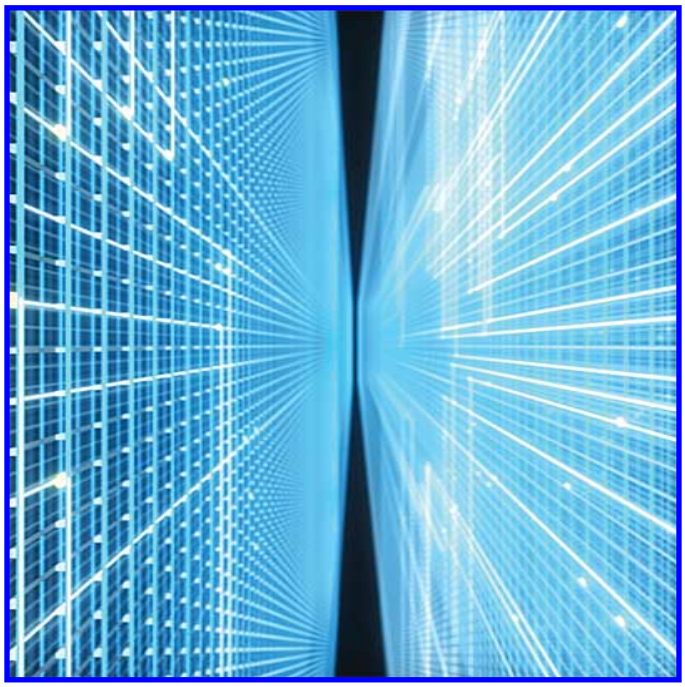
How will memristors change everything?

Wednesday, May 5, 2010 at 7:36AM

Todd Hoff in BigData, distribution, memristor



A non-random sample of my tech friends shows that not many have heard of memristors (though I do suspect vote tampering). I'd read a little about memristors in 2008 when the initial hubbub about the existence of memristors was raised. I, however, immediately filed them into that comforting conceptual bucket of potentially revolutionary technologies I didn't have to worry about because like most wondertech, nothing would ever come of it. Wrong. After watching Finding the Missing Memristor by R. Stanley Williams I've had to change my mind. Memristors have gone from "maybe never" to holy cow this could happen soon and it could change everything.

Let's assume for the sake of dreaming memristors do prove out. How will we design systems when we have access to a new material that is two orders of magnitude more efficient from a power perspective than traditional transistor technologies, contains multiple petabits (1 petabit = 128TB) of persistent storage, and can be reconfigured to be either memory or CPU in a package as small as a sugar cube (in a stacked configuration)?

I don't know, but it's worth thinking about, especially if you want to ride

the wave of the next decade's technological revolution (Bell's Law of Computer Classes). If you are looking to get ahead of the next revolution this just might be it. And as almost always revolutions are based on building a new material based on a fundamental discovery of how the world works. The memristor is such a material and discovery.

I will do a lot of "not pretending" in this article. I won't pretend I actually understand what memristors are or how they will change everything. But since the purpose of this blog is to explore scalability issues, I think it's worth taking a sip or two of the memristor kool-aid and see where it might take us.

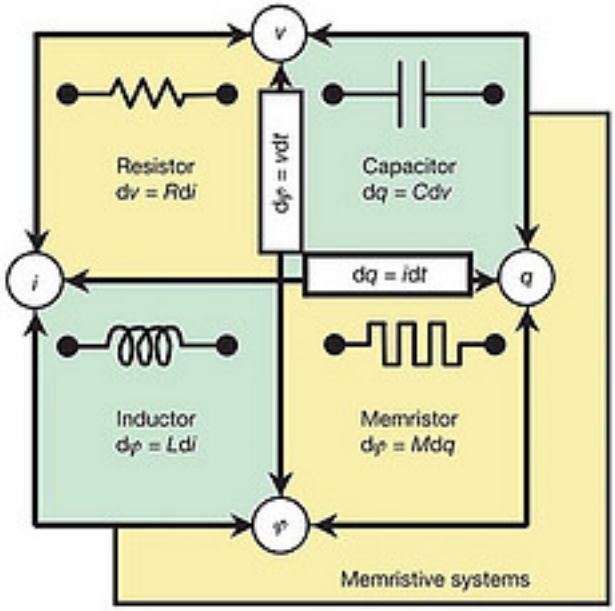
A Memristor is Like a Pipe *(seriously)*

Here's a simple analogy defining a memristor from How We Found the Missing Memristor:

> *A memristor is a pipe that changes diameter with the amount and direction of water that flows through it. If water flows through this pipe in one direction, it expands (becoming less resistive). But send the water in the opposite direction and the pipe shrinks (becoming more resistive). Further, the memristor remembers its diameter when water last went through. Turn off the flow and the diameter of the pipe ”freezes” until the water is turned back on. That freezing property suits memristors brilliantly for computer memory. The ability to indefinitely store resistance values means that a memristor can be used as a nonvolatile memory.*

For a more technical take on how a memristor works watch Finding the Missing Memristor. The video is excellent. It has three sections. The first

part of the video recounts the fascinating story of Leon Chua's discovery of the memristor, the fourth circuit element, in the 1960s. Williams says Chua is to circuit theory what Albert Einstein is to relativity. Chua postulated that the memristor existed based on symmetry. The resistor, capacitor, and inductor existed so there should be a fourth box to fill out the square. I imagine the process something like how missing elements in the periodic table were predicted to exist by looking at gaps in the table. A fifth Cylon was also predicted, but that turned out to be a completely different show.



At this point the memristor was still theoretical. What was needed was a blue fairy to bring this Pinocchio to life. That part in the story is played William and is the subject of the next part of the video which details how Williams and his team at HP Labs not only invented a working memristor, but also took the next step and invented a system architecture for making real field deployable products. The third section is a mind blowing exploration of what applications might be possible using memristors.

It Replaces RAM, Flash and Disk

Memristors are nano devices that remember information permanently, switch in nanoseconds, are super dense, and power efficient. That makes memristors potential replacements for DRAM, flash, and disk.

Williams projects that we'll reach the end of our ability to scale RAM, flash, and disk in the next few years. Fortunately for us the memristor is here to save the day :-) Memristors have the power and speed of the DRAM cell and the (potential) lifetime of a hard disk. Currently the memristor has a lifetime greater than flash, but they are working to extend that. In five years memristors could completely replace DRAM and disk and eventually CDs and DVDs. It is a universal non-volatile memory.

The characteristics of memristors are such that you have to rethink the *whole compute and storage paradigm*. How will it change your designs if you can have large enough amounts of SRAM like storage on the microprocessor such that you don't need DRAM? What if you can put huge amounts of storage near the processor and have enough bandwidth to exchange huge amounts of data? All at low power? Yet, until memristors are many times more durable, they can never replace DRAM and SRAM, they will become a flash only replacement.

It Requires Change

People love progress but they hate change. Memristors require change. They are not a plug compatible technology. You can't just drop a memristor chip or RAM module into an existing system and have it work. It will take a system redesign. The question is when will the pain point in industry be sufficient to cause a migration to a new technology? It's hard to tell as the competition will be fierce, but maybe we'll see memristors first used in a relatively standalone next generation product, like a new smart phone that will leapfrog the iPhone. It will be difficult to compete with Apple, Google, Amazon or other entrenched players while playing

on an even field, so someone will definitely be eager to throw in a little disruptive technology into the mix to see what shakes loose.

It is Big

How much storage are we talking about on a single chip? With an invention they hail as important as the memristor is a new architecture that allows the stacking of multiple crossbar memories on top of each other. This allows multiple petabits of memory (1 petabit = 128TB) to be addressed in one square centimeter of space. To get a feel for how much memory this is consider 1 terabyte is equal to 128 DVDs or 250,000 4 Meg images. We are talking about a lot of power efficient storage in a very little space. Just in time to handle new machine generated data sets that will blow away today's largest data warehouses.

It Computes

So far this is a good market, own the memory hierarchy. But wait, there's more! Memristors are not just stuck in they past, they don't just remember, they can perform logic!

I find this completely strange. We're not used to our memory also acting like a CPU. But it turns out memristors naturally implement something called material implication logic, which can be interconnected to create any logical operation, much the same way NAND gates were used to build early supercomputers because they were easier to build. Williams addresses the functional completeness problem in the video by showing how NAND can be derived from material implication and false. How this works I'm not completely sure, but it is a sufficient basis for executing programs and that's what matters.

So what we have now is a material that can be dynamically configured on

the fly to act as either memory or CPU.

It Flattens the CPU Memory Hierarchy Divide

With memristors you can decide if you want some block to be memory, a switching network, or logic. Williams claims that dynamically changing memristors between memory and logic operations constitutes a **new computing paradigm** *enabling calculations to be performed in the same chips where data is stored, rather than in a specialized central* processing *unit*. Quite a different picture than the Tower of Babel memory hierarchy that exists today.

It Learns

They are also exploring the emulation of brains because the properties of the memristor apparently mimic neurons and can learn without supervision. Synapses and axons are both effectively memristors. I have a feeling the brain is a little more complicated than that, neurons aren't just analog devices, they are essentially little molecular computers, but it's a fascinating direction.

It Talks Using Light

The kryptonite for large pools of storage is moving chunks of data around fast enough so cooperative work can be done. The performance bottleneck is in the interconnects, when data has to flow over wires. To get around the tyranny of the wire HP is working on an optical backplane using photonic interconnects. Every time a bit has to travel more than a 100 microns it will travel as a pulse of light. Over the next 10 years they project memristors + on chip photonic interconnects will improve the overall computational throughput of a computer system by two orders of

magnitude per unit of power, far outpacing what Moore's law and transistors can accomplish.

It Doesn't Exist

Yes, there's a lot of hype about memristors, but there also seems to be a lot of confidence memristors will be real viable products. But for now they don't exist. And we don't know a lot about memristors: unit cost; IO/s per device; performance on sequential/random access operations and read/write loads; reliability; error rates; ease of system integration; persistence lifetime; ease of programming; access times; instructions per clock cycle; power use; density.

There's a lot we don't know yet, but I'm cautiously optimistic, especially if you are looking to what's next on the horizon. HP can already make memristors that beat Flash: lower power, more durable, smaller, and faster. And it's still early days.

What Can We Do With These Things?

People much smarter than myself will figure out how to really use these things, but a few possibilities do come to mind. It's not easy to project the impact of memristors beyond the obvious because memristors challenge our common sense notion of system costs and capabilities. We are used to managing for scarcity, but with memristors we have material abundance. It may take a different way of thinking to fully exploit memristors. An example in how our thinking often has to change with new technologies is the use of sharded counters to reduce write contention when scaling on a BigTable type infrastructure. No doubt memristors will require more profound changes.

In Distributed Computing Economics Jim Gray lays out a model for

thinking about the components of a distributed system, the cost of each component, and how their ratios and relative costs can be used to guide architecture decisions. The components in the model are: networking, computation, database access, and database storage.

Mr. Gray discusses problems in terms of the size of the network input, the size of the network output, the amount of CPU required for a computation, and the required bandwidth. For example, the ideal mobile task, because of restricted bandwidth in a mobile network, is a problem that requires small network inputs, small network outputs, and lots of CPU. The examples given are cryptographic search and Monte Carlo simulations. SETI uses a petabyte of network bandwidth at a cost of about $1 million to access $1 billion of “free” CPU for a ComputeCost: NetworkCost ratio of 10,000:1. This is a good deal because SETI is so compute intensive. Web apps tend to be network or state intensive so won't work as mobile applications. More examples are given of different workloads.

The conclusion is: ***Put the computation near the data.*** *The recurrent theme of this analysis is that on-demand computing is economical only for very CPU-intensive (100,000 instructions per byte or a CPU-day-per-gigabyte of network traffic) applications. Pre-provisioned computing is likely to be more economical for most applications—especially data-intensive ones*.

The problem is without a commercially available device it can't be clear how to characterize system components or assign costs, but can we still make a guess how memristor based devices would fit into Mr Gray's model?

With petabits of persistent storage, colocated CPU and data, configurable numbers of dedicated CPUs, fast on device communication, presumably

fast inter-device communication, and slow WAN communication, we have what appears to be the equivalent of a largish cluster in a sugar cube sized device, maybe a data center will fit in the form factor of a brick. Without the high speed high bandwidth interconnects though these devices will stay relatively specialized because otherwise we won't be able to service high request loads.

A big concern is we have no feel for the latency characteristics of these devices. Many applications are highly latency sensitive, so their latency characteristics will have a big impact.

So it seems like we'll have an all purpose device that can handle small to large data input sizes, small to large output sizes, and small to large computational demands. But we still have the slow WAN divide. Many of the techniques for bridging data centers will still survive, but what will be different is that the size of the problems that can fit on a single system will grow immensely. This is the ultimate scale-up solution to scaling woes.

Designing for memristors may be a bit like the radical shift in our sense of space, time and causality that accompanied the move from classical Newtonian physics to the relativistic quantum perspective of Modern physics. Our common sense notions of classical physics dissolve and are replaced by what? What will systems and algorithms look like when our core assumptions have shifted so radically?

RAM is the New Disk

This is an obvious one. Even before 2006, when Jim Gray declared RAM is the new disk, many latency sensitive applications had moved their databases into RAM. The reasons are obvious: RAM is fast, disk is slow, disk is effectively sequential, RAM is random, RAM is colocated with the CPU, disk is far away.

The largest amount of RAM you can get on Amazon's EC2 is 68.4 GB. Systems capable of 128GB of RAM are now common and inexpensive. For a price 256GB, 512GB and even 1TB RAM systems are available today. And if you have Mariana Trench deep pockets a 128TB monster system can be yours.

So if your database fits in available RAM or can be partitioned across multiple boxes, and there's enough processing power to handle your app, and there's enough bandwidth to move all your data around, then RAM can already be your new disk.

What has kept in-memory databases from taking over the world of database management? Often problems are simply larger than than can fit in RAM, so a scale-out approach is required, and disk is by far the cheapest source of mass storage. And if you optimize for disks relatively fast streaming speeds, as do algorithms like MapReduce, then very large, well performing systems can be built on disk. Not so useful for low latency applications however.

RAM is also still expensive in large quantities, you also require double the number of systems for redundancy, and for a lot of people it's just too different.

Petabits of persistent memory storage attack these weaknesses and make in-memory databases a compelling value proposition. At this point why would you not want to keep your database in-memory? Well, what we don't know about these systems is greater than what we know, what is the real: cost per GB, size, power usage, sequential read and write speed, random read and write speed, persistence life time, IO/s, clock cycles per instruction?

But if these parameters come in as hoped then the face of

application architectures will truly change. RAM, flash, and disk are all unified and all databases are effectively in memory. That would change things up and level the technological playing field. You wouldn't have to be Google to implement truly large systems.

Locality is King

The is somewhat obvious too and revolves around the idea that when networks are expensive moving your computation close to storage is the optimal architecture. Again Jim Gray makes the world clear, from Distributed Computing Economics:

> *Today there is rough price parity between (1) one database access, (2) ten bytes of network traffic, (3) 100,000 instructions, (4) 10 bytes of disk storage, and (5) a megabyte of disk bandwidth. This has implications for how one structures Internet-scale distributed computing: one puts computing as close to the data as possible in order to avoid expensive network traffic.*

That's 2006. As early as 2003, in this ACM Queue article, Mr. Gray predicted:

> *Something that I'm convinced of is that the processors are going to migrate to where the transducers are. Thus, every display will be intelligent; every NIC will be intelligent; and, of course, every disk will be intelligent. I got the "smart disk" religion from you, Dave. You argued that each disk will become intelligent. Today each disk has a 200-megahertz processor and a few megabytes of RAM storage. That's enough to boot most operating systems. Soon they will have an IP interface*

> *and will be running Web servers and databases and file systems. Gradually, all the processors will migrate to the transducers: displays, network interfaces, cameras, disks, and other devices. This will happen over the next decade. It is a radically different architecture.*

Transducers change data in some way. Functions running on processors implement the transducer logic. So data must be fed to processors. Processors are so fast they spend most of their time waiting on RAM, moving data across a network to feed a processor would be a complete waste. So you want your data as close to the processor as possible in order to get the most throughput for the least cost.

We've seen this trend take root over the years. To scale, Google stores data across an uncountable number of disks using a distributed file system. Their storage problem is solved, but how will Google perform calculations on all that data? They could have a compute grid in which a cluster of CPUs run programs that access records over a distributed file system. This approach brings all that data over the network, which is what we want to avoid. So Google invented MapReduce. What MapReduce does is move computations in the form of code to the nodes where the data is stored. So the "transducers" are running as close as they can to the data stored on disk. Only after the data has been filtered and manipulated does the data cross the network for further processing.

While Mr. Gray's vision of transducers everywhere hasn't come true yet, we do see signs of it happening with the advent of flash storage. Flash is so fast it stops making sense to seek data on a flash device, move it to the CPU, run a computation, and write back the result. Why not ship the function down to the flash devices and have the flash run the computation? This is a scenario contemplated by the folks at RethinkDB, who are building a database specifically optimized for SSD.

In-memory databases are of course the ultimate example of moving computation to the data. The CPU has fast access to the RAM. No waiting for disks at all.

With memristors, and let's just say we now are using the memristors only as storage, we now have very large quantities of data directly accessible to the CPU. This effectively cuts out the need for MapReduce at a low granularity. Now let's take advantage of the fact the memristors can be configured as CPUs. This is as close as computation and data can possibly get.

Applying Functions to Data in Parallel

Bradford Cross from Flightcaster spoke recently at a Hadoop Meetup. He talked candidly and in some detail about their experiences trying to use Hadoop, Cascade, and other technologies to implement flight prediction algorithms. At the end of the talk Bradford said one thing that really stuck with me, in summary he said all they are trying to do is *apply functions to data*, it shouldn't be this hard.

When I look at memristor's ability to dynamically configure memory and logic devices on the fly, what I see is the perfect device for applying functions to data in massively parallel configurations. The potential for exploiting parallelism here is awesome.

As an example, let's consider the movie Avatar, a completely digital movie which reportedly requires over 1 petabyte of storage. I don't know if a movie of this type is still chunked up in frames, maybe it's just one time varying equation these days, but let's say it's still a series of a certain number of frames per second. Processing that amount of data would take a good size cluster. Now let's imagine laying each frame in one long array. Interestingly, the preferred data structure for scientific

data is the array. Each of the array cells is effectively parallelized. Now lets place transforms for each frame in series for each array cell. The first transform would operate on the frame and transform it in place, or maybe write it someplace new, then the next transform operates, and so on. You've just transformed the entire petabyte of movie frames in the snap of a finger. All those frames can be processed in parallel because you effectively have a dedicated CPU per frame and the CPU is colocated with the data. In my mind programs become much more geometric, much more structural in nature. More like laying ASICs across a space than logically coupling functions via a message bus.

Applications are currently seek limited. By shifting to a closer faster RAM solution the potential is to make applications CPU limited, but with the ability create CPUs and operate them in parallel we should not be CPU limited either.

I'm the opposite of an algorithms guru, but it seems to me there will be a great need to invent efficient algorithms that take advantage of the special properties of memristors. An algorithm like dynamic programming, which is a popular problem solving approach that solves complex problems by breaking them down into simpler steps, might really benefit from being implemented on memristors.

Is this how memristors really work? I'm not sure, but I haven't seen anything that says that's not how they work and that's how they appear to work to me. And if they work this way the potential is amazing. Let your imagination run a little. Imagine, for example, running an image recognition algorithm on all your images at once. And you know AI has to figure in here somewhere. There's a new trend in Machine Learning called Deep Learning, which moves us one step closer to AI, that uses dense matrix operations operating on hundreds of millions of variables. Sounds like something memristors might make manageable.

Another interesting angle to think about is shown in Comparing genomes to computer operating systems in terms of the topology and evolution of their regulatory control networks. They compare the transcriptional regulatory network of bacterium with call graph of the Linux kernel. Other than being just cool, it showed that human generated code changes more at the bottom of the call graph than at the top. The bacterium changed more at the top than at the bottom. Human thinking works top down, emphasizing reusability, biology works bottom up, emphasizing robustness. Will long lived programs in a materially abundant world shift more towards the biological model?

A Better FPGA (Field Programmable Gate Array)

This is also an easy call, given the ability of memristors to act as CPUs. Currently the processor hierarchy moving from most specialized to most general looks like: ASIC (Application Specific Integrated Circuits), FPGA, microprocessor. For a great talk on how memristors can implement FPGA like devices take a look at the video Hybrid CMOS-Memristor Reconfigurable Logic.

ASICs directly wire logic into hardware. Your 10 Gigabit ethernet interface, for example, will be a specially designed ASIC because a general microprocessor won't be fast enough and it will cost too much at volume. ASICs are completely designed for one purpose and one purpose only, they can't do anything else. Of all options ASICs are the fastest, densest, lowest power, have the highest upfront cost, and the cheapest per unit cost. But if you make just one little mistake your ASIC will have to be respun at considerable time and expense. This is a startup killer. Get your ASICs wrong and you are dead, which is why VCs like funding software, it's safer.

Microprocessors are the most general option. They interpret instructions step by step as dictated by a program. Microprocessors are universal so the cost can be spread out over many units, they are the least dense so they take a lot of space, and they use the most power. But their flexibility makes them the most practical option for system designers, even though you would rather not use them if you didn't have to. As a system designer you want low cost, high integration, and low power usage, but this is beyond the capability of the small guy so microprocessors are the default option in many cases. For example, if you are making a sprinkler system controller that uses wifi for networking, most of your expense will be in the separate components parts, which raises the price out of reach of a mass market. This is the primary reason most consumer products suck. To get them cheap enough so that people will buy and still leave a healthy profit margin you have to design dirt cheap and dirt stupid systems.

FPGAs occupy a middle ground. FPGAs are a collection of gates and that can be selectively connected by programming to build processors and custom hardware. Once programmed computations run fast. Using FPGAs it's possible to build parallel hardware that produces high parallel throughput. FPGAs are flexible and efficient. They are attractive for a lot of applications and are less dense than ASICs because 90% of logic is in the programmable interconnect.

Compared to CMOS, a technology used for constructing microprocessors, memristors can be made far denser, and they can be stacked. Compared to FPGAs memristors can be made two orders of magnitude denser.

Once upon a time I worked on an ATM switch where the key idea was to create blades that had FPGAs which were programmed on the fly to support different interface flavors. It was a really good idea. It would make so many things simpler and save the immense risk and expense of developing a series of ASICs. Unfortunately it didn't really work. FPGAs

were slow to program and the blades were still expensive. It was more cost effective to commit the capex upfront to design and build special purpose ASICs and separate boards around them.

Will memristors shift the capex to opex equation in the same way we've seen the cloud flip the capex of buying machines upfront to the opex of leasing on demand? Will memristors make it possible to make highly integrated devices that have fewer component parts and use lower power? If the technology holds true, it could revolutionize how embedded systems are built.

Low Power Sensors

Building on the previous section which played with the idea that memristors could be used to build highly integrated low power devices without the risk and expense of creating ASICs, if this were true it could finally usher in the era of sensors. Currently sensor devices are too big, too expensive, can't communicate worth a dang, and use too much power. Until we find a new way to build sensors the promise of sensor technology will remain unfulfilled. But watch out if we can unite supercapacitors, energy scavenging, low power devices, and high component integration (do I really need a separate wifi chip?). Sensors will explode.

Hardware Piracy

An interesting implication is that once hardware moves into the digital domain we'll have the same copyright and theft issues that we have with music and other valuable digital data. If I can read your memristor device I can simply recreate your designs on my own device. This is both good and bad in the complicated and confusing way all these issues are. I'd hate to get a DMCA take down notice on my smart grid :-)

Evolution of Control Structures

Ninety-five percent of DNA was once thought to be junk. Now we know what junk DNA are really controls for turning on and off genes, often in response to external environmental events. Genes are really just data whose expression is under the control of other sections of DNA. Doesn't it make you think that if evolution has focussed 95% of DNA on control that it might be important? We don't think about control that much. We focus all our efforts on data. I don't know how, but in my gut I think the memristor could shift the computing emphasis as much to control as data. And no, I don't really know what that means.

The Ambient Cloud

In Building Super Scalable Systems: Blade Runner Meets Autonomic Computing in the Ambient Cloud I developed the concept of the Ambient Cloud as an execution platform running on top of a massively distributed collection of compute resources constructed from wherever they can be found. Without an independent cloud infrastructure developers will have to align themselves with major vendors in order to planet scale. Memristors make the Ambient Cloud even more attractive as the resources available on an internet of memristor devices will be truly staggering. Writing applications on top of that infrastructure will give developers a very low cost structure and the ability to build planet scale applications, all while staying independent and capable of attacking new opportunities. That's truly inspiring.

Related Articles

- 2010 Second Memristor and Memristive Systems Symposium Video Series
- How We Found the Missing Memristor by R. Stanley Williams //

December 2008. *The memristor--the functional equivalent of a synapse--could revolutionize circuit design.*

- The missing memristor found by Dmitri B. Strukov, Gregory S. Snider, Duncan R. Stewart, R. Stanley Williams.
- The Mysterious Memristor by Sally Adee // May 2008. *Researchers at HP have solved the 37-year mystery of the memory resistor, the missing 4th circuit element*
- *Better I/O Through Byte-Addressable, Persistent Memory by a lot of people at Microsoft.*
- A hybrid nanomemristor/transistor logic circuit capable of self-programming by Julien Borghetti, Zhiyong Li, Joseph Straznicky, Xuema Li, Douglas A. A. Ohlberg, Wei Wu, Duncan R. Stewart, and R. Stanley Williams.
- Four-dimensional address topology for circuits with stacked multilayer crossbar arrays by Dmitri B. Strukov1 and R. Stanley Williams
- memristor.org
- Are Cloud Based Memory Architectures the Next Big Thing?
- Building Super Scalable Systems: Blade Runner Meets Autonomic Computing in the Ambient Cloud
- 'Digital Universe' Nears A Zettabyte by Rich Miller.
- The Memristor - Incredible! - a simple introduction to memristors.
- The SyNAPSE Project - uses memristors in their goal of developing a petascale machine that requires no more than a kilowatt of power and two liters of space.
- Identifying Suspicious URLs: An Application of Large-Scale Online Learning. About 31 minutes into this video they say it's not possible to keep a 1 million by 1 million training matrix in memory. Now it will.
