SAND2019-14547R

THE FUTURE OF COMPUTING: INTEGRATING SCIENTIFIC COMPUTATION ON NEUROMORPHIC SYSTEMS

LEAH REEDER*, JAMES B. AIMONE †, AND WILLIAM M. SEVERA‡

Abstract. Neuromorphic computing is known for its integration of algorithms and hardware elements that are inspired by the brain. Conventionally, this nontraditional method of computing is used for many neural or learning inspired applications. Unfortunately, this has resulted in the field of neuromorphic computing being relatively narrow in scope. In this paper we discuss two research areas actively trying to widen the impact of neuromorphic systems. The first is Fugu, a high-level programming interface designed to bridge the gap between general computer scientists and those who specialize in neuromorphic areas. The second aims to map classical scientific computing problems onto these frameworks through the example of random walks. This elucidates a class of scientific applications that are conducive to neuromorphic algorithms.

1. Introduction. Computer scientists have been fascinated by the brain and its connections to scientific computation for years. The father of theoretical computer science Alan Turing himself was tormented with the question "can machines think?" in his 1950 paper *Computing Machinery and Intelligence*. Now with modern technology, we are finally able to begin developing an answer to that question. The brain has inspired several projects that are paramount to our technological advancements today.

One such project is the EU's Human Brain Project, that has tasked itself with partially simulating a human brain. Using traditional computing methods, the brain is a computational nightmare due to its enormous number of neurons with their high connectivity [3]. However, as technology advances, we are able to simulate these neurons and their connections more efficiently using computing methods that are inspired by how the actual brain computes: neural networks.

Another common research project that has taken the world by storm and was first inspired by the brain is Artificial Intelligence (AI). The consequence of AI is that we are able to replicate human behavior on computers without having to hardcode that behavior explicitly. Machine Learning (ML) is a highly popular subset of AI that allows computers to take in data to solve an unknown algorithm. An exciting and popular result of ML is that computers, when exposed to more data, can distinguish items on their own without an explicit algorithm [2]. It has become hugely popular to use neural networks to assist this training of computers. ML techniques that are reliant on artificial neural networks are similar to human brains, as they get trained and essentially "learn".

In order to make these types of projects more scalable, numerous researchers in these fields have turned to neuromorphic computing, which is also known for its inspiration from the brain and use of neural networks. Although neuromorphic computing is widespread in many learning and neural applications, it is not found in many other engineering or computing applications. It is important to integrate these more general applications onto neuromorphic systems to see the benefits of neuromorphic spread into other areas.

1.1. Neuromorphic Computing. Neuromorpic computing blends two aspects of computing, both influenced by the brain. This nontraditional method of computing

*Colorado School of Mines, lreeder@mines.edu
†Sandia National Laboratories, jbaimon@sandia.gov
‡Sandia National Laboratories, wmsever@sandia.gov

combines software paradigms that are brain-inspired with physical computer hardware that uses artificial neurons as their computational elements. In this paper we are primarily referring to spikng neural algorithms due to their ability to relay a significant amount of information in small interactions [7, 10]. The spiking neural networks can be programmed onto hardware that is specifically designed to run these networks in an efficient manner with impressively low energy.

We can pair neural inspired algorithms that result in spiking neural networks with new computer architectures to achieve significant energy and volume efficiency [12]. This is especially desirable in the event that traditional computers could require more power than is reasonable due to the projected future of semiconductors and the end of Moore's Law [4, 16]. While spiking neural networks can run on traditional computing devices, when these networks are coupled with neuromorphic hardware, the computational advantages are significant [6]. Unfortunately, implementing these spiking neural algorithms and programming them on this specialized hardware is non-trivial, which has prevented neuromorphic computing from being wide spread in the general scientific community.

1.2. The Future of Computing. For decades, the scientific computing community has been focused on numerically solving challenging mathematical problems. The world's best supercomputers have been built in order to comply with the high demands scientists, engineers, and mathematicians have to compute these difficult problems. When this technology curve winds down due to the end of Moore's law, nontraditional methods of computing, such as neuromorphic, can rise up in the ranks [9].

Unfortunately, neuromorphic computing is not as ubiquitous as its traditional counterparts because a large knowledge base of neural computation is required. In this paper we will discuss several efforts at Sandia National Laboratories to try to bridge this gap. In Section 2 we will discuss Fugu, a high-level interface currently being developed to make neuromorphic computing more accessible to the general scientific computing community. In Section 3 we will then present a well known scientific computing technique, random walks, that has been implemented with a neural algorithm and on neuromorphic hardware. These two sections embody current efforts to show that not only can we implement non-neural and non-learning applications on these frameworks, but we can also achieve significant performance benefits on these computations as well.

2. Fugu. Fugu, a high-level tool for scientific applications, allows users to utilize and build spiking neural algorithms for arbitrary computations [1]. This is especially desirable because it allows users to be able to adopt neural algorithms for general computations, showing that neuromorphic paradigms can be used in strictly non-neural application spaces. In addition, Fugu significantly expands the population space of those who can benefit from neuromorphic computing. There are many potential contributors to Fugu, which we have grouped into the following three classes,

- A: Those without intimate knowledge of neural computing.
- B: Those who have experience with writing neural algorithms.
- C: Those with significant knowledge of neuromorphic hardware.

Our goal is to ensure that each type of user will be able to use Fugu with relative ease and contribute to it in any of the three manners. Fugu contains a library of different neural algorithms that are able to solve a variety of computations.

For example, several graph algorithms have been instantiated, such as shortest path and breadth first search. People in group B will be able to contribute to Fugu

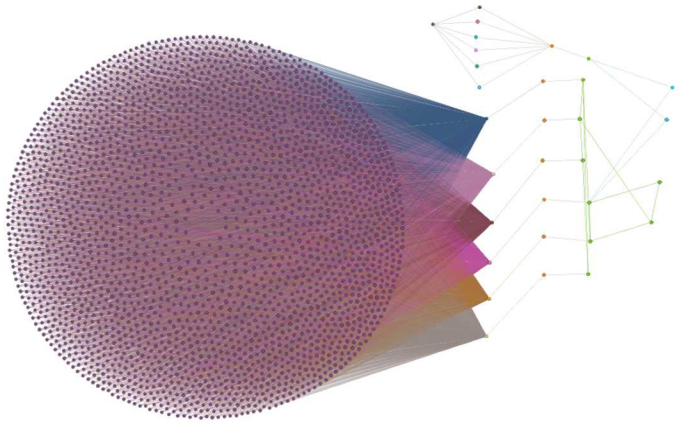

FIG. 2.1. *An example of the underlying neural structure of a large graph computation with Fugu. Each different colored node in the graph corresponds to a neuron of a particular brick. In this instance, the purple neurons represent the input map, the orange represents the classification, green the graph search, and blue the constraint satisfaction.*

and implement other useful algorithms identified by the people in group A's needs. In addition to algorithms, Fugu also contains several different neuromorphic backends that will be able to run these algorithms efficiently, as they have been designed and optimized for neural algorithms. People in group C will be able to develop new backends as more neuromorphic hardware becomes available. The end result is a plethora of spiking neural algorithms for a multitude of application spaces with plenty of backend options.

2.1. Structure. Fugu itself is composed of two main parts: bricks and scaffolds. The idea is that bricks are simple computations that users may want to use as a part of a larger computation. In this sense, users can combine bricks to form a scaffold.

For example, say a user wanted to determine if a destination on a map is within a certain vehicle's range. To compute this with Fugu, the following bricks would be needed: an input of the map, classification on that image, a resulting graph search, and then a constraint satisfaction. Each of these bricks would then be combined into a scaffold and computed in the specified order. The resulting network of neurons created from this example simulation can be seen in Figure 2.1.

2.2. Features. Fugu has several features that make it especially distinct compared to other platforms that run neural algorithms efficiently. The main distinction is that Fugu can run non-neural applications in addition to many learning and neural applications. In Section 3 we present one non-neural application, a Markov process random walk. This exemplifies a class of computations that fit nicely in spiking neural algorithms.

Another Fugu feature, shown in Figure 2.2, is the debugging tool created for Fugu. The idea behind this tool is that when users are creating a brick, it is important for them to determine if their brick works. This debugging tool can be used to ensure that the spikes are happening when expected and the bricks are connected in a reasonable way. A difficult part of not only neuromorphic, but computing in

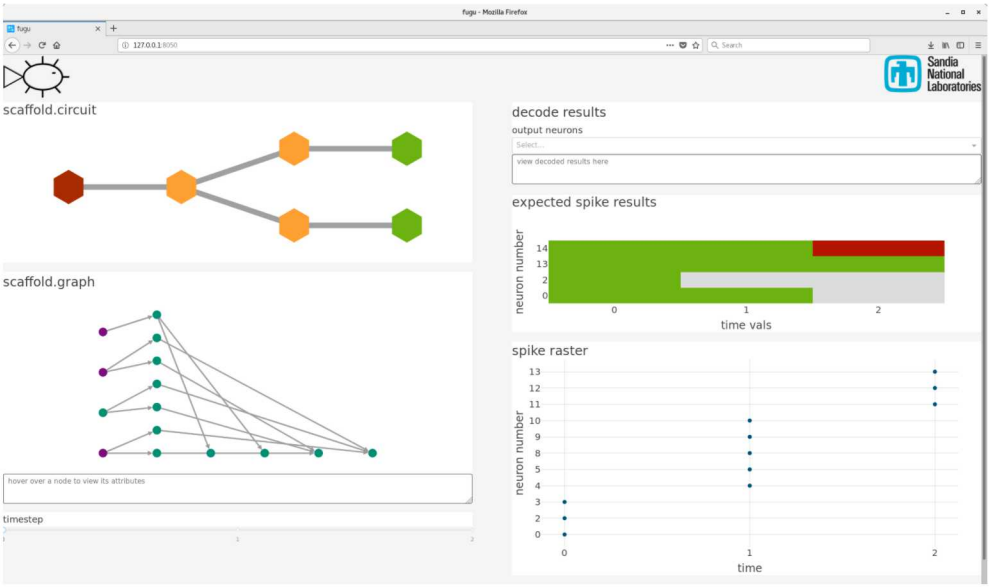

FIG. 2.2. *An example of a Fugu scaffold computation in the built in debugging tool. In this example, the input is a vector. Then, that vector is copied and two different dot products are applied to one copy of the vector. Finally, the resulting value of the dot products are checked against two thresholds to determine if the resulting value is above or below each threshold.*

general is the fact that debugging is difficult. As neuromorphic computing is not as widespread as traditional computing, there are not any best practices for debugging in place. We hope that this debugging visualization tool can be used effectively for any computation used on Fugu and can help set good practices for others in the neural network computing realm as well.

2.3. Goals. Eventually, we hope to have Fugu open to the general public. For now, it is being developed at Sandia National Laboratories with collaborators at Lawrence Livermore National Laboratory and Los Alamos National Laboratory. Our goal is to have Fugu be open source so a variety of people can use it and contribute to it, which would enhance the neuromorphic community as a whole.

3. Random Walks and Diffusion. Here we present a scientific application, random walks, that has been shown to map well onto a neuromorphic system. A random walk models particles moving in a diffusion scheme, or randomly in a free space. In a simplified scheme, particles start at a location on a one dimensional line and have defined probabilities of moving one step to the right or left. If we let the particle take one step at each time step, over time the motion of the particle would result in a random walk over its domain. A depiction of a 1-D random walk can be seen in Figure 3.1. This can extend easily to two or more dimensions. For an in-depth introduction on random walks, see [15].

The motion of particles in a random walk inherently solves the well known diffusion equation partial differential equation (PDE) [14]. The diffusion equation, also referred to as the heat equation, models the phenomenon of particles spreading throughout a domain. This is a well known PDE that arises in many scientific computing and engineering applications.

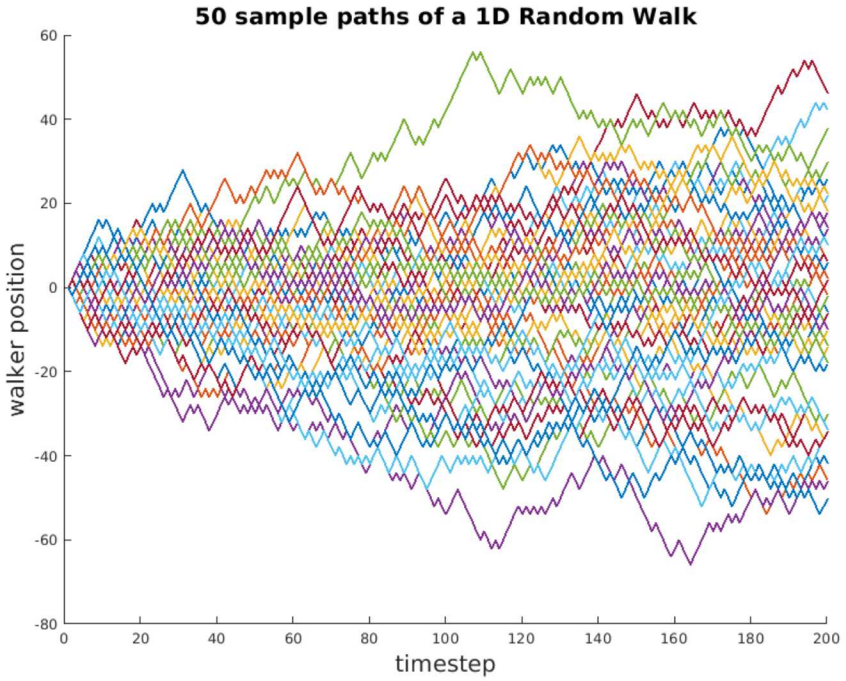

FIG. 3.1. *One dimensional random walk simulation depicting 50 particles.*

3.1. Spiking Neural Algorithm. The spiking neural algorithm that we use for random walks was first presented in [13]. This algorithm tracks positions of random walkers in time by tracking the nodes on a graph and counting the number of walkers at each node at any given point in time.

To do this, there is a spiking neural circuit embedded at each node in the graph that randomly determines which direction the walkers will go. A random walker's direction is determined by a specific neuron spiking in a probability gate. At a given time step, if a walker is at a node, neurons will propagate and spike throughout the circuit at that node, ending at this probability gate. The output neuron that spikes in the probability gate specifies which neighboring node the current node will send a walker to. This neighboring node is where the walker will be at the next time step. These probabilities are all user-determined, so this algorithm can be ubiquitous over a variety of applications.

3.2. Applications. The most common applications of random walks are a variety of graph algorithms and image processing tools such as path finding and image segmentation. More details on these types of applications and how they were implemented as spiking neural algorithms can be found in [11]. Here we focus on several more general engineering applications—radiation transport and electrical capacitance. Each of these applications can be modeled with different modifications of a simple random walk.

3.2.1. Radiation Transport. Radiation transport is an important and relevant application as many scientists try to model particles being emitted from a radioac-

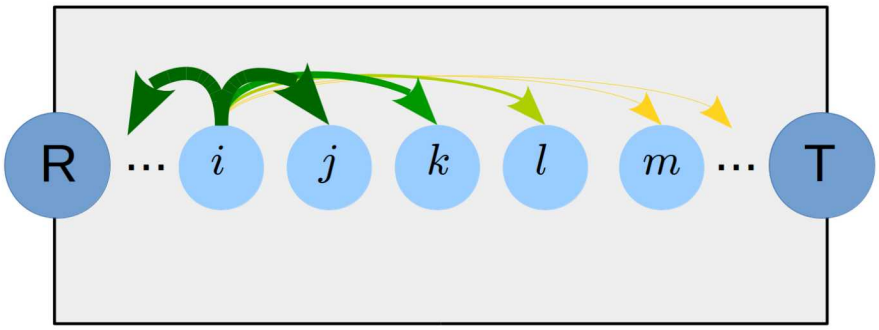

FIG. 3.2. *A potential 1D radiation transport scheme. The slab of material contains nodes along the x direction. A neural circuit is embedded at each node. Walkers have a probability to move from one node to another with a certain probability determined by a double tailed exponential distribution. Arrows represent transitions from node i. Transitions with higher probabilities are denoted in green with a thicker arrow. Transitions with low probabilities are denoted in yellow with a thin arrow. Nodes R and T represent reflection and transmittance nodes respectively.*

tive slab. Solving this radiation transport problem with random walks is not new; it is common to use Monte Carlo approaces to solve these types of particle transport problems [5]. Here we focus specifically on a one dimensional model problem. For a one dimensional radiation transport scheme, we consider random movement throughout a slab, where spatial locations are only defined along one direction. Particles are either absorbed or reflected at random spatial intervals along the slab. This results in three random numbers to consider at each particle location: distance particle traveled, direction of particle movement, and rate at which the particle is absorbed.

We are working on mapping this to our spiking neural algorithm. The first inherent difference from a simple random walk lies in the fact that we have more random events occurring. As it is, our algorithm can deal with particles moving in an arbitrary number of directions, but we will need to instantiate other random factors on top of that, such as a random absorption event. We also have particles that are randomly diffracted and change directions rapidly. As we are considering a one-dimensional case, we can represent this by creating highly connected neural circuits, so more potential directions are available at a single instance.

This does not require changes to the algorithm itself, but changes to how it is instantiated by the user. To model the possible directions and transition probabilities needed for this application, the length of the slab needs to be discretized. We set the transition probabilities following a double tailed exponential distribution, where walkers at a location are much more likely to move to closer neighbors rather than positions located on the other side of the slab. Here it is important to ensure that probability is conserved over all of the possible directions. Figure 3.2 shows how the model can be set up, with thicker arrows corresponding to connections with higher probabilities.

In the algorithm, we are primarily concerned with particles being either reflected, transmitted, or absorbed. We can keep track of this throughout the random walk simulation because if a particle reaches beyond the left-most point of the domain, we can qualify this as the particle being reflected back from where it first entered the

slab. If a particle reaches beyond the right-most point of the domain, then we can qualify this as the particle being fully transmitted throughout the slab. To quantify these positions in the actual simulation, we append two more possible locations to either end of the slab, one to simulate reflection and one to simulate transmittance.

Particles that are being absorbed fit more naturally to the original random walk model. To integrate absorption into the model, we create another random draw before spikes are sent to neighboring neurons. If we check the result of this draw to the specified probability of absorption, we can move spikes along if it does not meet the threshold and hold spikes if it does, signifying that the walker has been absorbed.

3.2.2. Electrical Capacitance. Using introductory electrostatic concepts, calculating the capacitance can be thought of as calculating the surface integral of the charge density of a capacitor. Let $S \in \mathbb{R}^3$ represent a conductor and ∂S be its surface in which we are calculating the charge density on. Additionally, let C represent the capacitance of S, and μ represent the charge density distribution. Then, capacitance can be found with the following equation,

$$C = \int_{\partial S} \mu(y) d\sigma(y). \tag{3.1}$$

To calculate the charge density $\mu(y)$ of a given object, a random walk on the surface of that object can be conducted. After n timesteps, the charge density can be calculated by determining the amount of particles at each position on the surface of the object [8].

In this random walk scheme, a particle starts at a particular location on the surface of a cube. The particle can then randomly move to another face of the cube with a random angle θ along the current face of the cube and random angle ϕ above the current face of the cube. This can be seen in Figure 3.3.

This general random walk model can be implemented quite easily with the neural algorithm that we already have; however, there will be a large bottleneck if we do not modify how the particles are connected. In the original scheme, each particle has the potential to move to any location on any remaining face of the surface. In our neural algorithm, this translates to building a network of neurons before the simulation begins and connecting a neural circuit at each node to circuits at all of the remaining nodes that can be reached. Although this can be done, our random walk algorithm will be very inefficient if we implement it at face value due to the significant number of highly connected neurons.

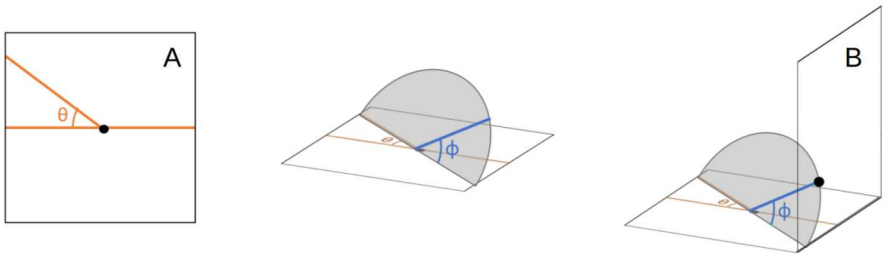

FIG. 3.3. *A depiction of how the particles move in a random walk scheme on a surface of a cube. A particle begins at face A then moves to face B as a result of two random draws: one for the angle* $\theta \in [0, 2\pi)$ *and one for the angle* $\phi \in [0, \pi]$.

Instead, we are exploring hierarchical connections of the nodes so we do not have to initialize a significant amount of neurons and connections that have a very low probability of being used. As we build our network of neurons before the simulations start, we need to prune the number of outcomes of walkers that arrive at each node so that there are a more reasonable number of connections at each node. This can be done in a variety of ways, but it is not clear just yet what the best implementation is. It is important when implementing these applications on neural platforms that we are able to adjust the algorithm so that it is able to work well throughout the application spaces and continually show performance benefits with this nontraditional method of computing.

4. Conclusion. We have shown that neuromorphic computing is a promising field with a significant number of unexplored research applications. Previous concerns regarding neuromorphic systems have accused it of being niche as it does not extend well to non-neural applications. However, we have discussed a variety of non-neural applications that show how neuromorphic can be used in addition to or instead of traditional architectures and paradigms. We hope to further this area of research and continue to develop useful algorithms that utilize the unique features of neuromorphic architectures to their fullest extent.

Acknowledgement. This research was funded by the Laboratory Directed Research and Development program at Sandia National Laboratories. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

REFERENCES

[1] J. B. Aimone, W. Severa, and C. M. Vineyard, *Composing neural algorithms with fugu*, arXiv preprint arXiv:1905.12130, (2019).
[2] E. Alpaydin, *Introduction to Machine Learning*, The MIT press, 2nd ed., 2010.
[3] A. Calimera, E. Macii, and M. Poncino, *The human brain project and neuromorphic computing*, Functional neurology, 28 (2013), p. 191.
[4] J.-A. Carballo, W.-T. J. Chan, P. A. Gargini, A. B. Kahng, and S. Nath, *Itrs 2.0: Toward a re-framing of the semiconductor technology roadmap*, in 2014 IEEE 32nd International Conference on Computer Design (ICCD), IEEE, 2014, pp. 139–146.
[5] S. A. Dupree and S. K. Fraley, *A Monte Carlo primer: A Practical approach to radiation transport*, vol. 1, Springer Science & Business Media, 2002.
[6] A. J. Hill, J. W. Donaldson, F. H. Rothganger, C. M. Vineyard, D. R. Follett, P. L. Follett, M. R. Smith, S. J. Verzi, W. Severa, F. Wang, et al., *A spike-timing neuromorphic architecture*, in 2017 IEEE International Conference on Rebooting Computing (ICRC), IEEE, 2017, pp. 1–8.
[7] W. Maass, *Networks of spiking neurons: the third generation of neural network models*, Neural networks, 10 (1997), pp. 1659–1671.
[8] M. Mascagni and N. A. Simonov, *The random walk on the boundary method for calculating capacitance*, Journal of Computational Physics, 195 (2004), pp. 465–473.
[9] D. Monroe, *Neuromorphic computing gets ready for the (really) big time*, Communications of the ACM, 57 (2014), pp. 13–15.
[10] W. Olin-Ammentorp, K. Beckmann, C. D. Schuman, J. S. Plank, and N. C. Cady, *Stochasticity and robustness in spiking neural networks*, arXiv preprint arXiv:1906.02796, (2019).
[11] L. E. Reeder, A. J. Hill, J. B. Aimone, and W. M. Severa, *Exploring applications of random walks on spiking neural algorithms*, Center for Computing Research Summer Proceedings - Technical Report SAND2019-5093R, (2018), pp. 145–156.

[12] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank, *A survey of neuromorphic computing and neural networks in hardware*, arXiv preprint arXiv:1705.06963, (2017).
[13] W. Severa, R. Lehoucq, O. Parekh, and J. B. Aimone, *Spiking neural algorithms for markov process random walk*, arXiv preprint arXiv:1805.00509, (2018).
[14] R. W. Shonkwiler and F. Mendivil, *Explorations in Monte Carlo Methods*, Springer Science & Business Media, 2009.
[15] L. Sjogren, *Chapter 2 : Random walks [pdf]*.
[16] M. M. Waldrop, *The chips are down for moore's law*, Nature News, 530 (2016), p. 144.