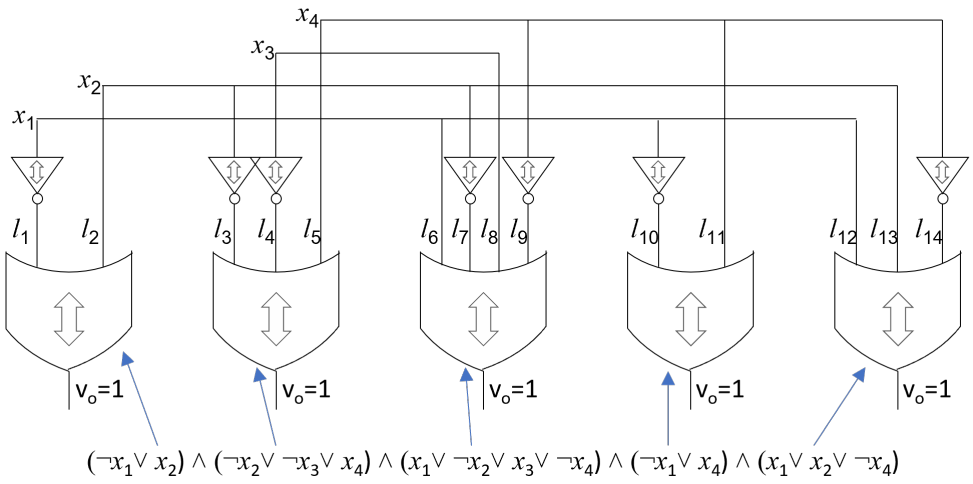


FIG. 3. A logic formula represented in conjunctive normal form (CNF) using Self-Organizing (SO) NOT and multiterminal OR gates. By setting the output of the SO multiterminal ORs to 1, the circuit finds, if it exists, the assignments at the SO OR gate terminals that satisfy all gates simultaneously and therefore the CNF formula. The SOLGs work cooperatively to find the assignment that satisfies all SOLGs, exchanging information about their state at each instant and adapting accordingly. This cooperation is achieved through superpositions of all signals arriving from the surrounding gates while simultaneously sending signals about their state, allowing for pure asynchronous communication.

their state at each instant so that they can adapt accordingly. This cooperation can be obtained only if each terminal of the gates receives signals carrying that information from the other gates. When designed in that way, the system can have the best response, because it has continuous knowledge of the rest of the circuit. That is, each gate dynamics is affected by changes to the surrounding gates as they occur.

To achieve the collaborative process just described, the terminals of the SOLGs must therefore support superpositions of all signals arriving from the surrounding gates while simultaneously sending signals about their state. This means that the SOLG terminals support superposition of both input and output signals (difference #2). The fact that each SOLG needs continuous knowledge of the state and internal dynamics of all other surrounding SOLGs means that the communication is continuous in time, and therefore it cannot be clocked. This ultimately implies pure asynchronous communication (difference #1).

As depicted here, the working principles of SOLGs are achieved using feedback loops. Each SOLG has what we call Dynamic Correction Modules (DCMs), one connected to each terminal, that generate feedback according to the signal it receives from the other terminals of the gate (see Fig. 1b). We use the term "self-organizing" to describe this activity: DCMs drive voltages autonomously via feedback loops, and once the SOLG satisfies the logical relation, the feedback ceases. Finally, where the terminal of one SOLG connects to one or more other SOLG terminals, the feedback from the DCMs of each terminal is propagated in the entire circuit, so the circuit behaves as a cohesive network to satisfy all SOLGs at once and minimize the global feedback [52, 74]. However, certain energy balance and signal propagation properties must be satisfied for the circuit to work properly [48, 76]. This requires finding design parameters that guarantee such conditions. Aspects of this are discussed in Section IV. (More information for particular applications is available in case studies, whitepapers, and peer-reviewed articles here [77]).

Finally, we introduce Self-Organizing Algebraic Gates (SOAGs) [53] designed to solve linear inequalities. Like the SOLGs, SOAGs have DCMs that create feedback on the terminal they are connected to based on signals from the other terminals, and they turn off if the gate is satisfied. The main difference is that an SOAG is satisfied when the state of its terminals (they still encode binary variables like SOLGs) satisfies a linear inequality. Using these gates, we can assemble circuits like the one in Figure 4 and use them to solve problems cast in the form of Integer Linear Programming (ILP), a much more versatile format for combinatorial optimization problems widely used in industry.

B. Other applications

We have applied the MEMCPU Platform to solve many combinatorial optimization problems relevant to academia [45, 48] as well as industry [47, 55, 57, 59], and military [56, 78]. We focus on problems that are usually hard, if not intractable, for state-of-the-art ILP solvers like IBM CPLEX [61] and Gurobi [60]. Our ap-

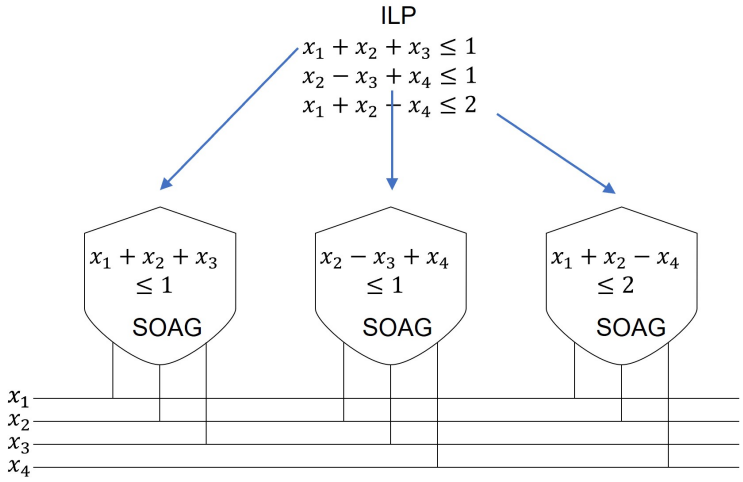


FIG. 4. A circuit assembled using Self-Organizing Algebraic Gates (SOAGs) designed to solve linear inequalities. Like SOLGs, SOAGs have DCMs that create feedback on the terminal they are connected to based on signals from the other terminals and turn off when the gate is satisfied. SOAGs are satisfied when the state of their terminals satisfies a linear inequality, allowing them to be used to solve problems cast in the form of Integer Linear Programming (ILP), a versatile format for combinatorial optimization problems widely used in industry.

proach usually outperforms those solvers for problems where it is hard to apply shortcuts or heuristics to simplify the problem (both CPLEX and Gurobi rely heavily on those methods to accelerate their solution when possible). Those solvers commonly rely on branch and cut, the "smartest" brute force method to solve ILPs. The compute time required and memory consumption can grow exponentially for these standard methods [79, 80]. We have proven the efficiency of the MEMCPU Platform on many scenarios for a variety of customers [77]. We summarize three of them in Appendix A to give an idea of the potential of our approach.

III. PRIME FACTORIZATION

RSA cryptography is a public-key cryptosystem that is widely used for secure data transmission. It was invented by Ron Rivest, Adi Shamir, and Leonard Adleman in 1971. The security of RSA relies on the practical difficulty of factoring the product of two large prime numbers, the "factoring problem" [62]. The RSA algorithm raises a message to an exponent e, modulo a composite number n, whose factors are not known. Thus, the task can be described as finding the $e-th$ roots of an arbitrary number, modulo n [62]. For large RSA key sizes (*e.g.* 1024 bits or more), no practical method for solving this problem is known; if an efficient method is ever developed, it would threaten the current or eventual security of RSA-based cryptosystems—both for public-key encryption and digital signatures [62, 63]. The most efficient method known to solve the RSA problem is by first factoring the modulus n [62, 63].

A. Existing Approaches

It is estimated that with current technology using the best-known algorithm (general number field sieve, GNFS), factoring a 2048-bit RSA key would take longer than the age of the universe. For reference, a 768-bit RSA key was factored by researchers in 2009, and that required more than two years of computation using hundreds of multicore CPUs [71]. After a decade, a new record was set in 2020 [72] where factoring an 829-bit number (RSA250) required 2700 CPU-years (it was factored parallelizing the GNFS over CPUs of supercomputer centers in France, Germany and California). Over more than a decade, we have only seen a very small improvement (i.e., 61 bits). This is mainly due to the lack of innovative new algorithms, and the fact that conventional computer performance is reaching a plateau. Factoring a 1024-bit number is considered to be potentially within reach of the resources of a large nation-state within the next decade using this method. However, factoring a 2048-bit number remains well beyond the capability of any known existing technology in the foreseeable future [71].

On the other hand, a hypothetical fault-tolerant quantum computer, if able to implement Shor's algorithm, could factor a 2048-bit RSA key in several hours or less [67]. Shor's algorithm is the best-known quantum algorithm for factoring large numbers and is exponentially faster than the most prominent classical algorithms. Shor's algorithm could run in polynomial time (specifically, time $\mathcal{O}((\log n)^3(\log\log n)(\log\log\log n))$, for a number n [81]). In contrast, the best-known classical factoring algorithm, the GNFS, runs in super-polynomial time complexity $\mathcal{O}(\exp(1.9(\log n)^{1/3}(\log\log n)^{2/3}))$ [70].

A fully fault-tolerant quantum computer is a significant technological challenge, and in 2021 industry consensus was that it was anywhere from 10 to 30 years in the future, and, even then it is still questioned whether it will break RSA 2048 encryption [82].

B. MemComputing Approach

MemComputing got limited funding for this project as part of a Phase II SBIR with an Air Force Intelligence Group in which we developed a MemComputing-based solution to perform prime factorization. It was built upon the cloud-based MEMCPU Platform. The MEMCPU Platform is programmed by submitting a mathematical model formulated using Integer Linear Programming (ILP) combined with specialized design and control parameters for the MemComputing Circuit. Part of the effort of properly programming the MEMCPU Platform is spent testing and profiling the mathematical model, identifying inefficiencies, and then iterating through improvements to reach the best fit. Once the mathematical model has been established, circuit design and control parameters need to be optimized to produce convergence as quickly and robustly as possible. Iterative testing and profiling is performed using advanced optimization techniques to improve the circuit's performance with knowledge of the mathematical formulation. Following this process, by the end of Phase II, the MemComputing solution was addressing up to 300-bit factorization problems.

1. Direct model: description and performances

We began with a direct approach to the factorization problem. This involved formulating solutions to the equation $pq = n$ where p, q are unknown integers and n is the known integer. There are many ways to transform this equation into an ILP model. They all involve binarization of the integers and then different ways to express the binary products and composing the equations. For example, the binary products $p_j q_k = s_{jk}$ can be expressed using 2 inequalities like,

$$p_j + q_k \leq s_{jk} + 1, \quad p_j + q_k \geq 2 s_{jk}. \tag{1}$$

However, it is easy to see that the first inequality can be substituted by $p_j + q_k \leq 2 s_{jk} + 1$ providing the same solutions. This is only an example of many different variations we can use to express the binary product. We tested many variations and eq. 1 worked better with the MEMCPU Platform [83].

Considering the expansion $p = \sum_{j=0}^{N_p - 1} 2^j p_j$, and similar ones for q and n and the binary products s_{jk}, we can express the product $pq = n$ through a set of equalities summing up all binary products belonging to groups of expansion coefficients. The general equations read

$$\sum_{g=MG}^{(M+1)G-1} 2^{g-MG} \left(\sum_{j+k=g} s_{jk} + \sum_{m=0}^{M-1} r_{mg} \right) = \sum_{g=MG}^{(M+1)G-1} 2^{g-MG} n_g + \sum_{g=G}^{G+\lfloor \log_2 \sup(\text{lhs}) \rfloor + 1} 2^g r_{Mg}, \tag{2}$$

where G is the size of the coefficient group, M is the M-th group of coefficients, r_{mg} are the binary coefficients of the remainders of the additions, and sup(lhs) is the upper bound that the left-hand side of eq. 2 can reach.

Also, in this case eq. 2 is not the only way to express the product $pq = n$ as an ILP model. We have tested many other variants, including more naive binary product representation using 3 bits adders or more involved variants using Karatsuba or Montgomery products. However, our tests showed that representation 2 with $G = 2$ was the best choice for the MEMCPU Platform [83].

Benchmark and Scaling results

The benchmarks we generated were designed to be among the hardest factorization problems. We used prescriptions to generate primes for strong RSA encryption [63]. The procedure involved generating a random string of bits and adding a 1 to both the beginning and end of the string to ensure that the number was odd and had a given length in bits. Then, we performed a primality test to select only prime random numbers. As a result, this selection would provide primes with a uniform distribution of bits. We then selected pairs of primes with the same length, whose product would produce an integer with a length exactly double that of the two primes.

In Figure 5, we report the timing results of this formulation. As the model is ILP, we also ran the same benchmark using the Gurobi ILP solver [60]. As expected, the performance from Gurobi quickly diverged and, in a log-log plot (time to solution *vs.* length of the product in bits), it was fit by a polynomial of degree 20.

On the other hand, the MEMCPU Platform displays two different scaling trends: one at a lower number of bits and the other at a higher number. This difference in scaling trends results from the ability of the preprocessing to simplify smaller problems more effectively, making them equivalent to even smaller ones. Indeed, the MEMCPU Platform includes a basic routine that simplifies, when possible, the ILP model submitted. For the factorization model, this routine is very effective up to about 40 bits, then it loses its efficiency. On the other hand, Gurobi has a much more advanced preprocessing routine that, looking at the log files, consistently works at any size we tested, providing a smoother scaling. This effect is further amplified by the fact that we were able to find better circuit design parameters for smaller problems, given the

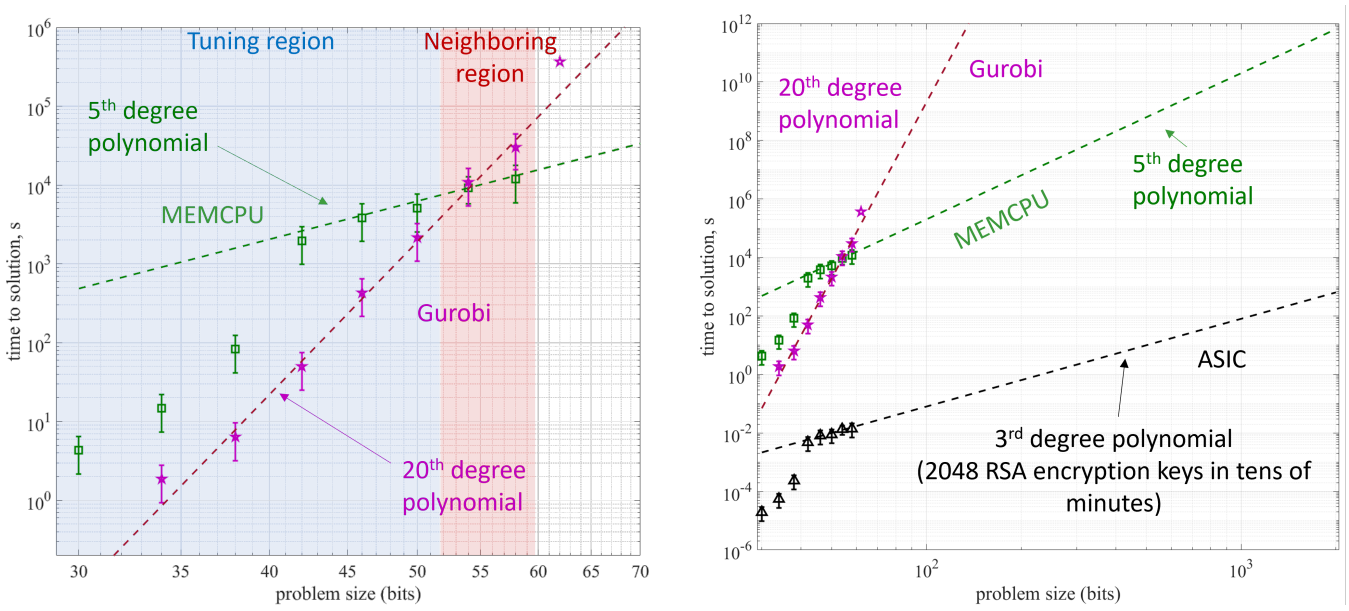


FIG. 5. Comparison of the scaling results for factorization problems between the MEMCPU Platform and the Gurobi solver. (a) Raw timings (symbols) and polynomial fits (dashed lines). The Tuning region is where the design parameters were fine-tuned for each size, while the Neighboring region employed the design parameters of smaller problem sizes, which, however, maintained about the same performance. (b) Same data, with extrapolation of the timing trend to 2048 bits, and the forecasted ASIC timings based on the MEMCPU Platform results.

cloud resource constraints during this SBIR and the need for further improvements to our in-house CAD toolbox (see section IV for details on the design procedure). The scaling reaches a more stable behavior above about 42 bits in length, fitted by a 5-degree polynomial.

As described in Section IV, the MEMCPU Platform is tuned for each size of factorization problem since the structure of the problem varies with the size. This procedure can be quite costly and is still being improved to increase its efficiency. We tuned for all sizes in the tuning region highlighted in figure 5a. However, due to limited resources, we performed only incomplete tuning starting from the previous size using continuation methods combined with parallel tempering (see section Section IV for details) up to 50 bits. Testing further (Neighboring region in figure 5a), up to about 60 bits, we were able to effectively use the the design found at lower bits and still keep the same scaling properties as reported in Figure 5a. These results represent already a major improvement, in terms of absolute timing (orders of magnitude faster) and scaling, with respect to the ones presented in [73] for motivations mentioned in the introduction. This proves that our approach is still prone to major improvements. For example, a more thorough parameter tuning and improvements to our in house CAD software will reduce much further the absolute timing and might even reduce the degree of the scaling. Also, tuning problems at larger sizes will extend the scaling to larger problem sizes. In addition, upgrades to the SOG design could also bring additional improvements.

In Figure 5b, the same scaling is extrapolated up to 2048 bits, assuming continued tuning. The timing for the ASIC realization of the MEMCPU Platform is also reported. The ASIC timing can be easily estimated since the MEMCPU Platform, being a circuit emulator, returns the full dynamics of the circuit, including the simulated runtime. It is worth noting that, at this point in our R&D, the forecast for the ASIC shows the possibility of solving a 2048-bit factorization problem in tens of minutes.

2. *Sieve model: description and performances*

Subsequently, we took a new approach based on a congruence method sharing similarities with the quadratic sieve or the general number field sieve (GNFS) method. In this case, we use the MEMCPU Platform to return unique congruences, i.e., special relations among integers, that ultimately are used to factorize the large biprime. Note that using these congruences to factor is a well-known and standard method [84] and the most effective known as of today [72]. A very interesting fact is that the sieve method has seen its computational complexity decrease because of smarter and smarter ways to compute congruences [85]. In fact, the basic Kraitchik's method has complexity $O(\exp(\sqrt{2\log n \log\log n})$ [85]. But by making the search for the congruences a bit

smarter we have the quadratic sieve, which has complexity $O(\exp(\sqrt{\log n \log\log n})$ [85]. Finally, employing the smartest sieve method, the GNFS, we arrive at the modern complexity of $O(\exp((64/9\log n)^{1/3}(\log\log n)^{2/3})$ [85]. Therefore, it would not be surprising if some new way to compute congruences will further decrease the complexity.

To use the MEMCPU Platform to find these congruences, we have developed an integer linear programming (ILP) formulation whose solutions are congruences. Therefore, it is enough to solve the ILP multiple times and then factorize the biprime using the same procedure as the standard sieve methods[70, 84]. An example of factorization using GNFS is reported in Figure 6 (a more detailed discussion is in the next subsection).

In this case, the problem we discuss is not unique, but it is the one that worked best during this project. More tests and other models/equations for finding useful congruences can be found in the biweekly reports from the SBIR [83]. The basic equation for congruences we consider is similar to the one used by Schnorr [86]. Although we use a similar form for the congruences, our approach to compute them is completely different and does not rely on any of the methods mentioned in that article. However it is useful as reference for the specific form of congruence we discuss here. The congruence reads:

$$x + kn = y, \tag{3}$$

where n is the integer to be factorized, k is a positive integer (in [86] k is required to be $b-$smooth, however it is not necessary in our case) and x and y are required to be $b-$smooth numbers, i.e., integers whose prime factors are all smaller than or equal to b. We denote with $\pi(b)$ the prime counting function, i.e., the number of primes smaller than or equal to b. Also, for any $m \leq \pi(b)$ let us assume we collected $m+1$ independent congruences containing exactly m unique primes in all x_j and y_j with $j = 1, .., m+1$. The set of these m primes is called the factor base F. By expanding $x_j = \prod_{p\in F} p^{\alpha_p,j}$ and $y_j = \prod_{p\in F} p^{\beta_p,j}$ in the factor base F, without loss of generality, we can multiply each congruence by $s_j = \prod_{p\in F} p^{\alpha_p,j \mod 2}$. The new congruences read

$$\begin{aligned} &s_j x_j + s_j k_j n = \\ &= x_j'^2 + k_j' n = s_j y_j = y_j' \end{aligned} \tag{4}$$

Therefore we are now in a situation equivalent to the quadratic form used in the quadratic sieve. We can therefore associate to each derived congruence the vector $v_j = \{\beta_p, j + (\alpha_p, j \mod 2)\}$ of the exponents of the factor base F. We can then efficiently select a subset of these congruences using Gaussian elimination to solve $\sum_j e_j v_j \mod 2 = \{0_p\}$, where e_j is a binary variable that is 1 if a congruence j is selected and 0 otherwise. Once the selection is evaluated, we can multiply side by side all congruences selected, and their product will produce a Fermat relation of the form $Y^2 - X^2 = Kn$. This can be efficiently used to find the factors of n by computing the $\gcd(Y - X, n)$.

Standard methods would search for smooth solutions of equation 3 employing sieving methods like the quadratic sieve or GNFS. However, by converting the problem in ILP we can directly require that the x and y are $b-$smooth. There are several ways to do this. The most generic one is requiring that both x and y are products of smaller and smaller integers, for example

$$x = \prod_k x^{(k)} \tag{5}$$

$$y = \prod_k y^{(k)}. \tag{6}$$

To decide how many $x^{(k)}$ and $y^{(k)}$ we need to determine the probability that a solution is b-smooth, let us consider that, in ILP formulation, we will use their binary representation to encode the products as we discussed in section III B 1. Therefore, if we implement each $x^{(k)}$ and $y^{(k)}$ using a certain number of bits, then we can determine what is the probability that a solution of the ILP version of the equation (3) is $b-$smooth. For example, if $b = 2^h$ and the length of each $x^{(k)}$ and $y^{(k)}$ in bits is exactly h, then the solution will be 100% $b-$smooth. If some of the $x^{(k)}$ and $y^{(k)}$ are larger than h, then the probability can be evaluated calling out the Dickman-de Bruijn function [87]. For example, if we leave all $x^{(k)}$ and $y^{(k)}$ of length h except one, let's say $x^{(1)}$, with length rh for some $r \geq 1$, then the probability that a solution is $b-$smooth is given by $\rho(r)$, where ρ is the Dickman-de Bruijn function. For rough estimates, it can be approximated as $\rho(r) \approx r^{-r}$. If more than one of the $x^{(k)}$ and $y^{(k)}$ are larger than h, then we can use the probability of independent events, that, in this case, would be the product of all Dickman-de Bruijn functions, one for each $x^{(k)}$ or $y^{(k)}$, exceeding h.

If we keep the probability high enough, we then shift the complexity to finding the solutions of the ILP problem. So, if we have an efficient solver for these kind of ILP problems, then we can solve the factorization problem efficiently.

We chose the form of the congruence like equation (3) because it offers the possibility to be implemented in ILP in a few different ways and also allows us to make some manipulations that help in keeping the problem size compact. Even if we discuss some of these aspects in this work, it is not intended to be an exhaustive description.

A useful first manipulation is writing eq. (3) as

$$\bar{x}x + kn = y, \tag{7}$$

where $\bar{x}$ is a fixed number. For each $\bar{x}$ chosen appropriately, we have a different ILP problem which can play a similar role as the multiple polynomial method for the quadratic sieve [88], allowing, for example, for independent parallelization. However, fixing $\bar{x}$ facilitates running multiple problems in parallel in a better way.

For example, let us consider $b = 2^h$ and $kn < 2^H$. If $2^{H-h-r} \leq \bar{x} \leq 2^{H-h-r+1}$, then x can be set of length (in bits) $h+r$, and therefore, if r is small enough, we will not need to further break x using (5) providing a non trivial simplification of the problem. It is worth noting that, if this scheme is used, r and the length of k need to be chosen carefully to guarantee that, at least statistically, the problem (7) has solutions. An estimate of the number of $b-$smooth solutions of eq. (7) as a function of r and H can be easily done using Dickman-de Bruijn functions.

An interesting question is how do we choose $\bar{x}$? A convenient way can be either a preset $b-$smooth number or a square. However, there is a better choice to speed up the computation of the congruences. If we leave x and/or some of the $y^{(k)}$ in the equation (6) larger than $b = 2^h$, then it is probable to obtain some of the congruences with one or more factors larger than 2^h. Typically, if there is only one large factor, the congruence can be stored and then combined with other congruence having the same large factor found by chance. This is typically employed in the standard sieve methods [84] since finding these collisions is actually quite likely, as in the birthday paradox. However, we can do better than relying on chance. We can create problems where the large factors are included in $\bar{x}$ providing a nontrivial speed up. A deeper analysis of the speed up deriving from this method will be a subject of future work since here we are not leveraging it yet.

It is worthwhile discussing some details of the ILP implementation of equation (6). If there are just two terms in the (6), then the implementation of the product would be identical to the implementation discussed in section III B 1. However, if there are more than two terms, we should implement it as a hierarchy of products. For example, we can use the iterative scheme

$$z^{(k+1)} = z^{(k)} y^{(k+1)} \tag{8}$$

with $z^{(0)} = y^{(0)}$. This implies $z^{(K)} = \prod_{k=0}^{K} y^{(k)}$ and each of the $z^{(k)}$ is a product of two integers that can be implemented as in section III B 1. This is not the only way to group the products, however we found that this provided better convergence with the MEMCPU Platform. Another important aspect to notice is that the sum of all lengths in bit of the $y^{(k)}$ must exceed $H = \log_2(\sup(kn))$. The more it exceeds H, the more solutions the ILP has (with no impact to the probability of smoothness). However, it also makes the ILP larger. Therefore, a trade-off between the ILP size and the solutions of the problems should be found. We did not do a deeper analysis to asses the optimal sum of all lengths in bit of the $y^{(k)}$ – it will be probably matter of future work – however we found that 20% larger than H provides a good trade-off in the range of the size we tested.

Finally, there is another way to express the equation (6) in ILP. In this case, we can simply implement (6) as a set of equations like

$$\begin{aligned} \bar{x}x + kn &= w^{(0)} y^{(0)} \\ &\vdots \\ \bar{x}x + kn &= w^{(K)} y^{(K)} \end{aligned} \tag{9}$$

coupled with the constraint $y^{(k)} \neq y^{(k')}$ for each $k \neq k'$. This condition can be achieved in ILP in many ways. However, tests with MEMCPU Platform have shown that, with proper MEMCPU design parameters, the SOG network naturally converges to a solution with negligible likelihood to have collisions among $y^{(k')}$ if we initialise the circuit with random initial conditions. It is worth noting that this approach might need more splits $y^{(k)}$ than (8) since even two different $y^{(k)}$ can have primes in common, therefore making it not necessarily true that $\bar{x}x + kn = \prod_k y^{(k)}$. Based on our tests discussed in the next section, we found that if the sum of all lengths in bits of the $y^{(k)}$ is 30% larger than H and $\log_2(\sup(w^{(k)})) + \log_2(\sup(y^{(k)})) \leq H + 2$ then the probability that $\bar{x}x + kn$ has factors other than the ones in the $y^{(k)}$ is negligible when we use the MEMCPU Platform.

Benchmark and Scaling results

We used the same benchmark described in the section III B 1. However, this time we were able to push to larger problems, reaching up to 300-bit problems. In this test, we implemented the ILP model of equation 9 and solved it to find $b-$smooth congruences with $b = 2^{21}$, statistically leveraging large factors, as normally done in standard sieve methods, with 27-bit cutoff . These values were kept independent of the problem size. The time reported in Figure 6 is the time to find $\pi(b)$ independent congruences with $\pi(b)$ the prime counting function. As shown in Figure 6a, using a best-in-class ILP solver like Gurobi, this formulation does not reduce complexity (as expected) since it is not efficient in solving this type of ILP problem and finding congruences. In fact, Gurobi's best fit for its scaling is still a 20^{th}-degree polynomial (the last point timed out after 8 hours without returning any congruence), which is expected since the ILP has similar structures as in the direct model.

We also compared against GNFS implemented in the Msieve software [89]. This implementation starts following the asymptotic GNFS sub-exponential scaling at around 150-bit factorization, as shown in Figure 6a. Msieve software was running on an Intel i9-11950H with 128GB RAM.

Our MEMCPU Platform proved to be very effective in finding congruences at fixed smoothness. In fact, its scaling is seen to follow a 4^{th}-degree polynomial for the emulation on CPUs. We were able to achieve a much lower scaling (2^{nd}-degree polynomial fitting) using our emulation on GPUs, as shown in Figure 6. The change in slope comes from the fact that the GPU implementation

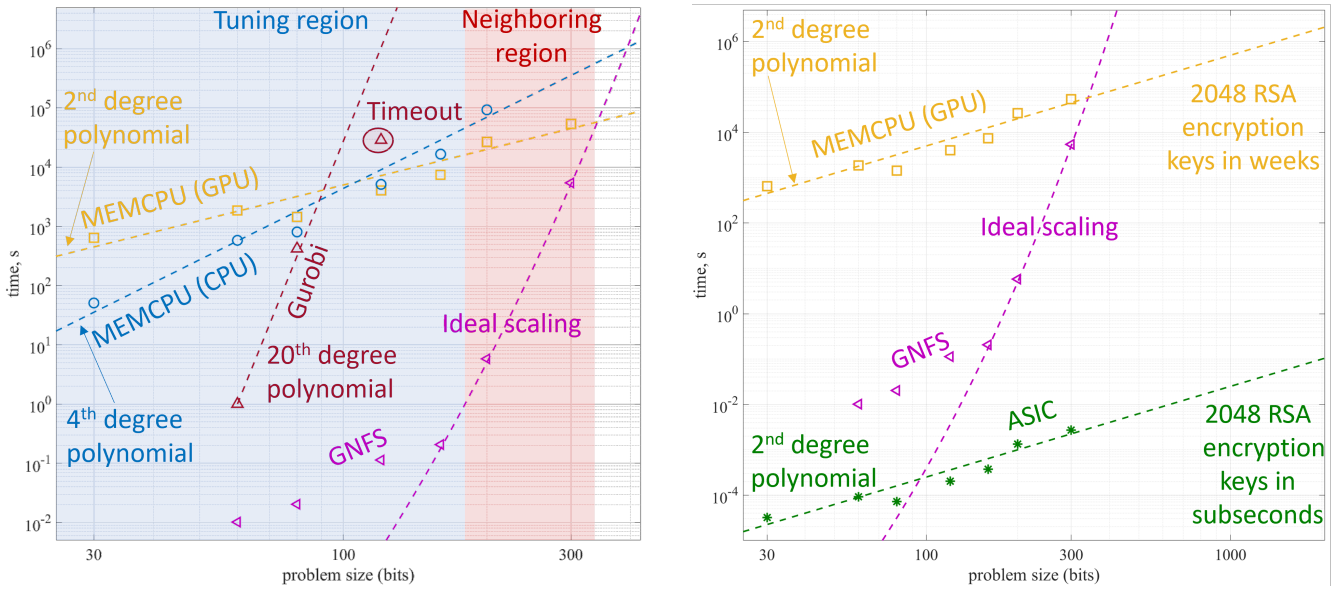


FIG. 6. (a) Comparison of scaling results for factorization problems up to 300 bits using the MEMCPU Platform and Gurobi solver to find congruences at fixed smoothness independent of the problem size, and GNFS implemented in Msieve software. The MEMCPU Platform exhibits efficient scaling, with a 4th-degree polynomial fit for CPU emulation and a 2nd-degree polynomial fit for GPU emulation. The Tuning region is where the design parameters were fine-tuned for each size, while the Neighboring region was able to use the design parameters of smaller problem sizes, which, however, maintained about the same performance. Gurobi timed out if pushed to larger problem sizes and has a 20th-degree polynomial fit like we had with the direct model. (b) The ASIC realization of the MEMCPU Platform shows a scaling similar to the GPU emulator but the extrapolation points to solving 2048-bit factorization problems sub-second. (This plot differs somewhat from an earlier version which used an older ILP formulation.)

of the MEMCPU Platform can efficiently distribute the calculation over the GPU cores. For problems with up to a few hundred thousand non-zeros in the ILP constraint matrix, the GPU implementation is almost independent of the size of the problem because it can distribute an entire time step simulation on the cores. However, once the problem size crosses the threshold of a few hundred thousand non-zeros in the ILP constraint matrix, the simulation time of each time step of the circuit starts growing with the size of the problem. For this work we used Virtual Machines with 8-NVIDIA V100 GPUs on Google Cloud to run the GPU tests. Even if the current implementation would efficiently distribute on the GPU cores for problem sizes up to several hundreds of bits, this threshold can be increased using either larger GPUs like Nvidia A100 or H100, or other distributed architectures.

We fully tuned the problem sizes up to 150 bits (Tuning region of figure 6a) and partially fine-tuned the ones at 200 bits. We stopped our tests at 300 bit problems because the limited resources from the SBIR did not allow us more cloud time to further tune or resources to finish upgrades to our CAD tools described in section IV D, which would enhance the tuning efficiency. In Figure 6b, the scaling from the converging region is extrapolated up to 2048 bits. The timing for the ASIC realization of the MEMCPU Circuit is also reported. The ASIC timing has been estimated as in the previous section since the MEMCPU Platform, being a circuit emulator, returns the full dynamics of the circuit, including the simulated run time. It is worth noting that the forecast for the ASIC shows the possibility of solving a 2048-bit factorization problem, in sub-second time once the tuning is extended. Further note that the MEMCPU Platform running on GPUs shows the potential to factor 2048-bits in a reasonable time without the ASIC if the proper tuning is extended.

IV. CIRCUIT DESIGN PROCEDURE

The design of the SOGs for factorization involves several aspects, from the design of the gates and circuit topology to the parameter tuning which establishes the correct behavior of the circuit. During this SBIR, we also added extra features to the MEMCPU Platform that allowed us to enhance the design of our circuit. In this section, we will go through the most significant and impactful functionality. It is worth reiterating that the MEMCPU Platform is essentially a CAD tool to design the circuit that can also be used directly as an ILP solver. Therefore, the design with our CAD tool is also the initial step of the ASIC development.

A. Circuit architecture and layout

The MEMCPU Platform functionality we discuss here implements the ILP problem of the quadratic form

$$(x + \lfloor\sqrt{n}\rfloor)^2 = yz \tag{10}$$

with x, y and z unknown integers. This equation is general enough to cover all aspects of the SOG design for all other ILP models discussed in this work. Figure 7a shows a sketch of the MEMCPU Circuit layout. The bit lines are vertical interconnects that traverse the circuit. For example, the light green band above x indicates the interconnects, i.e., the bit lines, for the binary coefficients x_j of x. The light blue regions indicate the areas containing gates. Figure 7b shows the types of gates in each region. The first two regions implement the bitwise product of two integers, as in ILP equation 1. The most natural way to implement this is using Self-Organizing AND gates as depicted in Fig. 7b. The bottom part of the circuit implements equations of the form shown in eq. 2, with additional terms to accurately represent the ILP problem 3. Equalities are implemented using a pair of SOAGs, splitting an equality into two inequalities with opposite directions. From an electronic circuit perspective, these gates are compact and possible to implement using CMOS technology since their function is a linear combination of binary variables with coefficients of either 1 or a power of 2. Further details about the design are not shared as they are partly trade secrets and partly patent pending [90].

B. Point Dissipativeness

Before proceeding further with the description of the circuit design, it is important to have a better understanding of the working principles of the MEMCPU Circuit. Even if the idea of the SOGs can be perceived as relatively simple at first, the details present the challenges. This is how we make sure that the MEMCPU Circuit, i.e., a network of interconnected SOGs, converges to a state where all gates are satisfied (i.e., the solution of the problem), and does it efficiently (i.e., as quickly as possible). There are many ways to achieve these goals. Our approach is two-fold. First, we ensure convergence by looking at the point dissipativeness, a property of dynamical systems [52, 74, 91]. Second, we try to address the convergence efficiency by leveraging the criticality hypothesis [92] which will be discussed in the next section.

Point dissipativeness is a property of some dynamical systems [91] that, when translated into electronic circuit terminology, implies that the systems maintain locally bounded energy [52] and have certain stability properties [52, 91]. The most important property is that the equilibria form a global attractor set [52], i.e., any system trajectory, for any initial condition, will end up into one of these equilibria. More details about this are out of the scope of this work but a great reference on the topic is [91]. However, it is important to notice that, if this property is satisfied, then the network of SOGs will certainly reach convergence [52] (that is, an equilibrium point, if it exists, in which all gates are satisfied [52]) and no spurious cycles [93] or chaos [94] can emerge. While the point dissipativeness is mathematically well defined [91], it is hard to prove that a dynamical system satisfies it. In the case of SOGs, an isolated SOG has a design that is easy to prove to be point dissipative [52, 74]. However, once connected in a network, it becomes challenging to prove the same for the whole system as point dissipativeness is a global property of dynamical systems [91]. Nonetheless, it can be proved that, if a circuit made of interconnected point dissipative SOGs converges to its equilibrium point for any possible initial condition, then the entire network is point dissipative [91]. This last statement may seem abstract, but it can be used to verify numerically that a MEMCPU Circuit design is point dissipative. In Figure 8 for example, two different design parameters for the SOGs are tested. The circuit tested in this case is for the direct factorization of a 30-bit factorization problem. In the charts, the distribution of the time to solution is shown (i.e., to the equilibrium point for the entire circuit) for random initial conditions of the circuit. We tested more than 100,000 initial conditions to produce these histograms. The simulations were stopped only if the circuit converged, and for both designs, 100% of the simulations converged. This can be interpreted as a numerical test to prove point dissipativeness. On the other hand, Figure 11c shows two designs for the same problem, with a more advanced circuit design that has much faster convergence. While Design 1 converges for 100% of the initial conditions, Design 2 only converges for 42% of the initial conditions, showing that the latter design is not point dissipative. A closer look at the distributions in Figures 8 and 11c shows that the point dissipative designs fit very well with a gamma distribution (dashed curves). We have found that this is common for all point dissipative designs and for any problems we have tested so far. However, if point dissipativeness is not satisfied, as demonstrated by Design 2 in Figure 11c, then the distribution largely deviates from the gamma distribution. We have found this scenario to be quite general, but we do not yet have a robust theory that explains it, although we are working on it.

This point dissipativeness result is not at all trivial. In fact, it shows that a large network of asynchronous, autonomous, active, coupled electronic circuits can always find the trajectory to an equilibrium point, regardless of the initial conditions. This can ultimately be interpreted as the SOGs working collectively to find a common equilibrium. This is exactly what underlies the singular capabilities of the technology. Also, this apparently general property (the time-to-solution distribution following a gamma distribution if the design is point dissipative) has many practical implications. For example, it can be leveraged to efficiently parallelize multiple initial condi-

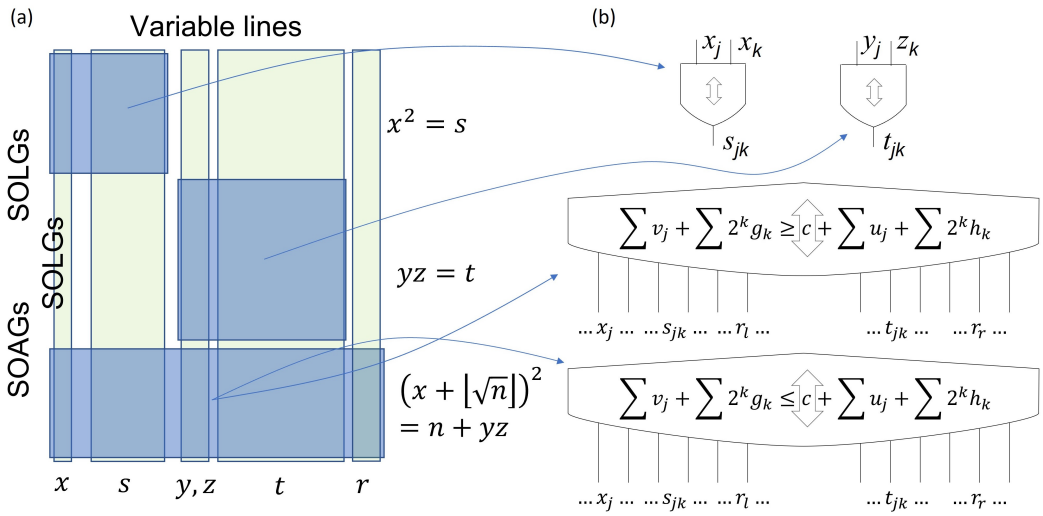


FIG. 7. (a) Sketch of the MEMCPU Circuit layout implementing the ILP problem (10), with vertical bit lines and gate-containing regions. (b) Types of gates in each region. The first two regions implement the bitwise product of two integers using Self-Organizing AND gates. The bottom part of the circuit implements equations with additional terms to accurately represent the ILP problem. Equalities are implemented using a pair of SOAGs.

tions and for an efficient restart process that accelerates convergence to the solution.

C. Criticality

Point dissipativeness is not the only crucial property that we need to guarantee in our design. The speed of convergence is also crucial. A classical approach to study the speed of convergence for point dissipative systems would start from the analysis of the convergence rate of the stable manifolds employing the classical stability theory methods [95] applied to point dissipative dynamical

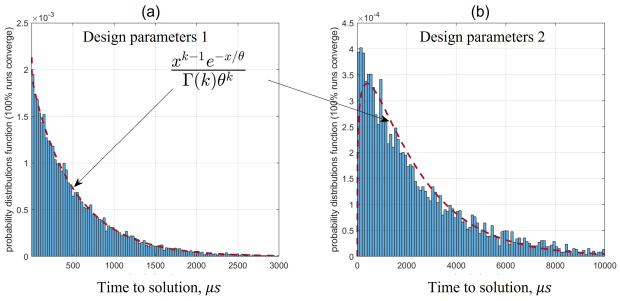


FIG. 8. Distribution of time to solution for two different SOG designs in a 30-bit factorization problem, direct model. Both distributions fit well with a gamma distribution. Design 2 (b) is more than 4 times slower to converge than Design 1 (a). The peak of Distribution 1 is almost at time 0, while Distribution 2 has a peak shifted at around 500µs. This suggests that point dissipativeness is not enough in practice and that there are designs where most initial conditions lead to an almost immediate collapse to the equilibrium point of the circuit.

systems [91]. For complex and large dynamical systems like ours, this approach wouldn't provide a useful outcome, both in practice and in theory, beyond some very general conclusions.

A less standard approach is to assume the so-called critical hypothesis [92, 96]. In short, it assumes that the most efficient form of computation for a distributed system occurs at the edge of chaos, where a 2^{nd} order-like order-disorder phase transition takes place [92, 96]. In this scenario, at the phase transition, the system enters a critical state where long-range correlations favor scale-free properties of the system [92, 96]. This mechanism is associated with optimal information flow through the system and is thought to occur in systems like the neural cortex, making brain information processing highly efficient [92]. By applying similar hypotheses to our circuit, we aim to unravel the mechanism for optimizing computational efficiency, which ultimately results in faster convergence to equilibria.

In the past, we have analyzed critical behavior by studying long-range correlations related to instantonic processes [76, 97], and some of the same researchers have also explored avalanche phenomena and critical branching processes [48, 98]. However, in this section, we would like to focus more on the communication aspect of the critical behaviors of memcomputing circuits. We believe it shows more promise for helping with circuit design and also provides a more practical understanding of the working principles governed by collective dynamics. The analysis reported here is not exhaustive and there is still work in progress, but it paves the way for more detailed studies and opens up new avenues for novel design techniques for SOGs.

Let's go back to Figure 8 and take a closer look at the distributions. Both distributions fit very well with

a gamma distribution (dashed curve in Figure 8) as discussed in the previous section. Here we discuss some other implications. We first notice that the average time to solution for the two different designs in Figure 8, is very different. On average, Design 2 is more than 4 times slower to converge than Design 1. Additionally, the peak of Distribution 1 is almost at time 0, while Distribution 2 has a peak shifted at around 500µs. This suggests that:

(a) Even if the system is point dissipative, we can have very different speeds of convergence depending on the design.

(b) There are designs where most of the initial conditions lead to an almost immediate collapse into the equilibrium point of the circuit.

Implication a) tells us that point dissipativeness is not enough in practice. We need to find, among all point dissipative designs, those that have faster convergence. Implication b), on the other hand, highlights a much more subtle and intriguing aspect of the system. To better understand this, let's keep in mind that this particular MEMCPU Circuit design has around 1,000 variables and a few thousand interconnected SOGs. If we then consider all electronic elements that are in the gates, this circuit has tens of thousands of state variables. Therefore, we are dealing with an autonomous dynamical system moving in a space with a dimensionality of tens of thousands. The fact that such a large dynamical system can collapse into its equilibrium point in just a few time steps for most random initial conditions is not common or trivial, especially for systems that are networks of autonomous dynamical systems.

The reasons for this behavior can be found looking at the correlations that are established in the system and what they imply. In Figure 9 for example, we show the correlations between the threshold crossing of the voltage for two different pairs of circuit nodes and two different designs. The correlations are defined as $C_{jk}(\tau) = R_{jk}(\tau)/\sqrt{(R_{jj}(0)R_{kk}(0))}$ where $R_{jk} = \int f(v_j(t))f(v_k(t+\tau))dt$, v_j is the voltage at the bit line j and f is the threshold crossing function defined as

$$f(x) = \begin{cases} 1 & \text{if } \max(x(t-\Delta t), x(t)) > th \quad \wedge \\ & \min(x(t-\Delta t), x(t)) < th \\ 0 & \text{otherwise} \end{cases}$$

for a threshold $th \in R$. These correlations indicate whether two SOG terminal voltages are crossing the threshold (and therefore switching their binary state) due to communication established by the SOGs. The correlations in Figure 9 are for circuits emulated for 10^5 time steps. We have reported the raw correlations calculated using the convolution theorem and filtered correlations using standard procedures such as polynomial fitting and baselining to the noise level. Visualizing internal correlations is a powerful analysis tool for SOGs. Figure 9 highlights two different designs. The first design (the good one) was tested for point dissipativeness and has a very good convergence speed (Figure 9a and 9b). It is similar to the design in Figure 8a. The second design (the bad one) could not converge to equilibrium within the allotted simulation time, regardless of the initial conditions (Figure 9c and 9d). It is important to note that both designs are for a network of interconnected point dissipative SOGs. However, as discussed in the previous section, point dissipativeness does not necessarily extend to the network as a global property of the dynamical system. Looking at correlations, we see that the good design exhibits high correlations that persist for up to a few hundred time steps. In contrast, the bad design does not show any correlation above the noise level, demonstrating no useful communication among nodes. Another important aspect is that the correlations are quite symmetric (i.e., for positive and negative τ), indicating that the good design has no preferential flow of information within the circuit and communication is mutual. Although Figure 9 only shows correlations for two pairs of circuit nodes, we checked most of the node pairs and all showed similar behavior.

Figure 9a and 9c show correlations for pairs of bit lines that are both connected to at least one SOG. On the other hand, Figure 9b and 9d show correlations of bit lines that require at least one SOG to be traversed to go from one line to the other. For the MEMCPU Circuit layout depicted in section IV A, it is unlikely for more distant lines in terms of SOGs to be traversed since large

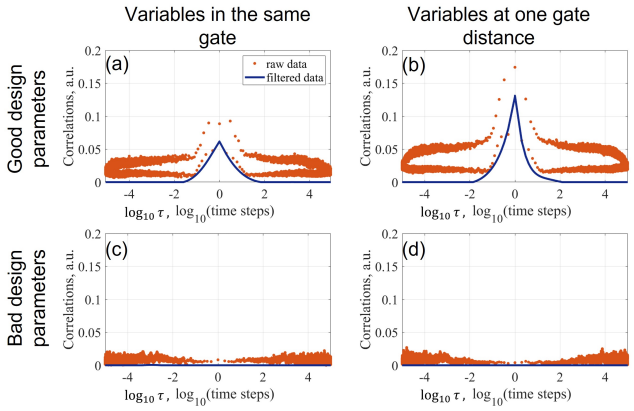


FIG. 9. Visualization of internal correlations for two different SOG designs. The first design ((a) and (b)) was point dissipative with a very good convergence speed, while the second design ((c) and (d)) did not converge to equilibrium within the allotted simulation time. The good design exhibits high correlations that persist for up to order-one-hundred time steps, while the bad design does not show any correlation above the noise level. Figures (a) and (c) show correlations for pairs of bit lines that are both connected to at least one SOG, while Figures (b) and (d) show correlations of bit lines that require at least one SOG to be traversed to go from one line to the other. For good designs, correlations extend beyond those internal to the gates and non-locally to the entire circuit, creating effective mutual communication among gates.

SOAGs, like the ones in Figure 7, are connected to many bit lines and the average distance in terms of SOGs to be traversed is 1. Therefore, this second node pair represents non-local correlations in terms of SOGs. Figure 9b shows that for good designs, correlations extend beyond those internal to the gates and non-locally to the entire circuit, creating effective mutual communication among gates.

In conclusion for this subsection, we can summarize that the convergence process is driven and accelerated by mutual communication among bit lines happening in and among the SOGs. This communication has non-local properties, as mutual correlations persist even if the bit lines are not directly electronically coupled through an SOG. The high persistent correlations, their mutual and non-local nature, are all fingerprints of critical behavior happening in the circuit. Our interpretation of critical behavior indicates that the circuit is able to have gates communicating at long distances, and the communication is mutual and balanced. Since we found these global features always for designs whose tests show point dissipativeness, we conclude that this criticality-driven communication is likely the mechanism that, once enhanced, accelerates the convergence to the equilibria. There is additional work in progress to quantify this mechanism and uncover critical parameters that can be used to enhance the design of our circuits.

D. Design Parameter tuning

An interesting theoretical question at this point is: What is the ideal design for SOGs and what would it imply? Answers to this question can be found in both classical stability theory of autonomous systems [95] and complexity and criticality theory [96]. However, this goes beyond the scope of this work. A more practical question is: How we can design a MEMCPU Circuit that approaches the ideal design. To this end, over the last few years, we have developed methods that are still being perfected and implemented within our in-house Computer-Aided-Design (CAD) platform. The design process can be summarized as follows:

(a) Each SOG is designed with a set of free parameters that can vary within a certain range. The design guarantees, within the parameter range, point dissipativeness of each gate if isolated. These parameters are typical design parameters for electronic components, ranging from resistances and capacitances to transistor model parameters.

(b) We input the ILP model into our CAD platform using standard formats such as .mps or .lp files. During the SBIR performance period, we upgraded the CAD platform to allow the input of extra information about the problem structure, as well. Specifically, subgroups of SOGs and SOG terminals can be demarcated as belonging to different groups that we call "families". This allows for the independent design of multiple families of components within the circuit.

(c) We use an internal general-purpose optimization method to explore the parameter space. It is a heavily modified parallel tempering Monte Carlo method [99] with the goal of quickly returning design parameters that could potentially make the entire circuit point dissipative and approach criticality. This method is implemented in our CAD platform and is used to explore the parameter space for the different SOG families.

(d) We further analyze the candidate designs from parallel tempering to filter out those that are point dissipative, have the best convergence, and exhibit the best generalizability to other problem instances.

(e) This process is repeated at increasing problem sizes to scale up to larger ones. Using a continuation algorithm to accelerate the design process, we use designs from the previous sizes as starting points for the larger ones.

As we mentioned, this process is still being perfected, even though it already provides reasonable results. To give an example of the challenges that we face, let us discuss one of the most important ones that were the subject of intense development during the SBIR. We would like the parallel tempering to return parameter designs that make the circuit point dissipative and have high convergence speed, which is highly critical. However, even though we are working on it, as of today we do not have a parallel tempering objective function that directly measures these two properties. So, we use an objective function that indirectly optimizes towards these two goals, which poses serious challenges. For example, the most basic idea would be to use some distance function defined for the SOGs that remain unsatisfied during the simulation. The simplest approach would be to just count the unsatisfied SOGs at each time step during the simulation and use the minimum attained. We can call this the number of "unsats". Even though this may seem like a good idea at first, let us describe the severe problems it generates.

The biggest issue arises from the fact that, since parallel tempering explores a large and complex space, it needs many iterations to reach a good region. An iteration is a short simulation to test the design parameters of each chain. The number of iterations varies depending on the number of SOGs families and the number of Monte Carlo chains we use, but it usually ranges between a few thousand and tens of thousands of iterations to achieve reasonable results. To give an idea of the size of the parameter space, let us consider the design in Figure 7. We can define several families of variables and SOGs. A basic classification would be 4 variable families ($x, [s,t], [y,z]$, and r) and 3 SOG families (the SOANDs for $x^2 = s$, the SOANDs for $yz = t$, and the SOAGs), which, it turns out, would translate into 121 free design parameters to

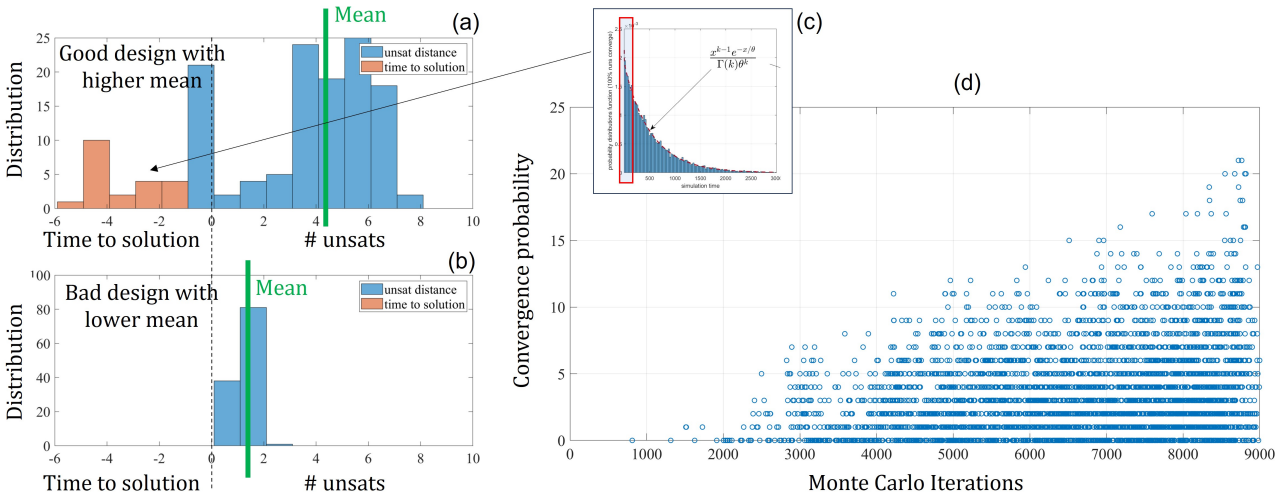


FIG. 10. Comparison of time to solution, unsats and convergence probability for different SOG designs. (a) Distribution of the number of unsats for the good design, which remains at high values before quickly reaching the solution. (b) Distribution for a bad design that can quickly arrive at a few unsats but never succeeds at converging. (c) Distribution of time to solution for a good design, with the part of the distribution that would be observed within a short simulation time highlighted by the red box. (d) A typical run of parallel tempering using an objective function (11) for a 38-bit problem direct model, with the probability of convergence for short simulations increasing as the number of iterations increases.

be optimized. The parameters are not generally independent (obviously, since being point dissipative and critical are global properties), so the parallel tempering must explore the entire space of 121 dimensions for continuous parameters. For a size like this, a good number of iterations for our parallel tempering would be a few tens of thousands with at least 180 chains divided into 6 independent parallel tempering routines of 30 chains each. This implies that long circuit simulations would be prohibitive, and therefore we test parameter designs with just short simulations.

Since we have a distribution of time to solution depending on the initial conditions, "short simulations" means that even good designs may not have enough time to arrive at the solution during an iteration of parallel tempering. For example, let's consider again the parameter set in Figure 8a, which is also reported in Figure 10c for convenience. Typically, for factorization, we simulate about 100 µs of circuit dynamics. In Figure 10c, the part of the distribution that we would observe within such short time is highlighted by the red box. Many initial conditions would end with a positive number of unsats. A distribution of the number of unsats can be found in Figure 10a. It is interesting to note that this parameter (already known to be point dissipative and sufficiently critical) has several unsatisfied clauses during most of the simulation. It remains at high values before finally quickly reaching the solution, collapsing into equilibrium. Turning to bad designs, there are many poor circuit designs that can quickly arrive at a few unsats but never succeed at converging. This is seen for example for the circuit design that produced the distribution in Figure 10b. These are designs in which some elements in the circuit may not have enough energy to trigger the correct dynamics, resulting in metastable states that screen the equilibrium points. Unfortunately, this means that using unsats (the number of unsatisfied SOGs) can lead parallel tempering into the wrong region, resulting in designs that are unlikely to be point dissipative.

We found a mitigation to this problem by using the following objective function for parallel tempering,

$$\text{obj} = \mu - \sqrt{\sigma^2} + \sqrt[3]{m_3}, \tag{11}$$

where $\mu = \langle \text{score} \rangle$, $\sigma^2 = \langle (\text{score} - \mu)^2 \rangle$, $m_3 = \langle (\text{score} - \mu)^3 \rangle$, and score is the value of any distance function that quantifies how unsatisfied the SOGs are. The objective function of Eq. 11 corrects the mean of the score (which, as explained earlier, can be very misleading) with the standard deviation and the third-order moment to statistically quantify the probability of reaching a solution within a short test. However, implementing this objective function in parallel tempering can be challenging because it requires an estimate of the aforementioned expectation values. Typically, we test a design only once and then accept or reject it based on the Metropolis-Hastings algorithm [99] before moving on. To overcome this issue, we estimate 11 by using previously tested design parameters close to the ones under testing on the current iteration. Using statistical inference, this method provides reasonable results, especially if the probability of convergence for short runs and good designs is above 5-10%. Figure 10d shows a typical run of our parallel tempering using 11 as the objective function for a 38-bit problem, for the direct factorization method, and a total of 82 free parameters. For designs that converged during parallel tempering, we also post-processed

the probability of convergence for short simulations by rerunning them with 100 different initial conditions and timing out at 100 µs of circuit dynamics. Figure 10d shows that as the number of iterations increases, this probability also increases, indicating that parallel tempering is converging to design parameter regions where the circuit designs are increasingly efficient in converging to their equilibria and thus solving the factorization problem.

Unfortunately, even this method is not perfect, as parallel tempering can still be attracted to regions of the design parameter space where the circuit is not point dissipative. An example is shown in Figure 11. Figure 11a shows the sampling distribution of our parallel tempering using 11 as the objective function for a 30-bit problem, the direct factorization method, and a total of 82 free parameters, with 180 chains divided into 6 independent parallel tempering runs, each running for 12,000 iterations. Each subplot reports the histogram of the trial samples (light blue), the samples that converged to equilibrium during the short run (light orange), and the post-processed probability of convergence for short runs (100 µs of circuit dynamics) for the design samples that converged during parallel tempering. To understand the issue, let us concentrate on just one free parameter, parameter 49, as shown in Figure 11b. Here, we can see that the projection of the distribution shows a peak at around 2. If we test some sample designs from that peak, we find that they are point dissipative and have very good convergence rates and high and symmetric correlations, indicating critical behavior (Figure 11c). However, the parallel tempering did not spend much time in that region, as the sample distribution suggests. On the other hand, if we test a parameter from the peak of the sample distribution, i.e., from the region where parallel tempering was most attracted, we see that the design associated with it is not point dissipative, showing only 42% of maximum convergence as reported in Figure 11c and also discussed in Section IV B. This demonstrates that even though we use 11 to get some good designs that provide the scaling reported in Section III B, the parallel tempering objective is still not perfect, and more work needs to be done to arrive quicker at ideal designs. We are currently exploring the use of an upgraded objective based on a critical parameter related to correlations that can be used to correct 11 and drive parallel tempering into regions where designs exhibit high correlations and are also point dissipative.

Although the effort to tune is currently somewhat onerous, we usually tune one problem instance per size, and then the design obtained is used for any other problem instances of the same size. For this reason, the total amount of compute for tuning was not quantified here and the computing burden was also mitigated by leveraging old gate designs to find new ones using continuation techniques. Such techniques that leverage design parameters from smaller sizes on larger sizes were seen to be very effective for tuning efficiently as size increases. The amount of tuning required is expected to continue with the same trend and at this point we do not see any roadblock that cannot be overcome with proper R&D and compute resources.

Finally, to summarize the effect of design parameter tuning, in Figure 12 we illustrated how the effect of the tuning extends beyond the tuning region. If the problem we are dealing with roughly maintains a constant structure in terms of node voltage connectivity, proportion of types of gates, terminals per gate, etc, then by tuning for only small problem sizes, the obtained design parameters will maintain the same circuit and convergence properties (point dissipativeness and criticality) at higher problem sizes. Therefore what we defined as the neighbouring region extends to any size. On the contrary, if the structure changes with the problem size, then the neighbouring region will extend only partially and to maintain the same circuit and convergence properties, we have to tune at higher and higher sizes as depicted in Figure 12. This is also remarked in sections III B 1 and III B 2.

E. Forecasted results from ASIC implementation

In the last few sections, we introduced and discussed the working principles of the MEMCPU Circuit and how our in-house CAD tool is used to optimize circuit design. Currently, our CAD tool implements generic compact models for electronic devices. The MEMCPU Platform results correspond to the time to perform the simulation of these generic components. How much time passes *within* the simulation provides the forecasted ASIC results. This is only an estimate of the future ASIC results, though, because the generic components (*e.g.* resistors, capacitors, transistors) currently simulated must be substituted for foundry models, making any necessary accommodations, before layout generation and manufacturing.

V. CONCLUSION

In this SBIR, we explored both direct and congruence approaches for prime factorization. We found that the MEMCPU Platform achieved best acceleration using the congruence method. Our variant of the sieve method is based on the same principles as the quadratic sieve or GNFS methods. However, it accelerates the solution by using the MEMCPU Platform to return unique congruences, i.e., special relations among integers, that are ultimately used to factorize large biprimes (numbers that have exactly two prime factors) [84].

To use the MEMCPU Platform as if it was a sieve machine to find these congruences, we have developed an integer linear programming (ILP) formulation whose solutions are congruences. Therefore, it is sufficient to solve the ILP multiple times to factorize the biprime. Sieve-based methods work similarly, but conventional sieving

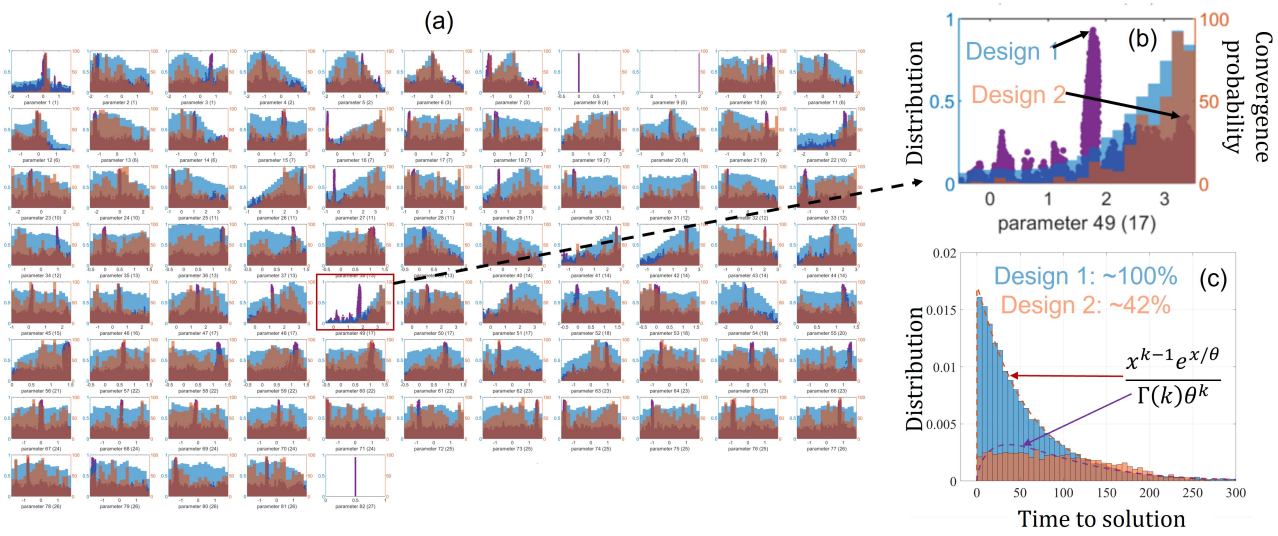


FIG. 11. Distributions of the parallel tempering iterations in the parameter space.(a) Sampling distribution of parallel tempering using 11 as the objective function. (b) Distribution for parameter 49. The design 1 parameters come from the peak at 2 and the design 2 parameters are taken from the edge as shown. (c) Convergence rates for sample designs from the peak of the distribution and from the region where parallel tempering was most attracted show very different behaviors.

has super-polynomial complexity [70, 84], as shown in Figure 6. Figure 6 also demonstrates that a best-in-class ILP solver, like Gurobi, does not reduce the complexity (as expected) since it is not efficient in solving the ILP problem to find congruences. However, a properly designed memcomputing circuit provides congruences very efficiently at a rate that is well described by a 2^{nd} order polynomial fit as a function of the problem size, as reported in Figure 6. Our approach also allows us to control the smoothness of the congruences, such that we can keep the number of congruences bounded with the size of the problem, as discussed in Section III B 2. This is not the optimal choice for standard sieve methods where they need an optimal trade off between smoothness and operations to find smooth congruences leading to their well known sub-exponential scaling.

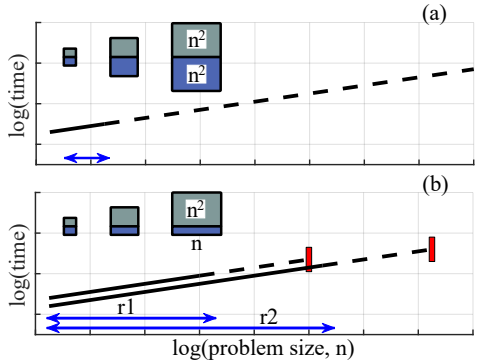


FIG. 12. Schematic illustrating that problems with size invariant structure can be tuned at a single problem size (solid line, tuning region), expecting to use the tuned parameters at all sizes (dashed line, neighboring region) (a). The inset depicts a circuit layout with two types of gates that grow at the same rate. In contrast, the structure of factorization changes with size (b), *e.g.* changing the ratio of product gates to sum gates. Tuning on example problems of a small range (r1) of sizes (solid line, tuning region) is seen to produce convergence not only for other problems in the range, but also up to a size somewhat larger than r1 (dashed line, neighboring region). Tuning further (r2) is seen to continue the scaling (red bar).

To understand the challenges, a quick review of the MEMCPU Platform was provided. The MEMCPU Platform emulates a circuit made of Self-Organizing Gates (SOGs) [52, 74]. Each SOG is an electronic component whose voltages at the terminals encode variables of the problem and the SOG drives these voltages to satisfy a relation among the variables [52, 74]. We call it "self-organizing" because it drives voltages autonomously via feedback loops and, once the SOG satisfies the feedback relation, it shuts down. When SOGs are connected via their shared terminals, the feedback from each SOG is propagated through the entire circuit. Hence, the circuit works in unison to satisfy all SOGs at once to minimize the feedback [52, 74]. However, it is important that energy balance and signal propagation properties be satisfied in order for the circuit to work properly [48, 76]. We have provided specific details about these design aspects through our analysis of point dissipativeness and criticality, which characterize the working principle of the MEMCPU Circuit. Additionally, we have discussed how we use and tune design parameters to ensure these properties.

During the SBIR, we were able to find these parameters for designs that can handle up to 300 bit factorization problems. It is worth noting that these parameters

are size dependent and not problem instance dependent. Continuing further requires finding parameters that scale this forward using continuation methods as discussed in this report.

ACKNOWLEDGMENTS

We gratefully acknowledge the financial support provided by the USAF Intelligence group through the SBIR Phase II, under Contract number FA864922P1131. Our sincere thanks to Dr. Massimiliano Di Ventra for his valuable insights and for reviewing this manuscript. We also extend our gratitude to Dr. Sergio Decherchi for his stimulating discussions.

Appendix A: Industrial applications

1. Oil and Gas Maritime Delivery Scheduling

Here is an example application of a complex problem in the Oil & Gas industry [55]. This work involved developing an optimized 30-day schedule for delivering 3500+ goods from a port in Louisiana to, from, and between offshore oil platforms in the Gulf of Mexico. The corporation employed several ships with various capacities for different types of goods. In addition, the ships also deliver fuel to the platforms and must ensure they do not run out of fuel. The goal was to provide a highly optimized shipping schedule while minimizing the number of ships and prioritizing slow over fast ships, since fast ships have limited capacity and are much more expensive to run. This type of scheduling problem in its full size is usually not amenable to classical optimization techniques, so companies manually create schedules based on simple operational rules, such as a first-come, first-served approach.

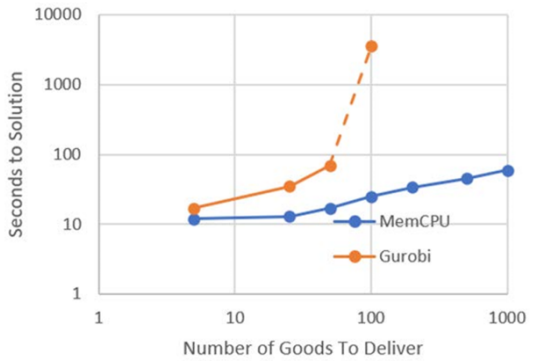


FIG. 13. Performance comparison of the MEMCPU Platform and Gurobi in solving an ILP problem for scheduling goods delivery in the Oil & Gas industry, showing linear scaling for the MEMCPU Platform and exponential scaling for Gurobi as the number of goods increased on a log-log scale.

This problem can be cast as ILP, transforming all scheduling constraints into linear inequalities invoking binary, integer, and continuous variables. The resulting problem is quite heterogeneous and complex but includes every aspect of the problem without approximations. Addressing the 40 different types of constraints results in hundreds of thousands of linear inequalities and hundreds of thousands of variables of various types. The resulting ILP is extremely hard, but its optimal solution will eliminate the inefficiencies of the first-come, first-served approach. The MEMCPU Platform and Gurobi (Fig. 13) were evaluated side-by-side. To perform a scaling analysis, we created multiple problems with increasing numbers of goods to be distributed. Gurobi, as expected, showed rapid exponential scaling, demonstrating the combinatorial hardness of the problem. At 100 goods, Gurobi's computational time exceeded 8 hours to find any feasible solution to the problem. The MEMCPU Platform showed linear scaling as the number of goods increased (Fig. 13). The MEMCPU Platform could be run over the full problem (3624 goods over 30 days).

The resulting schedule provided the wide array of benefits listed in (Table I) These optimizations should then deliver the following benefits:

- $0.5M monthly ($6M annual) reduction in ship lease payments.
- $1M monthly ($12M annual) reduction in fuel costs.
- 18kt reduction in greenhouse gases.
- Reduction in ship maintenance costs, crew costs, overtime, etc.
- These savings can then be replicated worldwide.

This work demonstrated that the MEMCPU Platform can efficiently solve combinatorial optimization problems considered intractable for today's best-in-class solutions while delivering significant efficiency improvements.

2. Aircraft, Passenger, Cargo, and Crew Scheduling

MemComputing has addressed several commercial aircraft scheduling problems. The Defense Innovation Unit (DIU) challenged us with an airlift scheduling problem specifically for the military [47]. They provided simulated airlift data representing worldwide military aircraft, passenger, cargo, and crew scheduling. The problem statement is quite complex, and details can be found in [47]. However, it can be summarized as follows.

- **Airlift requests**: There is a collection of airlift requests. Each of these requests a group of passengers and/or cargo. Each request can be fulfilled using multiple aircrafts distributing passengers and cargo. Each airlift has a priority code, departure

TABLE I. Results of using the MEMCPU Platform to optimize the delivery schedule, showing a decrease in the number of required ships, an increase in goods delivered by slow ships, a reduction in goods required to be delivered by expensive fast ships, and a decrease in the total number of transits, resulting in significant cost savings and reduction in greenhouse gases.

	# Slow Ships	# Transits	# Goods	# Fast Ships	# Transits	# Goods
Actual	12	176	2899	2	32	725
MEMCPU	7	92	3472	2	10	152
Improvement	41% less	48% less	20% less	same	68% less	79% less

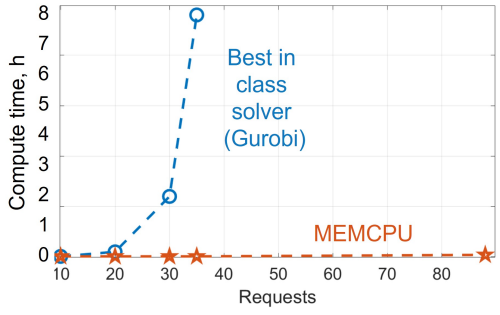
location, earliest pickup date, latest pickup date, latest drop-off date, destination, number of passengers, number of pallets of cargo and constraints on the type of aircraft.

- **Aircraft fleet**: The fleet consists of two types of aircraft. Each aircraft identifies the current airfield, the capacity for passengers and pallets of cargo, the cruising speed, and the maximum hours of continuous flight. There are also regulations that identify when, where and for how long an aircraft is unavailable for maintenance or other issues.

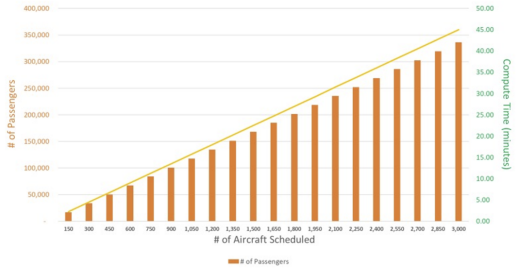
- **Crew**: There are regulations that must be met for the safety of the crew. These include the maximum number of hours the crew can be active, minimum down time after completing their active day and minimum time on the ground between legs.

- The goal is to create a schedule that successfully books all priority-1 requests with as many priority-2+ requests as possible while minimizing the overall flight time. Passengers and cargo may require multiple legs to reach their final destinations. These can be satisfied with the same aircraft or by switching aircraft as is most optimized.

We cast this problem in ILP format without the need for any approximations. In Fig. 14a, a performance comparison between the MEMCPU Platform and Gurobi, a best-in-class ILP solver, is shown when solving for varying numbers of airlift requests. Gurobi could solve small problems quickly, but its performance degraded exponentially as the number of airlift requests increased. On the other hand, the MEMCPU Platform solved all instances within minutes, exhibiting linear scaling. For example, it took approximately 1.5 minutes to calculate an optimized schedule for the full problem presented by the DIU that covered 2-weeks of airlift requests. This approach can scale much higher as depicted in Fig. 14b. There we tested scenarios of a scale like the meltdown of Southwest Airlines in December of 2022 [47, 100]. In summary, the work demonstrated that our approach can automate, optimize and help large commercial airlines recover quickly from major flight interruptions that can be brought on by large weather fronts. Savings in fuel and efficiency from fast, highly optimized scheduling can exceed hundreds of millions annually for these companies as well the DOD [47, 101].



(a)



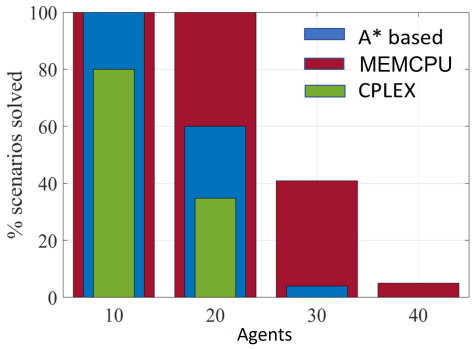
(b)

FIG. 14. (a) Performance comparison between the MEMCPU Platform and Gurobi in solving an ILP problem for military airlift scheduling, showing linear scaling for the MEMCPU Platform and exponential scaling for Gurobi as the number of airlift requests increased. (b) Results of using the MEMCPU Platform to optimize airlift schedules, demonstrating the ability to automate, optimize, and help large commercial airlines recover quickly from major flight interruptions, resulting in significant cost savings in fuel and efficiency.

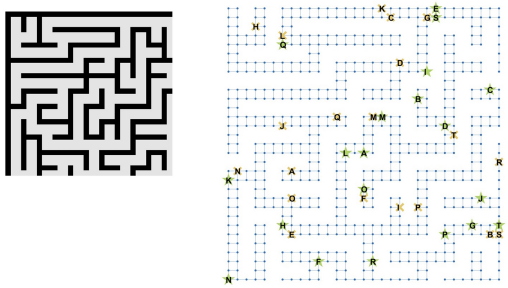
3. Drone Swarm Optimization

Lockheed Martin presented us a head-to-head benchmark study comparing the MEMCPU Platform vs. the best-in-class solution used to find the optimal path for a swarm of drones moving in a complex environment like a maze.

Problem Statement: A set of agents must move in an environment with obstacles, like a maze. Each agent has its own mission where it must travel from its starting point to its ending point. The goal is to find the optimal path for all agents that avoids deadlocks, collisions, or other interference [56]. This is a well-known problem in literature and goes under the name of Multi Agent Path



(a)



(b)

FIG. 15. (a) Results of a head-to-head benchmark study comparing the MEMCPU Platform, CPLEX, and an in-house method based on the A* algorithm in solving a Multi Agent Path Finding (MAPF) problem for a swarm of drones moving in a complex environment, showing that the MEMCPU Platform provided solutions to many more scenarios as the number of agents increased. (b) The maze used in the challenge (left) and the graph representation of the maze, with symbols representing the starting and ending points for a set of agents (right).

Finding (MAPF) problem [56]. The MAPF problem is known to be exponentially difficult with the number of agents, and the best-in-class solver used today is based on the A* optimization algorithm [56].

The head-to-head challenge that Lockheed Martin presented used the maze from Fig. 15b (left). The maze was represented in the form of a graph (Fig. 15b, right) so that trajectories of agents are described by indicating when they transit the nodes, and starting and ending points for all agents are constrained. Using graph theory, we developed an ILP formulation representing the MAPF problem. The ILP was suitable for both the MEMCPU Platform and CPLEX, which was run by Lockheed Martin. Lockheed Martin also used an in-house method based on the A* algorithm, considered to be the most efficient method known for MAPF problems. All solvers were given 10 minutes to find solutions. The MEMCPU Platform outperformed the other two solvers, providing solutions to many more scenarios especially as the number of agents increased. A summary of the final results is reported in Fig. 15a.

[1] J. J. P. Eckert and J. W. Mauchly, Electronic numerical integrator and computer, US patent # US757158A (1947).

[2] H. H. Goldstine and A. Goldstine, The electronic numerical integrator and computer (ENIAC), Mathematical Tables and Other Aids to Computation **2**, 97 (1946).

[3] J. von Neumann, First draft of a report on the EDVAC, Annals of the History of Computing, IEEE **15**, 27 (1993).

[4] J. L. Hennessy and D. A. Patterson, *Computer architecture, sixth edition: A quantitative approach* (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2017).

[5] S. Arora and B. Barak, *Computational complexity: A modern approach* (Cambridge University Press, 2009).

[6] J. Backus, Can programming be liberated from the von Neumann style?: A functional style and its algebra of programs, Communications of The Acm **21**, 613 (1978).

[7] J. L. Hennessy and D. A. Patterson, A new golden age for computer architecture, Communications of the ACM **62**, 48 (2019).

[8] M. Horowitz, 1.1 Computing's energy problem (and what we can do about it), in *2014 IEEE international solid-state circuits conference digest of technical papers (ISSCC)* (IEEE, 2014).

[9] J. Nickolls and W. J. Dally, The GPU computing era, IEEE Micro **30**, 56 (2010).

[10] D. Singh and C. K. Reddy, A survey on platforms for big data analytics, Journal of Big Data **2**, 10.1186/s40537-014-0008-6 (2014).

[11] Y. LeCun, Y. Bengio, and G. Hinton, Deep learning, Nature **521**, 436 (2015).

[12] I. Goodfellow, J. Bengio, A. Courville, and F. Bach, *Deep learning* (MIT Press Ltd, 2016).

[13] V. Sze, Y.-H. Chen, J. Emer, A. Suleiman, and Z. Zhang, Hardware for machine learning: Challenges and opportunities, in *2017 IEEE custom integrated circuits conference (CICC)* (IEEE, 2017).

[14] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, Efficient processing of deep neural networks: A tutorial and survey, Proceedings of the IEEE **105**, 2295 (2017).

[15] S. Yu, Neuro-inspired computing with emerging non-

volatile memorys, Proceedings of the IEEE **106**, 260 (2018).
[16] G. W. Burr, R. M. Shelby, A. Sebastian, S. Kim, S. Kim, S. Sidler, K. Virwani, M. Ishii, P. Narayanan, A. Fumarola, L. L. Sanches, I. Boybat, M. L. Gallo, K. Moon, J. Woo, H. Hwang, and Y. Leblebici, Neuromorphic computing using non-volatile memory, Advances in Physics: X **2**, 89 (2016).
[17] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, The SpiNNaker project, Proceedings of the IEEE **102**, 652 (2014).
[18] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild, Y. Yang, and H. Wang, Loihi: A neuromorphic manycore processor with on-chip learning, IEEE Micro **38**, 82 (2018).
[19] I. Boybat, M. L. Gallo, S. R. Nandakumar, T. Moraitis, T. Parnell, T. Tuma, B. Rajendran, Y. Leblebici, A. Sebastian, and E. Eleftheriou, Neuromorphic computing with multi-memristive synapses, Nature Communications **9**, 10.1038/s41467-018-04933-y (2018).
[20] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, Deep learning in spiking neural networks, Neural Networks **111**, 47 (2019).
[21] G. Singh, L. Chelini, S. Corda, A. J. Awan, S. Stuijk, R. Jordans, H. Corporaal, and A.-J. Boonstra, Near-memory computing: Past, present, and future, Microprocessors and Microsystems **71**, 102868 (2019).
[22] D. Ielmini and H.-S. P. Wong, In-memory computing with resistive switching devices, Nature Electronics **1**, 333 (2018).
[23] F. L. Traversa, F. Bonani, Y. V. Pershin, and M. Di Ventra, Dynamic computing random access memory, Nanotechnology **25**, 285201 (2014).
[24] Y. V. Pershin, F. L. Traversa, and M. Di Ventra, Memcomputing with membrane memcapacitive systems, Nanotechnology **26**, 225201 (2015).
[25] A. Sebastian, M. L. Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, Memory devices and applications for in-memory computing, Nature Nanotechnology **15**, 529 (2020).
[26] M. Le Gallo, R. Khaddam-Aljameh, M. Stanisavljevic, A. Vasilopoulos, B. Kersting, M. Dazzi, G. Karunaratne, M. Brändli, A. Singh, S. M. Müller, J. Büchel, X. Timoneda, V. Joshi, M. J. Rasch, U. Egger, A. Garofalo, A. Petropoulos, T. Antonakopoulos, K. Brew, S. Choi, I. Ok, T. Philip, V. Chan, C. Silvestre, I. Ahsan, N. Saulnier, V. Narayanan, P. A. Francese, E. Eleftheriou, and A. Sebastian, A 64-core mixed-signal in-memory compute chip based on phase-change memory for deep neural network inference, Nature Electronics , 1 (2023).
[27] F. Caravelli, F. C. Sheldon, and F. L. Traversa, Global minimization via classical tunneling assisted by collective force field formation, Science Advances **7**, eabh1542 (2021).
[28] F. L. Traversa and M. Di Ventra, Universal memcomputing machines, IEEE Transactions on Neural Networks and Learning Systems **26**, 2702 (2015).
[29] M. Di Ventra and F. L. Traversa, Perspective: Memcomputing: Leveraging memory and physics to compute efficiently, Journal of Applied Physics **123**, 180901 (2018).
[30] Y. R. Pei, F. L. Traversa, and M. Di Ventra, On the universality of memcomputing machines, IEEE Transactions on Neural Networks and Learning Systems **30**, 1610 (2019).
[31] K. Y. Camsari, R. Faria, B. M. Sutton, and S. Datta, Stochastic p-Bits for invertible logic, Physical Review X **7**, 10.1103/physrevx.7.031014 (2017).
[32] E. Goto, New Parametron circuit element using nonlinear reactance, KDD Kenyku Shiryo (1954).
[33] E. Goto, The Parametron, a digital computing element which utilizes parametric oscillation, Proceedings of the IRE **47**, 1304 (1959).
[34] J. von Neumann, Non-linear capacitance or inductance switching, amplifying, and memory organs, patent # US2815488A (1954).
[35] R. Wigington, A new concept in computing, Proceedings of the IRE **47**, 516 (1959).
[36] G. Csaba, A. Raychowdhury, S. Datta, and W. Porod, Computing with coupled oscillators: Theory, devices, and applications, in *2018 IEEE international symposium on circuits and systems (ISCAS)* (IEEE, 2018).
[37] T. Wang and J. Roychowdhury, Oscillator-based Ising machine, arXiv:1709.08102 (2017).
[38] J. Chou, S. Bramhavar, S. Ghosh, and W. Herzog, Analog coupled oscillator based weighted Ising machine, Scientific Reports **9**, 10.1038/s41598-019-49699-5 (2019).
[39] A. Mallick, M. K. Bashar, D. S. Truesdell, B. H. Calhoun, S. Joshi, and N. Shukla, Using synchronized oscillators to compute the maximum independent set, Nature Communications **11**, 10.1038/s41467-020-18445-1 (2020).
[40] G. Csaba and W. Porod, Coupled oscillators for computing: A review and perspective, Applied Physics Reviews **7**, 011302 (2020).
[41] K. Chen and D. Wang, A dynamically coupled neural oscillator network for image segmentation, Neural Networks **15**, 423 (2002).
[42] N. Shukla, A. Parihar, M. Cotter, M. Barth, X. Li, N. Chandramoorthy, H. Paik, D. G. Schlom, V. Narayanan, A. Raychowdhury, and S. Datta, Pairwise coupled hybrid vanadium dioxide-MOSFET (HVFET) oscillators for non-boolean associative computing, in *2014 IEEE international electron devices meeting* (IEEE, 2014).
[43] N. Mohseni, P. L. McMahon, and T. Byrnes, Ising machines as hardware solvers of combinatorial optimization problems, Nature Reviews Physics **4**, 363 (2022).
[44] H. Goto, K. Tatsumura, and A. R. Dixon, Combinatorial optimization by simulating adiabatic bifurcations in nonlinear Hamiltonian systems, Science Advances **5**, eaav2372 (2019).
[45] F. L. Traversa, P. Cicotti, F. Sheldon, and M. Di Ventra, Evidence of Exponential Speed-Up in the Solution of Hard Optimization Problems, Complexity **2018**, e7982851 (2018).
[46] F. Sheldon, P. Cicotti, F. L. Traversa, and M. D. Ventra, Stress-testing memcomputing on hard combinatorial optimization problems, IEEE Transactions on Neural Networks and Learning Systems **31**, 2222 (2020).
[47] F. L. Traversa, Aircraft Loading Optimization: MemComputing the 5th Airbus Problem (2019), arXiv:1903.08189 [cs, math].
[48] F. Sheldon, F. L. Traversa, and M. Di Ventra, Taming a

nonconvex landscape with dynamical long-range order: Memcomputing Ising benchmarks, Physical Review E **100**, 053311 (2019), publisher: American Physical Society.
[49] H. Manukian, F. L. Traversa, and M. Di Ventra, Accelerating deep learning with memcomputing, Neural Networks **110**, 1 (2019).
[50] S. R. B. Bearden, Y. R. Pei, and M. Di Ventra, Efficient solution of Boolean satisfiability problems with digital memcomputing, Scientific Reports **10**, 19741 (2020).
[51] F. L. Traversa, C. Ramella, F. Bonani, and M. Di Ventra, Memcomputing NP-complete problems in polynomial time using polynomial resources and collective states, Science Advances **1**, e1500031 (2015).
[52] F. L. Traversa and M. Di Ventra, Polynomial-time solution of prime factorization and NP-complete problems with digital memcomputing machines, Chaos: An Interdisciplinary Journal of Nonlinear Science **27**, 023107 (2017).
[53] F. L. Traversa and M. Di Ventra, MemComputing Integer Linear Programming (2018), arXiv:1808.09999 [nlin].
[54] H. Manukian, F. L. Traversa, and M. Di Ventra, Memcomputing numerical inversion with self-organizing logic gates, IEEE Transactions on Neural Networks and Learning Systems **PP**, 1 (2017).
[55] F. L. Traversa, Oil and Gas Maritime Delivery Scheduling (2020).
[56] F. L. Traversa and T. Walker, Drone Swarm Optimization.
[57] F. L. Traversa, Aircraft, Passenger, Cargo, and Crew Scheduling (2023).
[58] MemComputing (2023).
[59] F. L. Traversa, Optimizing Helicopter Transportation to Oil Rigs (2021).
[60] Gurobi Optimization (2023).
[61] Mathematical program solvers - IBM CPLEX | IBM (2023).
[62] E. Milanov, The RSA Algorithm (2009).
[63] N. I. o. S. a. Technology, *Digital Signature Standard (DSS)*, Tech. Rep. Federal Information Processing Standard (FIPS) 186-5 (U.S. Department of Commerce, 2023).
[64] P. Benioff, The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines, Journal of Statistical Physics **22**, 563 (1980).
[65] R. P. Feynman, Simulating physics with computers, International Journal of Theoretical Physics **21**, 467 (1982).
[66] Y. Manin, Computable and uncomputable, Sovetskoye Radio, Moscow (1980).
[67] P. Shor, Algorithms for quantum computation: discrete logarithms and factoring, in *Proceedings 35th Annual Symposium on Foundations of Computer Science* (1994) pp. 124–134.
[68] E. Anschuetz, J. Olson, A. Aspuru-Guzik, and Y. Cao, Variational Quantum Factoring, in *Quantum Technology and Optimization Problems*, Lecture Notes in Computer Science, edited by S. Feld and C. Linnhoff-Popien (Springer International Publishing, Cham, 2019) pp. 74–85.
[69] M. Dyakonov, When will useful quantum computers be constructed? Not in the foreseeable future, this physicist argues. Here's why: The case against: Quantum computing, IEEE Spectrum **56**, 24 (2019).
[70] A. K. Lenstra and H. W. Lenstra, eds., *The development of the number field sieve*, Lecture Notes in Mathematics, Vol. 1554 (Springer, Berlin, Heidelberg, 1993).
[71] T. Kleinjung, K. Aoki, J. Franke, A. K. Lenstra, E. Thomé, J. W. Bos, P. Gaudry, A. Kruppa, P. L. Montgomery, D. A. Osvik, H. te Riele, A. Timofeev, and P. Zimmermann, Factorization of a 768-Bit RSA Modulus, in *Advances in Cryptology – CRYPTO 2010*, Lecture Notes in Computer Science, edited by T. Rabin (Springer, Berlin, Heidelberg, 2010) pp. 333–350.
[72] P. Zimmermann, Factorization of RSA-250 - Discussion (2020).
[73] L. Rocutto, M. Maronese, F. L. Traversa, S. Decherchi, and A. Cavalli, Assessing the Effectiveness of Non-Turing Computing paradigms, IEEE Access , 1 (2023).
[74] M. D. Ventra and F. L. Traversa, Self-organizing logic gates and circuits and complex problem solving with self-organizing circuits (2018).
[75] M. D. Ventra, *MemComputing: Fundamentals and Applications* (Oxford University Press, Oxford, New York, 2022).
[76] M. Di Ventra, F. L. Traversa, and I. V. Ovchinnikov, Topological Field Theory and Computing with Instantons, Annalen der Physik **529**, 1700123 (2017).
[77] Resources | MemComputing (2020).
[78] F. L. Traversa, Proliferated LEO Satellite Optimization (2022).
[79] E. Klotz and A. M. Newman, Practical guidelines for solving difficult mixed integer linear programs, Surveys in Operations Research and Management Science **18**, 18 (2013).
[80] A. Basu, M. Conforti, M. Di Summa, and H. Jiang, Complexity of Branch-and-Bound and Cutting Planes in Mixed-Integer Optimization — II, Combinatorica **42**, 971 (2022).
[81] D. Beckman, A. N. Chari, S. Devabhaktuni, and J. Preskill, Efficient networks for quantum factoring, Physical Review A **54**, 1034 (1996).
[82] M. Mosca and M. Piani, 2022 Quantum Threat Timeline Report (2022).
[83] Detailed results can be found in the project reports. Limited availability upon request. Contact | MemComputing.
[84] C. Pomerance, Smooth numbers and the quadratic sieve (2008).
[85] C. Pomerance, A Tale of Two Sieves, in *Biscuits of Number Theory*, edited by A. Benjamin and E. Brown (American Mathematical Society, Providence, Rhode Island, 2009) pp. 85–104.
[86] C. P. Schnorr, Fast Factoring Integers by SVP Algorithms, corrected (2021).
[87] E. R. Canfield, P. Erdös, and C. Pomerance, On a problem of Oppenheim concerning "factorisatio numerorum", Journal of Number Theory **17**, 1 (1983).
[88] R. D. Silverman, The multiple polynomial quadratic sieve, Mathematics of Computation **48**, 329 (1987).
[89] Msieve (2023).
[90] F. L. Traversa, System on Chip Self-Organizing Gates And Related Self-Organizing Logic Gates And Methods - MemComputing Inc (2022).
[91] J. Hale, *Asymptotic behavior of dissipative systems*, 2nd ed., Mathematical surveys and monographs, Vol. 25

(American Mathematical Society, Providence, Rhode Island, 2010).
[92] J. M. Beggs, The criticality hypothesis: how local cortical networks might optimize information processing | Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences (2008).
[93] M. Di Ventra and F. L. Traversa, Absence of chaos in digital memcomputing machines with solutions, Physics Letters A **381**, 3255 (2017).
[94] M. Di Ventra and F. L. Traversa, Absence of periodic orbits in digital memcomputing machines with solutions, Chaos: An Interdisciplinary Journal of Nonlinear Science **27**, 101101 (2017).
[95] L. Perko, *Differential Equations and Dynamical Systems*, edited by J. E. Marsden, L. Sirovich, and M. Golubitsky, Texts in Applied Mathematics, Vol. 7 (Springer, New York, NY, 2001).
[96] K. Christensen and N. R. Moloney, *Complexity and Criticality*, Imperial College Press Advanced Physics Texts, Vol. 1 (2005).
[97] S. R. Bearden, H. Manukian, F. L. Traversa, and M. Di Ventra, Instantons in Self-Organizing Logic Gates, Physical Review Applied **9**, 034029 (2018).
[98] S. R. B. Bearden, F. Sheldon, and M. D. Ventra, Critical branching processes in digital memcomputing machines, Europhysics Letters **127**, 30005 (2019).
[99] D. J. Earl and M. W. Deem, Parallel tempering: Theory, applications, and new perspectives, Physical Chemistry Chemical Physics **7**, 3910 (2005).
[100] T. Dunn, Southwest Airlines Has Been Badly Broken (NYSE:LUV) | Seeking Alpha (2022).
[101] L. Josephs, Southwest Airlines' schedule stabilizes after holiday meltdown but costs are still piling up (2023).