SANDIA REPORT
SAND2014-18566
~~Official Use Only~~
Printed October 2014

Beyond Moore’s Law Computer Architecture

Erik P. DeBenedictis, Brad Aimone, Peter Sharma,
with additional participants acknowledged inside

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

~~OFFICIAL USE ONLY~~

~~May be exempt from public release under the Freedom of Information Act (5 U.S.C. 552), exemption number and category: 4 Commercial/proprietary~~

~~Department of Energy review required before public release.~~

~~Name/Org: Erik P. DeBenedictis~~ ~~Date: October 1, 2014~~

~~Guidance (if applicable): Proprietary information of another company for exemption 4.~~

DOES NOT CONTAIN OFFICIAL USE ONLY INFORMATION
Name/Org: Erik P. DeBenedictis/1425 Date: 1/17/2016

~~Further dissemination only as authorized to U.S. Government agencies and their contractors; other requests shall be approved by the originating facility or higher DOE programmatic authority.~~

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

SAND2014-18566
~~Official Use Only~~
Printed October, 2014

Beyond Moore's Law Computer Architecture

Erik P. DeBenedictis, Brad Aimone, Peter Sharma,
with additional participants acknowledged inside
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-MS 1319

Abstract

Final report for Beyond Moore's Law Computer Architecture LDRD 171060. It reports on the development of a computer scaling and computer architecture framework for computers that will apply once electronic devices stop scaling due to maturation of Moore's Law. The project also includes subprojects in materials and devices, which are either reported here or are projects in alliance with this project

Acknowledgements

The following people contributed to this project

John	Aidun	David	Frank	Peter	Mattern
Brad	Aimone	Mike	Frank	Setso	Metodi
Jim	Ang	Scott	Hemmert	Nancy	Missert
Ed	Barsis	Bruce	Hendrickson	John	Naegle
Bob	Benner	David	Henry	Murat	Okandan
Geoff	Brennecka	Vince	Hitalia	Ben	Payne
Bill	Camp	Scott	Holmes	Steve	Plimpton
Frances	Chance	John	Ihlefeld	Andrew	Pomerene
Patrick	Chu	Conrad	James	Fred	Rothganger
Jeanine	Cook	Jim	Laros	Peter	Sharma
Rich	Dondero	François	Léonard	Ann	Speed
Tim	Draelos	Rupert	Lewis	John	Sullivan
Bruce	Draper	Denis	Mamaluy	Alec	Talin
Serena	Elay	Marc	Manheimer	John	Wagner
Brian	Evans	Matt	Marinella	Noel	Wheeler

Table of contents

List of figures

List of tables

Nomenclature

Assessment method	Our method of attaching an expression multiplied by kT to a neuron, neural network, or neural network algorithm; the evaluating the expression yields the energy of executing a function in units of kT.
Artificial Neural Network (ANN)	An interconnection of neurons that computes. The phrase does not refer to the hardware that runs it.
ANN implementation approach	The abstract hardware design for a machine that evaluates an ANN. Does not refer to a specific implementation, but a parameterized design that can scale or be implemented with a varying number of neurons, layers, etc.
ANN computer	A specific computer constructed with the ANN implementation approach.
ANN technology stack	The stack of technology based on physical devices, including artificial but physical neural networks, and neural network algorithms.
Resistive crossbar neural network	An artificial neural network comprised of a crossbar of wires with resistors or memristors at the intersection points.
Artificial spiking neural network	An artificial neural network where signaling is via spikes.
Software-based neural network	An ANN neural network structure evaluated on a conventional computer using a software layer to emulate the ANN. Example: Deep Learning.
Neural network algorithm	Phrase coined by the authors to be an algorithm comprised of multiple steps, each step comprising use of a neural network. Usage is similar to the phrase “slide-rule algorithm,” which would be an algorithm executed by a person with a slide rule.

Beyond Moore’s Law Computer Architecture

Background

The Beyond Moore’s Law Computer architecture project is addressing portions of a crucially important hierarchy of ideas driving the computer industry. The computer industry is large but otherwise similar to industries that have an evolving sequence of products over time. The key product levels in computing will be summarized below, as Sandia’s role at these product levels becomes part of this project:

1. Sandia uses many products in computers and computing. These include activities in 1400 and 8900 on software and algorithms for supercomputers made from externally purchased computers. Center 1700 builds CMOS parts in the MESA fab that are of the same technology class as the base components of supercomputers, but Sandia only makes specialty parts that are not used for production computers. In addition, centers 1100 and 8900 do research into materials that can be used for enhancements to CMOS.

The computer industry engages in other activities to make commodity computers, which includes computer architecture and engineering and the production of infrastructure software like compilers and operating systems. Sandia is not very active in these areas.

2. However, computers are supported by an industry in continuous motion. It is essential that each generation of computer run software written in the past and on other types of computers. This means there is an important continuity or scalability property in computer designs, which is another level of abstraction on top of computer architecture. Where Sandia produces software, the software is almost universally released as a series of versions that form a product family.

3. At a higher level, somebody designs the product families. The design of a product family includes rules for the progression of common technology and interfaces from one member of the family to another. The progression sequence has an identity that is independent of the specific products by the length and predictability of the product family sequence and the dollar volume (or other impact measure) of the products that result from it.

At the broadest level, John von Neumann and Gordon Moore are credited with the architecture and technology that define the overall concept of modern computers. In simple terms, Gordon Moore described a product concept for a family of integrated circuits that improve year over year. Von Neumann described a computer design at the block diagram level, where the blocks could be filled in by integrated circuits that grew more powerful from one generations to the next. Depending on broadly the computer industry is defined, it can be sized at $4-7 trillion per year.

Sandia has had a real but limited role in the computer industry. Sandia invented the cleanroom, an instance of which is used for to manufacture every semiconductor product. Sandia also had a critical role in the parallel processing branch of the modern computer industry, accounting for the top few percent of the computer industry. The current thrust in “massively parallel supercomputers” originated at Caltech with a computer now called the “Cosmic Cube” and which center 1400 adopted in the early 1980s. Sandia has been one of the most influential entities in acquiring large massively parallel supercomputers and developing key algorithms and applications for them.

Sandians have participated in the management of the computer industry through employees participation over the years in the International Technology Roadmap for Semiconductors (ITRS). Current participants include Erik DeBenedictis PI of this project and Matt Marinella a participant in this project).

Current crisis in computers

The product family concept behind computers has lasted longer than expected but the concept itself is revealing limitations that threaten end the product family.

An example of this problem that Sandia helped fix in the 1980s was a limitation in speed set essentially by the speed of light. It became evident in the 1980s that the appetite for fast computers exceeded the speed possible with the von Neumann architecture due to limitations on clock rate and processor-memory latency. Sandia was instrumental in retrofitting the von Neumann architecture for greater speed by creating what is now called the parallel von Neumann architecture. It is nearly ubiquitous today that processors have multiple cores and multi-core processors are connected into clusters or supercomputers.

However, it has been evident since about 2003 that the underlying physics of transistors will prevent the continued growth in computer performance that society has come to expect. The current problem is that the energy required for computation cannot be lowered much further. The computer industry relies on new and innovative applications that are inevitably more complex than previous ones. If power consumption cannot be reduced, the new and inevitably more complex applications will use too much energy for users to afford, including the NNSA and other U.S. government agencies with national security missions.

There is a broad recognition that the world needs another major fixup or replacement for the computing concept of von Neumann and Moore. This offers opportunities at a range of sizes. An example at small scale might be more power efficient algorithms that can support new applications with the same computer hardware. An example at large scale would be new computing concept based on new physical devices (that augment or replace transistors) that make a new scalable computer architecture viable.

Sandia Beyond Moore activities and this LDRD

Sandia is expressing institutional interest in addressing the computing crisis through the Beyond Moore Computing Research Challenge, but this activity has not specified the response in terms of whether it addresses only specific products, specific product families, or designs new product families. A result of this LDRD has been to perform research at the highest level. Since the hierarchy makes it natural for high level activities to include lower level ones, this LDRD builds a research product out a wide swath of prior and current research projects across Sandia.

The top-level objective is to devise a computing concept that could succeed von Neumann and Moore's and substantiate it to the extent possible in 18 months. To create such a concept requires addressing the issues in Table 1.

Table 1: Properties of a computing concept

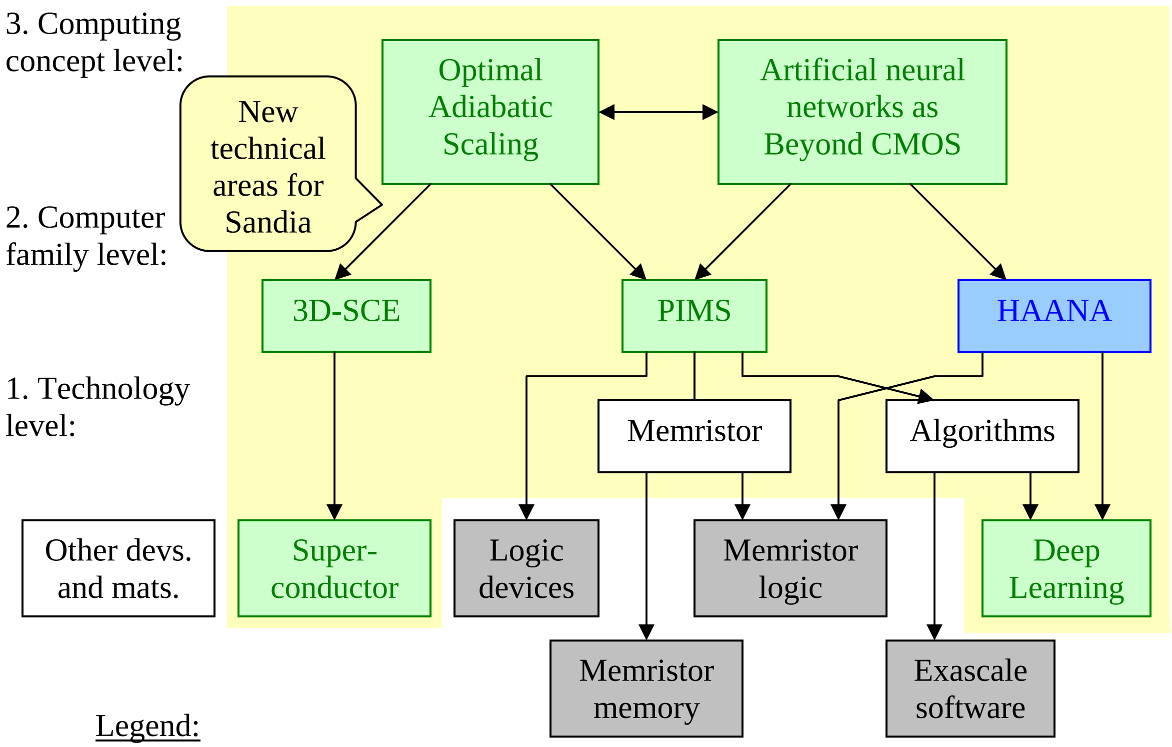
There must be a physical design for a computer that could persist across multiple generations. To have any chance of succeeding in today's environment, the generations must become progressively more power efficient.
There must be a method of programming the physical structure. A program written for one generation must work on successive generations.
While not essential, it would be a big bonus of the software written for today's von Neumann architecture would run at least moderately well.

Project organization

This LDRD project tapped existing activities and developed new ones. Various existing Sandia researchers in devices and materials participated to contribute their technology and expertise. This LDRD then started (what we believe is a) new activity at Sandia of using the device and materials research as building blocks to create the computing concepts with the properties in Table 1.

This new activity comprises devising and evaluating the computing concepts. Team members created or advanced 2-4 such candidates for the future of computing, including Processor-In-Memory-and-Storage (PIMS), and a 3D SuperConducting Electronics project (3D-SCE). This LDRD also created a neural network theory that was adopted by the HAANA Grand Challenge, although that project includes other activities unrelated to this LDRD. Each of these is a computation research activity in its own right, but also provides a framework and gives relevance to materials and device research.

The result is depicted graphically in Figure 1, showing the new activities in the colored area with arrows leading to existing activities in black and gray. The HAANA activity sees itself as a successor project to his LDRD, as opposed to part of it. The reader will see that this LDRD created a "superstructure" on top of existing research.



Legend:
Green: Result of Beyond Moore's Law LDRD
Gray: Preexisting activity cooperating with Beyond Moore's Law project
Blue: HAANA

Figure 1: New (color) and existing (white/gray) activities

"Beyond Moore's Law" Computing Concepts

We have defined Optimal Adiabatic Scaling as a potential replacement for the von Neumann architecture with one that is more compatible with the physical devices we can now manufacture. The von Neumann architecture separated a computer system in to processor, memory, and storage (disk drive). Even with the advent of parallel processing, data movement between these components needed to be a high speed. High-speed operation is less energy efficient than low speed computation for any of the technologies available to us. In Optimal Adiabatic Scaling, the number of devices in the processor increases rapidly enough from one generation of the product family to another that external memory and storage are not necessary. This eliminates the need to move data so quickly and allows more power-efficient operation.

In this project, Optimal Adiabatic Scaling has been applied to superconducting electronics through 3D-SCE and conventional room-temperature electronics. The Processor-In-Memory-and-Storage (PIMS) project devises an architecture that makes generous use of one of a class of non-volatile memory devices (Flash and the new options of memristors and Phase Change Memory (PCM)) as the internal memory and storage replacement. Appendix I includes a description of Optimal Adiabatic Scaling with the PIMS architecture, including performance upside calculations for the overall family. The

3D-SCE project is less mature, but comprises a new project start with activities in both 3D construction of superconducting processors and compatible memory devices.

The second high-level activity is a method of computing “ultimate limits” for neuromorphic computers, with the report in Appendix II. In the world outside of Sandia, there have been many ideas for using neural computing (sometimes called neuromorphic computing) as a successor to CMOS and the von Neumann architecture. These include Deep Learning, which is essentially programming simulations of neural networks on von Neumann computers or Graphics Processing Units (GPUs). Other approaches include using resistive arrays. Many of these other projects have been justified for reasons entirely different than computer throughput – such as research into the functioning of the human brain. Where these projects have claimed to support more effective computing, they have not necessarily had a way to predict potential effectiveness. The effort described in Appendix II can identify whether a neural implementation approach could be viable as a competitor against CMOS.

This effort may serve as a “divining rod” of sorts for many projects at Sandia and elsewhere. We give here a very brief description of the approach, by example. It is widely known that objects and information cannot move faster than the speed of light. If somebody has an idea for a machine and we can assess that it requires a component that moves faster than the speed of light, we could say the idea will not work without even constructing a test device. An “ultimate limits” analysis for neuromorphic computing uses somewhat more subtle principles, but along the same lines. There are minimum limits to the amount of heat that must be produced in computing due to thermodynamics. If the minimum heat of a proposed neuromorphic approach exceeds that of a computer readily available today emulating the same function in software, we will not need to construct a prototype to discover that the approach will not beat CMOS for energy efficiency.

The “ultimate limits” approach for neuromorphic computing has been applied to CMOS, PIMS, and some other approaches, showing PIMS to be a viable idea.

HAANA is the biggest successor project to this one, seeking to develop a hybrid computer that is especially efficient at neural network-class computations. The hybrid computer will comprise a standard von Neumann computer plus a neural network accelerator. The accelerator has yet to be defined, but it is likely to be based on work of this project.

This LDRD project has created an “alliance” with other projects. Physical science, materials, and devices projects have made their research available as building block for new architectures and computing concepts. The leading Sandia researchers in memristors, carbon nanotubes, lithium-based devices, spin-based logic, and SCE (superconducting) technology have participated in the meeting and device assessment discussions. This contributed to a new start SCE project and a paper on spin-based logic.

In conclusion, this project has pulled together many ideas from across Sandia into a computing concept and could possibly offer a power efficiency improvement for computing on the order of 80,000× (including both device-level improvements from physics and improvements due to computer architecture). This would be equivalent to about 16 generations of Moore's Law.

Project status

Projects will be listed below, along with publications.

Due to the short timeframe of this project, two likely publications are not yet ready for submission by the due date of this report. The manuscripts are in appendices I and II. The ideas in these manuscripts are due to be used in projects funded starting October 1, 2014. The intention is to publish the manuscripts once the ideas have been reviewed and vetted.

A review of spin wave logic is also included as Appendix III.

Subprojects in materials and device research

Materials and device research has been ongoing at Sandia for decades. Early efforts under this LDRD to extend a recently published comparison across a diverse set of emerging device technologies leveraged the deep expertise in device physics within Sandia. Device physics subject matter experts contributed to the review and comparison of emerging technologies and extended their own device physics research. The latter resulted in several additional publications on device technology development:

- François Léonard: Carbon nanotubes, papers cited
- Matt Marinella: Memristors, papers cited
- Nancy Missert: Superconductors, project started
- Peter Sharma: Magnetic Logic, document in Appendix III
- Alec Talin: Lithium-based devices, papers cited

Subprojects in computation research activities

The following ideas emerged during the time span of this LDRD, all of which represent computer science concepts that provide a justification for specific technologies

- Erik DeBenedictis: Optimal Adiabatic Scaling and Processor-In-Memory-and-Storage (PIMS). Fully due to this LDRD. Document in Appendix I.
- Erik DeBenedictis and Brad Aimone: Artificial neural networks as Beyond CMOS devices. Fully due to this LDRD. Document in Appendix II.
- Jeanne Cook: Processor-In-Memory-and-Storage (PIMS). Fully due to this LDRD. This idea has become a FY15 LDRD project.

The following activities emerged during the time span of this LDRD and with the connection specified

- Conrad James: Hardware Acceleration of Adaptive Neural Algorithms (HAANA), funded Grand Challenge project. The Artificial Neural Network power analysis and use of Deep Learning as a target applications area carried through to HAANA, although HAANA includes other activities not due to this LDRD.
- Nancy Missert: Three-dimensional SuperConducting Electronics (3D-SCE) (an implementation of Optimal Adiabatic Scaling, above), funded FY15 LDRD project. Nancy also developed an approach to superconductor memory unrelated to this LDRD.

References

Publications

1. V. Stavila, A. A. Talin, and M. D. Allendorf, “MOF-based electronic and optoelectronic devices.” Chem. Soc. Reviews DOI: 10.1039/C4CS00096J (2014)

2. A. A. Talin, A. Centrone, M. E. Foster, V. Stavila, P. Haney, R. A. Kinney, V. Szalai, F. El Gabaly, H. P. Yoon, F. Léonard, M. D. Allendorf, “Tunable Electrical Conductivity in Metal-Organic Framework Thin Film Devices.” Science 343, 66 (2014)

3. M.J. Marinella and V.V. Zhirnov, “Emerging Memory Devices: Assessment and Benchmarking,” in Emerging Nanoelectronic Devices, A. Chen, J. Hutchby, V. Zhirnov, G. Bourianoff, eds. Wiley, 2014.

4. V.V. Zhirnov and M.J. Marinella “Memory technologies: Status and Perspectives,” in Emerging Nanoelectronic Devices, A. Chen, J. Hutchby, V. Zhirnov, G. Bourianoff, eds. Wiley, 2014.

5. (invited) M.J. Marinella, “Emerging Resistive Switching Memory Technologies: Overview and Current Status,” in Proc. IEEE Int. Symp. on Circuits and Systems, Melbourne, Australia, June 2014.

6. A. Chen, M. Marinella, E. DeBenedictis, et. al. (about 100 authors in total), “Emerging Research Devices,” section of International Technology Roadmap for Semiconductors (ITRS) 2013 edition, http://www.itrs.net

7. Cummings, AW, J Varennes, F Léonard. (2013), “Electrical contacts to 3-D arrays of carbon nanotubes.” IEEE Transactions on Nanotechnology, 12, 1166.

8. Kane, AA, AC Ford, A Nissen, KL Krafcik, F Léonard. (2014). “Etching of Surfactant from Solution-Processed, Type-Separated Carbon Nanotubes and Impact on Device Behavior.” ACS Nano, 8, 2477.

9. Catalin Spataru and François Léonard. (2014). “Fermi-level pinning, charge transfer, and relaxation of spin-momentum locking at metal contacts to topological insulators.” Physical Review B, in press.

10. V. Klee, A. Talin, François Léonard et al. “Composition-Dependent Superlinear Photocurrent in CVD-Grown Monolayer MoS2(1-x) Se2x Alloy Devices,” submitted.

Documents in preparation

1. Erik DeBenedictis, “A Computing Paradigm Beyond Moore’s Law and Beyond the von Neumann Architecture,” appendix I.

2. Erik DeBenedictis and Brad Aimone, “Artificial Neural Networks as Beyond CMOS Systems,” appendix II.

3. Peter Sharma, “Spin wave computing,” appendix III.

4. Peter A. Sharma, “Obstacles for Collective-Spin Devices: Spin wave computing and Nanomagnet logic,” ERD Emerging Logic Assessment Workshop (August 26-28, 2014).

Presentations

1. (invited) Stanford University, Dept. of Materials Science and Engineering, “Molecule@MOF: A New Class of Electronic Materials.” 05/30/2014

2. P. Sharma, “Critique of spin wave computing and spin logic devices.” Invited presentation, ITRS Emerging Research Devices workshop, August 2014.

3. F. Léonard, “Electrical contacts to nanoscale materials and devices,” Stanford Electrical Engineering Seminar, October 2013.

4. F. Léonard, “Charge injection and transport in Nanomaterials.” UC Davis Physics Seminar, January 2014.

Workshops

ITRS Emerging Research Devices workshop, Albuquerque, NM. August 2014. Organized by Sandia (M. Marinella & E. DeBenedictis).

Note: The LDRD also hosted several internal workshops.

Appendix I: A Computing Paradigm Beyond Moore's Law and Beyond the von Neumann Architecture

A Computing Paradigm Beyond Moore’s Law and Beyond the von Neumann Architecture

Erik P. DeBenedictis, August 16, 2014

Abstract

Moore’s Law has driven the semiconductor industry for decades, principally through progressively more powerful microprocessors. The semiconductor scaling that has driven Moore’s Law is reaching physical limits and creating a crisis of sorts. We will show another set of principles that we call Optimal Adiabatic Scaling that has a similar effect to Moore’s Law, but which applies when manufacturing costs decline even if devices do not change at all.

We will use the effect above in conjunction with a new computer architecture, achieving a synergy that further boosts energy efficiency. Optimal Adiabatic Scaling will require a large number of devices. We propose to use this characteristic to advantage through a Processor-In-Memory-and-Storage (PIMS) architecture that stores persistent data in the same structure that does the computing. This will resolve certain issues limiting computer performance today, like the “memory wall.”

We build a systems software and algorithms approach around the physical rules in the paragraphs above. The combined effect preserves the idea of a computer with data files. However, the new approach distributes and reformats data to support algorithms that are more efficient in two ways. First, the algorithms may be more efficient in the conventional sense of having fewer steps. Second, the algorithms may run with higher power efficiency per operation by being a better match for Optimal Adiabatic Scaling.

A very rough analysis of performance suggests 80,000× improvement in energy costs for popular computing kernels could be possible over time, about half from electronics and half from architecture and algorithms.

Introduction

The concept of computing experienced an inflection point during WW II [Nordhaus 07]. Prior to WW II, computers comprised humans operating tools such as pencil and paper, abacus, etc. This meant compute power was limited by the speed and capacity of the human operator. Humans started to make programmable computers that could operate beyond the limits of a human operator during WW II. The von Neumann architecture was preeminent in these designs [von Neumann 45] as it allowed the growth of application complexity based on software, which was very scalable. The economic activity from the applications drove technology development of the underlying electronics and enabled exponentially more capable computers over time in a trend that continues through present.

In 1965 Gordon Moore wrote a paper [Moore 65] subjectively describing what later became Dennard Scaling [Dennard 74]. Moore projected exponential growth of various integrated circuit properties for 1965-1975 (but it continued well beyond 1975). The microprocessor was invented in 1970 (so it cannot be seen as inspiration for Moore’s

Law) and has over time supported the exponential path of the semiconductor industry. Moore's Law has now merged with the overall growth of computing and is nearly synonymous in today's lingo, but it is important to note that they are different.

There is a growing realization that the Dennard scaling that defined the most general form of Moore's Law has ended already. However, it is possible that the computer industry could switch to something other than CMOS and the von Neumann architecture and continue the larger technology trend that started in WW II.

Energy efficiency is currently seen as the limiting attribute of computer technology. Most applications can be made sufficiently fast by running computers in parallel, so speed is not a problem. The number of devices on a chip continues to grow into the third dimension at reasonable cost [Fujisaki 13], so hardware complexity is not a problem. However, energy per calculation is beginning to flat line. Eventually, this will cause new applications that are inevitably more complex to consume too much energy to be affordable to the user. This will stymie the business model that funds the industry and create a crisis.

Industry and government are responding in different ways. Industry [ITRS YY] is shifting attention from information processing throughput to mobile devices such as smartphones and the Internet of Things. This would shift industry to a new business model that may be profitable but would mark an end to the growth in computer capability. Some segments of government focus on other goals. Supercomputers rose through Gigaflops, Teraflops, Petaflops. Some government agencies (principally DOE) are funding sizeable technology advances to make supercomputers up to a few Exaflops at reasonable cost. There is related interest in similarly sized data centers that contain a scalable number of servers rather than a single scalable supercomputer. This would continue the computer performance growth trend.

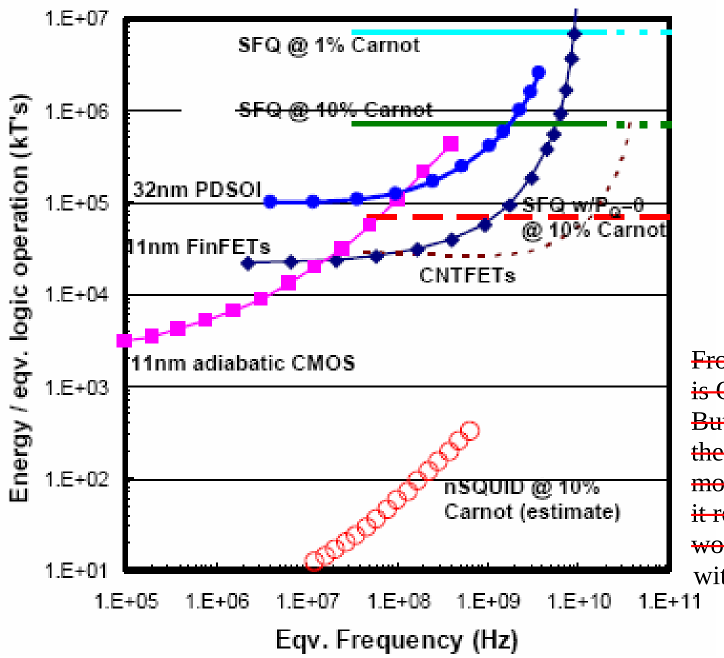
Other government agencies are pressing for neural networks or artificial intelligence, which are quite different.

Adiabatic and reversible computing represent technology options that offer the possibility of computing at "arbitrarily low energy levels." The options involve passing signals from one logic stage to another with loss of only a small fraction of the signal energy each time. This is in contrast to current (CMOS) logic, where all signal energy is turned into heat and regenerated from the power supply between each logic stage. Adiabatic computing is readily seen as cutting energy consumption 10-100× [Karakiewicz 12], although the number of devices per gate is greater than CMOS. For the purposes of this discussion, reversible computing builds upon adiabatic computing. After significant gains in power efficiency have been realized through adiabatic computing, reversible computing could offer even more gains. Neither adiabatic nor reversible computing has received much attention.

Memory and data storage have evolved in accordance with Moore's Law as well, with progress destined to slow for similar reasons. The forefront of memory and storage is the shift from moving media (disks and tapes) to technologies that use chips physically similar to those in processors but with novel devices such as Flash, Resistive RAM (ReRAM, e. g. memristors), Phase Change Memory (PCM), and Spin Torque Memory (STM). These technologies are being packaged in 3D, which allows continued growth in device complexity. Memories tend to dissipate less heat, which has been critical to practical exploitation of 3D without overheating.

Energy efficiency of logic families

The energy efficiency of most logic families varies by operating speed, as illustrated in Figure 2. While the curves illustrated in Figure 2 required extensive simulation or measurement to produce in detail, key high-level features are readily visible can be described in words. Most logic families have a minimum energy per operation that occurs at a low speed. This corresponds to the curves having zero slope on the left; all curves would go through zero slope if the graph were big enough. All logic families have a top speed. The slope of the energy per operation curve rises faster and faster as the top speed is approached, eventually reaching a singularity where it essentially shoots straight up. Obviously, the curves will go through a slope of 1 in between these extremes.



~~From D. Frank, IBM: "it is OK to use this figure. But please do not put it in the public domain, since most of the data points on it represent unpublished work."~~ From David D. Frank, with permission (1/17/2016)

Figure 2: Energy/op vs. speed tradeoff for various logic families.

Regions of the curves at or close to slope 1 are following something called the "adiabatic principle," although the tie to the adiabatic principle in physics has not been formally documented. The physical principal of a linear rise in energy for a computer appears to have first been described in Ref. [Bennett 73], p. 531 and more generally described in Ref. [Feynman 96] pp. 167-172. The use of the term "adiabatic" comes from adiabatic capacitor charging, apparently appears first in Ref. [Koller 92] in the context of practical circuits.

Optimal clock rate

We will devise a scaling rule by first discovering an economic principal from an exercise to optimize the clock rate and then applying the economic principle to scaling. The principle gives us a shortcut to picking the clock rate or frequency that yields optimal economic return over the lifetime of a computer. In today's environment, it makes sense to consider both the purchase price of a computer and cost of energy in operating a computer over its lifetime. These concerns lead to an optimization exercise. We would like to find the minimum cost per computer operation. This would be

$\min (\$_{purchase} + \$_{energy})/Ops_{lifetime}$.

Let us say the chips cost $\$_{purchase} = A$ to purchase.

For various reasons that the reader will certainly appreciate by the end of this analysis, the viable economic zone will be in a part of the curve in Figure 2 that is close to slope 1 – maybe slope ½ or 2 but not slope 20. Where the curves in Figure 2 have slope 1, the energy per operation will be proportional to f and the number of operations per second will be f as well, so the energy consumed over any fixed time period will be proportional to f^2. If we choose that time period to be the lifetime of the computer, there will be a constant factor B such that $\$_{energy} = Bf^2$; this will capture the frequency dependence of the energy cost even though we will never need to find the value of B.

The total number of ops over the machine's lifetime will be proportional to f, so we can say $Ops_{lifetime} = Cf$, where C is a constant factor.

We have now transformed the problem into finding f that minimizes $(A + Bf^2)/(Cf)$. To find the optimal value of f, we would set the derivative with respect to f to zero.

$\mathrm{d}/\mathrm{d}f\ (A + Bf^2)/(\mathrm{C}f) = 0$.

By simple calculus, this has the solution $f = \sqrt{}(A/B)$.

A simple economic principle emerges when we substitute the optimal value of f into the expressions:

$\$_{purchase} = A$, as before and

$\$_{energy} = Bf^2 = B\ (\sqrt{}(A/B))^2 = A$, which is the same value as in the line above.

So the economic principle is that wise users should pick their clock rate such that they spend the same amount of money buying the chips as they spend powering them!

Optimal Adiabatic Scaling

The optimal clock rate principle leads to a semiconductor scaling rule. Say that we have a computer that is operating at the optimal clock rate. The chip maker offers us upgraded chips with 4× as many of the same devices for the same cost per chip – in other words, a 4× reduction in cost per device. What do we do? If we simply stick the new chip in the socket intended for the previous generation, the system will consume 4× as much power and move away from the optimal balance of purchase price and energy cost.

The solution is to turn the clock down by a factor of $\sqrt{4}$× = 2×. With the clock turned down, power consumption will be reduced by 2^2× = 4× from the higher level. In fact, the per-chip power consumption will be exactly the same as the previous generation. The system will be returned to the optimal clock rate that minimizes the cost per operation.

By turning down the clock rate, system throughput will be lower than it might have been, but there would be 4× as many devices running at ½× the speed. This will still yield a 2× increase in throughput. This is like Moore's Law, but not as strong.

A scaling rule results if the specific factor of 4 used in the discussion is replaced by continuous variable β and squares and square roots are applied consistently. The factor α in typical semiconductor scaling refers to line width so the number of devices grows with α^2. To reduce confusion, we will say $\beta = \alpha^2$ and clarify that β represents device count. Table 2 comprises the table of scaling rules from [Theis 10] with Optimal Adiabatic Scaling on the right. Ignore the references to layers (which will be discussed later).

Table 2: Scaling rules from [Theis 10], with Optimal Adiabatic Scaling on right

	Const field	Constant V				Optimal Adiabatic Scaling
		Max f	Const f	Const f, N_{tran}	Multi core	
L_{gate}	$1/\alpha$	$1/\alpha$	$1/\alpha$	$1/\alpha$	$1/\alpha$	1[*]
W, L_{wire}	$1/\alpha$	$1/\alpha$	$1/\alpha$	1	$1/\alpha$	$\beta = \alpha^2$[†]
V	$1/\alpha$	1	1	1	1	1
C	$1/\alpha$	$1/\alpha$	$1/\alpha$	1	$1/\alpha$	1
$U_{stor} = ½CV^2$	$1/\alpha^3$	$1/\alpha$	$1/\alpha$	1	$1/\alpha$	$1/\sqrt{\beta} = 1/\alpha$[‡]
f	α	α	1	1	1	$1/\sqrt{\beta} = 1/\alpha$
N_{tran}/core	α^2	α^2	α^2	1	1	1
N_{core}/A	1	1	1	1	α	$\beta = \alpha^2$
P_{ckt}	$1/\alpha^2$	1	$1/\alpha$	1	$1/\alpha$	$1/\sqrt{\beta} = 1/\alpha$
P/A	1	α^2	α	1	1	1[§]
$f\,N_{tran}\,N_{core}$	α^3	α^3	α^2	1	α	$\sqrt{\beta} = \alpha$

[*] Term redefined to be line width scaling; 1 means no line width scaling

[†] Term redefined to be the increase in number of layers; previously was 1 for no scaling

[‡] Term redefined to be heat produced per step. Adiabatic technologies do not reduce signal energy, but "recycle" signal energy so the amount turned into heat scales down

[§] Term clarified to be power per unit area including all devices stacked in 3D

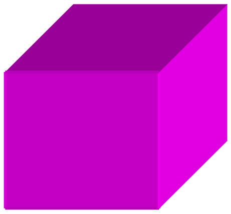
3D manufacture and distant future vision

Optimal Adiabatic Scaling fits nicely with current trends in semiconductor manufacturing, although it seems likely that the intergenerational interval will increase. There is a current reality that line width scaling will come to an end in a few more generations. Along with this realization, there is a lot of attention being given to exploitation of the third dimension. An extrapolation of current trends could possibly lead to chips reaching some “final” line width (say 7 nm) and associated parameters for a device at that size. Scaling would then proceed exclusively into the third dimension by having progressively more functional layers.

The number of layers could grow in accordance with a Moore’s Law-type phenomenon. However, it seems unlikely that historical growth rates will be maintained. Moore’s Law has doubled device count about every 18 months due to a $1/\sqrt{2}$ reduction in line width. While we lack rigorous data on the following statement, matching this rate with layering would require a doubling of the number of layers each 18 months. This seems unlikely, but maybe half the rate is feasible.

If we apply Optimal Adiabatic Scaling to 3D scaling, we could create a type of end-game vision as illustrated in Table 3. As this document is written, the largest DRAM chips are 8 GBits, about 10^{10} transistors, or nominally a square array of $10^5 \times 10^5$ devices. If we make the straightforward extension to 3D, we end up with 10^{15} devices of 100 nm^3. This would correspond to $\alpha = \sqrt{10^5} \approx 300$ in Table 2.

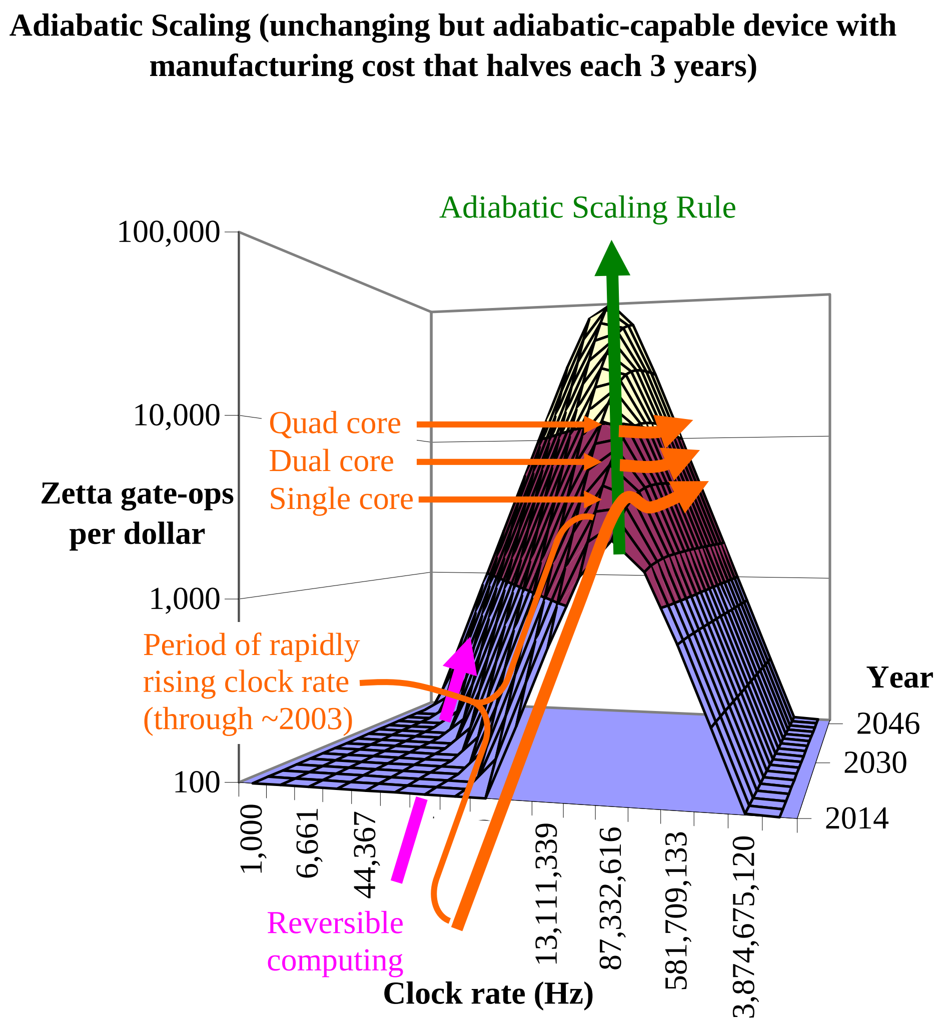
Table 3: Vision of 3D scaling

Timeframe	**Today**	**Changes**	**End-Game Vision**
Integration scale	10^{10} logic transistors	$\alpha = \sqrt{10^5} \approx 300$; $\beta = 10^5$	10^{15} logic transistors
Clock speed	3 GHz	$1/\alpha \approx 1/300\times$ (slower)	10 MHz
Performance	Chip is 2D comprised of 100 nm^2 gates. 	$\alpha \approx 300\times$ reduction in joules/op OR $\alpha \approx 300\times$ increase in energy efficiency	Chip is 3D comprised of 100 nm^3 gates. 
Power per cm^2	1	1	1

The remainder of the physical system may stay the same. In Optimal Adiabatic Scaling, the power emerging from the bottom face (say the heat sink is connected to the bottom face) stays the same. This would support the idea that the manufacturer offers a drop-in replacement with 4× as many components.

Progression over time

The discussion above has covered a series of ideas whose interactions are summarized in Figure 2. We warn the reader that there are too few dimensions on the printed page to illustrate both Optimal Adiabatic Scaling and device scaling, so the diagrams will not be accurate beyond the ideas explained in this document. Figure 2 is a partially calibrated plot of the economic viability (Ops/$) as a function of clock rate and time. The plot assumes Moore's Law will continue to produce devices at lower cost, but device performance will not change at all.



Plot based on equations developed in this document. The device must be adiabatic-capable, but with unchanging performance. However, manufacturing cost halves each 3 years.

Figure 3: Adiabatic system performance.

The purchase price of computers dominated energy costs prior to about the year 2000, so the best strategy was a rapid rise in clock rate. This occurred most conspicuously during the 1990s, as illustrated by the long rising segment of the orange arrow in the front-center of the plot. The rapid rise in clock rate went over the optimal speed around the year 2000. This led to dual-core processors being introduced in about 2003, which initially had a lower clock rate (2-2.5 GHz as opposed to 4 GHz in the single-core processors they replaced) and were closer to optimal. The number of cores has trended upwards at nearly constant clock rate since. The graph in Figure 2 shows a declining clock rate (which is not correct) because the semiconductor industry continued to improve devices in this era (which is not captured in the graph).

Figure 3 shows the path to the future as well. The green arrow at the "top of the ridge" for Optimal Adiabatic Scaling proceeds upwards in power efficiency and economic return. It proceeds downwards in clock rate at the square root of the scaling rate, just as projected in Table 2.

The projected trend can be viewed as a measured blend of the original Moore's Law (orange) and adiabatic or reversible computing (magenta).

Moore's Law (in orange) has been pushing up clock rate too fast to be economically optimal. Since 2003, Moore's Law has pushed processors over the crest of the crest of the hill (clock rate too high) and industry has responded by the architecture change of adding more cores.

Reversible computing (in magenta) takes power efficiency beyond the point of diminishing returns to a point where it is economically harmful. Reversibility is required to achieve an energy per gate operation below "on the order of kT." This condition would occur (if the proper designs are used) at some specific and low operating frequency, a frequency unrelated to the manufacturing cost of the chips. The magenta arrow on Figure 2 therefore follows the plotted surface at constant frequency (a frequency picked by the authors for artistic convenience). The reversible computer would have a very low clock rate and thus require a very large increase in the overall size of the computer to achieve a given throughput. Reversible computing thus appears to the far left of the graph in a region of rising but currently poor economic viability. The arrow for reversible computing will eventually intersect the arrow for optimal adiabatic computing.

Synergy of computer architecture and Optimal Adiabatic Scaling

The 3D computer vision illustrated in Table 3 could either be a cube of logic (as a microprocessor is a square of logic) or both storage and logic together (like a complete computer system today – or a brain). We will discuss both possibilities.

Figure 4 represents different scenarios of data that are stored and processed. Assume data are stored in the 3D computer in Table 3 by being evenly distributed across all or some subset of the physical structure of the computer. The Optimal Adiabatic Scaling derived previously and summarized in Table 2 will be correct for an entire computer, and it will also be correct if data is located on a constant fraction of the computer, like 50%, 1%, etc. However, it will not be correct if data is present on a progressively smaller subset of the entire computer as the problem scales, such as only $\sqrt{N}$, log N, or a fixed number of devices.

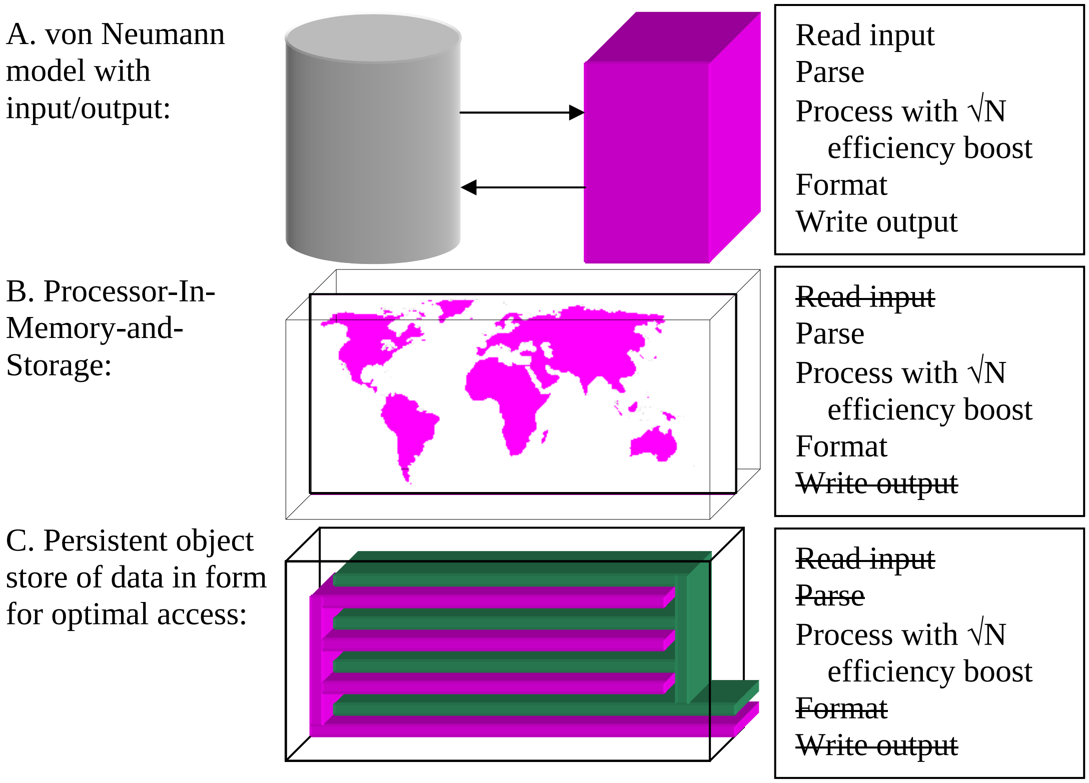

A. von Neumann model with input/output:
Read input
Parse
Process with √N efficiency boost
Format
Write output
B. Processor-In-Memory-and-Storage:
~~Read input~~
Parse
Process with √N efficiency boost
Format
~~Write output~~
C. Persistent object store of data in form for optimal access:
~~Read input~~
~~Parse~~
Process with √N efficiency boost
~~Format~~
~~Write output~~

Figure 4: Scaling scenarios

Figure 4A and Figure 4B illustrate data subsets that scale with different powers of a fundamental data element size.

Figure 4A illustrates data (in magenta) that is uniformly distributed across the entire device structure of N components during processing, which implies the data scales with N^1 (i. e. exponent 1). This is the common situation today where data is stored on disk drives. The cost of the disk drives is not considered along with the cost of the processor (and memory) chips, but input/output to the disk drive and associated data parsing and formatting contribute an overhead to the overall running time of the program. With Optimal Adiabatic Scaling, this scenario achieves the $\sqrt{N}$ boost in energy efficiency of the processing step.

Figure 4B illustrates a second scenario that has behavior different from a von Neumann computer. Say that the purpose of a computer system is to process maps. We shift from the current approach of storing maps on disk drives to storage in the same 3D structure as the processor (imagine a Hybrid Memory Cube but with Flash storage chips instead of DRAM [HMC 14]). In this scenario, a given map could be stored as a planar slice through the 3D structure of the system – with other maps on different slices. If the data is present in the processor before and after the processing, input and output to the disk drive can be eliminated. While this does not improve processing speed as defined for von Neumann computers, it introduces a new computer concept where certain functions previously attributed to overhead are eliminated altogether.

Including both the standard von Neumann scenario and the new scenario, the exemplary map processing could proceed in two ways:

1. The map processing example could proceed in accordance with Figure 4A by treating the system as an overlay of a storage system and a processor. The data in the planar slice could be redistributed to the entire processor before processing and back to the slice afterwards through the equivalent of input and output operations. This scenario would only benefit from the $\sqrt{N}$ boost in energy efficiency of the processing step.

2. However, map processing could proceed by using only the 2D slice of the 3D computer system that actually holds the map, as illustrated in Figure 4B. To define this scenario precisely, the 2D slice holding the map illustrated in Figure 4B must have a “thickness” (a concept that will be explained below). The devices in this slice would include both storage devices and processing devices. If the thickness were constant, the volume of the 2D slice would grow with $N^{2/3}$. Let us analyze this more precisely, but first generalize from a system using $N^{2/3}$ out of N devices to hold data to a system using N^{δ} out of N devices to hold data. Figure 4A and Figure 4B now illustrate δ=1 and δ=2/3 respectively.

Let us repeat the scenario of a chip maker offering an upgrade to a new generation of chip with 4× as many devices at the same cost, but where activity is on only N^{δ} of the devices. In this case, the number of active devices grows by a factor of 4^{δ}. This would cause the chip to move off the optimal cost per operation unless the clock rate was lowered. By multiplying the clock rate by $4^{-\delta/2}$, power per unit device would drop by 4^{δ}, precisely balancing the increase in number of active devices. This would result in a higher speed clock but less of an energy efficiency boost than in the previous explanation. Specifically, clock speed and energy consumption per operation would decline by the smaller amount $4^{\delta/2}$ rather than $4^{½}$.

Figure 4C expands the vision from Figure 4B to a form that we will later give an architectural realization. Let us assume the computer system is used to store several data sets, with two such data sets illustrated in magenta and green in Figure 4C. Each data set will be stored in devices distributed in a user- or operating system-definable subset of the overall system. Let us say the regions have volume N^{δ}, irrespective of their shape.

Figure 4C suggests a further innovation related to the representation of data in a storage medium. Data has been stored in “interchange formats” historically. In the past, data has been stored in storage media that have several attributes that strongly suggest using machine-independent data representations. For example, a disk drive or removable media could be taken to a computer with a different architecture, suggesting encoding formats should be independent of byte order and numerical representation. Data files are also loaded and stored to different memory addresses each time they are accessed by the same computer, arguing against putting pointers or integer offsets into data files. However, the structure in Figure 4C would make it impossible to separate stored data from the computing structure without cooperation of the computing structure. This change in architecture suggests an object store approach that stores data in an opaque internal format and provides access by functions that reformat the data as needed. The scenario in Figure 4B already redistributes data spatially to minimize data movement during computing. The innovation in Figure 4C is to manipulate the format of stored data to maximize compute efficiency.

There is precedent for this method of data storage in the way we humans store data in our brains. When humans input a pdf file into their brains by reading it, they do not store the bits of the pdf file in their brains, but rather parse and process the document's ideas and store the ideas in a form optimized for making use of the ideas. Humans have some limited ability to reconstruct a document from memory, but the ability to do so is not considered as important as understanding its contents.

The programmer now has a choice of how to use the data, with implications to performance.

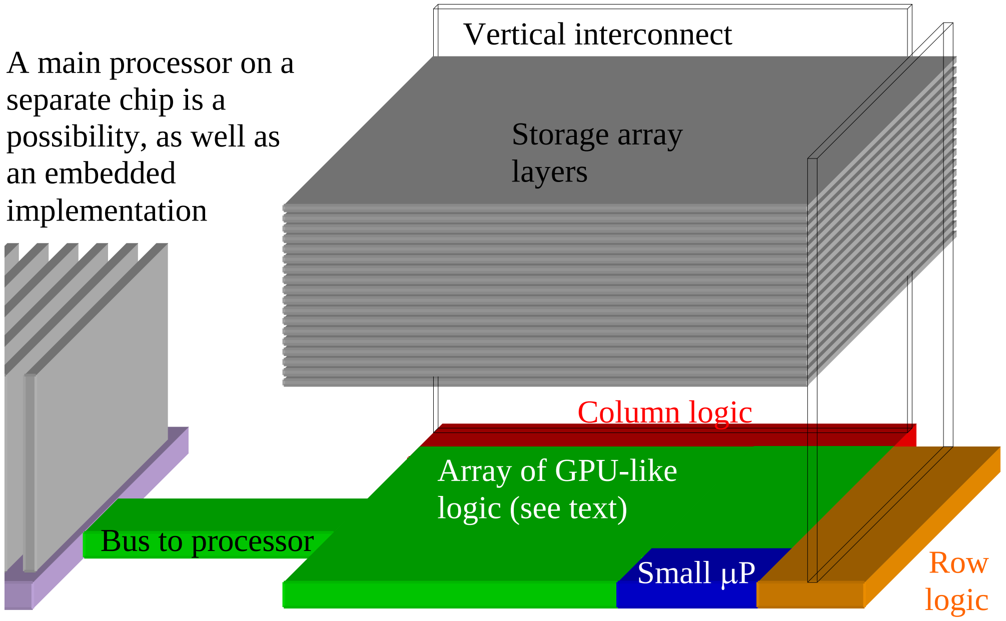
1. The programmer could choose to treat the illustrated 3D regions as storage and redistribute before processing as shown in Figure 4A. This would add overhead for input and output just like a von Neumann computer, but would achieve a $\sqrt{N}$ power efficiency improvement in the processing stage.

2. The programmer could leave the data in the original distributions as illustrated in Figure 4B (perhaps exploiting spatial locality between multiple data sets, as illustrated). This would eliminate the input and output steps and get $N^{\delta/2}$ power efficiency improvement. Since $\delta \leq 1$, $N^{\delta/2}$ will usually be less than then $N^{½}$ available in the first scenario.

3. The programmer could define an internal storage format that makes the data more amenable to the computations that will be performed on it. While this would require planning ahead, it could further eliminate the parsing and reformatting of data. For example, unstructured text files could be stored in an indexed form for fast searching. Like input and output, data parsing and reformatting are currently attributable to overhead and now counted as computing costs.

Processor-In-Memory-and-Storage (PIMS) implementation

Processor-In-Memory-and-Storage (PIMS) is the proposed architectural implementation and illustrated in Figure 5. Data would be stored in the 3D structure labeled "storage array layers." These layers could be Flash, ReRAM (memristors), Phase Change Memory (PCM), or some other option. The layers could also be DRAM as in [HMC 14], although such as system would not have persistent storage once the power was turned off. Each data set would be stored in a physical region like one of the structures in Figure 4C.


A main processor on a separate chip is a possibility, as well as an embedded implementation
Vertical interconnect
Storage array layers
Column logic
Array of GPU-like logic (see text)
Bus to processor
Small µP
Row logic

Proposed implementation of a single layer. In general, a full system would be a scalable array of the stacked structure replicated in 3D. The interface to a "main processor" on the left is optional. The small microprocessor (µP) illustrated in blue and the green structure described as array of GPU-like logic would form a viable computing system where control resides on the small microprocessor with energy-efficient computing performed on the GPU-like logic.

Figure 5: Processor-In-Memory-and-Storage (PIMS) implementation.

Data would be processed by an array of ALUs in an architecture like a stripped-down version of the GPU architecture widely used today. Data processing is likely to be programmed using languages descended from GPU languages like CUDA. It is well known that GPU architectures implement very high degrees of parallelism, which is a key requirement for adiabatic scaling. GPUs can have modest clock rates for a given degree of performance, which further boosts energy efficiency so they are viable on laptops without excessive power consumption. Data stored in formats like Figure 4C would be treated like 3D graphical objects (e. g. monsters in video games) or screen windows in a current GPU graphics card. The small µP and the GPU-like logic (ALUs) under control of a dual-architecture programming language (like CUDA) would manipulate the objects in the storage array layers (used like GPU memory).

The memory and ALUs comprising the GPU-like logic should scale adiabatically. The memory and ALUs will turn out to have similar levels of energy consumption (details on this appear later in the document). If either the memory or logic does not scale in accordance with Table 2, an imbalance will develop and it will not be possible to maintain the scaling rule.

In the context of Table 3, the vision would be a rapidly growing number of adiabatic ALUs operating at progressively lower clock rates, yielding more throughput and higher energy efficiency according to Table 2. However, we anticipate some architectural differences with current GPUs. GPUs devote many of their gates and resulting energy budget to functions specific to graphics, such as texture maps. Since the design space for Figure 5 is not specific to graphics, it is unlikely that functions designed for one purpose could be effectively repurposed. Due to these high-level considerations, we believe the proper design will be less complex than a typical GPU. The following example will illustrate this point.

Example of sparse matrices for neurons and meshes

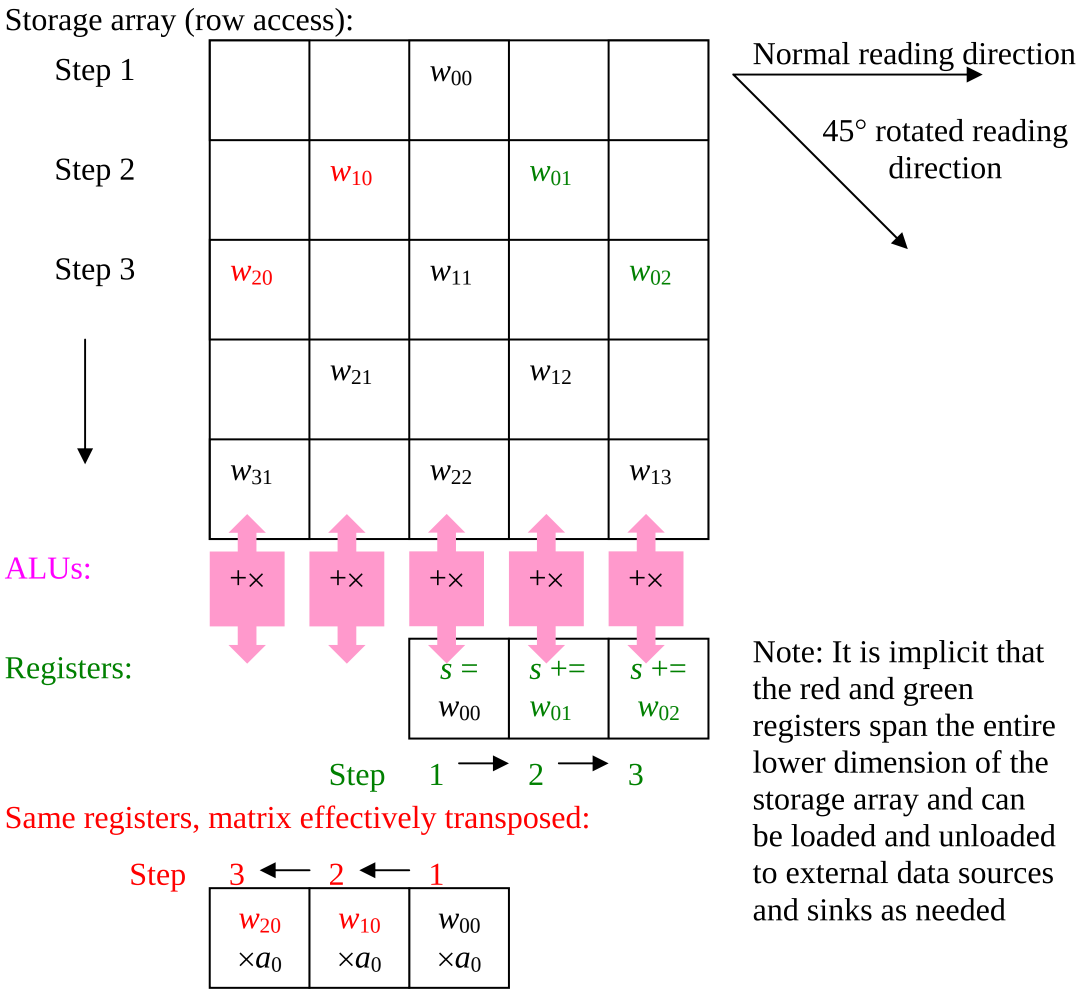
Sparse matrices illustrate many of the features described in this document and are key to both neural computing and supercomputer algorithms. Supercomputer algorithms include the Finite Element Method (FEM) and graph algorithms. The most compute-intensive activity in any of these is a sparse or dense vector-matrix multiplication. In an Artificial Neural Network (ANN), it is the multiplication of a vector representing neural axon output signals times a sparse matrix representing synapse weights. In current Deep Learning implementations of ANNs, the underlying arithmetic can be 8 or 16 bit integers or single precision floating point. However, multiplication by both a matrix and its transpose are required. Finite element methods on supercomputers create an irregular mesh around a structure to be simulated, such as a turbine blade. A compute-intensive step for simulating with meshes is the multiplication of a vector representing, say, force, at each point on the by a sparse matrix representing interconnections between mesh points. FEM simulations also require operation on both a matrix and its transpose but tend to use double precision floating point arithmetic. Graphs are often represented by matrices with nonzero elements for graph edges that exist and zeros where there are no edges. Graph algorithms can be represented by sparse matrices like ANNs or FEMs, but where the arithmetic may be non-existent (i. e. the sparsity pattern is the only data).

We will show how PIMS is an excellent structure for both the arithmetic and management of the memory layout. Due to the meshes being irregular, the computational effort is sometimes dominated by the non-arithmetic operations that manage the layout of numbers in memory as opposed to the actual arithmetic. The exposition of the computational structure will proceed in two stages: The first stage will be a dense matrix representation where both the matrix and its transpose can be accessed equally well. The second stage will extend the representation to sparse matrices.

The representation being discussed will be different from current GPU practice. In order to achieve the power efficiency necessary to support the conclusions of this document, the memory access pattern must be constrained. In the discussion below, the access pattern for the storage array is full rows, one at a time, from the top to bottom. This is compatible with demonstrated adiabatic memory technologies [Karakiewicz 12].

Directional transpose matrix representation

A key part of the strategy is to represent a matrix in a way that can be effectively transposed by reversing the processing direction from leftward to rightward. Figure 6 shows the matrix elements stored in a memory, but the matrix rows are rotated 45° clockwise from the memory rows and from the normal way matrices are illustrated. The storage array rows are read from top to bottom in subsequent steps, each step delivering all the matrix elements in the memory row to the rose-colored ALUs at the bottom.



Note: It is implicit that the red and green registers span the entire lower dimension of the storage array and can be loaded and unloaded to external data sources and sinks as needed

Figure 6: Matrix organization that transposes by processing direction

The processing algorithm will involve multidimensional movements similar to a square dance or a contra dance, where gents move one way across the ALUs and ladies move the other. In this description, the memory drops matrix elements onto positions on the dance floor corresponding to ALUs.

Let us describe a warm-up exercise for the required the motions of a vector-matrix multiply with an algorithm that merely sums the elements in all the rows of the matrix. Gents will be assigned accumulating sums, initially zero. Imagine you are a gent standing on the central ALU in Figure 6. In step 1, the matrix element w_{00} falls from the memory and is added to your accumulating sum (which is zero, so you compute $s = w_{00}$ as illustrated). In the next step, w_{01} falls from the memory to the ALU to your right. So you move to the right and add w_{01} to your accumulating sum (computing $s \mathrel{+}= w_{01}$ as illustrated). If the gents move one ALU to the right between each step, adding the values that fall from the memory will perform row-wise summation. To sum columns, the gents to go the left instead.

The ladies perform the second activity that will be needed for vector-matrix multiplication. Each lady will represent a vector element. A little thought reveals that the ladies should move in the opposite direction of the gents for vector-matrix multiplication. If addition is performed across a row, then each vector element will be multiplied by every matrix element in a column – and vice versa. Imagine you are a lady standing on the central ALU in Figure 6. In step 1, the matrix element w_{00} falls from the memory and you multiply it by your vector element. You thus compute $w_{00}\times a_0$. In the next step w_{10} falls from the ceiling to the ALU on your left. So you move to the left and compute $w_{10}\times a_0$. If the ladies move one ALU to the left during each step, multiplying the value they represent by the values falling from the memory will perform all the multiplies required for vector-matrix multiplication. To effectively transpose the matrix, the ladies to go the right instead.

The full vector-matrix multiply algorithm requires the ladies and gents move in opposite directions at the same time. In each step, a matrix element falls from the memory. The lady catches it and multiplies it by the vector element she represents. She hands the product to the gent, who adds it to his accumulating sum. The lady and gent then move in opposite directions for the next step.

Sparse matrices

The proposed enhancement to accommodate sparse matrices would make the human dance much less fun, but can be readily described in dance terms. In the enhancement, each matrix element would have two additional pieces of information that specify the sparsity pattern. The matrix elements in Figure 6 are enhanced with green and red pointers, as shown in Figure 7. Each pointer describes the location of the next element in the same row and same column. For example, a 2-bit codeword for the column could specify the next row element is (a) to the left, (b) left and down, (c) two to the left, (d) two to the left and down. (A second 2-bit codeword would be required for the row, for a total of 4 bits in this example.) In the terminology of the dance, the lady and gent will each treat one pointer or codeword as an appointment to show up at the specified time and place to perform their next math operation with a partner.

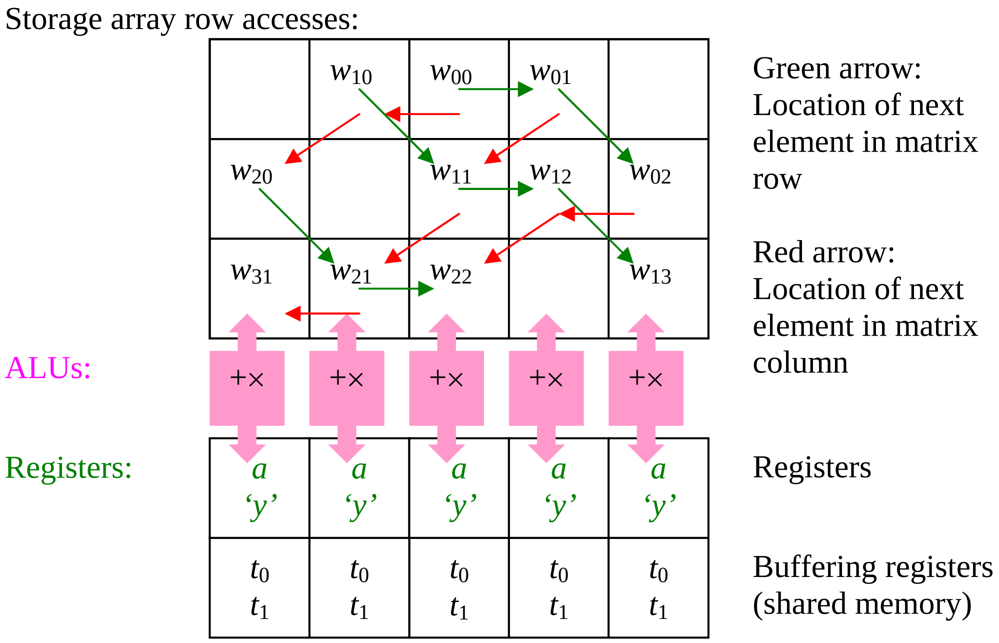


Figure 7: PIMS organization as extended for sparse matrix operations

A holding area will be required. As illustrated in Figure 7, the architecture is augmented with holding cells t_0, t_1, etc. If a lady or gent does not have a math operation in some step, they must wait in the ALU area. While the ladies and gents are fictions, the ALU will require storage locations to hold their vector elements or accumulating sums.

The method was described concisely and without consideration to resource conflicts. In actual practice, it would be necessary for the motion pattern corresponding to a sparse matrix to avoid multiple ladies or gents trying to get between the same pair of ALUs and causing overload of ALU-ALU connections. Furthermore, a system would be constructed with some specific number of cells in the holding area. A sparsity pattern could fail if it required more holding cells than had been constructed. To avoid failure, an algorithm could reallocate the positions of the matrix elements in the memory. This might include adding extra columns to the memory to make more holding cells available, a solution that would reduce efficiency (which is an inevitable tradeoff).

ALU complexity

The authors anticipate a GPU-like logic for the array of ALUs. To support this point, we will identify the required data flows for the ALUs in this example and let the reader decide if “GPU-like” is an acceptable description. A potential design for an ALU for sparse matrix processing is illustrated in Figure 8. The illustration uses 8- and 16-bit integers as are sufficient for some ANNs in Deep Learning, but the block diagram would be the same if the data types were change to single- or double-precision floating point.

Storage array format:

Synapse value: 8 bits as signed integer, but often interpreted at a higher level as a fixed point number	Green pointer code word	Red pointer code word
12 bits total: 8 bits +	2 bits +	2 bits

ALU (one for each 12 storage bits):

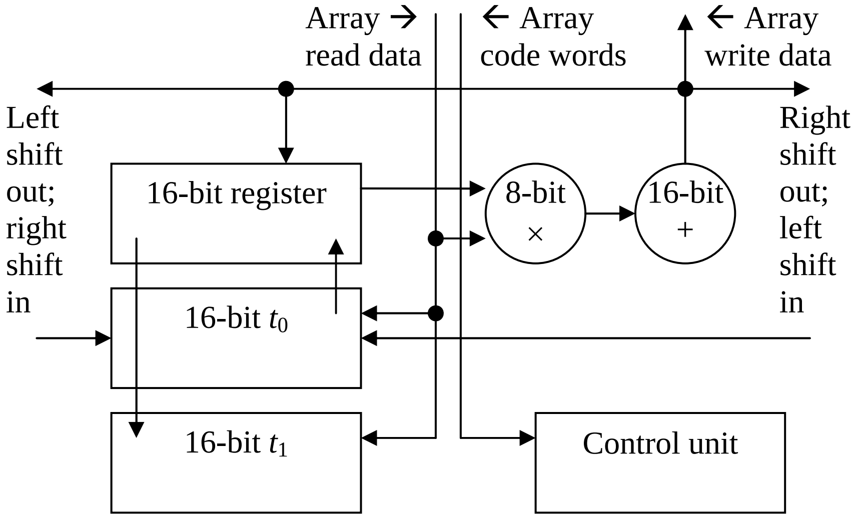


Figure 8: ALU data flows for sparse matrix calculations

While the overall array of ALUs would be Single Instruction Multiple Data (SIMD), each ALU would shift one of its intermediate products left or right so the intermediate products match up with the sparse matrix elements at the right place and time. Thus, each control unit would look at the code words encoding the sparsity structure and decide which intermediate products to send left, right, and into the temporary registers *t*. The structure shown has the basic elements of a stream processing unit in a GPU, but the illustration is only a single application. Anticipated designs would likely have several times the complexity in order to achieve the necessary degree of generality.

Power consumption

Let us make an argument that the design shown in Figure 5 and Figure 8 could be power efficient in the short term and progressively more power efficient in the future. Let us identify technologies that have been demonstrated, albeit not in combination. This will yield an optimistic but not impossible result and with some guidance for each sub technology. Let us do this for an ANN system of human brain proportions of 10^{11} neurons, 10^{15} synapses, and a 20 Hz update (i. e. brain wave cycle) rate. Let us initially assume the system is a collection of chips like Figure 5; whether these chips are themselves stacked will be considered later.

A rough sketch of such a system is shown in Figure 8. Let us assume an equivalent storage density to a state-of-the-art Flash memory chip containing 64 GBytes or 512 Gbits (e. g. Micron MT29F512G08CUCAB [MT29F512G]). Since a synapse is represented by 12 bits, this implies $64\times2^{30}\times8/12 = 4.6\times10^{10}$ synapses per chip, or 10^{15} / 4.5×10^{10} = 21,800 chips will be required to implement 10^{15} synapses. If a memory or storage bank is 1000 × 1000 bits, each PIMS chip could contain 55,000 × 10 one-megabit banks, representing 4.58×10^{6} × 10,000 synapses. The structure just described would have the correct ratio of 10,000 synapses to each neuron.

PIMS chip equivalent to 64GByte Flash:

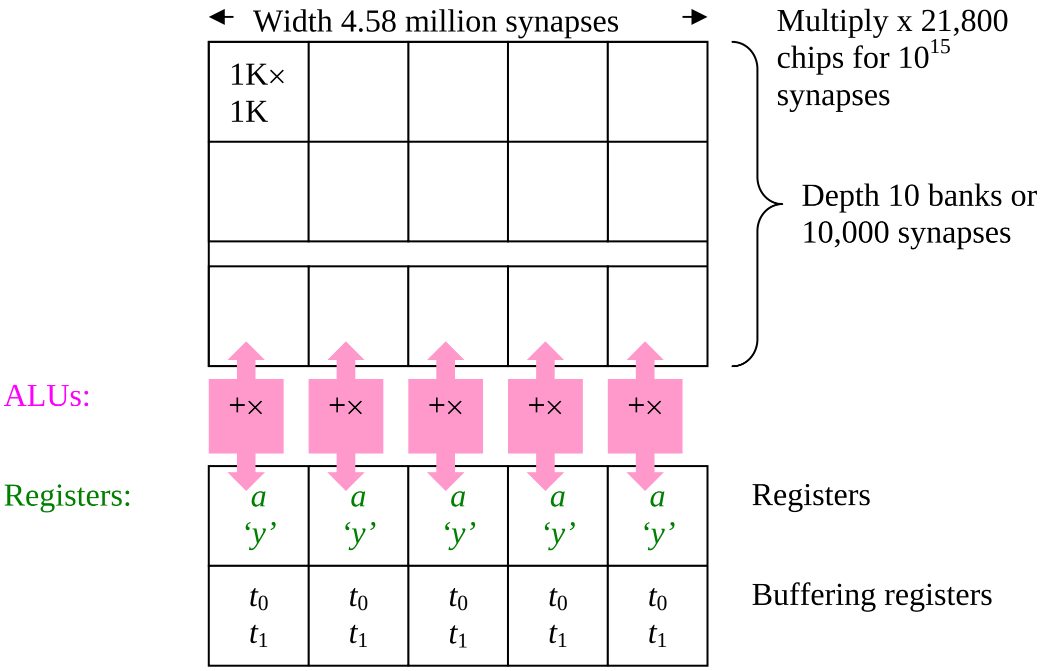

Width 4.58 million synapses
Multiply x 21,800 chips for 10^{15} synapses
1K× 1K
Depth 10 banks or 10,000 synapses
ALUs:
+×
Registers:
a
'y'
Registers
t_0
t_1
Buffering registers

Figure 9: One of 21,800 chips in a brain-sized structure

The assessment approach will be to separately evaluate memory and logic options, followed by a cross-product table for systems. In more detail: While the memory and logic will be tightly integrated, their power efficiency can be evaluated separately. A system in our analysis will be specified by three parameters:

1. $E_{memoryBIT}$, the energy for the memory to access each bit in the whole-row-at-a-time mode
2. N, the number of bits in a synapse, including 4 bits for pointers
3. $E_{logicALUP}$, the energy to process a synapse (for all the bits in the synapse)

The energy to evaluate a synapse will be

$$E_{synapse} = N\,E_{memoryBIT} + E_{logicALUOP}$$

We will analyze two memory options and four logic options. These will create eight combinations. We will also analyze a commercial GPU, which yields a single result with no breakdown by logic and memory.

Memory Access Energy

We will need to estimate energy access energy, which we will do for both conventional DRAM and an extrapolation of an adiabatic research device [Karakiewicz 12].

The most energy-efficient method of accessing large amounts of memory in a production chip occurs during DRAM refresh. A DRAM such as the 4 GBit Micron MT41K256M16 (we will consider the 1066 MHz version) [DDR3L] refreshes its entire contents in 8192 refresh cycles, or $2^{32}/2^{13} = 2^{19}$ = 512K bits each refresh cycle.

Equation (22) on page 15 of Ref. [TN-41-01] is:

Pds(REF) = (I_{DD}5 - I_{DD}3N) × V_{DD}

In this equation Pds(REF) is the power of the subcomponent REF, where REF stands for “refresh.” For more details, see Ref. [TN-41-01].

From the datasheet for the part above [DDR3L], I_{DD}5B = 205 mA, I_{DD}3N = 68 mA, and V_{DD} = 1.35v (table 19, p. 41). This yields 185 mw power during refresh. The time for each refresh cycle is 139 clocks (table 8, p. 30) at a clock rate of 1066 MHz, or 130 ns.

The refresh energy per bit is therefore

$E_{DRAMBIT}$ = 185 mw×130 ns/2^{19} joules/bit = 46 fJ/bit

The second option is a unique adiabatic DRAM described in Ref. [Karakiewicz 12]. The paper reports that when adiabatic charge recycling is turned on, energy efficiency improves by 85×, which implies about 98.8% of the energy driving rows and columns is recovered. Unfortunately, this 256-row device is used in logic mode where on average 128 rows are accessed in parallel, with the number of “1” bits added in analog. We will now make a wild assumption. Most of the power in this research memory is in the row drive, as opposed to the logic. So we will assume/speculate that we could achieve the same power efficiency driving 128 memory banks one row at a time. In this case, the adiabatic memory approach would yield extremely good results.

The research chip in Ref. [Karakiewicz 12] achieves 1.1×10^{12} ops/second/milliwatt. If the ops can be separated across banks and the device operated purely as a memory, energy per bit will be

$E_{TMACSBIT}$ = .001 / 1.1×10^{12} = .91 fJ/bit

The nVidia GTX 750 Ti is a state-of-the-art consumer GPU today. Memory bandwidth is limited to 86.4 GBytes/sec. The system consumes 60W, yielding 86.8 pJ/bit. If a synapse is 12 bits, the energy requirement will be 1.04 nJ/synapse to load the synapse value from memory to the processor. This subsystem is memory bandwidth limited in this application, so we will assume the computing energy is zero.

Compute Energy

The publication [Nikonov 13] contains energy projections for Beyond CMOS devices, including 32 bit adders. We will consider two devices: the “HomJTFET,” which is the device with lowest energy and “CMOS HP,” which is the common device in microprocessors. The energies per operation are:

$E_{\text{TFET-32Add}}$ = .15 fJ/add (figure 47, p. 65),
$E_{\text{HP-32Add}}$ = 2.5 fJ/add (figure 47, p. 65),

for a 32-bit adder. Dividing by 32, this is

$E_{\text{TFET-FA}} = .15\text{ fJ}/32 = 4.7\text{ aJ}$
$E_{\text{HP-FA}} = 2.5\text{ fJ}/32 = 78\text{ aJ}$

Each synapse evaluation includes an 8×8 multiply, a 16-bit add, and what we will estimate as a tripling of energy for control and bit transfer. Let us assume an N-bit multiplier consumes $1.2N^2$ times the energy of a full adder. The 20% addon is due to the need for gated full adders. This would lead to

$E_{\text{TFET-mul}} = (64 + 20\%)\ E_{\text{TFET-FA}} = 77\ E_{\text{TFET-FA}} = .36\text{ fJ}$
$E_{\text{HP-mul}} = (64 + 20\%)\ E_{\text{HP-FA}} = 77\ E_{\text{TFET-FA}} = 6\text{ fJ}$

However, we will also include an estimate for 21-bit multipliers to be closer to single precision floating point

$E_{\text{TFET-mul-21}} = (21^2 + 20\%)\ E_{\text{TFET-FA}} = 529\ E_{\text{TFET-FA}} = 2.4\text{ fJ}$
$E_{\text{HP-mul-21}} = (21^2 + 20\%)\ E_{\text{HP-FA}} = 629\ E_{\text{TFET-FA}} = 41\text{ fJ}$

Let us assume the adder is 16 bits

$E_{\text{TFET-add}} = 16\ E_{\text{TFET-FA}} = 75\text{ aJ}$
$E_{\text{HP-add}} = 16\ E_{\text{HP-FA}} = 1.25\text{ fJ}$

This will lead to energies per synapse evaluation in an ALU

$E_{\text{TFET-ALUOP}} = 3\ (E_{\text{TFET-mul}} + E_{\text{TFET-add}}) = 1.3\text{ fJ}$
$E_{\text{HP-ALUOP}} = 3\ (E_{\text{HP-mul}} + E_{\text{TFET-add}}) = 22\text{ fJ}$
$E_{\text{TFET-ALUOP-21}} = 3\ (E_{\text{TFET-mul-21}} + E_{\text{TFET-add}}) = 7.7\text{ fJ}$
$E_{\text{HP-ALUOP-21}} = 3\ (E_{\text{HP-mul-21}} + E_{\text{TFET-add}}) = 128\text{ fJ}$

System Energy

If extended to 10^{11} neurons and 10^{15} synapses updating at 20 full cycles per second, we construct Table 4 showing overall system energy in orange for the pertinent combinations of memory and logic options. Four of the rows list a logic energy in blue and two of the columns list an memory energy in red. Each row-column intersection four constituent energies (see legend) and the system energy in kilowatts. The right column for GPUs represents a memory-bound situation and is modeled by zero additional energy for computation.

Table 4: Combinations of technologies

		Memory	DRAM	TMACS	nVidia GTX 750 Ti
			46 fj/bit	9.1 fj/bit	87 pj/bit
Logic	fj/ synapse	bits needed			
TFET	1.3	12	12×46=552 (fj memory) 1.3 (fj logic) 553 (fj mem+logic) **11 KW (kilowatts)**	12×9.1=11 (fj memory) 1.3 (fj logic) 12 (fj mem+logic) **240 W**	12×86=1 (nj memory) **21 MW (megawatts)**
HP	21.8	12	12×46=552 (fj memory) 21.8 (fj logic) 574 (fj mem+logic) **11 KW**	12×9.1=11 (fj memory) 21.8 (fj logic) 33 (fj mem+logic) **650 W**	12×86=1 (nj memory) **21 MW**
TFET 21	7.7	21	21×46=1149 (fj memory) 7.7 (fj logic) 1158 (fj mem+logic) **23 KW**	21×9.1=23 (fj memory) 7.7 (fj logic) 30 (fj mem+logic) **610 W**	21×86=2 (nj memory) **43 MW**
HP 21	128	21	21×46=1149 (fj memory) 128 (fj logic) 1278 (fj mem+logic) **26 KW**	21×9.1=23 (fj memory) 128 (fj logic) 150 (fj mem+logic) **3 KW**	21×86=2 (2nj memory) **43 MW**
Legend: Line 1: femto joules to access memory for one synapse Line 2: femto joules logic energy to act on a synapse Line 3: Total energy (line 1 + line 2) **Line 4: System energy at specifiedscale (watts, kilowatts, megawatts)**					

A conspicuous result of the Table 4 is that all the PIMS approaches beat a contemporary GPU by 2000× or more.

A second result is that DRAM energy dominates logic, when DRAM is used. The energy to access a synapse with DRAM is much higher than the energy to process the synapse. Specifically, for the four scenarios, the memory access energy is 9-425 times larger than the logic energy. This means the logic is “essentially free” in this scenario. If this situation were likely to persist, the wise architect would add complexity to the logic.

If our wild assumption that the adiabatic DRAM is feasible, the TMACS row achieves a substantial further improvement.

However, the extremely low access energy demonstrated in Ref. [Karakiewicz 12] raises the possibility that the logic in Figure 8 is actually overbuilt. GPU and PIMS are subject to different design constraints. GPUs have always had an external memory with an access energy limited by the DDR or DDR-class bus between chips. It would be silly for a GPU architect to reduce GPU energy below that of the memory feeding it. It would be more sensible for the GPU architect to add additional GPU features even if the features were marginally useful. This is presumably the reason for texture maps, double precision, etc. While the thought process in the sentences above applies to PIMS, the memory access energy is much less and as a result the amount of logic that will balance the memory access energy will be much less. The logic illustrated in Figure 8 might benefit from adiabatic implementation (see next section) which has not been considered in the power calculation.

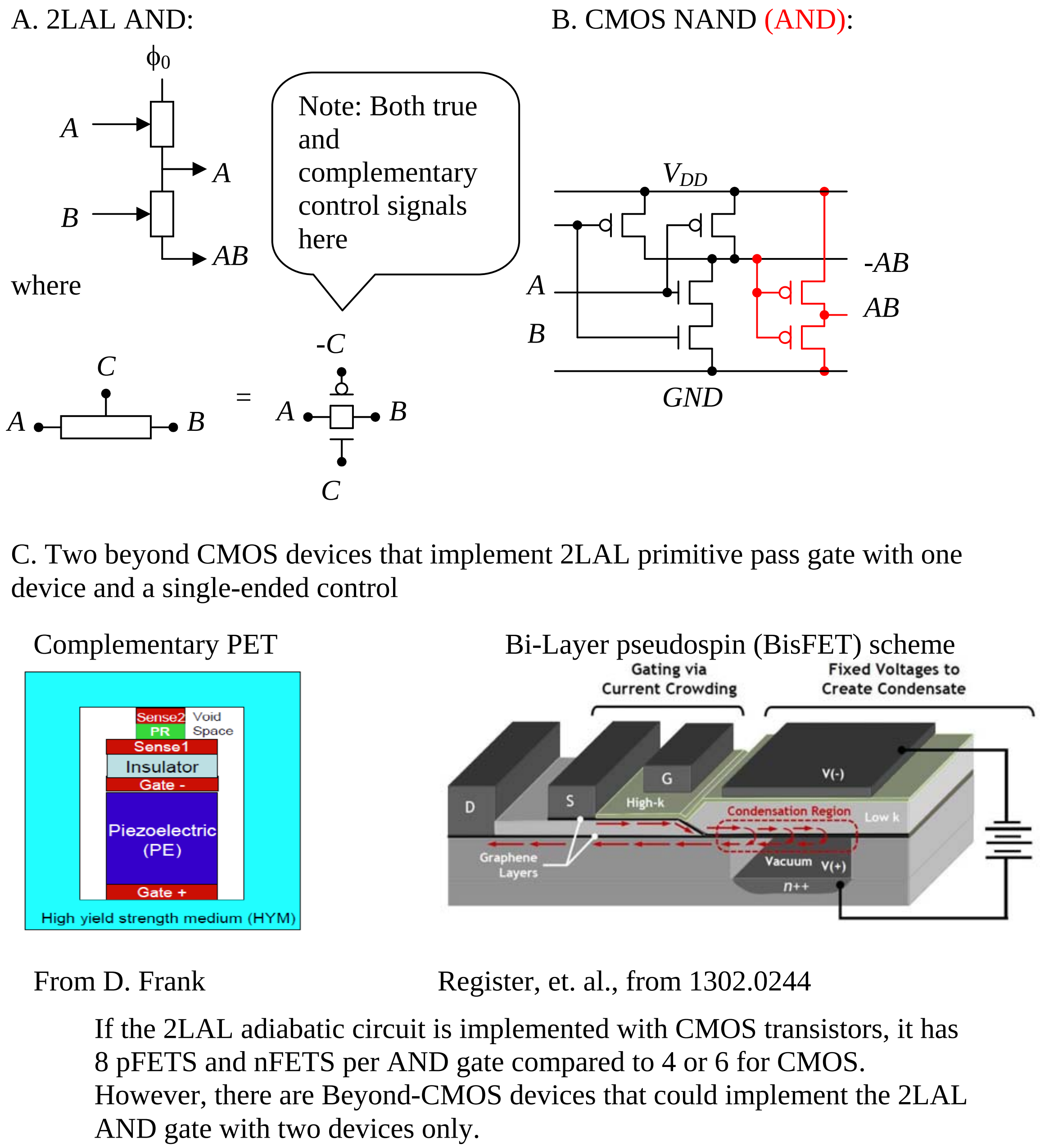
With the adiabatic memory (as wild a speculation as it is), the system is more power efficient that a modern GPU by around a factor of 80,000.

Adiabatic circuit families

Adiabatic circuitry has not been popular to date, but the reasons for this appear to be addressed for Optimal Adiabatic Scaling and PIMS. Adiabatic circuitry is believed to have higher device count and more wiring complexity than equivalent CMOS circuits. While the author admits this is true for CMOS implementations of adiabatic circuits, the PIMS design space “floods” the processor structure with enormous numbers of devices for the purpose of storage. This will tend to reduce the impact of a modestly larger number of devices per equivalent circuit.

Furthermore, the idea that adiabatic circuits have larger device count is to some extent an artifact of optimizing devices to serve the needs of CMOS. The devices are mis-optimized for the current application, suggesting we should discuss the possibilities of devices that are actually optimized for PIMS.

Figure 10A and B shows AND circuits with both the adiabatic logic family 2LAL and CMOS, showing how the device count overhead advantage reverses between CMOS and 2LAL depending on subtle properties of the underlying device



A. 2LAL AND:

B. CMOS NAND (AND):

C. Two beyond CMOS devices that implement 2LAL primitive pass gate with one device and a single-ended control

Complementary PET

Bi-Layer pseudospin (BisFET) scheme

From D. Frank

Register, et. al., from 1302.0244

If the 2LAL adiabatic circuit is implemented with CMOS transistors, it has 8 pFETS and nFETS per AND gate compared to 4 or 6 for CMOS. However, there are Beyond-CMOS devices that could implement the 2LAL AND gate with two devices only.

Figure 10: Adiabatic and CMOS circuits.

If the 2LAL circuit is implemented with CMOS transistors as in Figure 10A, 4 transistors would appear to be required for an AND gate, but actually the number is 8. The reader's attention is called to the implementation of the primitive 2LAL circuit element (which is actually a CMOS pass gate). The circuit element requires both true and complementary control signals (*C* and -*C* in the diagram). To consistently create both true and complement signals in a circuit would require augmenting each circuit with the complementary circuit. Both the true and complement circuits will be of equal size, so the effective transistor count doubles.

In contrast, the CMOS circuit in Figure 10B has 6 transistors for an AND gate or 4 transistors for a nearly equally functional NAND gate.

However, Figure 10C shows two “Beyond CMOS” devices that would appear convey substantial benefit to the 2LAL circuit only. The benefit accrues from subtle properties about the devices’ behaviors:

- The control signal on these 4-terminal devices is isolated from the source-drain connection in both voltage and current. This would permit the equivalent of a CMOS pass gate with a single device as opposed to two devices. This would cut the number of devices for 2LAL by 2×.

- These devices can be controlled by the voltage difference between a signal and ground, a signal and the supply voltage, or a signal and the clock. This avoids the need for an entire second circuit to generate the complementary logic signals. This would cut the number of devices for 2LAL by an additional 2×.

Conclusions

We have devised an approach to increasing computer performance beyond the limits of Moore’s Law and the microprocessor. We specifically use energy efficiency as the computer performance metric.

The first step was to provide greater clarity to an adiabatic scaling rule. It is ubiquitously known that clock rates in production microprocessors rose so fast is the 1990s that the resulting power dissipation exceeded customer tolerance. Multi-core processors with slower clocks are now more effective in the marketplace. It has also been claimed that adiabatic and reversible computing could have extremely high energy efficiency, but these technologies never received a lot of attention. This document shows the connection between the emergence of multi-core processors and the non-viability of reversible computing: Reversible computing is an idea in the right direction, but the community needs to go in that direction at the right pace.

We identify a clock rate that optimizes economic return for a computer. Since the emergence of multi-core processors around 2003, the Moore’s Law has driven clock rates above the optimum point. The remedy is to mix just the proper amount of adiabatic “slow down” – an amount that changes over time.

If manufacturing costs continue to decline (as predicted), the optimal mix of Moore’s Law and adiabatic computing will show rising power efficiency over time. We call the effect Optimal Adiabatic Scaling, and it is like Moore’s Law but less powerful. Optimal Adiabatic Scaling would have a doubling period that is twice as long as Moore’s Law under the most optimistic assumptions about 3D scaling.

However, Optimal Adiabatic Scaling has another property that can yield big benefit. Optimal Adiabatic Scaling predicts an O(N) increase in device count will raise both throughput and power efficiency by O($\sqrt{N}$). If followed naturally, this will lead to an excess of devices. This excess of devices can resolve a long-standing division (of sorts) in computing. For a long time, computers have been divided into a processor and a memory or storage subsystem. A sizeable amount of time, energy, and resource is devoted to transferring data between these units. Storage technology has become more like processor technology in recent years, changing from rotating disks to Solid State Disk (SSD) chips using almost the same technology as the processor. Optimal Adiabatic Scaling is thus remarkably compatible with the idea of a new architecture that merges processors and storage, using the excess devices resulting from adiabatic scaling to become the storage medium.

As mentioned, the brain and neural systems work this way.

We then discussed high-level computer system organization that could exploit Optimal Adiabatic Scaling in a system that merged storage with processing. This eventually led to the idea that stored data could be distributed within the structure of a computer to provide sufficient data near compute elements to enable the power efficiency boost and also eliminate steps like input, parsing, formatting output after computation, and output. The collective performance boost comes from a combination of more energy-efficient logic in conjunction with algorithms that circumvent the need for certain algorithmic steps altogether. These steps are essentially the I/O and data manipulation.

We finally presented a practical architecture for realizing the gains described above. The release of 3D (or maybe 2½D) chips that combine logic and memory is imminent [HMC 14] [HP The Machine]. If the logic in these stacks could be switched to something like a GPU, the resulting system would support the proper system structure and demonstrate the concepts in this document. The architecture was illustrated in a form that might apply to an ANN like Deep Learning or sparse matrices on supercomputers. The efficiency was very high using estimates of technology now on roadmaps, enabled by the tight integration of processing and memory. Furthermore, the architecture proposed had the correct parallelism and scaling to embody Optimal Adibatic Scaling over time.

We performed an energy efficiency analysis of the PIMS approach in comparison with a current production GPU. With projections based on production DRAM, PIMS outperformed the GPU by about 2000×. If a really far-out adiabatic memory concept could be moved to production, the boost rises to about 80,000×. At 80,000×, the historical rate of improvement due to Moore’s Law could be maintained for another 16 generations or so. The energy efficiency analysis illustrated why a hugely stripped down GPU architecture using adiabatic low-level technology would make sense.

As future work, the authors suggest building a system like Figure 5, programming it to operate on data like shown in Figure 4. If successful, the system should scale like Table 2 and eventually produce a device like the 3D structure in Table 3.

References

[Bennett 73] C. Bennett, “Logical Reversibility of Computation,” IBM Journal of Research and Development, Volume 17 Issue 6, November 1973, Pages 525-532

[DDR3L] Micron datasheet, DDR3L SDRAM MT41K1G4 – 128 Meg x 4 x 8 banks MT41K512M8 – 64 Meg x 8 x 8 banks MT41K256M16 – 32 Meg x 16 x 8 banks

[Dennard 74] R. H. Dennard, et. al., Design of ion-implanted MOSFET's with very small physical dimensions, IEEE J. Solid-State Circuits, Vol. SC-9, pp. 256-268, 1974.

[Feynman 96] RP Feynman, JG Hey, RW Allen, Feynman lectures on computation, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1998

[Fujisaki 13] Y. Fujisaki, Review of Emerging New Solid-State Non-Volatile Memories, Jpn. J. Appl. Phys. 52 040001

[HMC 14] Hybrid memory cube is an emerging product by Micron. See the Internet.

[HP The Machine] "The Machine" is an announced research program by Hewlett Packard. See the Internet.

[ITRS YY] http://www.itrs.net

[Koller 92] Koller, J.G.; Athas, W.C., "Adiabatic Switching, Low Energy Computing, And The Physics Of Storing And Erasing Information," Physics and Computation, 1992. PhysComp '92., Workshop on , vol., no., pp.267,270, 2-4 Oct 1992
doi: 10.1109/PHYCMP.1992.615554

[Karakiewicz 12] R. Karakiewicz, R. Genov, G. Cauwenberghs, 1.1 TMACS/mW Fine-Grained Stochastic Resonant Charge-Recycling Array Processor, IEEE Sensors Journal, Vol. 12, No. 4, April 2012, p. 785

[Moore 65] G. Moore, Cramming More Components onto Integrated Circuits, Electronics pp. 114-117, April 19, 1965.

[MT29F512G] Micron data sheet, NAND Flash Memory MT29F64G08CBAA[A/B], MT29F128G08C[E/F]AAA, MT29F128G08CFAAB, MT29F256G08C[J/K/M]AAA, MT29F256G08CJAAB, MT29F512G08CUAAA, MT29F64G08CBCAB, MT29F128G08CECAB, MT29F256G08C[K/M]CAB, MT29F512G08CUCAB

[Nikonov 13] D. Nikonov and I. Young. Overview of Beyond-CMOS Devices and A Uniform Methodology for Their Benchmarking, arXiv 1302.0244.

[Nordhaus 07] W. D. Nordhaus, "Two centuries of productivity growth in computing," Journal of Economic History, vol. 67, no. 1, p. 128, 2007.

[Theis 10] T. Theis, In Quest of the "Next Switch": Prospects for Greatly Reduced Power Dissipation in a Successor to the Silicon Field-Effect Transistor, Proceedings of the IEEE, Volume 98, Issue 12, 2010

[TN-41-01] Micron technical note, TN-41-01: Calculating Memory System Power for DDR3.

[von Neumann 45] J. von Neumann, First draft of a report on the EDVAC, Moore School of Elec Eng, U. of Pennsylvania, Philadelphia, Pa, June 30, 1945 (101 pp). (This draft was written in March-April 1945.)

Appendix II: Artificial Neural Networks as Beyond CMOS Systems

Artificial Neural Networks as Beyond CMOS Systems

Erik P. DeBenedictis and James B Aimone, August 18, 2014

Abstract

We present a method to assess artificial neural network (ANN) implementation approaches as candidates for a Beyond CMOS technology. Neural networks are indisputably highly capable information processors with remarkably low power consumption. However, real biological neural networks are unsuitable as a Beyond CMOS technology because they grow organically as opposed to being manufactured. We investigate the possibility that an ANN technology stack could be devised using biologically inspired methods but which would otherwise be manufacturable.

We present a method to estimate top-to-bottom energy efficiency of technology stacks for brain-inspired computing. This is based on a method of assessing the minimum energy consumption of the underlying computational primitive (a neuron) as a function of kT (the base value of thermodynamic energy). We then apply computational complexity theory to data representations, architectures, and algorithms built from the artificial neurons.

The result is believed to be a novel method of assessing minimum energy in an information processing structure that combines ideas from Landauer’s minimum gate energy, channel capacity in communication theory, and complexity theory. The method was purpose-built for assessing ANN implementation approaches as candidate Beyond CMOS technologies, and permits cross-comparison between ANN and CMOS technology stacks.

Two conclusions are apparent from applying the new method to proposed artificial neural concepts: (1) The common resistive crossbar neural network scales significantly less well than a digital emulation of itself. Resistive crossbar scaling could possibly be improved by adding sparse coding of data (biology is known to do this), but sparse coding is not dealt with in the literature in any detail. (2) There is a possibility that artificial neurons based on spiking could scale very well, but there is no evidence of a compatible artificial synapse.

Introduction

Neuro-inspired computing has been receiving attention from both neuroscientists to accelerate simulations of neural circuits and the computer industry as a Beyond CMOS technology option. The brain’s potential benefits as an inspiration for computation are widely cited. Biological neural systems are capable of performing highly complex functions that are difficult using conventional algorithmic approaches. Furthermore, the brain performs these functions quickly and at low power.

Despite this increasing attention, there remains considerable uncertainty about which features of neural circuits would be necessary in an effective Beyond CMOS technology. Biological neural systems have a number of computational properties that are distinct from computer systems, such as integrating inputs in the analog domain, communicating using spikes which are effectively digital in amplitude but analog in time, distributing information over many nodes operating in parallel, and leveraging sparse coding of representations that often progress through hierarchical layers producing progressively more complex features. Computer scientists have achieved some neural-like functionality in software through recent developments in techniques such as Deep Learning; however Deep Learning and previous artificial neural network (ANN) algorithms are generally only based on the equivalent of level-based neurons and not spiking [Hinton 2006]. Other research has focused on producing hardware instantiations of neurons, often by replicating the complex dynamics of spike generation or synaptic transmission in silicon [Indiveri 2011].

If neural-based architectures are to become a key contributor to Beyond CMOS technology the community would be seeking a manufacturable technology based on those properties and structures from neuroscience that contribute to the needed function and power efficiency. However, there is no consensus as to whether it would be best to extend development of resistive crossbar neural networks to more power-efficient devices and sparse coding [Nikonov 2013] [Taha 2013] or to restart with more neurobiologically realistic spiking approaches [Dayan 01], or to select an intermediate strategy [Merolla 2014] [Eliasmith 2012]. The differences between these approaches can be substantial, both in terms of potential algorithm functionality and hardware efficiency, and to date we are unaware of any systematic analysis weighing the advantages of these different approaches. It is not assured that the conventional approaches of computing technology development will be successful in this domain without an objective means to evaluate the advantages of emphasizing different aspects of neural fidelity.

In this study, we describe an approach that we believe will provide an initial framework for rigorously developing neural computation hardware. First, we quantify the costs associated with representing neural algorithms on analog architectures, such as resistive crossbar arrays. Second, we tie back to digital logic by considering potential strategies for expressing neural computation in a digital framework. Finally, we show how the framework described here enables these different implementation strategies (analog, digital, and spiking) to be quantified for specific algorithms, thus facilitating a direct comparison for the purpose of neural computing engineering. While we consider only a selection of proposed neural systems, we believe that this approach is generalizable to a number of potential neural computing systems.

Reasoning about the limits of technology

There has been an information revolution underway since WW II, driven by a progression of electronic technologies at the base, the von Neumann architecture, and software for implementing applications. Integrated circuits have been the electronic driver, but the current generation of this technology (CMOS) shows the approach is reaching its physical limits. There is a search for an alternative approach that could maintain the year-to-year rise in performance essential to the information revolution. This search includes the ANN implementations [ITRS 20XX] [DARPA Synapse] as the central component of a stack of technologies (devices, artificial neurons, and learning algorithms) below and above it. It is thus important to know how an ANN technology stack would perform in comparison to CMOS.

However, CMOS is such a formidable competitor that upstarts will need a new comparison approach to get meaningful results. With an investment of around a trillion dollars, industry has improved CMOS incrementally to the point where it is approaching the limits of the underlying physics in several dimensions. While ANN implementations have many interesting attributes, today's systems work only at small scale and have not been optimized for power efficiency or much of anything else. This results in CMOS winning in comparisons without the ANN implementation even having a chance. This is not helpful since we are specifically looking for a successor to CMOS. We need a comparison method that is not biased in favor of the more mature technology.

It is sometimes possible to find the ultimate physical limits of a technology theoretically. Comparing the limits CMOS with the limits of ANNs would neutralize CMOS's maturity advantage. Essentially, a physics-level energy analysis is applied to two approaches assuming all the components are being implemented by the best components possible without violating the laws of physics. Where this method applies, it leads to minimum energies typically in units of kT (which is about 4 zetpojoules at room temperature, or 4×10^{-21} Joules). Since we know CMOS will approach its physical limits (since it already is close), investment in an ANN implementation will only be justified if the limits of the specific ANN approach are higher.

We use a unique process in this document for identifying the limits of ANN technology, although the process is derived from several methods that have accomplished similar functions in other fields. Table 1 summarizes the contributing approaches along with our proposed approach.

Table 5: Comparison of technology limit approaches

	Incoming presumption	Freedom	Result of method	Prohibition
Landauer's gate energy	Computer and algorithm are implemented as a specific network of gates	Pick any circuit for a gate, with infinitely advanced components available	Minimum overall energy (signal + system)	May not redesign gate-level implementation (e. g. may not invent reversible computing)
Communications limits	System comprised of transmitter, channel, and receiver.	Pick any design for transmitter and receiver, with infinitely advanced components available.	Minimum energy of signal (does not count transmitter or receiver energy)	May not use energy freely-available in the receiver to increase signal energy
Complexity theory	Application decomposable to operations on computer words (e. g. 32-bit integers)	Pick any sequence of operations	"Big O" performance of algorithm in limit as size → infinity	May not redesign the computer for better performance (self-enforcing prohibition: it won't improve the result)
Proposed method for analyzing artificial neural networks. Also applies to CMOS realizations, thereby allowing comparison with CMOS	A nanostructure that holds synapse values (usually)	Design control electronics, electrical protocol, and algorithm	Minimum overall energy for algorithm at scale (signal + system)	May not change the nanostructure or anything specified in the artificial neural network concept

Landauer's principle and minimum gate energy

Landauer [Landauer 1961] proposed that the minimum energy for a gate operation was "on the order of kT," where k is Boltzmann's constant of 1.38×10^{-23} j/K and T is the absolute temperature in degrees Kelvin. Landauer assumed a computer was defined as a specific network of gates. Landauer's contribution was to identify the lowest possible energy for any implementation of a gate, given that the implementer was free to use any technology consistent with the laws of physics. In Landauer's case, energy included both the signal energy in the 0’s and 1’s and the energy to power the gate. While the implementer had considerable freedom, he or she was not eligible to alter the gate-level specification of the computer or to improve upon algorithms.

Landauer’s principal has been useful in technology planning. With expenditure of around a trillion dollars, CMOS had moved considerably closer to the minimum energy predicted by Landauer. It is now accepted that even another trillion dollars will not continue the reduction below “on the order of kT” per gate operation and that we must proceed by going outside Landauer’s assumptions if we want to go anywhere at all.

Communications limits

Shannon [Shannon1949] and Johnson-Nyquist [Johnson 1928] [Nyquist 1928] developed theory for the amount of information that may be transmitted over a communications channel as a function of the signal energy in units of kT. The communications limit gives the implementer freedom to choose any implementation of the transmitter and receiver consistent with the laws of physics. Unlike Landauer’s principle, the energy assessment applies only to the signal energy in the channel and does not include energy to power the transmitter or receiver. While the implementer has considerable freedom, he or she may not use energy freely available in the receiver to bolster the input signal.

The communications limits apply to neural systems as follows. Biological neural signals are a waveform, at least in part. Landauer’s principles apply to discrete units of information, whereas communications limits apply to signals that convey information over time. Communications limits thus apply to biological inspired manmade systems that retain the waveforms of their biological inspiration – as well as a comparison baseline for manmade systems that emulate waveforms digitally.

Computational complexity theory and algorithms

Many people contributed to the development of computational complexity theory for algorithms, which identify which algorithms run faster than others at large scale. The presumption behind complexity theory is that the user is most concerned about the running time of algorithms as the data size becomes very large, called scalability. The implementer is free to choose any sequence of operations, but changes in computer architecture, instruction set, or processing speed will have no effect on the outcome because constant factors are removed from the result (due to “Big-O” notation).

Complexity theory will be needed to understand scaling of current ANN implementations from the level of current experiments to the expected scale of real systems. Experimental data on ANN computers has been at the level of tens of thousands of synapses (active elements). The largest software implementations have been much larger at around a billion variables [Le 2013]. However, the human brain is often cited as a scaling target. The human brain has around 10^{15} synapses, or a million times larger than anything demonstrated so far.

Proposed method

We propose a method in the style of those listed above, but specific to the effectiveness of an ANN technology stack as a Beyond CMOS candidate.

Our method accepts and analyzes ANN implementation approaches as an input. Usually, these approaches are a nanotechnology structure that holds the synapse state, including, for example, memristor arrays [Taha 2013], Charge Injection Device (CID) arrays [Karakiewicz 2012], and CMOS implementations where synapse state is stored in DRAM [Merolla 2014]. There are exceptions, such as neuristor concepts, which are based on a replacement for the neuron body (soma) as opposed to synapse. Just as Landauer's principal would not apply if one rearchitected the gate-level implementation of a computer, our method presupposes that we will not redesign the ANN implementation approach to give a better result.

However, we will presume that future engineers will use infinitely advanced components. For example: If the ANN implementation approach has a resistor or capacitor in some spot, we will assume the component can be the best that could ever possibly be constructed. This means the resistor will generate precisely the amount of thermal noise predicted by engineering principles, but not zero noise. This approach allows us to find minimum energy limits for implementation approaches as functions of kT. This kT energy measure allows us to accommodate the intermixing of physical and algorithmic behavior. Use of such a measure is unusual but not unique [Peng 2008].

We will also view ANNs as building blocks for larger systems, with a neural network algorithm being the principal with which the building blocks are assembled. The majority of today's complexity theory (i. e. software algorithms for computers) is based on counting the number of operations on a computer word (e. g. 32 bit integer). While ANNs can have different primitives, these primitives are close enough to computer operations that complexity theory can be adapted. Specifically, here we focus on the common ANN function of the sparse vector-matrix multiply on signals, which are essentially bandwidth-limited variable-precision fixed point sequences.

There will be an important difference in the nature of complexity expressions, as illustrated in Table 6. The energy efficiency of computers today uses expressions from physics and engineering to compute the energy of gates, such as in Ref. [Nikonov 2013]. The energy of a gate operation would be C kT or C joules, for some constant C. The remainder of the energy efficiency computation is essentially based on counting gate operations, so the energy of a computation would be of the form $f(N)$ C kT.

Table 6: Comparison of analog and digital systems

Level	Digital computer	Neuron
Algorithm	$f(N)$ C kT	$f_{algorithm}(N)$ $f_{neuron}(N)$ kT → $F(N)$ kT
Primitive building block	C kT (gate)	$f(N)$ kT (neuron)

In contrast, a neuron is an analog structure of variable size, specifically N synapses. The energy consumption of a neuron will be of the form $f_{neuron}(N)$ kT, where f is not a constant. When multiple neurons are grouped into algorithms, the overall energy will be $f_{algorithm}(N)$ $f_{neuron}(N)$ kT, which involves two functions of N and is ambiguous in general. Therefore, we will group both functions together to say the energy is $F(N)$ kT. This latter expression becomes the computational complexity of the neural algorithm using a kT energy metric [Peng 2008].

Utility of the method

The assessment method could provide useful information for determining whether one ANN implementation approach has more or less promise than another approach or a conventional digital implementation. The information would be in the form of an energy estimate, either as numbers or algebraic expressions, and could be realized in the following forms, among others:

- The method might reveal minimum energy for some computation as a function of problem size N to be

  $E_{minimum} = (1.52\times10^{12}\ N^2 + 3.33\times10^{20}\ N)$ kT (note numbers are examples)

  Each of these would be specific to the ANN implementation approach but implementation-independent with respect to devices, and thus allow comparison of the approaches on the basis of ultimate limits to energy.

- A second use would be to understand algorithmic complexity, which is the order of the energy consumption using Big-O notation, such as

  $\mathrm{O}(E_{minimum}) = N^2\ L^2$ kT (note $N^2\ L^2$ is an example expression)

  Each Big-O expression would be specific to the ANN implementation approach but implementation-independent with respect to devices. These expressions will allow comparisons of scalability of energy efficiency. It is common to teach students in college computer science classes things like “quicksort is $N \log N$ but bubble sort is N^2.” In the future, students might be taught that “a level-based neuron is $N^2\ L^2$ kT.”

- A third use of the procedure is to produce practical energy efficiency results. In many cases, the expressions for energy efficiency of practical systems has the same algebraic structure as the expressions for ultimate limits [Llewellyn 1931] [IRE 1957], but there are differences in constant factors. The power efficiency of technologies is road mapped. This could lead to a variant of the above expression like

  $E_{practical} = (1.52\times10^{12}\ N^2 + 3.33\times10^{20}\ N)\ R(2014)$ kT (note numbers are examples)

  where R stands for “roadmap.” $R(2014)$ kT is the energy dissipation of the gate or neuron (in this context) in the year 2014. With the help of a roadmap, an engineer could use the expression above to compute the practical energy efficiency at some other year and for a different value of *N*.

Analysis of a level-based ANN

To illustrate our analysis approach, we explore the theoretical performance of an ANN technology stack based on the resistive crossbar implementation and algorithms built on it. We then compare the resistive crossbar implementation with digital and spiking implementations (but we do not reanalyze the entire technology stack for these other implementations).

The resistive crossbar neural network shown in Figure 11 is present in the literature with many variants, but with the resistance consistently representing ANN synapse values. As shown in Figure 11A, the electrical structure comprises an array of row and column wires with programmable resistors (e. g. memristors) at the cross points and an amplifier. As shown in Figure 11B, the array wires are analogous to the axons (rows) and dendrites (columns) of a living neural system, with the programmable resistors being analogous to synapses.

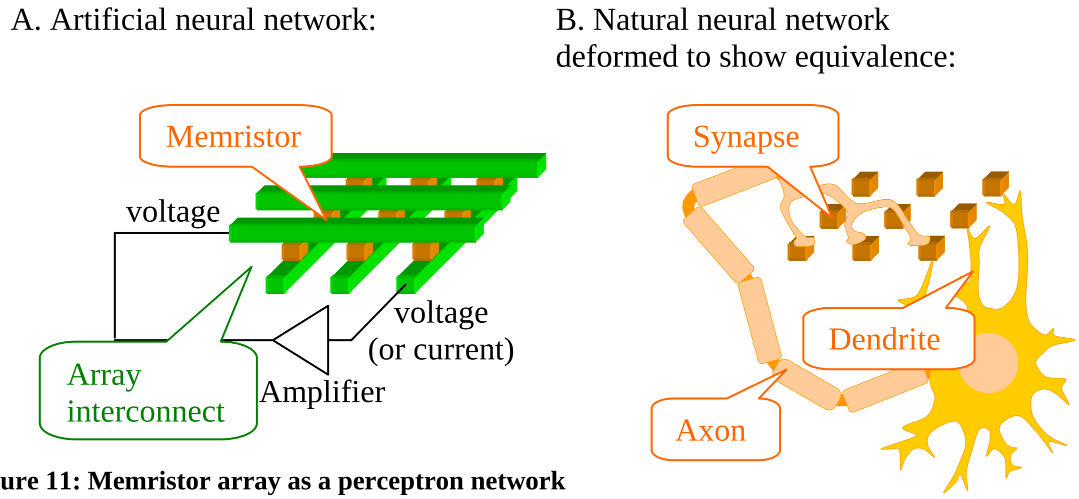
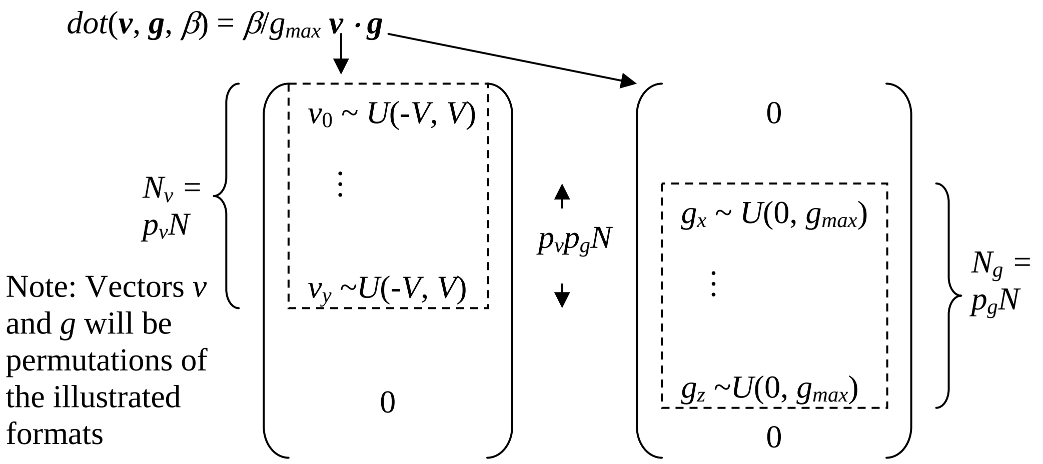

A. Artificial neural network:
B. Natural neural network
deformed to show equivalence:
Memristor
voltage
voltage
(or current)
Array
interconnect
Amplifier
Synapse
Dendrite
Axon

Figure 11: Memristor array as a perceptron network

The objective of the structure in Figure 11A is to perform a vector-matrix multiply, which is equivalent to each column performing a dot product. A dot product comprises a series of multiplies and adds, and is a fundamental computation to most ANN algorithms and is a close approximation of synaptic integration in biological systems. The strategy is to use Ohm’s law applied to the memristor to multiply and Kirchhoff’s current law on the column conductor to add.

Voltages v_i driving each row of the memristor array are uniformly distributed in the range [-V, V] and comprise vector $\boldsymbol{v}$, as illustrated in Figure 12. (The notation “$v_i \sim U(a, b)$” means v_i is a random variable uniformly distributed between a and b.) The other vector $\boldsymbol{g}$ is defined by the resistance state of memristors in a column and comprises conductances (1/resistance) g_i uniformly distributed in the range [0, g_{max}].



Note: Vectors *v* and *g* will be permutations of the illustrated formats

Figure 12: Definition of the dot function as *β/g* times the dot product of *v* and *g*

The vectors have *N* elements and may be sparse, as illustrated in Figure 12. Exactly p_vN elements of ***v*** and p_gN elements of ***g*** are nonzero, with p_vp_gN pairs of corresponding elements both being nonzero (meaning the multiplication step in the dot product will yield a nonzero result in exactly p_vp_gN cases). While p_v and p_g may serve as probabilities in practice, in this analysis we interpret them as exact ratios.

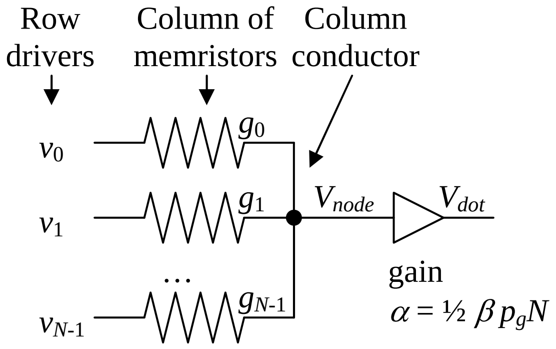
We define a *dot* function that addresses two issues with the dot product.

1. The straightforward definition of a dot product yields awkward physical units, so we define *dot* with different units. The dot product of a voltage vector and a conductance vector would yield a current vector, which would have the wrong units to be used again by the same function. To address this issue, the *dot* function divides the mathematical dot product by the scalar g_{max}.

2. Biological neural networks often modulate gain, which will turn out to have an effect on energy consumption. The *dot* function will also multiply the result by β. A reader uninterested in gain may assume $\beta \equiv 1$.

Some algorithms require changing the sparsity pattern. An element of ***v*** can be moved into the sparse region easily by grounding a row conductor. Moving an element of ***g*** into the sparse region will require adding and using a select device. This asymmetry will be discussed later in more detail.

The circuit in Figure 13 can perform a dot product a described above, albeit with some caveats. The resistive combining network on the left forms the weighted average of the input voltages as opposed to the dot product. However, the weighted average appearing on node V_{node} is mathematically equivalent to the dot product of ***v*** and ***g*** divided by the sum of the elements of ***g***. We will now assume (with discussion in the next paragraph) that the average memristor conductance is ½ g_{max}, which would make the sum of the weights ½ g_{max} p_gN. If the amplifier's gain is set to α = ½ β p_gN, voltage V_{dot} will match the definition of the *dot* function above.



$$V_{node} = \frac{\sum_i v_i g_i}{\sum_i g_i}$$

Assume
$\sum_i g_i$ = ½ g_{max} p_gN

$$V_{dot} = \beta/g_{max} \sum_i v_i g_i$$

Figure 13: Circuit

The assumption that the average memristor conductance is ½ g_{max} requires comment. The authors have encountered the circuit in Figure 13 the literature [Taha 2013] and it appears reasonable in the context of a neural network. While this assumption would require normalization of the ***g*** vector, the circuit would thereafter perform dot products as needed.

There will be a discussion below about the interaction between the Johnson-Nyquist noise and system speed or clock rate. At this point, let us assume the circuitry in Figure 13 is bandlimited to frequency *f*. The noise power according to the Johnson-Nyquist noise theorem will be 4kT *f* at the input to the amplifier. For the specific situation in Figure 13, this would be

$$\overline{P_{noise}} = 4\text{ kT } f = \overline{V_{noise}}^{2}\ ½\ Np_g g_{max}$$

which yields

$$\overline{V_{noise}} = \left[\frac{8\,kT}{Np_g} \frac{f}{g_{max}} \right]^{½}$$

In accordance with previous discussion, the noise will be amplified before appearing on the output

$$\overline{V_{noiseout}} = \overline{V_{noise}}\ \alpha = \overline{V_{noise}}\ ½\ \beta\, Np_g$$

The number of resolution levels *L* will be the output range 2*V* divided by the noise voltage $V_{noiseout}$.

$$L = \frac{2V}{V_{noiseout}} = \left[\frac{Np_g}{8\,kT} \frac{g_{max}}{f} \right]^{½} \frac{4V}{\beta Np_g}$$

Which by squaring yields

$$L^2 = \frac{2\ N_p g}{kT}\,\frac{g_{max}}{f}\,\frac{V^2}{\beta^2 N^2 p_g^{\ 2}}$$

And then by rearrangement to a form that has units of energy and will be useful later

$$\frac{V^2 g_{max}}{f} = \frac{\beta^2\ L^2\ N^2\ p_g^{\ 2}\ \mathrm{kT}}{2\ N p_g}$$

The power consumption of the circuit is addressed now. Only the energy turned into heat in a resistor is irrecoverable in this situation, so we will analyze the average power dissipated by all the resistors. We will express the power as a base value plus small-scale correction.

V_{node} moves asymptotically to zero as N increases, with the base value for power assuming $V_{node} = 0$. If V_{node} is grounded, there will be $p_v p_g N$ uniformly distributed voltages [-V, V] across resistors with average conductance $g_{max}/2$. This yields the base power of 1/6 $V^{\ 2}\ g_{max}$ per resistor and total power

$$\overline{P_{neuron}}^{(B)} = 1/6\ V^{\ 2}\ g_{max}\ p_v p_g N$$

We will designate the base power with the superscript $^{(B)}$ and use it in subsequent discussion of higher level functions. However, we have numerically computed the small-scale correction. Given that the v's and g's in Figure 12 have well-defined distributions, the average heat produced by the resistors in Figure 13 can be computed as

$$\overline{P_{neuron}} = P_{simulate}(p_v p_g N,\ (p_v\text{-}p_v p_g)N)\ V^{\ 2} g_{max}\ \ p_v p_g N,$$

where $P_{simulate}(M, Z)$ is the result of a numerical computation. The authors wrote a computer program that rolls uniformly distributed random numbers in the range [-1, 1] for v's and [0, 1] for g's and computes the power dissipation as a function of the number of uniformly-distributed drive voltages $M = p_v p_g N$ and additional zero voltages due to sparseness $Z = (p_g\text{-}p_v p_g)N$ and given $V = g_{max} = 1$. A graph of this function is illustrated in Figure 14.

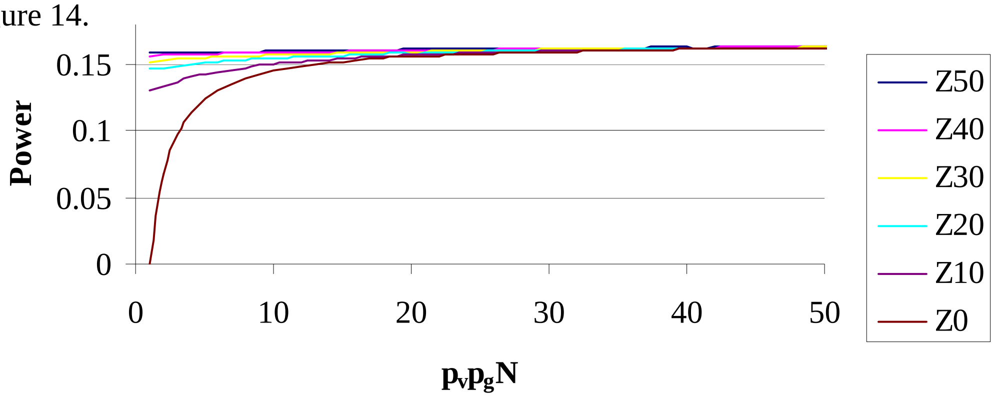


Figure 14: Computation of average power ($P_{simulate}$)

All curves in Figure 14 are asymptotic to 1/6, with the interesting behavior on the left. The lowest curve labeled Z0 is the power when all the applied voltages are uniformly distributed in the range [-*V*, *V*] and there are no additional grounded signals applied due to sparsity. This curve shows less average power due to V_{node} shifting away from zero towards the applied voltages and reducing power. The other curves labeled ZNN include the addition of NN grounded signals applied due to sparse values in the voltage vector. Tying V_{node} to additional grounds would be expected to reduce fluctuation in V_{node} and cause the curve to approach the asymptotic value more quickly – which is what is observed.

We must now establish a connection between operating speed and the Johnson-Nyquist noise. We had previously assumed the circuitry would be limited to frequencies below *f*, but *f* has so far been just an algebraic symbol. We are now free to engineer a specific value for *f* using algebraic manipulations. To identify the limiting case, the Nyquist sampling theorem states that the maximum rate at which voltages could be applied to the rows and dot products obtained from the columns would be 2*f*. This would imply the limiting case of a clock period of 1/(2*f*). In this limiting case, the energy to evaluate a neuron would be the power P_{neuron} multiplied by the clock period.

The equation below is a rearrangement of terms from the equation for power above, divided 2*f*.

$$\overline{E_{neuron}}^{(B)} = \frac{\overline{P_{neuron}}^{(B)}}{2f} = \frac{V^2 g_{max}}{f} \frac{p_v p_g N}{12}$$

Substitute

$$\overline{E_{neuron}}^{(B)} = \frac{\beta^2 L^2 N^2 p_g^2 \text{ kT}}{2 N p_g} \frac{p_v p_g N}{12} = \frac{\beta^2 L^2 N^2 p_v p_g^2 \text{ kT}}{24}$$

The equation above is notable because of the absence of *V* and g_{max}. The equation is thus an implementation-independent representation of the minimum energy $E_{neuron}^{(B)}$ as a function of the nature of the problem being solved and the thermodynamic quantity kT.

There is redefinition of terms that may yield insight. Expressing $E_{neuron}^{(B)}$ in terms of $N_v = p_v N$ and $N_g = p_g N$ the nonzero signals in the vectors,

$$\overline{E_{neuron}}^{(B)} = \frac{\beta^2 L^2 N_v^2 N_g \text{ kT}}{24 N}$$

In conventional computer terminology, the system will perform $p_v p_g N$ multiply operations. The energy per operation will be $E_{neuron}^{(B)}$ divided by $p_v p_g N$.

Energy/op = 1/24 $\beta^2 L^2 N_g$ kT

Which tells us the energy per equivalent multiply operation is proportional to the number of nonzero elements in a column of the weight matrix. In subjective terms, this means the cost of a multiply depends on how many similarly computed products might be added up afterwards.

Further note the implication to software-based ANNs, such as Deep Learning. As mentioned above, Deep Learning typically does not assume any sort of sparse coding or gain β. This would imply $\beta = p_v = p_g = 1$ and the energy per neural evaluation would be

$$\overline{E_{neuron}}^{(B)} = 1/24 \; L^2 N^2 \text{ kT}$$

The above expressions are for a dot product. If we multiply by M, which is the number of output neurons, we get the energy of an $N \times M$ vector-matrix multiply, as may occur in software-based methods such as Deep Learning.

$$\overline{E_{vmm}}^{(B)} = 1/24 \; \beta L^2 N^2 M \text{ kT},$$

where E_{vmm} is the energy of a vector-matrix multiply.

A neural-network algorithm to improve energy efficiency

The previous section analyzed the minimum energy of an analog vector-matrix multiply system. We will now use the analog array as a building block to construct algorithms. In this first step, we will show that the analog vector-matrix multiply circuit is part of a more general class of hybrid vector-matrix multiply systems that embody both analog and digital principles. We will discover that the energy efficiency of the analog array just analyzed can be improved by executing it in sections – in fact the greatest energy efficiency is obtained when it is transformed into a digital emulation of itself.

We will analyze a hybrid vector-matrix multiply system for a dense N-element vector and a dense $N \times M$ matrix, using L-level arithmetic. We previously found a minimum energy of order $E_{vmm} = O(N^2 L^2 M)$ kT. We will analyze an algorithm that subdivides the matrix into horizontal slices and processes the slices sequentially. While the matrix starts out being dense, we will view the act of subdividing the matrix into j slices as creating a series of j matrices where each matrix is the original matrix with all but the j^{th} slice set to zero by control of the sparsity pattern. This will result in $p_g = 1/j$ for the new matrices.

This algorithm will require j sequential additions in L-level arithmetic. The addition could be done in either analog or digital given the hybrid assumption in this analysis. To establish an energy bound on this function, we know that analog-digital converters consume $O(L)$ kT [Murmann 2004] and digital addition is $O(\log L)$ kT. We thus assume $O(E_{add}) \leq O(L)$ kT.

The energy efficiency of the subdivided vector-matrix multiply is described with help of Table 7, followed by a discussion of engineering consequences. For large L, subdivision by j will reduce energy consumption by a factor of j, since E_{add} will be dominated by L^2. For $j=N$, the algorithm reduces the exponent of N from 2 to 1 for the overall vector-matrix multiply. However, fully extending this process to $j=N$ essentially transforms the memristor array into a memory feeding an external processor.

Table 7: Improving energy efficiency of vector-matrix multiply.

	E_{mvm}	p_g	Clock cycles	Clock period	P = *Power*
Analog array	$O(N^2 L^2 M)$ kT	1	1	T_{clock}	$\propto 1$
j steps × $1/j$ th size	$j\, O(N^2 L^2 p_g^2 M)$ kT + jME_{add} $= O(1/j \times N^2 L^2 M)$ kT + jME_{add} $\leq O((N^2 L + j^2)/j\, ML)$ kT	$1/j$	j	T_{clock}/j	$\propto 1/j$
$j = N$ full extension	$O(N L^2 M)$ kT + NME_{add}	$1/N$	N	T_{clock}/N	$\propto 1/N$

It is notable that the subdivision into sequential steps need not make the algorithm slower. There is a time-power tradeoff available to the engineer, but the authors suggest that the algorithm could be performed in j clock periods where the time of each clock period is reduced from T_{clock} to T_{clock}/j. This would maintain the same total computation time and throughput at lower power.

For small L, the situation is different but with interesting consequences. E_{vmm} in the middle row of the table can be factored as $O((N^2 L + j^2)/j\, ML)$ kT. This expression has an extremum at $O(N^2 L) = O(j^2)$, or $O(j/N) = O(\sqrt{L})$. This implies there is some j that optimizes energy efficiency, which could possibly be $1 < j < N$ and reveal a hybrid architecture to be most efficient. However, identifying the optimal value of j would require technology details beyond the scope of this document.

Implementing the energy efficiency improvement suggested by Table 7 will require some engineering changes. The changes will be summarized, but a full description is beyond the scope of this document.

It will be necessary to put a “select device” [ITRS 2012] in series with the memristors. While a select device will be an additional R&D activity, select devices may become available through the natural flow of events.

The behavior indicated by Table 7 could be implemented by an array of memristors if rows of memristors could be connected and disconnected as part of the algorithm. This is not the same as just “not using a row” by setting the drive voltage to zero. As stated previously, the divider network in Figure 13 computes the dot product of $\boldsymbol{v}$ and $\boldsymbol{g}$ divided by the sum of the elements of $\boldsymbol{g}$. While setting a drive voltage to zero will remove terms from the dot product, the memristor now connected to ground will continue to contribute to the sum of the elements in $\boldsymbol{g}$, reducing the amplitude of the output signal. In contrast, a select device will electrically disconnect a memristor so it does not contribute additional loading on the column conductor.

Current roadmaps indicate that manufacturing processes for memristors will be developed initially for storage applications like memory sticks and Solid State Disks (SSDs), after which the manufacturing capability could be repurposed for neuromorphic systems. Some form of select device is believed to be necessary to scale memristor memories to commercially viable size. Therefore it would be a reasonable to expect some form of select device to be available to the engineer of a neuromorphic system.

The middle row in Table 7 discloses an architectural generalization of the analog network described previously in this document and a digital implementation of the same function. By varying parameter *j* from 1…*N*, this architecture incrementally changes itself from analog to digital, covering intermediate stages where it does analog processing of submatrices that are summed digitally.

Extending the hierarchy of neural-network algorithms

The development of computing was enabled by the ability to write programs in a hierarchy of loops, subroutines, and so forth. Biological studies of brains indicate a multi-level structure with feedback, global timing (e.g., neural oscillations or brain waves), and sleep/wake cycles. This observed behavior of living neural systems is more complex than any of the systems discussed so far in this paper, but we will now create more complex neural network algorithms and estimate the energy efficiency.

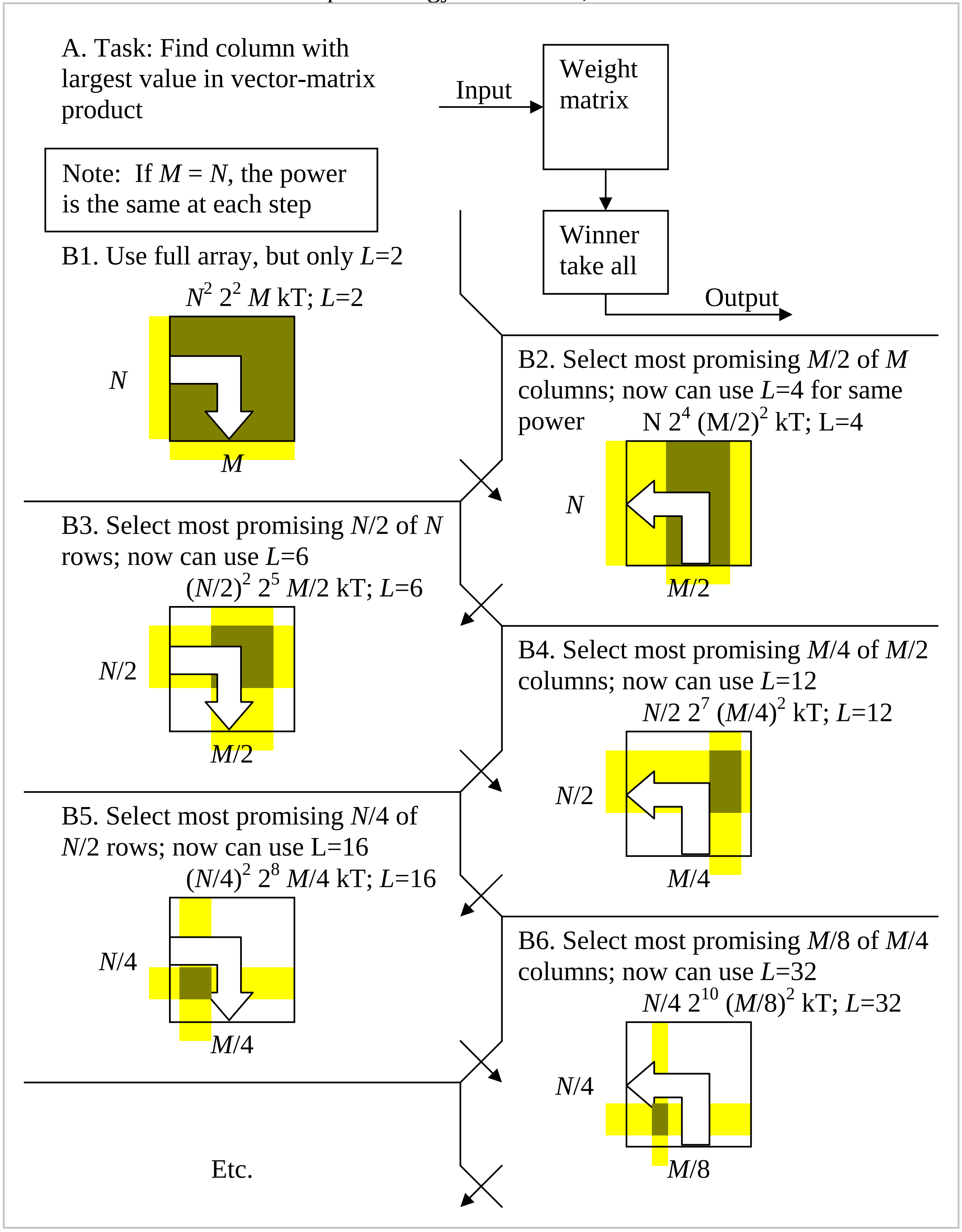
In many cases, a single-level neural network (i. e. what we have discussed in this document) is followed by a "winner take all" (WTA) function. WTA has been suggested as both the function of specific neural regions, such as the dentate gyrus in episodic memory algorithms [de Almeida 2009], thalamocortical modules in confabulation algorithms [Solari 2008], as well as representing the core decision boundary of neural machine learning techniques such as adaptive resonance theory (ART) and radial basis function (RBF) algorithms. The idea here is that the full vector-matrix multiply is not necessary for neural function, but rather finding the output with the largest value (or simply one of the largest values) is often sufficient. Given that the full vector matrix multiplication is not necessary for the intended computation, we can consider a more efficient algorithm that finds only the desired result.

This section will continue to use the memristor array in Figure 11A, but will presuppose more sophisticated drive electronics with the following new features:

1. *L* will be controllable. The drive electronics will be able to change clock periods, voltage drive, and perhaps amplifier bandwidth such that the *L* can be selected as needed for an algorithm (within an allowable range).
2. The drive electronics will be duplicated with row and column function swapped. Mathematically, this will change an operation on a matrix to an operation on the transpose of the matrix.

The strategy we describe here provides a WTA function, but leverages varying precision to achieve this target in a uniquely energy-efficient manner. The algorithm consists of running the vector-matrix multiply algorithm multiple times, initially starting with all the rows and columns but low precision. As the algorithm proceeds, rows and columns will be turned off. This would reduce *N* and *M* and potentially cut power. However, we will raise *L* as the algorithm proceeds to keep power constant. The algorithm would complete when the system is operating at the maximum precision on as many rows and columns as are feasible given power constraints.

Figure 15 illustrates the method. Figure 15A shows a rectangular weight matrix feeding a WTA network. This would require energy $N^2 L^2 M$ kT, which we seek to reduce.



A. Task: Find column with largest value in vector-matrix product
Input
Weight matrix
Winner take all
Output
Note: If $M = N$, the power is the same at each step
B1. Use full array, but only L=2
$N^2 2^2 M$ kT; L=2
N
M
B2. Select most promising M/2 of M columns; now can use L=4 for same power
N 2^4 (M/2)^2 kT; L=4
N
M/2
B3. Select most promising N/2 of N rows; now can use L=6
$(N/2)^2\ 2^5\ M/2$ kT; L=6
N/2
M/2
B4. Select most promising M/4 of M/2 columns; now can use L=12
$N/2\ 2^7\ (M/4)^2$ kT; L=12
N/2
M/4
B5. Select most promising N/4 of N/2 rows; now can use L=16
$(N/4)^2\ 2^8\ M/4$ kT; L=16
N/4
M/4
B6. Select most promising M/8 of M/4 columns; now can use L=32
$N/4\ 2^{10}\ (M/8)^2$ kT; L=32
N/4
M/8
Etc.

Figure 15: Neural network algorithm.

Figure 15B shows the improved algorithm. Vector-matrix multiply is performed originating both from the left and bottom edges of a memristor-type array and flowing in the direction of the elbow-shaped arrow. The sparsity of each operation is indicated by the yellow and dark-yellow regions. The dark-yellow regions are correctly scaled to show the size of the active region of the sparse matrix, but are shown as contiguous for illustration convenience. Figure 15B1 shows all N rows and M columns being activated at once but with just two levels. The energy expression shows consumption of $N^2\ 2^2\ M$ kT, which is must less energy than required for Figure 15A. In each of the subsequent steps, half of the rows or columns will be turned off. The 90° arrow reverses direction to show the reversal of the direction of vector-matrix multiply.

The idea is that each step reveals the half of the rows or columns that are most strongly coupled to the driving side. These become the most likely candidates to be the ultimate winner. So, the algorithm turns off the most likely losers and proceeds to test the winners again with higher precision.

If we assume $N = M$, the power will be the same at each step.

The number of levels increases in the sequence 2, $\sqrt{}2$, 2, $\sqrt{}2$, … This means the number of iterations required to reach L levels is about $\log_{1.68} L$.

This means the algorithm has reduced energy of $O(N^2\ L^2\ M)$ kT to $O(\log_{1.68}L\ N^2\ M)$ kT, a reduction by a factor of $O(L^2/\log_{1.68}L)$.

This approach illustrates a neural network algorithm generally inspired by the WTA algorithms often used in neural computation [de Almeida 2009] (de Almeida, Idiart, and Lisman 2009), but this is not a specific WTA algorithm. Importantly, it is unlikely that this algorithm, or similar variants, would be guaranteed to always provide the *best* answer; instead, as with real neural systems, it is simply assured to provide a *good* answer. Nonetheless, this approach allows us to use the kT complexity to compare the energy efficiency of the algorithm to the more straightforward approach of simply performing a vector-matrix multiply and finding the largest result and make design decisions balancing accuracy and energy consumption accordingly.

Digital computer implementations

This section will discuss energy consumption of a digital implementation comparable to Figure 11, which will lead to an understanding of a hybrid analog-digital architecture that can be changed from analog to digital in a series of steps. The digital implementation in this section would be essentially the result of taking the analog neural network in Figure 11 and evaluating groups of rows in parallel and then multiple groups sequentially. The digital implementation would be the result of the maximum number of groups, which would be one row at a time. This implementation becomes the last row of Table 7. Furthermore, the digital implementation must be sufficient to execute algorithms such as in Figure 15, which require running the analog circuit backwards.

The complicated part of this section will be to assess both computational energy and memory access energy.

Figure 16 shows C-language software for the dot product of two N-element vectors of ints, ints in C being data words of an implementation-dependent number of bits treated as an integer. Let us assume that the ints are $B = \log_2 L$ bit of precision and that we can ignore integer rounding issues.

```
int v[N], w[N], sum = 0;
for (int i = 0; i < N; i++)
    sum += v[i] * w[i];
```

Figure 16: Software dot product

Each B-bit arithmetic unit will require B^2 gated one-bit full adders, each comprising about a half-dozen gates (there are different adder designs). This leads to a complexity of $N B^2 = N \log_2^2 L$ gate operations for the multiply.

Each synapse in a digital implementation would require the energy from a $B \times B$ array of gated full adders multiplied by, say, 3× for overhead. This would allow computation of the energy $E_{digital}$ of the digital implementation

$$E_{digital} = 3 \log_2^2(L)\, E_{fulladd}, \qquad (1)$$

where $E_{fulladd}$ is the energy of a full adder. Example energies of full adders are given in Table 8.

Table 8: Full adder energy. Top two entries from [Nikonov 2013]

	Energy/32-bit Adder Fig 47	$E_{fulladd}$ units of kT
CMOS HP	3 fJ	22,000 kT
HomJTFET	.15 fJ	1100 kT
Thermal limit 100 kT/gate		9 gates × 100 kT = 900 kT
Landauer limit		3 kT

The description just given does not address energy associated with accessing memory, so one of the authors (DeBenedictis) has a separate paper [DeBenedictis 2014] dealing with a vector-matrix multiply structure that is compatible with the adiabatic memory disclosed in Ref. [Karakiewicz 2012]. This other work will be summarized here with the reader referred to Ref. [DeBenedictis 2014] for more detail.

The approach used in Ref. [DeBenedictis 2014] and illustrated in Figure 17 allows a matrix to be accessed either by row or by column and for that access to use highly energy efficient methods of driving voltages to the rows

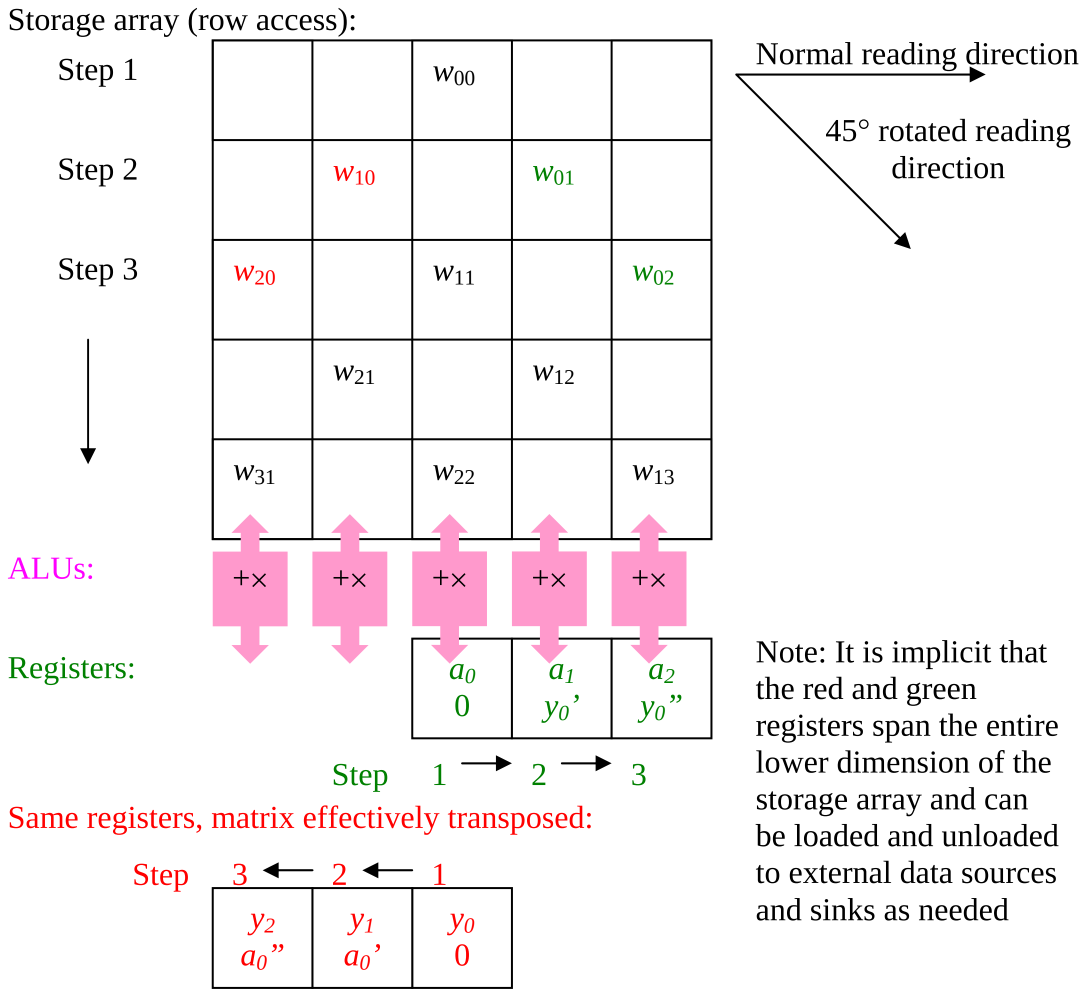

Storage array (row access):
Step 1
Step 2
Step 3
w00
w10
w01
w20
w11
w02
w21
w12
w31
w22
w13
Normal reading direction
45° rotated reading direction
ALUs:
+×
+×
+×
+×
+×
Registers:
a0
0
a1
y0’
a2
y0”
Step 1 → 2 → 3
Same registers, matrix effectively transposed:
Step 3 ← 2 ← 1
y2
a0”
y1
a0’
y0
0
Note: It is implicit that the red and green registers span the entire lower dimension of the storage array and can be loaded and unloaded to external data sources and sinks as needed

Figure 17: Matrix organization that transposes by processing direction

A key part of the strategy is to represent a matrix in a way that can be effectively transposed by reversing the processing direction from leftward to rightward. Figure 17 shows the matrix elements stored in the storage array rotated 45° clockwise from the normal way matrices are illustrated. The green colored portions of the diagram show the processing of a row: The storage array rows are read from top to bottom in subsequent steps. The rose-colored ALUs process data from the selected row and a data register below it. The w_{00} matrix element goes to the center ALU in step 1, which also processes the register data (labeled a_0, 0). The state in each processing unit is shifted right after each second step, with the vector element shifted left.

The approach in Figure 17 is then enhanced by associating with each matrix element w_{ij} two codewords of perhaps 2 bits each (4 bits total) that indicate the relative location of the next element in the same row and column.

The consequence of this approach is that the memory can be accessed by accessing full rows one at a time from top to bottom. In addition to this access pattern being the same as the one needed for Table 7, the access pattern is compatible with the charge-recycling logic in Ref. [Karakiewicz 2012]. The benefit of the charge recycling is projected to be an 85× reduction in energy to (an aggressively projected $E_{TMACSBIT}$ = .91 fJ/bit access).

The overall energy for vector matrix multiply is therefore projected to be ETMACSBIT × (synapse bits + 4) + $E_{FULLADD} \times B^2$. (check this over)

Dot product using signal representation as spikes

It is widely recognized in the literature that the representation of information by spikes can be very efficient for certain computational functions [Berger 2010], but the kT energy complexity will require discussion. While there has been considerable research in hardware solutions for spike generation [Indiveri 2011] [Pickett 2013], we feel engineered solutions for spiking logic are incomplete. We will therefore create a simplified spike representation for this energy analysis.

We propose the self-consistent number representation illustrated in Figure 18. In this simplified representation, numbers are represented by a stream of spikes that are randomly distributed with the Poisson distribution. A continuous variable $0 \le v \le 1$ would be represented by a spike stream with an average of $v\,L^2$ spikes in a reference time interval T_{ref}. Variable v_j is therefore represented by a stream of spikes at rate $v_j\,L^2/T_{ref}$.

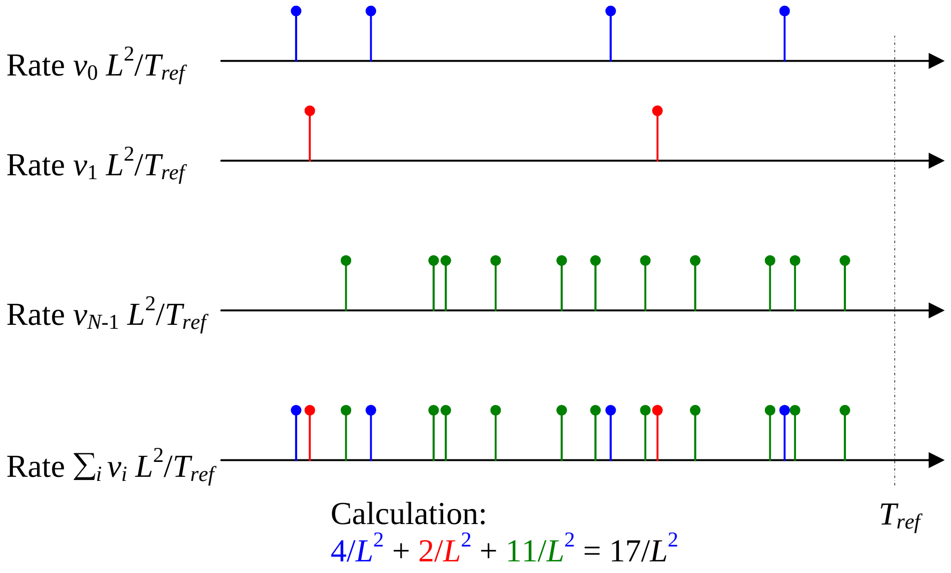
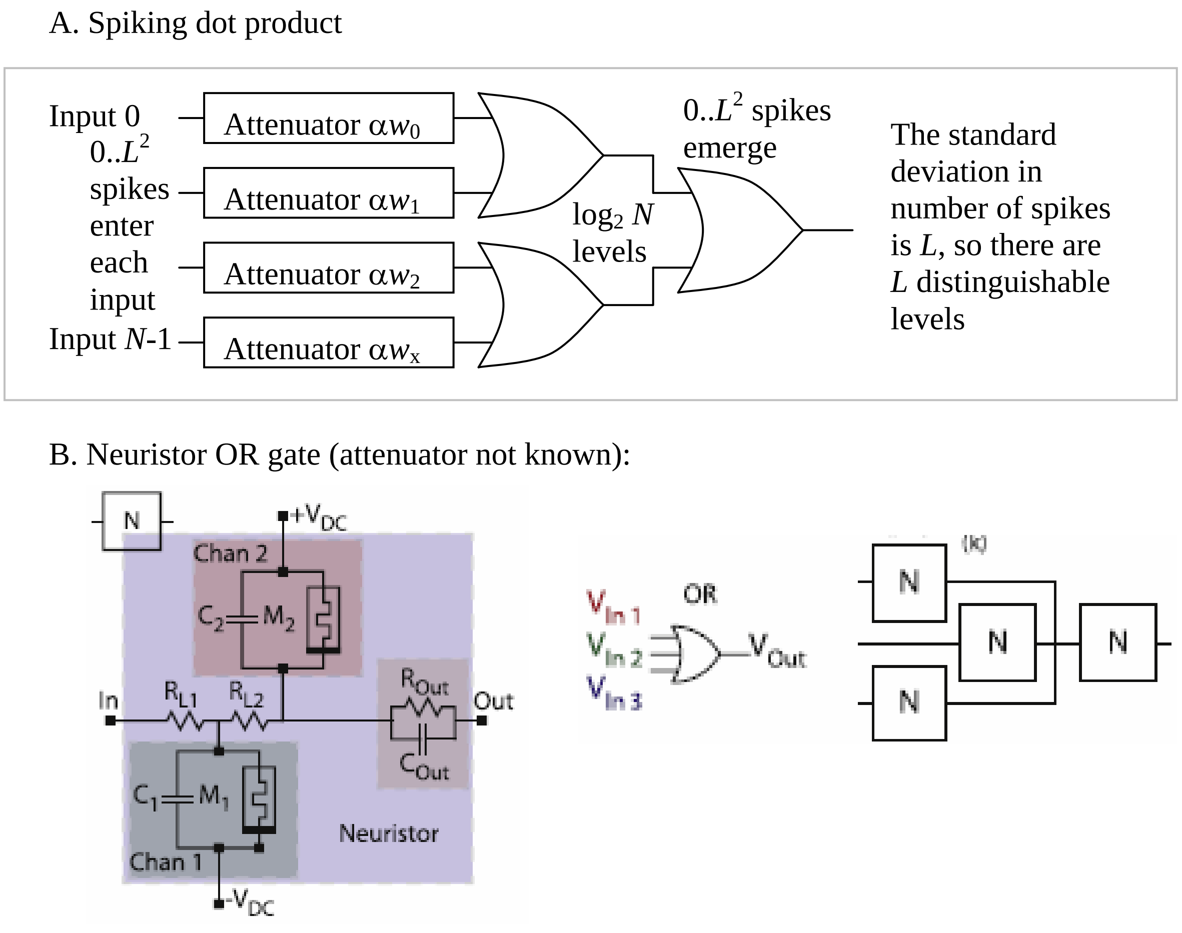

Rate $v_0\,L^2/T_{ref}$
Rate $v_1\,L^2/T_{ref}$
Rate $v_{N\text{-}1}\,L^2/T_{ref}$
Rate $\sum_i v_i\,L^2/T_{ref}$
T_{ref}
Calculation:
$4/L^2 + 2/L^2 + 11/L^2 = 17/L^2$

Figure 18: Summing with spikes

Figure 18 illustrates how addition in the number system can be accomplished simply by aggregating spike streams. If streams with rates $v_0\,L^2/T_{ref}$ .. $v_{N\text{-}1}\,L^2/T_{ref}$ are simply aggregated, the result will be a stream with a spike rate that is the sum of the rates of the input streams. Thus, stream aggregation adds the rates.

The precision of a number in the rate-modulated code depends on the number of spikes. The uncertainty in the measurement will be the standard deviation in the number of spikes, which will be the square root of the number of spikes. The largest standard deviation will be when v = 1 and there are L^2 spikes. This point of maximum uncertainty will have a standard deviation of L spikes out of L^2, for an equivalent of L levels. L thus has the same definition as used previously in this document.

Spikes can be aggregated with remarkably high energy efficiency as shown in Figure 19. Ignoring the attenuators for now, Figure 19A comprises a binary OR-gate tree of logarithmic depth. While biological neurons do not have a binary tree of OR gates, the biological equivalent would be a dendrite structured as a (generally non-binary) tree of branches.


A. Spiking dot product

B. Neuristor OR gate (attenuator not known):

Figure 19: Dot product for rate modulated spikes

According to the theory outlined in Ref. [Landauer 1961], heat must be created when the information entropy of a gate's output is less than the inputs. A gate in Figure 19A receiving no spike produces no change in entropy when it computes an output with no spike. If a gate in Figure 19A receives a spike on either or both inputs and produces a spike on its single output, it has "forgotten" which input combination it received the spike and must generate heat on the order of kT.

With the attenuators, Figure 19A actually performs a dot product of the weights w_j in the attenuators and the input signals. The vision for how such a circuit would work (should somebody engineer the required attenuator) is that the attenuators would randomly delete spikes such that the fraction αw_j pass through. If each attenuator passes a random fraction αw_j of a random spike stream of Poisson distribution with rate $v_j\, L^2/T_{ref}$, the attenuator's output will be a spike stream with Poisson statistics but rate $\alpha w_j v_j\, L^2/T_{ref}$, thus performing the multiplication $w_j v_j$ as needed in computing the dot product.

Following biological inspiration, the idea is that the attenuators would either pass a spike or completely ignore it without consuming energy [Mainen 1995]. This attenuation is thus comparable to the probability of release at the pre-synaptic terminal of a biological synapse. An important note is that this correlation of the probability of release to the weight, while effectively assuming uniform conductances, is somewhat complementary to that of biological systems in which the probability of release is often assumed to be uniform (aside from short term plasticity effects) with variable post-synaptic conductances.

The energy dissipation of the dot product circuit Figure 19A will depend on the aggregate number of spikes that pass through all the gates during the computation of a dot product. We assume α is adjusted to scale the number of spikes using methods outside the scope of this document. Irrespective of how the number of spikes is reduced to the range $0..L^2$, the total number of gates activated will be $0..L^2 \log_2 N$ for a theoretical minimum energy of $0..L^2 \log_2 N$ kT, or in more practical implementations $0..L^2 \log_2 N\, E_{spike}$, where E_{spike} is the energy of a spike.

While the OR gates in Figure 19A are useful for illustration, some form of "digital restoration" would be required to have a complete system. While a spike entering one of the OR gates in Figure 19A would produce something that looks like a spike on the output, the shape of the spike (e. g. the spike's width) would gradually change due the buildup of circuit and timing imperfections and noise. A mechanism would be needed to restore spikes to a canonical form (a concept called digital restoration). The neuristor [Pickett 2013] could be the device/circuit for implementing spikes such as in Figure 19B. The energy/spike in (Williams) is 6-60 fJ. The neuristor circuit in Figure 19B performs the same logical function as an OR gate, but restores the output to a pulse instead of a level.

The reader should note the profound difference in the scalability of spiking representation. The energy of spiking representation scales with $L^2 \log N$, which has slow growth with N. The other methods in this document scale at least linearly with N. However, the price spiking's efficient scaling with N would appear to be quadratic scaling with L.

Summary of scaling

We will make comparative plots of scaling of various implementation approaches, after defining those approaches. As summarized in Table 9, we analyze (in approximate order of top-to-bottom in the plots – which is most efficient at bottom) a consumer GPU, the resistive crossbar neural network, Landauer's limit, a spiking implementation based on neuristors, and a PIMS system [DeBenedictis 2014].

Table 9: Summary of vector-matrix product energies.

Implementation	Energy ($N \times M$ matrix, $N = M$, L-level values, $B = \log_2 L$)
nVidia GTX 750 Ti	A simple analysis indicates the nVidia GTX 750 Ti (a state of the art consumer GPU at the time of this writing, costing $150) will be memory bandwidth limited. The computational strategy is the assume it will consume its standard 60 Watts and process data as fast as it can be read from memory. The memory rate will be based on synapse values of the specified number of bits + 4 bits of sparsity control information from [(appendix 1)].
Resistive crossbar	1/24 N^2 L^2 p^3 M kT, as discussed in text.
Landauer's Limit	Full adder is 3 kT. Energy is 3 N^2 p^2 B^2 kT ln 2
Neuristor spike train	L^2 $\log_2 N$ spike energy = 2^{2B} $\log_2 N$ E_{spike}; 6-60 fJ/spike for a neuristor (we use 6 fJ), as disclosed
PIMS	See author's other paper; TFET logic and adiabatic memory

Energies per vector-matrix multiply are plotted in Figure 20 for four scenarios, with all scenarios have $N = M$ (square matrix). The top row represents $L = 65536$ levels or $B = 16$ bits = $\log_2 L$ arithmetic, which is generally representative of ANNs that were developed for implementation on digital computers. (While single precision floating point with 21 bits precision may be more common, using 21 bits instead of 16 would simply strengthen and already strong distinction). The bottom row represents $L = 8$ levels or $B = 3$ bits = $\log_2 L$ arithmetic, perhaps more representative of biology.

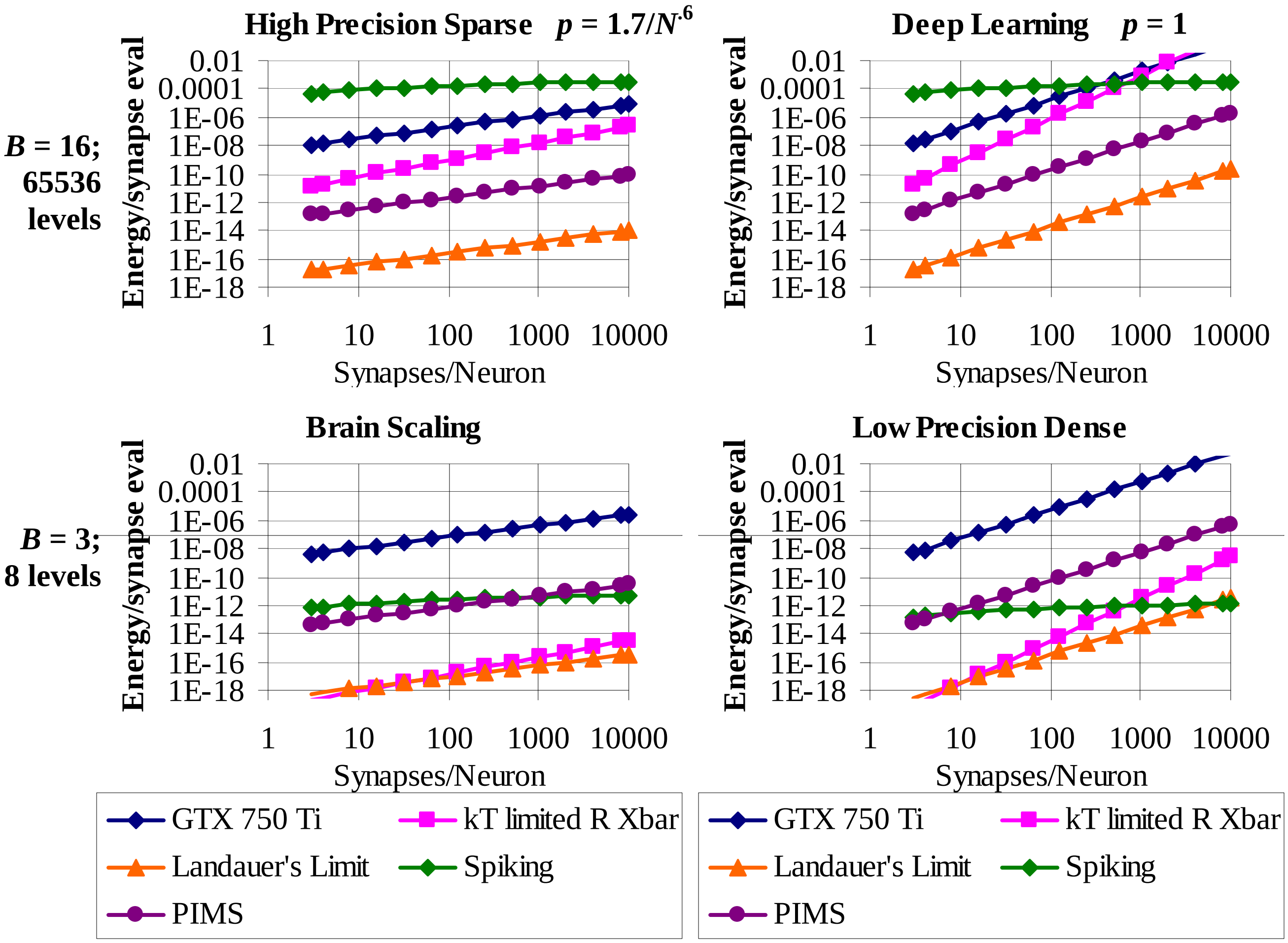


Figure 20: Comparison of implementation approaches

The right column represents the obvious dense vectors and matrices (p = 1), but the left column represents a sparsity pattern somewhat representative of biology. The specific equation is $p = 1.7/N^{.6}$, which was developed by hand to represent the sparsity the authors subjectively observe in the literature. Specifically, the product Np varies from 2.6 to 68 as N varies from 3-10,000. This is consistent with the authors understanding of the sparse coding biology uses.

The reader’s attention is called first to the green curves representative of spiking. Spiking is nearly unaffected by the number of synapses, but has poor performance at high precision. Interestingly, the human-manufactured and measured neursitor matches Landauer’s limit in the low-precision, dense, scenario, at scale. This remarkable achievement is due to the better algorithmic scaling of spiking overcoming the handicap of having been implemented with components created in a real laboratory as opposed to the infinitely advanced components assumed by Landauer’s method.

The performance of the resistive crossbar neural network deserves comment. It is the worst performer in the Deep Learning scenario and an excellent performer in the Brain Scaling scenario. It would seem reasonable for the human programmers that created Deep

Learning technology to have adapted their information coding and data structures to the properties of digital computers. Due to the digital signals being restored (as opposed to analog signals that accumulate noise that cannot be removed) and place-value number representation having L levels of precision for $\log_2 L$ resources (bits), software for digital logic can use large quantities of high-precision numbers with relatively low impact on energy efficiency. However, this type of data will exacerbate the power of the analog neural system due to the $N^2 L^2 M$ scaling being quadratic in precision and cubic in scale (assuming square matrices where $N = M$).

However, the curve for the resistive crossbar neural network is always steeper than the digital implementations.

The resistive crossbar neural network will have issues in two classes to succeed as a Beyond CMOS option. First, the analysis in this document is based on best performance theoretically possible. Actual implementations of the resistive crossbars will need to improve. Second, implementers will need to cope with the less effective scaling. This analysis shows that even with perfect devices, a consumer-grade GPU will be more energy efficient for data derived from Deep Learning. The remedy to the second issue would be sparse coding, small scale, or perhaps a hybrid architecture.

Conclusions

We developed a method of cross-comparing the energy efficiency of ANN technology stacks between themselves and CMOS. The method can compare large amounts of technology, but is not very precise. CMOS is highly mature due to an investment of around a trillion dollars whereas ANNs have received just a miniscule fraction of that amount. To avoid the comparisons being biased by either overhyping immature concepts or the large amount of data available for highly mature concepts, we developed an approach based on theoretical analysis of the physical limits. This approach could make broad statements like “concept A will have worse energy efficiency than concept B by an amount that grows over time” but would be unable to say that “concept A has 1.23× better energy efficiency than concept B,” if both were in fact true.

A conclusion can be drawn about the comparison between resistive crossbar neural networks and CMOS digital implementations. The energy to perform a vector-matrix multiply of dense $N \times M$ arrays of fixed point numbers of L levels of precision is $O(N^2 L^2 M)$ kT for the resistive crossbar neural network and $O(N \log^2(L) M)$ kT for digital. This clearly favors the digital implementation for large N and large L (large L equating to high precision).

However, we also analyzed a sparse data representation where only the proportion p (let us assume here $p=p_v=p_g$) of the numbers were nonzero. The resistive crossbar energy is actually $O(N^2 L^2 p^3 M)$ kT in this case. The expressions indicate that a resistive crossbar could possibly beat CMOS if sparse data representations were exploited. While the proponents of resistive crossbar neural networks often claim that they are exploiting sparse data representations, this is rarely, if ever, used as a design criterion with a formal quantification (e. g. $p=1/\sqrt{N}$). We can conclude that resistive crossbar neural networks are thus most likely to be viable at small scale or if sparse coding can be added to the technology mix.

We analyzed the scalability of an artificial spiking neural network and found it to have major advantages – although the absence of a proposed realization of a suitable manufactured synapse is a current flaw. In the spiking vision, a vector-matrix multiply would be O(log(N) L^2 M) kT. This would significantly beat the O(N $\log^2(L)$ M) kT energy of a digital implementation for large N and small L. However, the vision includes a synapse that duplicates the probability of release at the pre-synaptic terminal of a biological synapse. In other words, the synapse passes a spike to its output with probability p or ignores it with probability 1-p it (where “ignoring” means not consuming any energy). In contrast, what we see in the literature are synapses that absorb a proportion p of the spike’s energy. Thus, we conclude artificial spiking neural networks have upside potential if a physical realization of the necessary synapse can be found.

(By the way, we have not analyzed the use of sophisticated software algorithms on a conventional computer. These could offer advantage over sparse matrix algorithms.)

We have also applied the method to higher-level algorithms. The vector-matrix multiply function and the expressions for its energy consumption of a resistive crossbar neural network were used as a building block for a higher-level algorithm. The example algorithm was equivalent to a vector-matrix multiply followed by a “winner take all” function. The combination outputs the largest element in the product. The algorithm involves multiple applications of the resistive crossbar where the precision varies, rows and columns are turned off, and the system is run both forwards and backwards. The algorithm reduces the energy consumption from O(N^2 L^2 M) kT (per above) to O($\log_{1.68}L$ N^2 M) kT. The fact that higher-level algorithms have an advantage is widely understood in biology and computer science, but we have developed a framework that can quantify the advantage for ANN implementations.

The method of quantifying energy efficiency can be compared to other methods of performance estimation.

The energy of a neuron operation can be compared to the energy-per-gate-op in conventional computing logic gates. However, the equivalent neuron parameters are functions of the number of synapses instead of simply being scalars. We show how to express the energy of a neuron operation as a multiple of kT, which makes it cross-comparable to computer logic gates in accordance with the principles outlined by Landauer.

An important outcome of this analysis is a strategy for assessing the advantages of neurobiological properties. While the brain appears to leverage a number of unique approaches, such as hybrid analog-digital spiking, temporal coding, highly parallel computation, the decision to implement these in conventional computational systems has often been *ad hoc* due to the lack of a rigorous way of quantifying their contribution. The sparse coding of biological neural circuits is a notable example of this.

This limitation of analog hardware on accelerating dense neural algorithms allows digital approaches to be advantageous. The crossbar analog neural network and a CPU/Graphics Processing Unit (GPU)/Processor-In-Memory (PIM) system are both elements of a more general architecture. The more general architecture would process batches of rows in groups, with one batch corresponding to an analog neural network and *N* batches of size 1 corresponding to digital hardware. The digital computer is more energy efficient in key cases, disputing unproven assertion that neuromorphic hardware is simply superior to von Neumann-class computation.

As future work, it should be possible to devise more ANN technology stacks and quantify their performance. These systems would comprise a series of interconnected ANNs, like the layers in a Deep Learning system or a cortical network. However, these ANNs would be executed by Beyond CMOS components (i. e. neither Deep Learning software nor living cells) and would be able to execute algorithms where the physical components are run forward, backwards, turning elements on and off, shifting thresholds, etc. The methods in this document would allow the algorithms to be assigned a computational/energy complexity in the form $F(N)$ kT that would quantify the algorithm's quality. From a study of multiple algorithms of this type, perhaps devices and architectures could be devised that provide good performance over a variety of algorithms, thereby forming a possible Beyond Moore's Law computer system.

References

[Berger 2010] Berger, Toby, and William B. Levy. "A Mathematical Theory of Energy Efficient Neural Computation and Communication." Information Theory, IEEE Transactions on 56 (2): 852–74.

[DARPA Synapse] A DARPA program announced by "Broad Agency Announcement for Systems of Neuromorphic Adaptive Plastic Scalable Electronics (Synapse)," DARPA Defense Sciences Office DARPA-BAA 08-28

[Dayan 2001] Dayan, Peter, and Laurence F. Abbott. Theoretical Neuroscience. Cambridge, MA: MIT Press. http://f3.tiera.ru/2/Cs_Computer%20science/CsGn_Genetic,%20neural/Dayan%20P.,%20Abbott%20L.F.%20Theoretical%20neuroscience%20(2002)(432s).pdf.gz.

[de Almeida 2009] De Almeida, Licurgo, Marco Idiart, and John E. Lisman. "A Second Function of Gamma Frequency Oscillations: An E%-Max Winner-Take-All Mechanism Selects Which Cells Fire." The Journal of Neuroscience 29 (23): 7497–7503.

[DeBenedictis 2014] E. DeBenedictis, "A Computing Paradigm Beyond Moore's Law and Beyond the von Neumann Architecture," companion paper.

[Eliasmith 2012] C. Eliasmith, "A large-scale model of the functioning brain," Science, vol 338, no 6111, pp. 1202-1205, 2012.

[Hinton 2006] Hinton, Geoffrey E., and Ruslan R. Salakhutdinov. “Reducing the Dimensionality of Data with Neural Networks.” Science 313 (5786): 504–7.

[Indiveri 2011] Indiveri, Giacomo, Bernabé Linares-Barranco, Tara Julia Hamilton, André Van Schaik, Ralph Etienne-Cummings, Tobi Delbruck, Shih-Chii Liu, et al. “Neuromorphic Silicon Neuron Circuits.” Frontiers in Neuroscience 5. http://www.ncbi.nlm.nih.gov/pmc/articles/pmc3130465/.

[IRE 1957] “IRE Standards on Electron Tubes: Definitions of Terms, 1957”, Proc IRE Vol 45 983-1010, July 1957.

[ITRS 2012] ITRS Emerging Research Devices. “Emerging Memory Select Device Workshop.” http://www.itrs.net/ITWG/Beyond_CMOS/2012April/ERD%20Select%20Device%20Workshop%20Report.pdf.

[ITRS 20XX] International Technology Roadmap for Semiconductors, http://www.itrs.net. Note: ITRS covers neuromorphic options, notably in the Emerging Research Devices section.

[Johnson 1928] J. Johnson, “Thermal Agitation of Electricity in Conductors,” Phys. Rev. 32, 97, 1928

[Karakiewicz 2012] R. Karakiewicz, R. Genov, G. Cauwenberghs, “1.1 TMACS/mW Fine-Grained Stochastic Resonant Charge-Recycling Array Processor,” IEEE Sensors Journal, vol. 12, no. 4, April 2012.

[Landauer 1961] Landauer, R. 1961. “Irreversibility and Heat Generation in the Computational Process.” IBM Journal of Research and Development 5 (3): 183–91.

[Le 2013] Q. Le, “Building high-level features using large scale unsupervised learning,” 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).

[Llewellyn 1931] F. Llewellyn, “A Rapid Method of Estimating the Signal-to-Noise Ratio of a High Gain Receiver,” Proceedings of the Institute of Radio Engineers Volume 19, Number 3 March, 1931. Note: The concept of noise factor or noise figure developed in multiple stages. A much more final definition is in [IRE 1957].

[Mainen 1995] Mainen, Zachary F., and Terrence J. Sejnowski. “Reliability of Spike Timing in Neocortical Neurons.” Science 268 (5216): 1503–6.

[Merolla 2014] P. Merolla, et. al., “A million spiking-neuron integrated circuit with a scalable communication network and interface,” Science 345, 668 (2014).

[Murmann 2004] B. Murmann, B. Boser, “Digitally Assisted Pipeline ADCs,“ Kluwer Academic Publishers.

[Nikonov 2013] Nikonov, Dmitri E, and Ian A Young. 2013. “Overview of Beyond-CMOS Devices and A Uniform Methodology for Their Benchmarking.” arXiv Preprint arXiv:1302.0244.

[Nyquist 1928] H. Nyquist, “Thermal Agitation of Electric Charge in Conductors,” Phys. Rev. 32, 110, 1928

[Peng 2008] Peng, Wang. 2008. “Energy Analysis Method of Computational Complexity.” In Computer Science and Computational Technology, 2008. ISCSCT’08. International Symposium on, 2:682–85. IEEE. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4731715.

[Picket 2013] Pickett, Matthew D., and R. Stanley Williams. “Phase Transitions Enable Computational Universality in Neuristor-Based Cellular Automata.” Nanotechnology 24 (38): 384002.

[Shannon 1949] Shannon, C., “Communicaiton in the presence of Noise,” Proceedings of the IRE (Volume:37, Issue: 1)

[Solari 2008] Solari, S., Smith, A., Minnett, R., & Hecht-Nielsen, R. (2008). “Confabulation theory,” Physics of Life Reviews, 5(2), 106-120.

[Taha 2013] Taha, Tarek M., Raqibul Hasan, Chris Yakopcic, and Mark R. McLean. “Exploring the Design Space of Specialized Multicore Neural Processors.” In Neural Networks (IJCNN), The 2013 International Joint Conference on, 1–8. IEEE. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6707074.

[Williams 2014] Williams, R. Stanley. “Neuromorphic Circuits: Nonlinearity & Neuristors.” http://nice.sandia.gov/documents/RS%20Williams%20NICE%202014.pdf.

Appendix III: Spin Wave Computing

Spin Wave Computing

Peter Sharma's document "Spin wave computing, Peter Sharma, 8656, Materials Physics," (received by e-mail 9/13/2013)

171060 SAND Report (PI: DeBenedictis)
Spin wave computing, Peter Sharma, 8656, Materials Physics

Overview of spin wave computing

Why spin waves? What are spin waves? Properties of spin waves. Spin wave logic. Experimental progress for logic functions. Spin wave logic materials. Logic requirements and spin wave computing. Advantages of spin wave computing. Disadvantages of spin wave computing. Spin wave computers and optical computers.

Benchmarks for spin wave computing

Nikonov and Young's uniform benchmark methodology. Six figures of merit, Energy for excitation/detection of spin waves. 1-bit adder using spin waves. Logic delay and critical path in 1-bit adder. Systems level evaluation of spin wave computing using Rent's rule.

Overview of spin wave computing

The energy and speed of CMOS integrated circuits is largely determined by the charging and discharging of planar wire interconnects. For instance, charging long wires is estimated to make up more than 90% of the ~10^6 k*T* estimated energy cost of CMOS[1]. An important part of going "beyond" Moore's law is to devise a way of computing that does not involve charge. The spin of the electron has long been proposed as an alternative state variable to charge and there are numerous schemes for using spin for computing[2]. Devices based on spin-polarized charge currents are unlikely to have any advantages over CMOS technology[3]. Thus, one common aphorism in this field is that transporting only the spin degree of freedom will avoid the charging energy in a CMOS approach and thus lead to low power logic designs. We do not discuss spin-based memory technologies here.

Spin systems possess hydrodynamic modes[4] similar to conventional fluids, and can be both diffusive and propagating. Therefore, spin-based information transport is possible. Spin waves are propagating hydrodynamic modes of ordered ferromagnetic materials corresponding to transverse fluctuations around a uniform spin direction defined by a global magnetic field. There are several types of spin waves depending on wavelength, magnetic field, and the shape of the ferromagnetic material[5]. Hertel et al.[6] were the first to propose that logical operations could be performed through the interference of magnetostatic spin waves in ferromagnetic microstructures. Binary logic is accomplished by interfering two or more incoming spin waves and mapping the phase of the output spin wave to logic '0' and '1' for a phase of 0 and π, respectively. The phase of the outgoing spin wave is inferred from the outgoing amplitude.

Two groups were the first to explore the use of spin waves to perform simple logic functions for classical computing[7,8]. Using spin waves in this way was motivated by the discovery that Gaussian wave packets of magnetostatic spin waves could propagate over distances of over 50 μm in ~45 nm thick permalloy thin films[9] through microwave excitation. In this experiment, the magnetostatic modes of the permalloy thin film had wavelengths of ~18 μm and traveled at a group velocity of ~2.5×10^4 m/s. The group velocity of spin waves observed by Covington et al. is on the order of the speed of a surface acoustic wave in diamond and about three orders of magnitude less than electrical signal propagation speeds in metallic interconnects. The characteristic decay time of such spin waves was ~1 ns.

A NAND gate was demonstrated by Schneider et al.[10]. Spin waves were launched into the two arms of a Mach-Zehnder interferometer made of a low loss insulating ferromagnetic material, YIG. A relative phase is introduced between the spin waves using two separate DC currents, which induce a small difference in local magnetic field. The output phase is thus controlled by the difference in input DC current. A majority gate was demonstrated by Shabadi et al.[11] in a 20 nm thick NiFe alloy on a Si substrate, where three input spin waves with different phases were made to interfere, and the phase of the output spin wave possessed the majority phase. An inverter is required to make majority logic Boolean complete, but is considered to be easily constructed by fabricating a spin waveguide of length equal to an odd integer multiple of the wavelength. All logical operations using spin waves were demonstrated by microwave excitation using asymmetric coplanar microstrip antennas patterned directly on magnetic features. Typical spin waveguide widths in these logic experiments are microns. Spin wave propagation in a dense array of planar permalloy wires has been observed for widths down to 300 nm with ~100 nm spacing[12]. Arrays of wires exhibit different spin wave frequencies and propagation velocities than single wires. Circuits involving more than one logical operation have not yet been demonstrated, but have been extensively modeled[13,14]. Spin wave computing appears in the latest 2012 ITRS roadmap document[15].

The materials used in these experiments are either ferromagnetic metallic NiFe alloys (Permalloy), Co-Fe alloys, or ferrimagnetic insulator $Y_3Fe_5O_{12}$ (YIG). Spin wave computing materials should be ferromagnetic or ferromagnetic in order to have as large a magnetic moment as possible and low magnetic damping to ensure long coherence lengths for spin wave packets. A large magnetic moment is necessary for coupling a passing spin wave packet to proposed detection/amplification schemes. Any magnetic material with these properties should be useful for spin wave computing. Metals or insulators can be used, although insulators tend to have lower magnetic damping due to the lack of electron-magnon interactions. NiFe alloys are easily fabricated on Si substrates, making them in principle CMOS compatible. YIG requires a lattice matched substrate ($Gd_3Ga_5O_{12}$) to achieve the highest quality and thus the lowest magnetic damping. Deposition on Si may be possible with suitable buffer layers between Si and $Gd_3Ga_5O_{12}$, but this fabrication process has not been developed. We found no studies relating the microstructure of these materials to their spin wave propagation properties, so it is not clear what level of purity or crystallinity are needed for spin wave computing.

Computers carry out ~10^9 instructions (e.g. floating point operations) composed of many more sequential and parallel logical operations per second, which imposes several constraints on any new logic devices[16,17]. Logic devices must possess Boolean completeness, concatenability (multiple inputs/outputs must be possible to construct), gain, signal restoration, and input/output isolation. Nonlinearity is sometimes identified as a separate logic requirement[18], but here we conflate this idea with gain.

Spin wave devices can be made Boolean complete through NAND gates or majority and inversion gates and both approaches have been demonstrated experimentally for single logic operations. Spin wave computing encodes logical bits into the phase of a spin wave. Multiple spin waves can be made to interfere at different frequencies in parallel, so spin wave computing is believed to have a high fan in/fan out capability[19,20]. Covington et al. demonstrated that spin wave packets obey linear superposition[9], so this idea has some merit. The main disadvantages of spin wave computing are gain, signal restoration, and input/output isolation.

Spin wave logic relies on encoding a bit in the phase of a spin wave packet. We define the amplitude of a spin wave packet as positive for a phase of 0 and negative for a phase of π. In this way, the amplitude of a spin wave packet that has undergone interference between two or more input wave packets is a function of phase. A continuous range of amplitudes and phases are possible during a logical operation. There is no intrinsic gain (or nonlinearity) in a spin wave device. Gain must be built into spin wave devices using CMOS components.

There is no known method of achieving signal restoration for the phase of a spin wave packet. Many schemes have been proposed for detection/amplification of spin waves[13,21-23]. Spin waves have been also been proposed to switch the direction of a magnetic domain above a certain phase threshold[13]. However, none of these schemes enforce a phase of 0 or π at the output of a spin wave logic device. Signal restoration might be achieved using CMOS components but amplification will be necessary since spin wave detector voltages are not compatible with CMOS switching voltages. The lack of signal (phase) restoration is probably the most fundamental flaw in spin wave computing.

Spin wave majority logic gates as proposed by Khitun[7] have no explicit input/output isolation. Spin waves could, in principle, reflect from the boundary of a logic device and feedback into the input nodes. Clocking the inputs and outputs using CMOS technology may be one (unproven) way of getting around this problem, but implies additional unknown power, delay, and complexity overheads. Another effect not considered in the literature is that all wave phenomena are susceptible to localization[17,24], where multiple interference in the presence of disorder (in some general sense) can halt wave propagation entirely, thereby halting a logical operation. Clocking would not solve the problem of localization. Localization has not been observed in experiments at the micron level, but could pose a serious problem when scaling to submicron dimensions.

There is an even more fundamental problem with spin wave computing outside of the basic requirements for logic. It is difficult to guide spin waves around corners of an interconnect[25]. All previously cited logic device experiments interfere multiple spin waves using straight, line-of-sight, spin waveguides. The difficulty here is that the local magnetization direction is determined by two different factors: the shape anisotropy of a thin planar magnetic interconnect and the global magnetic field needed to remove magnetic domains. When an interconnect curves away from the global magnetic field direction, an inhomogenous demagnetizing field distribution arises in the curve region. The internal demagnetizing field will be inhomogeneous for any non-ellipsoidal-shaped contact geometry[26]. The magnetostatic spin wave dispersion is highly anisotropic, so an inhomogenous field distribution will cause different frequencies of a spin wave packet to slow down or speed up and further distort a spin wave packet beyond that implied by the intrinsic material magnetic damping.

Vogt et al.[25] showed that in the presence of a magnetic field, spin wave propagation halts at the curve of a magnetic waveguide. Vogt et al. were able to solve this problem by pinning the global magnetization direction in a waveguide to an underlying gold contact with an applied current in the absence of an applied magnetic field. This result implies that an underlying pinning layer will be required to route spin wave packets around curves. Such a layer will further complicate fabrication.[25] An alternative method of guiding spin waves around a right angle was shown by Bracher et al[27]. Two spin waves are generated at the ends of a “T” configuration. The output spin wave was shown to have the correct phase relationship for interference and passed at right angles to the input spin waves.

The results of Vogt et al. and Bracher et al. suggest that interference and propagation of multiple spin wave packets in a majority gate is possible but may be very sensitive to the local geometry. Thus, there may be a large variability in each logic operation and/or higher system complexity due to geometrical constraints on the shape and relative alignment of magnetic waveguides. Fan in/fan out may also be effected by the limitations suggested by Vogt et al. and Bracher et al. Finally, large scale architectures may be limited to a very narrow range of topologies.

Spin wave computing, as it relies on the wave nature of spins in a material, shares many of the same characteristics as optical computing, which has been considered as a CMOS alternative for many years[28]. The reader may find this analogy interesting. Many of the advantages and disadvantages of spin wave computing probably have analogs in optical computing. One advantage of using spin waves over light waves is that the size of optical components are limited by the wavelength of light, which is one of the barriers to large scale optical integration. Spin waves exist at wavelengths comparable to lattice spacing, and may have better scaling properties than optical computers.

Benchmarks for spin wave computing

Nikonov and Young’s single device benchmarks. Six Figures of Merit. 1-bit adder using spin waves. Energy for spin wave logic. Energy cost for excitation/detection. Signal propagation and latency for spin waves. 1-bit adder using spin waves. Systems level evaluation of spin wave computing using Rent’s rule.

Nikonov and Young recently compared many different devices under common benchmarks[29,30]. In their work, the energy and speed of single devices was compared for common operations, including inverters and full adders. Spin wave computing was compared in this work with CMOS and other non-CMOS approaches at an abstract level. For instance, possible CMOS-based overheads that may be needed for spin wave logic operations or excitation/detection are not considered. Furthermore, a systems level analysis was not attempted, which was beyond the scope of that work. We attempt to modify Nikonov and Young's analysis along these lines. Only a 1-bit adder is considered due to space limitations. We also attempt a systems level analysis for spin wave computing using Rent's rule to simulate interconnect length distribution. We benchmark spin wave computing in the present work with respect to six figures of merit as proposed by Debenedictis: (1) physical dimension, (2) speed, (3) energy per operation, (4) ratio of on energy to off energy, (5) propagation energy, and (6) propagation velocity.

Size

MOSFET performance can be limited by dimension, as for example determined by short channel effects. We do not discuss these here, except to note that 5 nm is believed to be the smallest feature size compatible with a CMOS approach. Electromigration limits current densities in metal interconnects to ~ 10^6 A/cm^2, which imposes another size limitation. There are also significant manufacturing challenges in lithography at feature sizes less than 100 nm.

Spin wave transmission is significantly changed at submicron sizes[31]. Spin wave propagation implies that the wave vector components of a Gaussian wave packet are well defined. When confined to sizes below typical magnetostatic wavelengths (~1−10 μm as observed in present experiments), Gaussian wave packets will no longer propagate. One way to see this is that as magnetic elements are reduced in size, the frequency becomes independent of wave vector and so the group velocity (dω/dk) becomes zero. Translational invariance is broken, and magnetostatic modes of vibration have nodes at the boundaries of a magnetic feature, which by definition is where spin waves must be detected. Thus, neither phase nor amplitude information can be transmitted across small structures. This limitation is the same as that found in optical computing. Spin waves exist at wavelengths on the order of a lattice constant (<1 nm) due to the exchange interaction so spin wave computing is still possible in principle at very small dimensions. However, experiments have not demonstrated coherent propagation of these exchange spin waves. Intermediate wavelength (exchange/dipolar) spin waves may determine the propagation at feature sizes between 1 nm and 100 nm. Due to a lack of experimental data, we analyze spin wave computing based on the known properties of long wavelength magnetostatic modes, for which logic operations have been demonstrated experimentally. We set the size limits for spin wave computing at the same dimensions as for CMOS, as assumed by Nikonov and Youg. Thus, our assumptions are inconsistent, but our analysis can be updated given new work in this field.

Speed

CMOS speeds are determined by the time needed to charge a capacitor, given by RC, where R is a resistance and C is a capacitor, typically a fraction (1/10) the speed of light or ~10^7 m/s. Covington et al. found that propagating Gaussian spin wave packets travel at ~10^4 m/s in permalloy, which we assume is a typical speed. Higher speeds are possible in YIG, which has a lower damping. Spin waves are thus at least 1000 times slower than CMOS. Latency or delay is a more important parameter than speed, so slower speeds can be compensated for by reducing dimension. Ignoring the previous problems with scaling spin wave transmission, delay for a 45 nm structure is ~10 ps, which is comparable to the corresponding CMOS node[13]. Speed will be a problem for longer interconnects using spin wave buses. This problem is treated later.

Energy per operation

The main contributors to energy for spin wave logic circuit are spin wave generation and signal restoration, which are all driven by CMOS. The energy for a spin wave logic operation is then determined by how many CMOS gates are needed. Spin wave logic circuits can be built out of majority gates, which require fewer components, and thus fewer CMOS gates than any corresponding homogenous CMOS operation. We quote energy values in units of k*T*. CMOS computing presently involves energies of ~10^6 k*T* at a systems level. 1 kT is ~4×10^{-21} J or 0.004 aJ at room temperature.

We assume that a simple CMOS inverter will be used for generation. Generation and detection need not have the same mechanism. Detection schemes may or may not be compatible with a typical CMOS switching voltage (~1 V), and so amplification may be needed. Spin waves are generated by microwave signals in previous logic experiments, but this method is not scalable to the submicron length scale. Generation of microwaves given typical magnetization densities in the appropriate frequency range costs an energy of ~10^9 k*T*[32]. We ignore microwave generation/detection as a feasible mechanism for computing

Another suggested method involves using piezoelectric/magnetoelectric heterostructures[19], also known as magnetoelectric cells. Magnetization dynamics can also be driven by a spin polarized current[33], at current densities of ~10^6 A/cm^2 typical of CMOS. Simulations of a magnetoelectric cell coupled to a spin wave packet lead to typical output voltages of ~1–10 mV, given typical magnetization densities in permalloy. While this switching voltage is lower than in CMOS, such heterostructures have no intrinsic gain and so cannot be used to switch a subsequent spin wave excitation without amplification. We assume that the amplification cost would be large since a ~1 mV signal cannot be used to directly switch a CMOS inverter. This problem is similar to that for superconducting computing using Josephson junctions, which work at ~mV signals. We borrow a design from the literature that uses 12 MOSFETs[34] for hybrid Josephson junction/CMOS memory to amplify a piezoelectric/magnetoelectric heterostructure as an order of magnitude estimate of CMOS overhead. The latency associated with this design is ~550 ps using a 350 nm process, which we assume scales as expected for CMOS. For instance, a 15 nm process is ~23 times smaller than a 350 nm process, leading to a latency of ~24 ps. Simpler designs may be sufficient for amplification of piezoelectric/magnetoelectric heterostructures, but there is a lack of research in this area.

Thus, in the present analysis we assume current-induced magnetization switching as the excitation method given the compatibility with CMOS, which is the only way to achieve other logic requirements. Current-driven magnetization dynamics may not posses full phase coherence[35] and it is not clear how current controls phase, but we ignore these complications due to the lack of basic research in this field. A CMOS inverter (e.g., as described in Nikonov and Young) will be assumed to drive the right amount of current to induce magnetization dynamics. A CMOS inverter may also be considered an upper bound for magnetoelectric generation of magnetization dynamics.

Ratio of on energy to off energy

Spin wave logic consumes no energy when not used. The ratio of on to off energy will entirely be determined by the surrounding CMOS components. This figure of merit is assumed to be the same as in CMOS.

Propagation energy and velocity

In the present analysis, a spin wave is generated by a current-induced spin torque at a current density of ~10^6 A/cm^2. While a spin wave packet has a very small characteristic energy of ~1000 kT [29], there is a threshold current required for excitation of magnetization dynamics. There is no analog of the CV^2 energy for charging a metal interconnect for spin wave propagation. Propagation velocity for a spin wave packet is ~10^4 m/s as discussed previously. For long interconnects this will lead to a large penalty in latency. Thus, there could be a large energy savings but a large penalty in speed using spin wave interconnects. Parallelism is an often quoted solution to beyond-CMOS technologies with large latency. We do not explicitly treat parallelism as a way to overcome the high latency of spin wave computing architectures, but point out that the foundational results of Covington et al.[9] imply that multiple spin wave packets can pass through a single interconnect without signal degradation over distances of at least ~10 μm, potentially enhancing throughput as originally pointed out by Khitun et al.[36]

We only consider a 1-bit adder here as an example. We use the design described by Khitun et al.[13] A 1-bit adder is reproduced in Fig. 1, with points of generation and detection labeled by G and D, respectively. This adder functions in a sequential manner, and so the lengths of the spin wave guide must be carefully controlled. The spin wave packets from A, B, and Cin arrive at Cout simultaneously, and represent a majority gate that calculates the carry out. However, for the sum to be correctly computed in this design, the signal from Cin is assumed to arrive at S before A and B. The S terminal is assumed to be set to 0 before computation. This setting should be accomplished with a clock signal driving an inverter, whose output adjusts the magnetoelectric cell at S. For Cin with a phase of 0 or π (logic 0 or 1), terminal S is 0 and π, respectively. Signals A and B have a relative phase shift of p. Khitun et al. assume that a pinning layer underneath one of the A signal arms will allow a phase shift of π, but still allow signals A and B to arrive simultaneously. If the length of A is adjusted to accomplish a phase shift of π, then signals A and B will no longer arrive at the same time. The 1-bit adder shown in Fig. 41 of Nikonov and Young would not operate correctly, but should have a similar area to the design in Fig. 1.

All terminals labeled G are assumed to be the output of a CMOS inverter. All terminals labeled D are assumed to involve amplification using the 12 MOSFET design of Yoshikawa et al.[34] The delay of this operation is given by the critical path from A to S (T_{SW}), the delay of an inverter for driving A, B, or Cin (T_{INV}), and the delay for amplifying and standardizing the output terminal S (T_{STD}). The total delay is $T_{ADD} = T_{SW} + T_{INV} + T_{STD}$. T_{SW} is about $10F/v_{SW}$, where F is dimension of the half pitch defined by Nikonov and Young and shown in Fig. 41 of that reference, and v_{SW} is the group velocity of a spin wave packet (~10^4 m/s). We used intrinsic CMOS properties for a 15 nm node, as reported by Nikonov and Young in Table 10 in Ref. 29. We set the half pitch F to 15 nm for all estimates. In this way, T_{SW} is 15 ps, T_{INV} is 0.25 ps, and T_{STD} is ~24 ps, leading to a total delay of ~64 ps for a one bit adder, and ~2000 ps for a 32 bit ripple carry adder, approximately one order of magnitude lower than that shown in Fig. 50 of Nikonov and Young.

We similarly sum the energy for required CMOS components. There are three energy terms for generation (E_G) and two terms for detection (E_D). The G terminals use one inverter each, so E = (2x20)x3=120 aJ according to Table 10 of Nikonov and Young. The D terminals each have a cost of 12 MOSFETs, so ED = 2x(12x20) = 480 aJ. The total energy is $E_G + E_D$ = 600 aJ per one bit adder, or 19,200 aJ for a 32 bit adder. There are many implementations of a one bit CMOS adder, but we assume ~20-30 MOSFETs are needed. Thus, in terms of the number of required MOSFETs, the one bit spin wave adder is not clearly better than a CMOS adder, as 12 MOSFETs are needed for each D terminal and 1 for each G terminal in Fig. 1. There is an additional delay of 15 ps in the spin wave 1-bit adder.

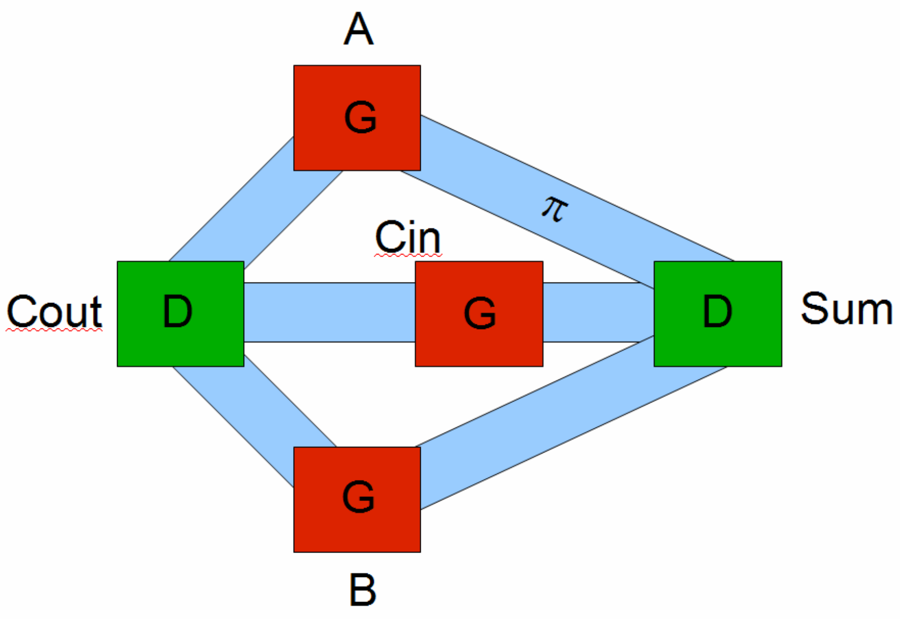


Fig. 1 1-bit adder as proposed by Khitun[13]. The spin wave packet in the top arm of the adder must be phase shifted by π in order for this design to function correctly. Signals A, B, and Cin arrive simultaneously at Cout. Signal Cin must arrive at S before signals A and B. Signals A and B should arrive simultaneously at S for the sum to be computed correctly. We assume the area of this design is similar to that in Nikonov and Young.

As pointed out previously, spin wave packets have no analog of the CV^2 energy in an electrical interconnect. The lack of a charging energy suggests that a great deal of energy can be eliminated relative to the 10^6 kT figure for CMOS. We ignore the 1000x slower propagation speeds of spin waves relative to electrical interconnects, relying on a hypothetical parallel solution for this problem. While there is no charging energy along a long spin wave waveguide, spin waves may only travel a certain distance before falling below the noise floor. The spin wave signal will need to be restored after a certain length. We assume the same detection mechanism as in the discussion surrounding the 1-bit adder, where amplification is needed. To re-propagate a spin wave signal, we assume an additional CMOS inverter is needed.

Long electrical interconnects obey the diffusion equation[37], and the time to charge a wire scales with the square of the interconnect length. Inserting repeaters (such as an inverter consisting of 2 MOSFETs) reduces latency and adds an energy penalty.

We treat the propagation length of spin waves as an adjustable parameter, L_{SW}, which is in units of gate pitch We assume that a repeater for a spin wave interconnect requires an amplification step (e.g. 12 MOSFETs) and an inverter step (e.g. 2 MOSFETs). The energy cost for a long spin wave interconnect is then given by the charging energy for supplying the repeater voltage, which we assume takes place on electrical wires of similar lengths.

Calculations for spin wave interconnect energies are compared with CMOS assuming a 15 nm node leading to an interconnect capacitance per unit length, ε_{IC}, of 126 aF/μm, as given by Nikonov and Young in eq. (26), page 48. Charging energy is $\sim CV_{dd}^2$. Assuming V_{dd} ~1 V, the energy is given by C_{eff} as defined by:

$$\frac{\int_1^{2\sqrt{N}} l\varepsilon_{IC}\, g(l)\, dl}{\int_1^{2\sqrt{N}} g(l)\, dl} \quad (1),$$

where l is wire length in units of gate pitches, N is the number of gates, and $g(l)$ is an interconnect density function[38]. The interconnect density function is based on Rent's rule and depends on the rent exponent, rent prefactor, and fanout. For $N = 10^9$, a fanout of 4, rent prefactor of 4, the energy, in units of kT depends on the Rent exponent as shown in Fig. 2. Higher Rent exponents imply a larger density of long wires. Eq. (1) is probably not the most accurate way to estimate energy based on Rent's rule, however we use this equation to compare CMOS with spin wave architectures in a relative way.

The energy for a spin wave interconnect is assumed to be given by an interconnect capacitance per unit length, ε_{SWIC} of:

$$\theta(L_{SW} - l)\varepsilon_{IC} + \sum_{j=1}^{2\sqrt{N}/L_{SW}} \delta(l - jL_{SW})\varepsilon_{IC} \quad (2),$$

where ε_{IC} is 126 aF/μm, N is the number of gates, and L_{SW} is the effective spin wave propagation length in units of gate pitch. The symbol δ is the Dirac delta function, which is 1 if $l=jL_{SW}$ and 0 otherwise. j is an integer ranging from 1 to $2N^{1/2}/L_{SW}$, the number of repeaters for a given number of gates. θ is the Heaviside function, which is 1 for $l<L_{SW}$ and 0 otherwise. We assume that below a spin propagation length, most of the architecture will be CMOS based given the discussion surrounding the 1-bit adder. This interconnect capacitance function per unit length is then integrated using the same interconnect density function shown before. For the same parameters, the energy of a spin wave architecture is compared to that for CMOS in Fig. 2. We chose several L_{SW} values from 5–100 μm.

In the simulations in Fig. 2, incorporating spin wave interconnects reduces the overall energy relative to CMOS. The energy savings is greater at larger Rent exponents, meaning that when the system uses more long wires, more energy is saved, as expected. For smaller spin propagation lengths, the energy also decreases. There is a peak in energy around a particular Rent exponent for smaller spin wave propagation lengths. This effect is probably an artifact of the crude approximation made in Eq. (2).

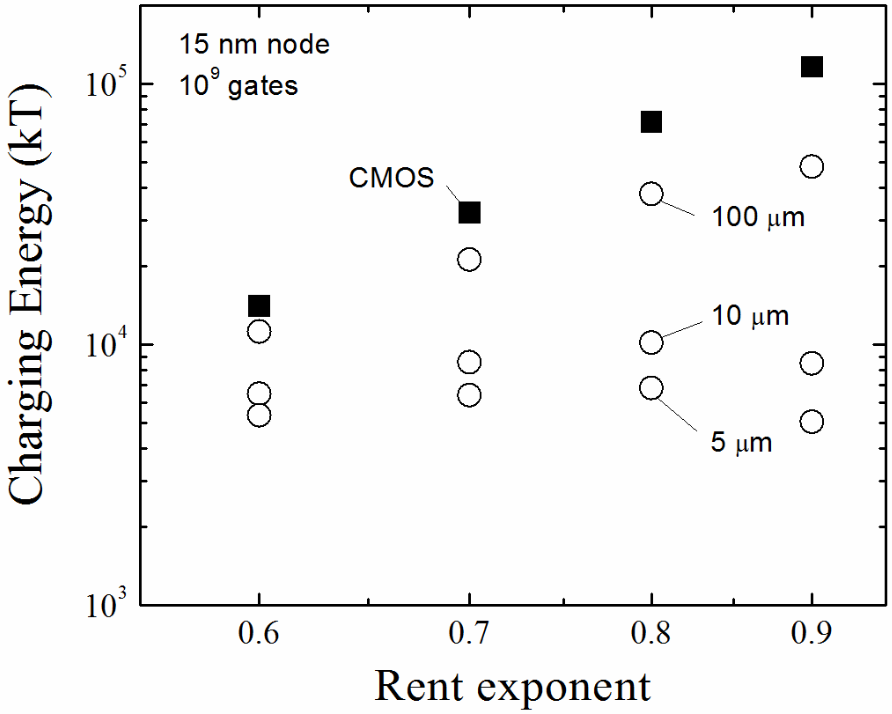


Fig. 2 Charging energy averaged over an interconnect length distribution. CMOS is compared to spin wave architectures. The drop in energy at small propagation lengths is likely an artifact of the crude approximation used for the interconnect density distribution.

In Fig. 3, we plot the spin wave charging energy as a function of spin propagation length. In the top panel of Fig. 3, the total spin wave energy is shown for different Rent exponents. At small spin wave propagation lengths, higher Rent exponents lead to a larger energy drop, while energy increases at larger Rent exponents for larger spin propagation lengths.

The bottom panel in Fig. 3 shows the change in energy of the two terms in Eq. (2) with spin propagation length at a fixed Rent exponent. The θ term (black symbols) is the charging energy associated with CMOS interconnects below the spin propagation length. We assume this because most of the logic functions will require CMOS components, and we found no clear advantage in terms of power or energy relative to a homogenous CMOS design. The δ term in Eq. (2) (red symbols) is meant to account for the charging of CMOS repeaters, assumed to be associated with similar wire lengths as the spin wave interconnect. Thus, at higher spin propagation lengths, there is a higher CMOS logic cost and a smaller spin wave interconnect cost (fewer repeaters for a given number of gates). Eq. (2) implicitly assumes that wire lengths greater than a spin propagation length, there will be no logic functions. The interconnect density distribution may need to be modified to take into account the heterogeneous nature of spin wave computing. Nevertheless, the simulations in Fig. 2 and 3 do indicate that there is a potential cost savings using spin wave interconnects, even accounting for CMOS-based repeaters.

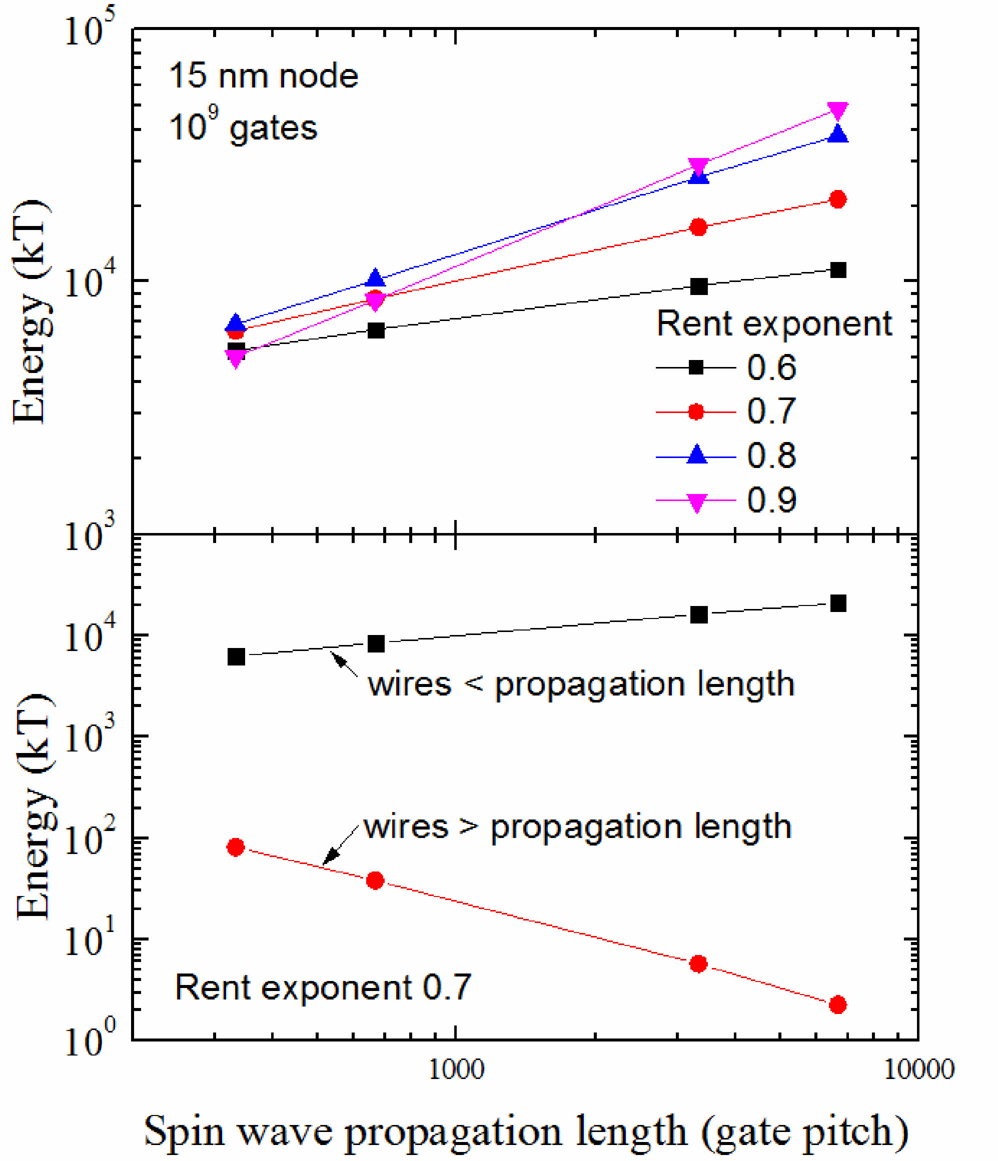


Fig. 3 Spin wave charging energy as a function of spin wave propagation length in units of gate pitch. Top panel: total energy for different Rent exponents. Bottom panel: two terms in Eq. (2) for a fixed Rent exponent of 0.7.

References

1 Mukhopadhyay, S., Raychowdhury, A. & Roy, K. *Switching Energy in CMOS Logic: How far are we from physical limit?*, < https://nanohub.org/resources/1250> (2006).

2 Bandyopadhyay, S. & Cahay, M. Electron spin for classical information processing: a brief survey of spin-based logic devices, gates and circuits. *Nanotechnology* **20**, 412001 (2009).

3 Bandyopadhyay, S. & Cahay, M. Reexamination of some spintronic field-effect device concepts. *Applied Physics Letters* **85**, 1433-1435 (2004).

4 Kadanoff, L. P. & Martin, P. C. Hydrodynamic equations and correlation functions. *Annals of Physics* **24**, 419-469, doi:http://dx.doi.org/10.1016/0003-4916(63)90078-2 (1963).

5 Lévy, L.-P. *Magnetism and superconductivity*. (Springer, 2000).

6 Hertel, R., Wulfhekel, W. & Kirschner, J. Domain-Wall Induced Phase Shifts in Spin Waves. *Physical Review Letters* **93**, 257202 (2004).

7 Khitun, A. & Wang, K. L. Nano scale computational architectures with Spin Wave Bus. *Superlattices and Microstructures* **38**, 184-200, doi:http://dx.doi.org/10.1016/j.spmi.2005.07.001 (2005).

8 Kostylev, M. P., Serga, A. A., Schneider, T., Leven, B. & Hillebrands, B. Spin-wave logical gates. *Applied Physics Letters* **87**, 153501-153503 (2005).

9 Covington, M., Crawford, T. M. & Parker, G. J. Time-Resolved Measurement of Propagating Spin Waves in Ferromagnetic Thin Films. *Physical Review Letters* **89**, 237202 (2002).

10 Schneider, T. *et al.* Realization of spin-wave logic gates. *Applied Physics Letters* **92**, 022505-022503 (2008).

11 Shabadi, P. *et al.* in *Proceedings of the 2010 IEEE/ACM International Symposium on Nanoscale Architectures* 11-16 (IEEE Press, Anaheim, California, 2010).

12 Neusser, S. & Grundler, D. Magnonics: Spin Waves on the Nanoscale. *Advanced Materials* **21**, 2927-2932, doi:10.1002/adma.200900809 (2009).

13 Khitun, A. & Wang, K. L. Non-volatile magnonic logic circuits engineering. *Journal of Applied Physics* **110**, 034306-034311 (2011).

14 MORITZ, C. A., KHASANVIS, S., SHABADI, P. & RAJAPANDIAN, S. N. DESIGN OF SPIN WAVE FUNCTIONS-BASED LOGIC CIRCUITS. *SPIN* **02**, 1240006, doi:doi:10.1142/S2010324712400061 (2012).

15 www.itrs.net

16 Keyes, R. W. Information, computing technology, and quantum computing. *Journal of Physics: Condensed Matter* **18**, S703 (2006).

17 Landauer, R. Is Quantum Mechanics Useful? *Philosophical Transactions of the Royal Society of London. Series A: Physical and Engineering Sciences* **353**, 367-376, doi:10.1098/rsta.1995.0106 (1995).

18 Behin-Aein, B., Datta, D., Salahuddin, S. & Datta, S. Proposal for an all-spin logic device with built-in memory. *Nat Nano* **5**, 266-270, doi:http://www.nature.com/nnano/journal/v5/n4/suppinfo/nnano.2010.31_S1.html (2010).

19 Alzate, J. G. *et al.* in *Nanoscale Architectures (NANOARCH), 2012 IEEE/ACM International Symposium on.* 196-202.

20 Khitun, A. Multi-frequency magnonic logic circuits for parallel data processing. *Journal of Applied Physics* **111**, 054307-054309 (2012).

21 Rovillain, P. *et al.* Electric-field control of spin waves at room temperature in multiferroic BiFeO3. *Nat Mater* **9**, 975-979, doi:http://www.nature.com/nmat/journal/v9/n12/abs/nmat2899.html#supplementary-information (2010).

22 Wang, Z., Sun, Y., Wu, M., Tiberkevich, V. & Slavin, A. Control of Spin Waves in a Thin Film Ferromagnetic Insulator through Interfacial Spin Scattering. *Physical Review Letters* **107**, 146602 (2011).

23 Demidov, V. E. *et al.* Magnetic nano-oscillator driven by pure spin current. *Nat Mater* **11**, 1028-1031, doi:http://www.nature.com/nmat/journal/v11/n12/abs/nmat3459.html#supplementary-information (2012).

24 Anderson, P. W. Absence of Diffusion in Certain Random Lattices. *Physical Review* **109**, 1492-1505 (1958).

25 Vogt, K. *et al.* Spin waves turning a corner. *Applied Physics Letters* **101**, 042410-042413 (2012).

26 Jackson, J. (John Wiley & Sons: New York, 1998).

27 Bracher, T. *et al.* Generation of propagating backward volume spin waves by phase-sensitive mode conversion in two-dimensional microstructures. *Applied Physics Letters* **102**, 132411-132415 (2013).

28 Keyes, R. W. The cloudy crystal ball: Electronic devices for logic. *Philosophical Magazine Part B* **81**, 1315-1330, doi:10.1080/13642810108205810 (2001).

29 Nikonov, D. E. & Young, I. A. in *Electron Devices Meeting (IEDM), 2012 IEEE International.* 25.24.21-25.24.24.

30 Nikonov, D. E. & Young, I. A. Overview of Beyond-CMOS Devices and A Uniform Methodology for Their Benchmarking. *arXiv preprint arXiv:1302.0244* (2013).

31 Demokritov, S. O. *Spin wave confinement*. (Pan Stanford Publishing, 2009).

32 Khitun, A., Nikonov, D. E., Bao, M., Galatsis, K. & Wang, K. L. Feasibility study of logic circuits with a spin wave bus. *Nanotechnology* **18**, 465202 (2007).

33 Kiselev, S. I. *et al.* Microwave oscillations of a nanomagnet driven by a spin-polarized current. *Nature* **425**, 380-383 (2003).

34 Yoshikawa, N. *et al.* Characterization of 4 K CMOS devices and circuits for hybrid Josephson-CMOS systems. *Applied Superconductivity, IEEE Transactions on* **15**, 267-271, doi:10.1109/tasc.2005.849786 (2005).

35 Tsoi, M. *et al.* Generation and detection of phase-coherent current-driven magnons in magnetic multilayers. *Nature* **406**, 46-48 (2000).

36 Khitun, A., Bao, M. & Wang, K. L. Magnonic logic circuits. *Journal of Physics D: Applied Physics* **43**, 264005 (2010).

37 Feynman, R. P., Hey, J. & Allen, R. W. *Feynman lectures on computation*. (Addison-Wesley Longman Publishing Co., Inc., 1998).

38 Davis, J. A., De, V. K. & Meindl, J. D. A stochastic wire-length distribution for gigascale integration (GSI). I. Derivation and validation. *Electron Devices, IEEE Transactions on* **45**, 580-589, doi:10.1109/16.661219 (1998).

Distribution

1	MS0899	Technical Library	9536 (electronic copy)
1	MS0359	D. Chavez, LDRD Office	1911

For CRADA reports add:

1	MS0115	OFA/NFE Agreements	10012

Sandia National Laboratories