TuringX: A scalable and reliable platform for computing beyond Moore's law

TuringX Developers

September 20, 2022
v1.1

Abstract

With the **end of Moore's law** approaching and **Dennard scaling ending**, the computing community is increasingly looking at new technologies to enable continued performance improvements. A neuromorphic computer is a non-von Neumann computer whose structure and function are inspired by biology and physics. Today, such systems can be built and operated using existing technology, even at scale, and are capable of **outperforming current quantum computers**[1,2].

TuringX is a next-generation **platform for neuromorphic computing** based on a new flexible blockchain protocol. It is designed for the development of software applications and algorithms that utilize neuromorphic hardware and are capable of accelerating computation. To accomplish this goal, the platform connects hosts that are running clusters of neuromorphic chips with users and applications that utilize this next-generation hardware. On the TuringX platform, computation time is exchanged for the TuringX native token.

TuringX has also developed a proprietary circuit design, the **TuringX Neuromorphic Chip**, that complements the TuringX ecosystem and turns any modern field programmable gate array (FPGA) based chip into a neuromorphic computing chip that can perform orders of magnitude faster than classical or quantum methodologies for a wide range of applications. Due to the dominance of ASICs in the proof-of-work token mining industry, there is a large amount of dormant FPGA infrastructure available which can be converted into high performance next-generation neuromorphic computing clusters.

[1] Mniszewski, S. M. Graph partitioning as quadratic unconstrained binary optimization (QUBO) on spiking neuromorphic hardware. In *Proc. International Conference on Neuromorphic Systems* 1–5 (ACM, 2019).

[2] Yakopcic, C., Rahman, N., Atahary, T., Taha, T. M. & Douglass, S. Solving constraint satisfaction problems using the Loihi spiking neuromorphic processor. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)* 1079–1084 (IEEE, 2020).

1. Introduction

With the end of Moore's law approaching and Dennard scaling ending, the computing community is increasingly looking at new technologies to enable continued performance improvements. Among these new technologies are neuromorphic computers. **Neuromorphic computing** was coined by Carver Mead in the late 1980s[3,4] and at the time primarily refers to analogue-digital implementations of brain-inspired computing. In recent years, however, the term neuromorphic has come to encompass a broad range of hardware implementations as the field continues to evolve and large-scale funding opportunities have become available for brain-inspired computing systems, including the DARPA Synapse project and the Human Brain Project of the European Union.

The term neuromorphic computer refers to **non-von Neumann computers** whose structure and function are **influenced by biology and physics**. Data and instructions are stored in the memory units of Von Neumann computers, which consist of separate CPUs and memory units. On the other hand, in a neuromorphic computer, both processing and memory are governed by neurons and synapses. Unlike Von Neumann computers, neuromorphic computers define their programs based on the structure of the neural network and the parameters of the network rather than by explicit instructions. Also, while von Neumann computers encode information as numerical values expressed in binary terms, neuromorphic computers receive spikes as input, which are encoded numerically by the associated time at which they occur, the magnitude and the shape of their output.

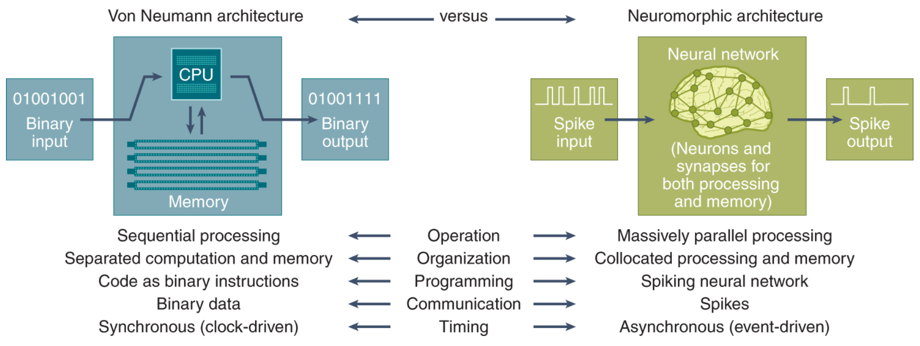
Von Neumann architecture
versus
Neuromorphic architecture
01001001
Binary input
CPU
Memory
01001111
Binary output
Spike input
Neural network
(Neurons and synapses for both processing and memory)
Spike output
Sequential processing
Separated computation and memory
Code as binary instructions
Binary data
Synchronous (clock-driven)
Operation
Organization
Programming
Communication
Timing
Massively parallel processing
Collocated processing and memory
Spiking neural network
Spikes
Asynchronous (event-driven)

Figure 1: Comparison of the von Neumann architecture with the neuromorphic architecture

[3] Mead, C. Neuromorphic electronic systems. *Proc. IEEE* 78, 1629–1636 (1990).

[4] Mead, C. How we created neuromorphic engineering. *Nat. Electron.* 3, 434–435 (2020).

As a result of the contrasting characteristics between the two architectures (Fig. 1), neuromorphic computers offer a number of fundamental operational differences:

- **Inherently parallel operation** is a characteristic of neuromorphic computers, where all neurons and synapses can potentially operate simultaneously; however, when compared with the parallelized von Neumann systems, neurons and synapses perform relatively simple computations.

- **Memory and processing are co-located:** in neuromorphic hardware, there is no concept of separating memory and processing. In many implementations, neurons and synapses perform processing and store values in tandem, despite the fact that neurons are sometimes thought of as processing units and synapses as memory units. By combining the processor and memory, the von Neumann bottleneck regarding processor/memory separation is mitigated, resulting in a reduction in maximum throughput. Furthermore, this collocation reduces the need for data access from the main memory, which consumes a large amount of energy compared to compute energy.[5].

- Neuromorphic computers have **inherent scalability** since adding more neuromorphic chips increases the number of neurons and synapses. . In order to run larger and larger networks, it is possible to treat multiple physical neuromorphic chips as a single large neuromorphic implementation. Several large-scale neuromorphic hardware systems have been successfully implemented, including SpiNNaker[6,7] and Loihi[8].

- Neuromorphic computers use **event-driven computation** (meaning, computing only when available data is available) and temporally sparse

[5] Sze, V., Chen, Y.-H., Emer, J., Suleiman, A. & Zhang, Z. Hardware for machine learning: challenges and opportunities. In *2017 IEEE Custom Integrated Circuits Conference (CICC)* 1–8 (IEEE, 2017).

[6] Mayr, C., Hoeppner, S. & Furber, S. SpiNNaker 2: a 10 million core processor system for brain simulation and machine learning. Preprint at https://arxiv.org/abs/1911.02385 (2019).

[7] Furber, S. B., Galluppi, F., Temple, S. & Plana, L. A. The SpiNNaker project. *Proc. IEEE* 102, 652–665 (2014).

[8] Davies, M. et al. Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99 (2018).

activity to achieve extremely high computational efficiency[9,10]. There is no work being performed by neurons and synapses unless there are spikes to be processed, and typically spikes are relatively sparse in the network operation.

- **Stochasticity** can be incorporated into neuromorphic computers, for instance when neurons fire, to accommodate noise.

Neuromorphic computers are well documented in the literature, and their features are often cited as motivating factors for their implementation and utilization [11,12,13,14]. An attractive feature of neuromorphic computers is their **extremely low power consumption**: they consume orders of magnitude less power than conventional computers. This low-power operation is due to the fact that they are event-driven and massively parallel, with only a small portion of the entire system being active at any given time. Energy efficiency alone is a compelling reason to investigate the use of neuromorphic computers in light of the increasing energy costs associated with computing, as well as the increasing number of applications that are energy constrained (e.g. edge computing applications). As neuromorphic computers implement neural network-style computations inherently, they are a natural platform for many of today's artificial intelligence and machine learning applications. The inherent computational properties of neuromorphic computers can also be leveraged to perform a wide variety of different types of computations[15].

[9] Mostafa, H., Müller, L. K. & Indiveri, G. An event-based architecture for solving constraint satisfaction problems. *Nat. Commun.* 6, 1–10 (2015).

[10] Amir, A. et al. A low power, fully event-based gesture recognition system. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* 7388–7397 (IEEE, 2017).

[11] Schuman, C. D. et al. A survey of neuromorphic computing and neural networks in hardware. Preprint at https://arxiv.org/abs/1705.06963 (2017).

[12] James, C. D. et al. A historical survey of algorithms and hardware architectures for neural-inspired and neuromorphic computing applications. *Biol. Inspired Cogn. Archit.* 19, 49–64 (2017).

[13] Strukov, D., Indiveri, G., Grollier, J. & Fusi, S. Building brain-inspired computing. *Nat. Commun.* 10, 4838–2019 (2019).

[14] Davies, M. et al. Advancing neuromorphic computing with Loihi: a survey of results and outlook. *Proc. IEEE* 109, 911–934 (2021).

[15] Aimone, J. B. et al. Non-neural network applications for spiking neuromorphic hardware. In *Proc. 3rd International Workshop on Post Moores Era Supercomputing* 24–26 (PMES, 2018).

2. The TuringX Vision

A TuringX neuromorphic chip and its protocol are flexible, and the community can modify them in the future. In this section, we define the main principles which TuringX should adhere to. This can be referred to as TuringX's social contract. When any of these principles are intentionally violated, the resulting chip and protocol should not be referred to as TuringX.

- **Decentralization comes first.** It is important that TuringX be as decentralized as possible: any parties (social leaders, software developers, hardware manufacturers, miners, funds, and so on) whose absence or malicious behavior may compromise the security of the network should be avoided. During the lifetime of TuringX, if any of these parties appear, the community should consider measures to reduce their impact.

- **Designed for regular people.** TuringX is a platform for ordinary people, and their interests should not be compromised to benefit big parties. As such, centralized mining should be prevented, and regular people should have the opportunity to run full nodes and mine blocks (albeit with a small probability).

- **A platform for neuromorphic computing of the future.** TuringX serves as the foundation for applications and algorithms built on top of it. Although it is designed for a variety of applications, its primary objective is to provide an efficient, secure and easy way to utilize highly efficient next generation computing systems.

- **Reducing energy consumption**: In light of the growing threat of climate change to our environment and our future, it is imperative that we take every measure necessary to reduce our energy consumption. An accelerated adoption of neuromorphic computing will therefore benefit our entire society since neuromorphic computing uses orders of magnitude less energy compared to traditional computing systems.

- **Long-term perspective.** In order to ensure the long-term success of TuringX, all aspects of development should be viewed from a long-term perspective. The TuringX project should be able to survive for centuries without any hard forks, hardware or software improvements, or any other unpredictable changes. Due to the fact that TuringX is designed as a platform, it should also be possible for applications and algorithms built on top of TuringX to survive over the long term. Due to TuringX's resiliency and long-term survivability, it may also have the potential to serve as a good store of value.

- **Open and permissionless.** TuringX neuromorphic chips and TuringX protocol do not restrict or limit any categories of usage. A user should be able to join the network and participate in the protocol without taking any preliminary steps. TuringX does not allow discrimination or limited access at the core level, as is the case with traditional supercomputer systems. In contrast, application developers are free to implement any logic they like, as long as they are responsible for the ethical and legal implications of their work.

3. TuringX Neuromorphic Chip

A TuringX machine is a class of general-purpose computing machines based on memory systems, where information is processed and stored at the same physical location. We analyze the memory properties of the TuringX machine to demonstrate that they possess **universal computing power**—they are Turing-complete—, **intrinsic parallelism**, **functional polymorphism**, and **information overhead**, namely that their collective states can support exponential data compression directly in memory through their collective states. Moreover, we show that the TuringX machine is capable of solving NP-complete problems in polynomial time, just like a non-deterministic Turing machine. The TuringX machine, however, requires only a polynomial number of memory cells due to its information overhead. It is important to note that even though these results do not prove NP=P within the Turing paradigm, the concept of TuringX machines represents a paradigm shift from the current von Neumann architecture, bringing us closer to the concept of brain-like neural computation.

Since Alan Turing invented his ideal machine in 1936[16,17], mathematicians have been able to develop this concept into what is now known as computational complexity theory[18], a powerful tool essentially employed to determine how long does an algorithm take to solve a problem with given input data. It is now known as the universal Turing machine (UTM) and serves as the conceptual foundation for all of today's digital computers. The practical realization of a UTM is commonly

[16] A. M. Turing, "On computational numbers, with an application to the entscheidungsproblem," Proc. of the London Math. Soc., vol. 42, pp. 230–265, 1936.

[17] A. M. Turing, The Essential Turing: Seminal Writings in Computing, Logic, Philosophy, Artificial Intelligence, and Artificial Life, Plus The Secrets of Enigma. Oxford University Press, 2004.

[18] S. Arora and B. Barak, Computational Complexity: A Modern Approach. Cambridge University Press, 2009.

done using the von Neumann architecture[19], which apart from some inconsequential details, it can be viewed as a device that requires a central processing unit (CPU) that is physically separate from the memory. The CPU contains both the control unit that directs the operation of the machine, as well as the logic gates and arithmetic functions needed for execution (arithmetic/logic unit). A large amount of data must be transferred between the CPU and the memory, thereby limiting the machine's performance both in terms of time (von Neumann bottleneck[20]) and energy consumption[21].

While **parallel computation** mitigates some of these problems, it **does not resolve them**: several processors manipulate portions of the whole data, using a physical "closed" memory. As a result, all the processors will eventually have to communicate with each other in order to solve the whole problem, still requiring a substantial amount of data transfer between them and their memory. A fundamentally new method of manipulating and storing data would be required in order to overcome this "information latency issue".

Recent research has proposed a new computing paradigm, inspired by the operations of our own brain, that does not rely on the UTM concept and puts the **entire computation in the memory**. The paradigm is known as memcomputing[22]. In the same way as the brain, memcomputing machines would compute in memory without the requirement of a separate processor. The memory allows learning and adaptive capabilities[23,24], bypassing broken connections and self-organizing the computation into the solution path[25,26], much like the brain is able to sustain a

[19] J. von Neumann, "First draft of a report on the edvac," Annals of the History of Computing, IEEE, vol. 15, no. 4, pp. 27–75, 1993.

[20] J. Backus, "Can programming be liberated from the von neumann style?: A functional style and its algebra of programs," Commun. ACM, vol. 21, pp. 613–641, Aug. 1978.

[21] J. L. Hennessy and D. A. Patterson, Computer Architecture, Fourth Edition: A Quantitative Approach. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2006.

[22] M. Di Ventra and Y. V. Pershin, "The parallel approach," Nature Physics, vol. 9, pp. 200–202, 2013.

[23] Y. V. Pershin, S. La Fontaine, and M. Di Ventra, "Memristive model of amoeba learning," Phys. Rev. E, vol. 80, p. 021926, Aug 2009.

[24] F. L. Traversa, Y. V. Pershin, and M. Di Ventra, "Memory models of adaptive behavior," Neural Networks and Learning Systems, IEEE Transactions on, vol. 24, pp. 1437–1448, Sept 2013.

[25] Y. V. Pershin and M. Di Ventra, "Solving mazes with memristors: A massively parallel approach," Phys. Rev. E, vol. 84, p. 046703, Oct 2011.

[26] Y. V. Pershin and M. Di Ventra, "Self-organization and solution of shortest-path optimization problems with memristive networks," Phys. Rev. E, vol. 88, p. 013305, Jul 2013.

certain amount of damage and still operate seamlessly. In practice, memcomputing can be implemented by utilizing physical properties of many materials and systems that exhibit a degree of time non-locality (memory) in their response functions at particular frequencies[27,28,29].

This type of machine has been mathematically proven to have the same computational power as a **non-deterministic Turing machine**[30], but unlike the latter, it is fully deterministic and, as such, **can be constructed**. These three properties are responsible for its computational power: **intrinsic parallelism** - interacting memory cells change their states simultaneously and collectively when performing computations; **functional polymorphism** - the same interacting memory cells can calculate different functions based on the signals applied; and finally **information overhead** - memory cells interacting can store a quantity of information in a manner that is not directly proportional to the number of memory cells.

This property is derived from a different type of architecture: the topology of this architecture is described by a network of interacting memory cells, and its dynamics are described by a collective state that is capable of storing and processing information simultaneously. Collective states are similar to the collective (entangled) states of many qubits in quantum computation, where the entangled state can be used to solve certain types of problems efficiently.

3.1 Memprocessors

In a TuringX architecture, memprocessors are the basic building blocks. We define a memprocessor as an object defined by the fourtuple *(x, y, z, σ)* where x is the state of the memprocessor, y is the array of internal variables, z the array of variables that connect from one memprocessor to other memprocessors, and σ an operator that defines the evolution

[27] T. Driscoll, H.-T. Kim, B.-G. Chae, B.-J. Kim, Y.-W. Lee, N. M. Jokerst, S. Palit, D. R. Smith, M. Di Ventra, and D. N. Basov, "Memory metamaterials," Science, vol. 325, no. 5947, pp. 1518–1521, 2009.

[28] Y. V. Pershin and M. Di Ventra, "Memory effects in complex materials and nanoscale systems," Advances in Physics, vol. 60, no. 2, pp. 145– 227, 2011.

[29] M. Di Ventra and Y. V. Pershin, "On the physical properties of memristive, memcapacitive and meminductive systems," Nanotechnology, vol. 24, p. 255201, 2013.

[30] F. L. Traversa and M. Di Ventra. Universal Memcomputing Machines. (preprint on arXiv:1405.0931) IEEE Transaction on Neural Networks and Learning Systems, DOI: 10.1109/TNNLS.2015.2391182, 2015.

$$\sigma[x, y, z] = (x', y').$$

When two or more memprocessors are connected, we have a network of memprocessors (computational memory). In this case we define the vector *x* as the state of the network (i.e., the array of all the states x_i of each memprocessor), and $z = \cup_i z_i$ the array of all connecting variables, with z_i the connecting array of variables of the memprocessor *i*-th.

Let z_i and z_j be respectively the vectors of connecting variables of the memprocessors *i* and *j*, then if $z_i \cap z_j \,!= \emptyset$ we say that the two memprocessors are connected. Alternatively, a memprocessor is not connected to any other memprocessor (isolated) when we have $z = z(x, y)$ (i.e., *z* is completely determined by *x* and *y*) and

$$\sigma[x, y, z(x, y)] = (x, y),$$

which means that the memprocessor has no dynamics. A network of memprocessors has the evolution of the connecting variables z given by the evolution operator Ξ defined as

$$\Xi[x, y, z, s] = z'$$

where $y = \cup_i y_i$ and *s* is the array of the external signals that can be applied to a subset of connections to provide stimuli for the network. Finally, the complete evolution of the network is defined by the system

$$\begin{aligned} &\sigma[x_1, y_1, z_1] = (x'_1, y'_1) \\ &\ldots \\ &\sigma[x_n, y_n, z_n] = (x'_n, y'_n) \\ &\Xi[x, y, z, s] = z'. \end{aligned}$$

The evolution operators σ and Ξ can be interpreted either as discrete or continuous evolution operators. The discrete evolution operator interpretation includes also the artificial neural networks[31], while the continuous operator interpretation represents more general dynamical systems.

[31] S. Haykin, Neural Networks: A Comprehensive Foundation. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2nd ed., 1998.

3.2 TuringX machine

An ideal TuringX machine consists of an interconnected bank of memory cells (memprocessors) capable of performing either digital (logic) or analog (functional) operations under the control of a control unit. The computation with and in memory can be illustrated as follows. When two or more memprocessors are connected, a signal sent by the control unit causes the memprocessors to change their internal states according to both their initial states and the signal, producing intrinsic parallelism as well as functional polymorphism.

We define the TuringX machine as the eight-tuple

$$\text{TuringX machine} = (M, \Delta, P, S, \Sigma, p_0, s_0, F),$$

where M is the set of possible states of a single memprocessor. It can be either a finite set M_d (digital regime), a continuum or an infinite discrete set of states M_a (analog regime), thus M can be expressed as $M = M_d \ OR \ Ma$. Δ is a set of functions

$$\delta_\alpha : M^{m_\alpha} \backslash F \times P \rightarrow M^{m'_\alpha} \times P^2 \times S$$

where $m_\alpha < \infty$ is the number of memprocessors used as input of (read by) the function δ_α, and $m'_\alpha < \infty$ is the number of memprocessors used as output (written by) the function δ_α; P is the set of the arrays of pointers p_α that select the memprocessors called by δ_α and S is the set of indexes α; Σ is the set of the initial states written by the input device on the computational memory; $p_0 \in P$ is the initial array of pointers; s_0 is the initial index α and $F \subseteq M$ is the set of final states.

Note that the two important features of the TuringX machine, namely **parallelism** and **polymorphism**, are clearly embedded in the definition of the set of functions δ_α. Indeed the TuringX machine, unlike the UTM, can have more than one transition function δ_α (functional polymorphism), and any function δ_α simultaneously acts on a set of memprocessors (intrinsic parallelism). The TuringX machine also differs from the UTM in that it does not distinguish between machine states and symbols recorded on tape. It is rather the states of the memprocessors that encode this information. It is imperative to have this component in order to build a machine that is capable of storing data and performing computations simultaneously.

As another important point, unlike a UTM, which has a finite number of discrete states and unlimited tape storage, a TuringX machine can operate, in principle, on an **infinite number of continuous states**, even if the number of memory processors is limited. In essence, each memprocessor is an analog device with a continuous set of state values.

Finally, it can be noticed that the formal definition of memprocessor and network of memprocessors is compatible with the function δ_α defined above. In fact, the topology and evolution of the network is associated with the stimuli s, while the control unit defines all possible $\delta_\alpha \in \Delta$ in the sense that those can be obtained by applying a certain signal s_α (which selects the index vector p_α) to the network. The network evolution then determines x' while β and p_β (or better s_β) are defined by the control unit for the next processing step.

3.3 TuringX Neuromorphic Chip

The TuringX machine can be realized physically as a **non-linear dynamical system**, which is composed of point attractors that represent the solutions to the problem. It is possible to numerically integrate TuringX machines' equations of motion, since they are non-quantum systems. The performance of similar machines on a wide variety of combinatorial optimization problems has already been demonstrated to be orders of **magnitude faster** than that of traditional algorithmic approaches[32,33,34,35,36].

Subsequently, by employing topological field theory[37], it was shown that the physical reason behind this efficiency rests on the dynamical long-range order that develops during the transient dynamics where avalanches (instantons in the field theory language) of different sizes are generated until the system reaches an

[32] Massimiliano Di Ventra and Fabio L. Traversa. Perspective: Memcomputing: Leveraging memory and physics to compute efficiently. *Journal of Applied Physics*, 123(18):180901, 2018.

[33] F. L. Traversa, P. Cicotti, F. Sheldon, and M. Di Ventra. Evidence of expo- nential speed-up in the solution of hard optimization problems. *Complexity*, 2018:7982851, 2018.

[34] F. L. Traversa and M. Di Ventra. Memcomputing integer linear programming. *arXiv:1808.09999*, 2018.

[35] Forrest Sheldon, Fabio L. Traversa, and Massimiliano Di Ventra. Taming a nonconvex landscape with dynamical long-range order: Memcomputing ising benchmarks. *Phys. Rev. E*, 100:053311, Nov 2019.

[36] Haik Manukian, Fabio L Traversa, and Massimiliano Di Ventra. Accelerating deep learning with memcomputing. *Neural Networks*, 110:1–7, 2019.

[37] M. Di Ventra, Fabio L. Traversa, and Igor V. Ovchinnikov. Topological field theory and computing with instantons. *Ann. Phys. (Berlin)*, 529:1700123, 2017.

attractor[38]. The transient phase of the solution search therefore resembles that of several phenomena in Nature, such as earthquakes[39], solar flares[40] or quenches[41].

Our **TuringX Neuromorphic Chip** utilizes field programmable gate arrays (FPGAs) to achieve close to real-time performance of a TuringX machine. It can be adapted freely to the problem to be computed and it can also be interconnected and operated as part of a cluster.

Using the TuringX Neuromorphic Chip, problems that cannot be solved with classical or quantum methods can be solved, therefore **eliminating** the barrier posed by the **von Neumann bottleneck**. It may be used to solve hard optimization problems, to implement integer linear programming (ILP), to carry out machine learning (ML), to train deep neural networks or to improve computing efficiency generally.

4. TuringX Protocol

Bitcoin has been a successful implementation of the concept of p2p electronic cash. Both professionals and the general public have come to appreciate the convenient combination of public transactions and proof-of-work as a trust model. Today, the user base of electronic cash is growing at a steady pace; customers are attracted to low fees and the anonymity provided by electronic cash and merchants value its predicted and decentralized emission. Bitcoin has effectively proved that electronic cash can be as simple as paper money and as convenient as credit cards.
Unfortunately, Bitcoin suffers from several deficiencies.

The core component of any blockchain system is its consensus protocol and TuringX utilizes an egalitarian Proof of Work (PoW) consensus protocol which shows several advantages over traditional one-CPU-one-vote algorithms:

[38] Forrest Sheldon, Fabio L. Traversa, and Massimiliano Di Ventra. Taming a nonconvex landscape with dynamical long-range order: Memcomputing ising benchmarks. *Phys. Rev. E*, 100:053311, Nov 2019.

[39] Per Bak and Chao Tang. Earthquakes as a self-organized critical phenomenon. *Journal of Geophysical Research: Solid Earth*, 94(B11):15635–15637, 1989.

[40] E. T. Lu and R. J. Hamilton. Avalanches and the distribution of solar flares. *The Astrophysical Journal*, 380:L89–L92, October 1991.

[41] Gunnar Pruessner. *Self-Organised Criticality: Theory, Models and Characteri- sation*. Cambridge University Press, 2012.

It is well known that one of the most significant problems with a PoW system is the development of specialized hardware (ASICs), which allows a small group of ASIC-equipped miners to solve PoW puzzles orders of magnitude faster and more efficiently than anyone else. Memory-hard PoW schemes can solve this problem by reducing the disparity between ASICs and commodity hardware. We believe that the most promising approach is to use asymmetric memory-hard PoW schemes that require significantly less memory to verify a solution than to discover it[42,43].

Secondly, a PoW network's decentralization is threatened by the fact that even large miners tend to form mining pools, leading to a situation in which just a few pool operators (5 in Bitcoin and 2 in Ethereum at the time of writing) control more than 51% of computing power. Our protocol is both **memory-hard** and **pool-resistant**.

4.1. Egalitarian Proof-of-work

In this section we detail our proof-of-work algorithm. Our primary goal is to close the gap between CPU (majority) and GPU/FPGA/ASIC (minority) miners. It is appropriate that some users can have a certain advantage over others, but their investments should grow at least linearly with the power. More generally, producing special-purpose devices has to be as less profitable as possible.

The original Bitcoin proof-of-work protocol uses the CPU-intensive pricing function SHA-256. It mainly consists of basic logical operators and relies solely on the computational speed of processor, therefore is perfectly suitable for multicore/conveyer implementation. However, modern computers are not limited by the number of operations per second alone, but also by memory size. While some processors can be substantially faster than others, memory sizes are less likely to vary between machines.

Memory-bound price functions were first introduced by Abadi et al and were defined as “functions whose computation time is dominated by the time spent accessing memory”. The main idea is to construct an algorithm allocating a large block of data (“scratchpad”) within memory that can be accessed relatively slowly (for example, RAM) and “accessing an unpredictable sequence of locations” within it. A block should be large enough to make preserving the data more advantageous than recomputing it for each access. The algorithm also should prevent internal

[42] A. Biryukov and D. Khovratovich, “Equihash: Asymmetric proof-of-work based on the generalized birthday problem,” Ledger, vol. 2, pp. 1–30, 2017.

[43] Ethash. [Online]. Available: https://github.com/ethereum/wiki/wiki/Ethash/6e97c9cea49605264c6f4d1dc9e1939b1f89a5a3

parallelism, hence N simultaneous threads should require N times more memory at once.

Dwork et al investigated and formalized this approach leading them to suggest another variant of the pricing function: "Mbound". One more work belongs to F. Coelho, who proposed the most effective solution: "Hokkaido". To our knowledge the last work based on the idea of pseudo-random searches in a big array is the algorithm known as "scrypt" by C. Percival. Unlike the previous functions it focuses on key derivation, and not proof-of-work systems. Despite this fact scrypt can serve our purpose: it works well as a pricing function in the partial hash conversion problem such as SHA-256 in Bitcoin.

By now scrypt has already been applied in Litecoin and some other Bitcoin forks. How- ever, its implementation is not really memory-bound: the ratio "memory access time / overall time" is not large enough because each instance uses only 128 KB. This permits GPU miners to be roughly 10 times more effective and continues to leave the possibility of creating relatively cheap but highly-efficient mining devices. Moreover, the scrypt construction itself allows a linear trade-off between memory size and CPU speed due to the fact that every block in the scratchpad is derived only from the previous. For example, you can store every second block and recalculate the others in a lazy way, i.e. only when it becomes necessary. The pseudo-random indexes are assumed to be uniformly distributed, hence the expected value of the additional blocks' recalculations is 21 ·N, where N is the number of iterations. The overall computation time increases less than by half because there are also time independent (constant time) operations such as preparing the scratchpad and hashing on every iteration. Saving 2/3 of the memory costs 31 · N + 13 · 2 · N = N additional recalculations; 9/10 results in 1 ·N +...+ 1 ·9·N = 4.5N. It is easy to show that storing only 1 of all blocks 1010 s increases the time less than by a factor of s−1 . This in turn implies that a machine with a CPU 2 200 times faster than the modern chips can store only 320 bytes of the scratchpad.

Our algorithm is a memory-bound algorithm for the proof-of-work pricing function. It relies on random access to a slow memory and emphasizes latency dependence. As opposed to scrypt every new block (64 bytes in length) depends on all the previous blocks. As a result a hypothetical "memory-saver" should increase his calculation speed exponentially. It requires around 2MB per instance:

- fits in the L3 cache (per core) of modern processors, which should become mainstream in a few years;
- A megabyte of internal memory is an almost unacceptable size for a modern ASIC pipeline;

- GPUs may run hundreds of concurrent instances, but they are limited in other ways: GDDR5 memory is slower than the CPU L3 cache and remarkable for its bandwidth, not random access speed.
- Significant expansion of the scratchpad would require an increase in iterations, which in turn implies an overall time increase. “Heavy” calls in a trust-less p2p network may lead to serious vulnerabilities, because nodes are obliged to check every new block’s proof-of-work. If a node spends a considerable amount of time on each hash evaluation, it can be easily DDoSed by a flood of fake objects with arbitrary work data (nonce values).

4.2. Untraceable Transactions

In this section we describe the scheme of fully anonymous transactions satisfying both untraceability and unlinkability conditions. An important feature of our solution is its autonomy: the sender is not required to cooperate with other users or a trusted third party to make his transactions; hence each participant produces a cover traffic independently.

Our scheme relies on the cryptographic primitive called a group signature. First presented by D. Chaum and E. van Heyst[44], it allows a user to sign his message on behalf of the group. After signing the message the user provides (for verification purposes) not his own single public key, but the keys of all the users of his group. A verifier is convinced that the real signer is a member of the group, but cannot exclusively identify the signer. The original protocol required a trusted third party (called the Group Manager), and he was the only one who could trace the signer. The next version called a ring signature, introduced by Rivest et al. in[45], was an autonomous scheme without Group Manager and anonymity revocation. Various modifications of this scheme appeared later: linkable ring signature[46,47,48] allowed to determine if two signatures were produced by the same group member, traceable ring signature[49,50] limited excessive anonymity by providing possibility to trace the signer of two messages with respect to the same metainformation (or “tag” in terms of[50]). A similar cryptographic construction is also known as a ad-hoc group

[44] David Chaum and Eug`ene van Heyst. Group signatures. In EUROCRYPT, pages 257–265, 1991.

[45] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In ASIACRYPT, pages 552–565, 2001.

[46] Joseph K. Liu, Victor K. Wei, and Duncan S. Wong. Linkable spontaneous anonymous group signature for ad hoc groups (extended abstract). In ACISP, pages 325–335, 2004.

[47] Joseph K. Liu and Duncan S. Wong. Linkable ring signatures: Security models and new schemes. In ICCSA (2), pages 614–623, 2005.

[48] Man Ho Au, Sherman S. M. Chow, Willy Susilo, and Patrick P. Tsang. Short linkable ring signatures revisited. In EuroPKI, pages 101–115, 2006.

[49] Eiichiro Fujisaki. Sub-linear size traceable ring signatures without random oracles. In CTRSA, pages 393–415, 2011.

[50] Eiichiro Fujisaki and Koutarou Suzuki. Traceable ring signature. In Public Key Cryptography, pages 181–200, 2007.

signature[51,52]. It emphasizes the arbitrary group formation, whereas group/ring signature schemes rather imply a fixed set of members. For the most part, our solution is based on the work “Traceable ring signature” by E. Fujisaki and K. Suzuki[53]. In order to distinguish the original algorithm and our modification we will call the latter a one-time ring signature, stressing the user’s capability to produce only one valid signature under his private key. We weakened the traceability property and kept the linkability only to provide one-timeness: the public key may appear in many foreign verifying sets and the private key can be used for generating a unique anonymous signature. In case of a double spend attempt these two signatures will be linked together, but revealing the signer is not necessary for our purposes.

4.2.1 Elliptic curve parameters

As our base signature algorithm we chose to use the fast scheme EdDSA, which is developed and implemented by D.J. Bernstein et al.[54]. Like Bitcoin’s ECDSA it is based on the elliptic curve discrete logarithm problem, so our scheme could also be applied to Bitcoin in future.

Common parameters are:

q: a prime number; q = 2255 − 19;
d: an element of Fq; d = −121665/121666;
E: an elliptic curve equation; −x 2 + y 2 = 1 + dx2y 2 ;
G: a base point; G = (x, −4/5);
l: a prime order of the base point; l = 2252 + 27742317777372353535851937790883648493;
Hs: a cryptographic hash function {0, 1} ∗ → Fq;
Hp: a deterministic hash function E(Fq) → E(Fq).

4.2.2 Terminology

Enhanced privacy requires a new terminology which should not be confused with Bitcoin entities.

private ec-key is a standard elliptic curve private key: a number a ∈ [1, l − 1];
public ec-key is a standard elliptic curve public key: a point A = aG;
one-time keypair is a pair of private and public ec-keys;
private user key is a pair (a, b) of two different private ec-keys;
tracking key is a pair (a, B) of private and public ec-key (where B = bG and a 6= b);

[51] Ben Adida, Susan Hohenberger, and Ronald L. Rivest. Ad-hoc-group signatures from hijacked keypairs. In in DIMACS Workshop on Theft in E-Commerce, 2005.
[52] Qianhong Wu, Willy Susilo, Yi Mu, and Fangguo Zhang. Ad hoc group signatures. In IWSEC, pages 120–135, 2006.
[53] Eiichiro Fujisaki and Koutarou Suzuki. Traceable ring signature. In Public Key Cryptography, pages 181–200, 2007.
[54] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. J. Cryptographic Engineering, 2(2):77–89, 2012.

public user key is a pair (A, B) of two public ec-keys derived from (a, b);
standard address is a representation of a public user key given into human friendly string with error correction;
truncated address is a representation of the second half (point B) of a public user key given into human friendly string with error correction.

The transaction structure remains similar to the structure in Bitcoin: every user can choose several independent incoming payments (transactions outputs), sign them with the corresponding private keys and send them to different destinations.

Contrary to Bitcoin's model, where a user possesses unique private and public key, in the proposed model a sender generates a one-time public key based on the recipient's address and some random data. In this sense, an incoming transaction for the same recipient is sent to a one-time public key (not directly to a unique address) and only the recipient can recover the corresponding private part to redeem his funds (using his unique private key). The recipient can spend the funds using a ring signature, keeping his ownership and actual spending anonymous. The details of the protocol are explained in the next subsections.

4.3. Unlinkable payments

Classic Bitcoin addresses, once being published, become unambiguous identifier for incoming payments, linking them together and tying to the recipient's pseudonyms. If someone wants to receive an "untied" transaction, he should convey his address to the sender by a private channel. If he wants to receive different transactions which cannot be proven to belong to the same owner he should generate all the different addresses and never publish them in his own pseudonym.

Our solution is allowing a user to publish a single address and receive unconditional unlinkable payments. The destination of each TuringX output (by default) is a public key, derived from recipient's address and sender's random data. The main advantage against Bitcoin is that every destination key is unique by default (unless the sender uses the same data for each of his transactions to the same recipient). Hence, there is no such issue as "address reuse" by design and no observer can determine if any transactions were sent to a specific address or link two addresses together.

First, the sender performs a Diffie-Hellman exchange to get a shared secret from his data and half of the recipient's address. Then he computes a one-time destination key, using the shared secret and the second half of the address. Two different ec-keys are required from the recipient for these two steps, so a standard TuringX address is nearly twice as large as a Bitcoin wallet address. The receiver also performs a Diffie-Hellman exchange to recover the corresponding secret key.

A standard transaction sequence goes as follows:

1. Alice wants to send a payment to Bob, who has published his standard address. She unpacks the address and gets Bob's public key (A, B).

2. Alice generates a random $r \in [1, l-1]$ and computes a one-time public key $P = Hs(rA)G+ B$.

3. Alice uses P as a destination key for the output and also packs value $R = rG$ (as a part of the Diffie-Hellman exchange) somewhere into the transaction. Note that she can create other outputs with unique public keys: different recipients' keys (Ai , Bi) imply different Pi even with the same r.

4. Alice sends the transaction.

5. Bob checks every passing transaction with his private key (a, b), and computes $P 0 = Hs(aR)G + B$. If Alice's transaction for with Bob as the recipient was among them, then $aR = arG = rA$ and $P 0 = P$.

6. Bob can recover the corresponding one-time private key: $x = Hs(aR) + b$, so as $P = xG$. He can spend this output at any time by signing a transaction with x.

As a result, Bob gets incoming payments, associated with one-time public keys which are **unlinkable** for a spectator. Some additional notes:

• When Bob "recognizes" his transactions (see step 5) he practically uses only half of his private information: (a, B). This pair, also known as the tracking key, can be passed to a third party (Carol). Bob can delegate her the processing of new transactions. Bob doesn't need to explicitly trust Carol, because she can't recover the one-time secret key p without Bob's full private key (a, b). This approach is useful when Bob lacks bandwidth or computation power (smartphones, hardware wallets etc.).

• In case Alice wants to prove she sent a transaction to Bob's address she can either discloser or use any kind of zero-knowledge protocol to prove she knows r (for example by signing the transaction with r).

• If Bob wants to have an audit compatible address where all incoming transaction are linkable, he can either publish his tracking key or use a truncated address. That address represent only one public ec-key B, and the remaining part required by the protocol is derived from it as follows: $a = Hs(B)$ and $A = Hs(B)G$. In both cases every person is able to "recognize" all of Bob's incoming transaction, but, of course, none can spend the funds enclosed within them without the secret key b.

4.4. One-time ring signatures

A protocol based on one-time ring signatures allows users to achieve unconditional unlinkability. Unfortunately, ordinary types of cryptographic signatures permit to trace transactions to their respective senders and receivers. Our solution to this deficiency lies in using a different signature type than those currently used in electronic cash systems.

We will first provide a general description of our algorithm with no explicit reference to electronic cash. A one-time ring signature contains four algorithms:

GEN: takes public parameters and outputs an ec-pair (P, x) and a public key I.

SIG: takes a message m, a set S 0 of public keys {Pi}i6=s, a pair (Ps, xs) and outputs a signature σ and a set S = S 0 ∪ {Ps}.

VER: takes a message m, a set S, a signature σ and outputs "true" or "false".

LNK: takes a set I = {Ii}, a signature σ and outputs "linked" or "indep".

The idea behind the protocol is fairly simple: a user produces a signature which can be checked by a set of public keys rather than a unique public key. The identity of the signer is indistinguishable from the other users whose public keys are in the set until the owner produces a second signature using the same keypair.

GEN: The signer picks a random secret key x ∈ [1, l − 1] and computes the corresponding public key P = xG. Additionally he computes another public key I = xHp(P) which we will call the "key image".

SIG: The signer generates a one-time ring signature with a non-interactive zero-knowledge proof using the techniques from [21]. He selects a random subset S 0 of n from the other users' public keys Pi , his own keypair (x, P) and key image I. Let 0 ≤ s ≤ n be signer's secret index in S (so that his public key is Ps). He picks a random {qi | i = 0 . . . n} and {wi | i = 0 . . . n, i 6= s} from (1 . . . l) and applies the following transformations:

$$L_i = \begin{cases} q_i G, & \text{if } i = s \\ q_i G + w_i P_i, & \text{if } i \neq s \end{cases}$$

$$R_i = \begin{cases} q_i \mathcal{H}_p(P_i), & \text{if } i = s \\ q_i \mathcal{H}_p(P_i) + w_i I, & \text{if } i \neq s \end{cases}$$

The next step is getting the non-interactive challenge:

$$c = \mathcal{H}_s(m, L_1, \ldots, L_n, R_1, \ldots, R_n)$$

Finally the signer computes the response:

$$c_i = \begin{cases} w_i, & \text{if } i \neq s \\ c - \sum_{i=0}^{n} c_i \mod l, & \text{if } i = s \end{cases}$$

$$r_i = \begin{cases} q_i, & \text{if } i \neq s \\ q_s - c_s x \mod l, & \text{if } i = s \end{cases}$$

The resulting signature is σ = (I, c1, . . . , cn, r1, . . . , rn).

VER: The verifier checks the signature by applying the inverse transformations:

$$\begin{cases} L_i' = r_i G + c_i P_i \\ R_i' = r_i \mathcal{H}_p(P_i) + c_i I \end{cases}$$

Finally, the verifier checks if

$$\sum_{i=0}^{n} c_i \overset{?}{=} \mathcal{H}_s(m, L_0', \ldots, L_n', R_0', \ldots, R_n') \mod l$$

If this equality is correct, the verifier runs the algorithm LNK. Otherwise the verifier rejects the signature.

LNK: The verifier checks if I has been used in past signatures (these values are stored in the set I). Multiple uses imply that two signatures were produced under the same secret key. The meaning of the protocol: by applying L-transformations the signer proves that he knows such x that at least one Pi = xG. To make this proof non-repeatable we introduce the key image as I = xHp(P). The signer uses the same coefficients (ri , ci) to prove almost the same statement: he knows such x that at least one Hp(Pi) = I · x −1 .

If the mapping x → I is an injection:

1. Nobody can recover the public key from the key image and identify the signer;
2. The signer cannot make two signatures with different I’s and the same x.

4.5. A TuringX transaction

By combining both methods (unlinkable public keys and untraceable ring signature) Bob achieves new level of privacy in comparison with the original Bitcoin scheme. It requires him to store only one private key (a, b) and publish (A, B) to start receiving and sending anonymous transactions. While validating each transaction Bob additionally performs only two elliptic curve multiplications and one addition per output to check if a transaction belongs to him. For his every output Bob recovers a one-time keypair (pi , Pi) and stores it in his wallet. Any inputs can be circumstantially proved to have the same owner only if they appear in a single transaction. In fact this relationship is much harder to establish due to the one-time ring signature.

With a ring signature Bob can effectively hide every input among somebody else's; all possible spenders will be equiprobable, even the previous owner (Alice) has no more information than any observer.

When signing his transaction Bob specifies n foreign outputs with the same amount as his output, mixing all of them without the participation of other users. Bob himself (as well as anybody else) does not know if any of these payments have been spent: an output can be used in thousands of signatures as an ambiguity factor and never as a target of hiding. The double spend check occurs in the LNK phase when checking against the used key images set. Bob can choose the ambiguity degree on his own: n = 1 means that the probability he has spent the output is 50% probability, n = 99 gives 1%. The size of the resulting signature increases linearly as O(n+ 1), so the improved anonymity costs to Bob extra transaction fees. He also can set n = 0 and make his ring signature to consist of only one element, however this will instantly reveal him as a spender.

5. Resiliency and Survivability

Due to its nature as a platform, TuringX is expected to support **long-term** contracts for at least the lifetime of an average person. Despite this, even young smart contract platforms are experiencing performance degradation and inability to adapt to external conditions. Therefore, a cryptocurrency will depend on a small group of developers to provide a hard-fork to fix this problem, or else it will not be able to survive. As an example, the Ethereum network has been using the Proof-of-Work consensus algorithm and promises to switch to Proof-of-Stake in the near future. Nevertheless, delays in Proof-of-Stake development have resulted in several fixing hard forks[55] and the community is still reliant on the core developers to implement the next hard fork.

The first common survivability issue is that developers often implement ad-hoc solutions in pursuit of popularity without conducting adequate research and testing. Inevitably, such solutions will result in bugs, which will in turn lead to hasty bug fixes, which will then lead to bug fixes of those bug fixes, etc., making the network unreliable and even less secure. Rather than seeking short-term innovation, TuringX focuses on using **stable, well-tested solutions**. Many of the solutions used in TuringX have been formalized in papers that have been presented at peer-

[55] Ethereums blockchain is once again feeling the difficulty bomb effect. [Online]. Available: https://www.coindesk.com/ ethereum-blockchain-feeling-the-difficulty-bomb-effect

reviewed conferences[56,57,58,59,60,61] and have been widely discussed in the community.

Decentralization (and thus survivability) is also challenged by the absence of secure trustless light clients. TuringX aims to solve this problem without creating new ones. Since TuringX is a Proof-of-Work blockchain, a small header can easily be extracted from the block content. This header alone allows for the validation of the work done on it, and a headers chain is sufficient to select the optimal chain for synchronization with the network. **Headers-chains**, although much smaller than the full blockchain, still grow linearly over time. Recent research on light clients has demonstrated a way for light clients to synchronize with the network by downloading an even smaller amount of data, thus enabling untrusted low-end devices, such as mobile phones, to join the network[62,63]. TuringX uses an authenticated state and allows clients to download proofs of the correctness of transactions included in a block. In this way, TuringX is accessible to anyone using a mobile phone, regardless of the blockchain size.

There is also a third potential problem, namely that while light clients solve the problem for TuringX users, they still do not solve it for TuringX miners, who must

[56] L. Reyzin, D. Meshkov, A. Chepurnoy, and S. Ivanov, "Improving authenticated dynamic dictionaries, with applications to cryptocurrencies," in International Conference on Financial Cryptography and Data Security. Springer, 2017, pp. 376–392.

[57] D. Meshkov, A. Chepurnoy, and M. Jansen, "Short paper: Revisiting difficulty control for blockchain systems," in Data Privacy Management, Cryptocurrencies and Blockchain Technology. Springer, 2017, pp. 429–436.

[58] A. Chepurnoy, V. Kharin, and D. Meshkov, "A systematic approach to cryptocurrency fees," IACR Cryptology ePrint Archive, vol. 2018, p. 78, 2018.

[59] A. Chepurnoy, V. Kharin, and D. Meshkov, "Self-reproducing coins as universal turing machine," in Data Privacy Management, Cryptocurrencies and Blockchain Technology. Springer, 2018, pp. 57–64.

[60] A. Chepurnoy and M. Rathee, "Checking laws of the blockchain with property-based testing," in Blockchain Oriented Software Engineering (IWBOSE), 2018 International Workshop on. IEEE, 2018, pp. 40–47.

[61] T. Duong, A. Chepurnoy, and H.-S. Zhou, "Multi-mode cryptocurrency systems," in Proceedings of the 2nd ACM Workshop on Blockchains, Cryptocurrencies, and Contracts. ACM, 2018, pp. 35–46.

[62] A. Kiayias, A. Miller, and D. Zindros, "Non-interactive proofs of proofof-work," Cryptology ePrint Archive, Report 2017/963, 2017. Accessed: 2017-10-03, Tech. Rep., 2017.

[63] L. Luu, B. Buenz, and M. Zamani, "Flyclient super light client for cryptocurrencies," IACR Cryptology ePrint Archive, 2019. [Online]. Available: https://eprint.iacr.org/2019/226

still keep the entire state for efficient transaction validation. Currently, blockchain systems allow users to store arbitrary data in this state. Due to the fact that this data lasts forever, it creates a lot of dust, which grows in size infinitely over time[64]. In situations where the state is too large for random-access memory, an adversary may be able to generate transactions that are very slow to validate as they require random access to the miner's storage. As a result, DoS attacks such as the one that occurred on Ethereum in 2016 may occur[65]. The community's fear of such attacks as well as the problem of "state bloat" without compensation for miners and users may have prevented scaling solutions from being implemented (such as larger block sizes, for instance). For this reason, TuringX contains a **storage rent component**: if an output remains in the state for four years without being moved, a miner may charge a small fee per byte.

Similarly to regular cloud storage services, this concept has only recently been proposed for cryptocurrencies[66] and has several important implications. In the first place, it ensures that TuringX mining will always be stable, as opposed to Bitcoin and other proof-of-work currencies, where mining may become unstable once emission is completed[67]. Second, the growth of the state's size becomes predictable and controllable, so TuringX miners are able to manage their hardware requirements more effectively. Finally, by collecting storage fees from outdated boxes, miners can return coins to circulation, thus preventing the steady decrease of circulating supply due to lost keys[68]. It is expected that all of these factors will support TuringX's long-term viability, both technically and economically.

---

[64] C. P´erez-Sol`a, S. Delgado-Segura, G. Navarro-Arribas, and J. HerreraJoancomart´ı, “Another coin bites the dust: an analysis of dust in utxobased cryptocurrencies,” Royal Society open science, vol. 6, no. 1, p. 180817, 2019.

[65] Ethereum network attackers ip address is traceable. [Online]. Available: https://www.bokconsulting.com.au/blog/ ethereum-network-attackers-ip-address-is-traceable/

[66] A. Chepurnoy and D. Meshkov, “On space-scarce economy in blockchain systems.” IACR Cryptology ePrint Archive, vol. 2017, p. 644, 2017.

[67] M. Carlsten, H. Kalodner, S. M. Weinberg, and A. Narayanan, “On the instability of bitcoin without the block reward,” in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2016, pp. 154–167.

[68] E. Krause, “A fifth of all bitcoin is missing. these crypto hunters can help,” 2018.

6. TuringX's Native Token

TuringX's native token is called **TRGX** and can be divided into 10^9 smaller units called nanoTRGX (one nanoTRGX equals one billionth of a TRGX). There will be a total of maximum **100,000,000.0 TRGX** available. TRGX are vital to the stability and security of the TuringX platform for a number of reasons as outlined below. Tokens will be emitted according to a predetermined and hard-coded schedule.

6.1. Emission

There's **no initial coin offering** („ICO"), no **pre-mining** and **no coin drop** for developers or any other hidden incentive built into the token. Upon launch of the TuringX mainnet, all 100,000,000.0 TRGX tokens will be available according tot he emission schedule. To ensure the smoothness of the emission process we use the following formula for block rewards:

$$BaseReward = (\mathrm{MSupply} - \mathrm{A}) \gg 18,$$

where A is amount of previously generated coins. The following graphic displays the emission graphically:

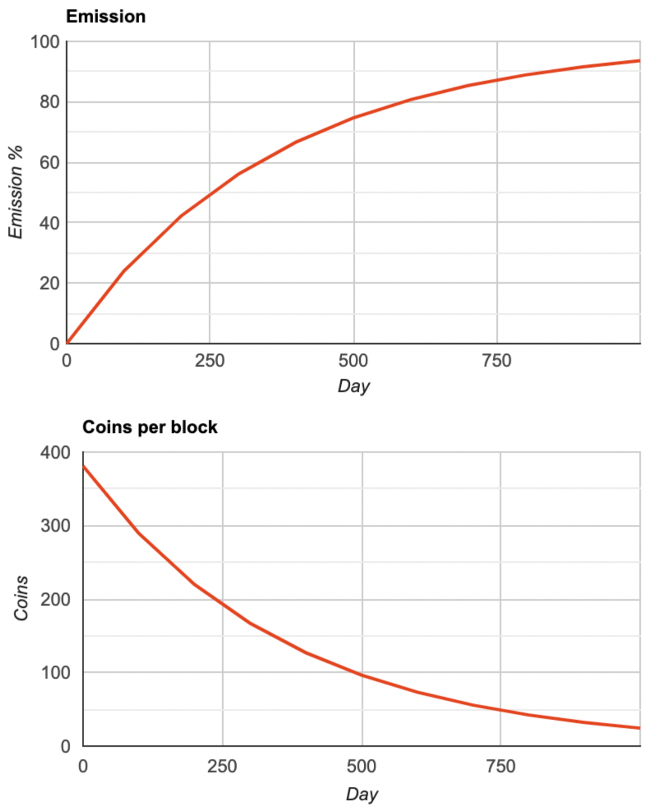

Figure 2: The TRGX Emission Schedule

7. Next Generation Neuromorphic Computing

The next generation of computing capabilities should be **accessible to everyone**, according to our opinion. As a platform for liberating and accelerating neuromorphic computing, which we believe will be the future of computing, TuringX combines **proprietary circuitry** (such as TuringX's Neuromorphic Chip) with **general available** neuromorphic computing **infrastructure** (such as Intel Lohi[69], IBM TrueNorth[70] or the University of California's NeuRRAM[71]) and **software and algorithm** developers.

TuringX provides the **foundation for applications** and algorithms that will be built upon it. A platform enables users and applications to access this next-generation hardware by connecting hosts that are operating clusters of neuromorphic chips. The TuringX native token is used to exchange computation time on the platform.

- **TuringX operators** maintain and operate neuromorphic computing infrastructure. This can be achieved by using general hardware such as Intel Lohi, IBM TrueNorth or University of California‘s NeuRRAM, or by programming FPGAs with the TuringX Neuromorphic Chip circuit design. Due to the dominance of ASICs in proof-of-work token mining, there is a significant amount of dormant FPGA infrastructure available, which can be repurposed into high performance next generation neuromorphic computing clusters. TuringX's operators offer their computing resources in exchange for TRGX, TuringX's native token.

- **Software developers** develop applications that run on neuromorphic computing infrastructure provided by operators. With TuringXscript, a simple and easy-to-learn scripting language, problems and computational tasks can be reformulated for execution on neuromorphic computing clusters. Having access to this infrastructure allows software developers and researchers to implement high-performance, highly efficient computing systems that can outperform current and Quantum methodologies. TuringX's native token TRGX is used as compensation for the use of its computing resources.

[69] https://www.intel.com/content/www/us/en/research/neuromorphic-computing.html

[70] Krishna, R. & Nandini, Usha & Mayan, J. & Sawarn, Nidhi. (2021). Neuromorphic Computing – The Principal of Development. 10.4108/eai.7-6-2021.2308573.

[71] https://www.sciencedaily.com/releases/2022/08/220817114253.htm

- Besides the environmental benefits associated with repurposing hardware, **users** may also obtain TuringX's native token TRGX and thus take part in the growing post Moore's law computing market segment, a market that will likely grow exponentially in the near future as a result of repurposing hardware. Our vision is for ordinary people to be able to benefit from the rapidly increasing computing power in the future.

7.1. Example

To illustrate the superior performance of neuromorphic computing, the following example showcases an implementation of a constraint satisfaction problem, where a problem formulation with complexity $O(n^{100,000})$ is being solved using the TuringX Neuromorphic Chip. The problem consists of 100,000 unique variables. Existing methodologies based on current and Quantum technology (reducing the complexity with Shor's algorithm[72] to $O(n^{50,000})$ cannot solve this problem today as it would require **longer than the existence of the universe** to find a solution. The TuringX Neuromorphic Chip solves the problem **in 2.23s** because of its inherent parallelization, it's long-range order and its capability to utilize instantons (Fig.2).

Current Method	**Quantum Method**	**TuringX Neuromorphic Chip**
$O(n^{100,000})$	$O(n^{50,000})$	2.23s
*Longer than the universe exits	*Longer than the universe exists	

Note: these results can be verified and reproduced with our reference implementations published in our GitHub repository.

[72] Mosca, M., Verschoor, S.R. Factoring semi-primes with (quantum) SAT-solvers. *Sci Rep* 12, 7982 (2022). https://doi.org/10.1038/s41598-022-11687-7