Bayesian Memory: Hardware Implementation Issues

Dan Hammerstrom
Electrical And Computer Engineering
Motivation

What are we doing?

- Looking for a modular building block to allow us to build very large (scalable) networks for Intelligent computing
  - Bayesian Memory, the NAND gate of cognition?
- We want models that map cleanly to the hardware technology
- On the plus side, due to their massive, fine-grained, parallelism and relatively low precision, neural models allow for significant implementation optimizations
- Scaling to very large networks is an important, if not the most important, requirement
- The result is a modest proposal in machine learning based on projected hardware capabilities and constraints, the Bayesian Memory (BM)
A Quick Review Neural Circuits
Why We Are Interested!

- Are fine-grained and massively parallel
- Are built from slow, low-power, unreliable components
- Are tolerant of manufacturing defects
- Are robust in the presence of faulty and failing hardware
- Adapt instead of being programmed, can adapt around a variety of implementation problems
- Are asynchronous
- Consist of networks of sparsely connected and sparsely activated nodes
- Utilize low precision computation
- Are self-repairing
- Are extremely tolerant of noisy environments
Additional Characteristics

- **Self-organizing** – in fact, system design becomes more the provisioning of organizing principles (Prof. Christoph von der Malsburg), than the specification of all operational aspects of the models
  - [www.organic-computing.org](http://www.organic-computing.org)

- Exhibit **Generalization**, and graceful degradation

- **Low power** - the processing power of the brain is roughly $10^{15}$ operations per second which it accomplishes at a power dissipation of about 25 watts
Challenges Facing the Semiconductor Industry
Opportunities For Biologically Inspired Models

- The semiconductor industry has been following Moore’s law for almost 30 years
  - It is not really a physical law, but one of faith
  - The fruits of a hyper-competitive $300 billion global industry

- Then there is Moore’s lesser known 2nd law
  - The 1st law requires exponentially increasing investment

- And what I call Moore’s 3rd law
  - The 1st law results in exponentially increasing design errata
- Intel is now manufacturing in their new, innovative 45 nm process
  - Effective gate lengths of 37 nm (HkMG)

- Transistors of this size are no longer acting like ideal switches

- And there are other problems …
Challenges Facing the Semiconductor Industry

- **Power Density** – the primary constraint on continued performance improvements

- **Performance overkill** - the highest volume segments of the market are no longer performance/clock frequency driven

- **Density overkill** – How do we use all these transistors? And how do we build correct designs?

- **The end of Moore’s law** – Is this really a problem, or will we hit other limits first?
'Tech overshoot' under fire - Chip industry chafes under Moore's Law-era models

EE Times: Latest News, Brian Fuller

BOSTON — The private-equity buyout movement is just one sign of a tectonic shift in the semiconductor industry as businesses that have chased Moore's Law now grapple with "technology overshoot," two Harvard professors said last week.

Clayton Christensen and Woodward Yang told EE Times that designers deliver more technology than consumers can figure out how to use, forcing businesses to adjust from stem to stern—from how they design and manufacture devices to how they service customers and structure their corporations.

"There are some people seeing that just relentlessly pursuing Moore's Law is not giving them the return on investment they had previously seen ..."
Parallelism

- Because of power and interconnect limitations, ever increasing processor performance will need to come more from parallel execution than by raw clock speed.

- However, with the exception of multi-media and perhaps games, there are still few opportunities to leverage parallelism in volume market desktop applications.

- And there is no way to automatically parallelize arbitrary applications.

- Consequently multiple cores will be far more difficult to leverage than faster clock speeds were.
The Complexity Crisis

- In spite of the heroic efforts of computer scientists and engineers around the globe, we are slowly losing the complexity battle.

- And the complexity of these systems is growing exponentially.
  - According to a recent study by the NIST, “Software bugs” cost the U.S. economy an estimated $60B annually, 0.6% of the GNP.

- As Bill Wulf said once,
  - “software is getting slower faster than hardware is getting faster”
  - It’s also getting buggier!
The Design Productivity Gap

- Complexity is a problem for hardware too

- The **Design Productivity Gap** results from the fact that the number of transistors is increasing faster than our ability to design them
  - ASICs passed this point a while back, and FPGAs are there now ("Butts’ Law")

- How do we create a 100% guaranteed correct design of several billion transistors?
“Post-CMOS”

- And if Moore’s law does slow down, can we, by moving to molecular scale “electronics,” buy a little more shrinkage?
  - Is it possible? Is it economical?
  - Will it enable new applications? Or will it be more of the same?
  - What should the research agenda be?
  - And most importantly, What will we do with it?

- Will we hit the complexity or parallel programming walls before Moore’s law runs out?
The Challenges of Nanoelectronics

- Of the various problems facing the semiconductor industry, which ones does nanotechnology solve?
  - The end of Moore’s law
  - Anything else?

- It severely aggravates several, not the least of which is the design complexity problem
  - Having trouble verifying your 1 Billion transistor design?
  - Well we’re going to give you 1 Trillion devices!

- The current design tools and methodologies will not stretch that far

- But, the $64K question is (which is seldom asked and even more seldom answered), what exactly will we use all these devices for?
Die sizes are shrinking, have we run out of ideas on how to use all those transistors?
So, There Are Lots Of Questions Of An Architectural/Systems Nature

- Can we assume that computation, algorithms, and applications will continue more or less as they have? Should we?

- Will most volume applications be able to effectively utilize hundreds of cores on a single chip?

- The effective use of nanoelectronics will require solutions to more than just increased density, we need to consider applications and system solutions
And for any silicon we build commercially, we need to out run the Silicon Steamroller.
A Key Issue: Virtualization

- An important issue in implementing biologically inspired models in silicon is the degree of *virtualization* utilized by the underlying hardware.

- The models create what we call the **c-graph**, the model is then emulated on a hardware structure represented by what we call the **p-graph**.

  - Both graphs represent “data flow”
  - The nodes in the p-graph represent the physical computational entities and the arcs represent the communication paths
  - In a computer cluster the p-graph is the physical network of processors and their interconnect
  - The c-graph can be thought of as a “virtual” network being emulated by a physical network
The **degree of virtualization** is roughly the “degree” of multiplexing of computational and communication resources by a “virtual” model

- Multiplexing implies several virtual (c-graph) nodes sharing a single physical compute element (p-graph), and multiple c-graph arcs mapped into a single p-graph arc

- At one end of the spectrum is a single fast processor (a p-graph of one node) emulating the entire c-graph (virtualization)
- At the other end of the virtualization spectrum, the c-graph is implemented intact on an identical p-graph (directly into hardware with no virtualization)

- A single p-graph node may emulate many c-graph nodes

- Likewise each p-graph connection may contain many “virtual” c-graph connections

- If each c-graph node is a subnetwork (the case for our models) then one can imagine even spreading a c-graph node across many p-graph nodes
Programming flexibility vs. non-flexibility

- Microprocessor
- DSP
- PC cluster
- FPGA
- Digital general purpose NN proc
- Digital specific-purpose VLSI/Nano
- Mixed-signal VLSI/Nano with virtual interconnect

Max time multiplexing vs. max structural parallelism

- Coarsest PN
- Coarse PN
- Fine PN
- Finest PN

Coarser vs. Finer
The Performance – Flexibility Trade-Off

![Graph showing the trade-off between virtualization/flexibility and performance/price. The graph indicates that as virtualization/flexibility increases, performance/price decreases, with a technology barrier in between. General purpose processors are placed on the left side and algorithms wired into silicon on the right side.](image)
Since it promises the highest performance, why not put ourselves at the direct implementation (little or no virtualization) end of the spectrum?

Is that the best performance/price for our models?

As long as price is part of the equation, then the dynamic behavior of the models determines the optimal virtualization.

In computer engineering the 80/20 rule states that 20% of the hardware is active 80% of the time – perhaps we can leverage that with neural systems too.
Another Consideration, Amdahl’s Law

\[ Speedup = \frac{SerialOps + ParallelOps}{SerialOps + \left( \frac{ParallelOps}{N} \right)} \]

- More “great” computer architectures have succumbed to Amdahl’s law, than to any other effect – this I know from experience
- You need the flexibility to solve a large percentage of the total problem
- In the hardware business there is a kind of “Occam’s Razor” where the “simpler” implementations tend to be the best
Neural net chips in the 1990s significantly sped up the neural component, but neural nets rarely operate alone in a real application, so what about the remaining 80% of non-neural operations?

Say you have a chip that speeds up 20% of the problem 1000x versus a chip that speeds up 80% problem 8x,

- \[ Sp1 = \frac{1}{0.8 + 0.2/1000} = 1.25 \]
- \[ Sp2 = \frac{1}{0.2 + 0.8/8} = 3.33 \]

This is where cost comes into the equation, that fast chip may still be worth it if it is inexpensive.
Virtualization And The Great Analog Vs. Digital Debate

- Analog computation has an advantage in low precision computations, low power applications, and impressive computational density (significant functionality in a few transistors)

- Lots of good research on floating gate / single transistor synapses

- Digital can multiplex scarce computational and communication resources more efficiently

- Digital is less dependent on manufacturing process variations and operating temperature
- Digital technology is much less area efficient, especially for certain types of functionality (e.g., leaky integration)

- It is also power intensive

- The representation of time is generally difficult in virtualized designs, often requiring that it be represented and updated explicitly
  - As Carver Mead used to say, “let’s let time be its own representation”

- In spite of digital’s weaknesses, our analyses are generally favorable to using digital computation
Then There’s Connectivity

- Reasonable people can disagree about whether the state and local computation should be analog or digital

- BUT for large association models and traditional CMOS, multiplexed connectivity is the only alternative -- Why?

- Assume a rectangular array of silicon neurons where
  - each neuron receives input from its $N$ nearest neighbors
  - each such connection consists of a single metal line
  - the number of metal layers is much less than $N$
  - The area required by the metal interconnect is of $O(N^3)$
    - Literally several square meters for billions of connections
So even with connectivity patterns that are much sparser than cerebral cortex, the use of dedicated metal lines for each connection quickly overwhelms the available connection resources.

But if we multiplex connections, then don’t we need to store and respond to large numbers of connection addresses? Yup.

And we need some kind of routing based interconnect.

Imagine a 1M node network and assume the weight matrix is 50% full, which is $500 \times 10^9$ connections.

- If they are 1-bit connections, then we have 8 per byte, or 62.5 Gbytes.
- But if they are 32 bit addresses, that is 2 Terabytes.
- Metal vs. RAM for addresses, address RAM vs. metal, it is a trade-off that cannot be avoided

- And even non-multiplexed analog computation requires multiplexed communication

- The Address Event Representation (AER) is used extensively in aVLSI neuromorphic circuits
Front End Vs. Back End

- As we move up the hierarchy, the computation for most higher level models becomes more interconnect driven, leveraging larger amounts of more diffuse data

The “Front End” - **DSP**
- Front End Signal Processing
- Feature Extraction
- Motor Control Subprogram

The “Back End” - **ISP**
- Higher Order Feature Extraction
- Association “Context Info”
- Decision Making
- Motor Control Programs
Front end algorithms tend to involve applying the same computation over a large array of elements, for which analog representation is particularly effective.

Connectivity is generally topographic and typically involves neighborhood connectivity to physically adjacent elements.

The neuromorphic, aVLSI, subthreshold CMOS approach developed by Carver Mead and his group is an excellent example of this kind of front end processing.
- The Annual Telluride Summer Workshop is dedicated to this technology.
Another example is the CNN architecture developed by Chua, Roska et al.

A 2D array of programmable, analog “analogic” processors, with local NEWS interconnect.
However, these technologies do not work as well for back end algorithms for a number of reasons:

- The networks represent more complex “higher order” relationships
- In some models they are much more sparsely activated
- They tend to require longer range connectivity
- There is more topographic invariance
Summary Implementation Constraints

So to map to our hardware we need algorithms that:
- Scale
- Require low precision
- Are modular
- Utilize mostly local connectivity

Things that would be nice include
- Operations that are “associative”
- Sparse representations
- Self-organization
- Bayesian based
Review: Bayesian Memory module (BM)

- **A - Code Book (CB) of vectors/patterns**
  - Clustering or Quantization Mechanism
  - Learn Vectors
  - Store Vectors
  - Compare Vectors
  - Map Input Vectors ➔ Output Vectors

- **B - Probabilistic Framework**
  - Mechanism – learn joint/conditional probabilities CPT, Priors
  - CPT – probabilistic association between parent-child Codebooks
  - Bidirectional Belief Propagation – Pearl’s Algorithm BBP-PA
Basic Operations of BM

- **Training**
  - Learn Code Book, e.g., via clustering
  - Learning CPT and priors (child CB to parent CB probability relation)

- **Hierarchical Structure**
  - Requires layer by layer, systematic training
  - For $n$ layer hierarchical system, requires $n+1$ epochs

- **Inference**
  - Observe causal and diagnostic messages (top-down and bottom-up evidence respectively)
  - Update all the modules using these evidences, according to BBP-PA (probabilistic framework)
Most Likely Computation: A - The Codebook

- The computation of the most-likely codebook vector can be complicated

- Which is a result of the influence of “belief” propagated both from above and from below

- But superficially it looks like a Vector Quantizer with sparse indices as output

<table>
<thead>
<tr>
<th>Input Vector</th>
<th>W</th>
<th>i</th>
<th>Output Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000100111000</td>
<td>0.12</td>
<td>0</td>
<td>10010101000</td>
</tr>
<tr>
<td>0001101001010</td>
<td>0.05</td>
<td>1</td>
<td>01011100110</td>
</tr>
<tr>
<td>0111010101000</td>
<td>0.001</td>
<td>n</td>
<td>10001000100</td>
</tr>
</tbody>
</table>
Part B – Is Just The Basic BBP Equations
- BMs are modular
- They are ideal for hierarchical structuring and region based processing
- BBP-PA assumes such a structure
- Example system tested on MNIST handwritten digit dataset and USPS dataset
Systems Architecture

High-Level Knowledge/ Reward Association and Working Memory (Orbitofrontal & dorsolateral prefrontal cortices)

High-level responses and working memory

Execution commands and Dopamine

Motor outputs

Sensorimotor controller (Basal ganglia/ Ventral tegmental area)

Execution Commands

Sensory & motor predictions (Cb)

Sensory & execution commands

Sensorimotor gateway relay (Truncus)

Visual, Ultra-sound, Infra-red, bump and heading/tilt.

Sensory inputs:

Sensory & motor predictions

Motor and sesensory prediction (Cerebellum)

Proprioceptors: Deadreckoning sensors.

Visual Feature extractors (Visual cortical pathways including V1, V4 and IT)

IT Modulation

IT

Visual from IT

Sensory Significance Association (Amygdala)

Ultra-sound, Heading/Tilt and Bump Sensory, Significance associations
Putting It All Together …

- Our Hypothesis: A useful model of cortex uses locally connected, hierarchical “Bayesian Memories” with distributed information representation to efficiently capture higher-order, highly structured probabilities, and to perform efficient inference over those probability structures.

- Or, put another way, a network of small BMs allows us to approximate arbitrarily complex higher order Bayesian Memory while minimizing operations:
  - \( m \times BM(n), \ m \ll n \), with a large number of lower order BMs, \( n \times BM(m) \), and perform inference more efficiently.
  - Goal: if cost is \( O(x^2) \) for a \( BM(x) \), let \( k=m/n, \ m \ll n \), then \( km^2 \ll n^2 \).
Associative Memory as a BM
The BM can always be implemented directly.

And we have looked at the various possible implementations and associated costs.

But are there simpler approximations to the BM that are even simpler to implement and are better match to the implementation technology?

One such approach is to use a sparsely activated associative memory, such as those of Palm, Willshaw, or many others.
Best-match association finds the “closest” match according to some metric

Vector quantization, as we saw earlier, is a common best-match operation, where the metric is usually Euclidean distance in a metric space

This approximating behavior of **Associative Memories** with the right kinds of representations can lead to useful generalization
Input, output, and weight vectors are assumed to be binary.

To implement Palm auto-association we define the weights as:

\[
W = (w_{ij}) = \left( \sum_{\mu=1}^{M} R(x_{i}^{\mu}, x_{j}^{\mu}) \right)
\]

Or

\[
W = (w_{ij}) = \left( \prod_{\mu=1}^{M} R(x_{i}^{\mu}, x_{j}^{\mu}) \right)
\]

Palm defines a number of different correlation rules \( R \) and sometimes referred to as a "clipped" Hebbian rule.

The output computation in response to an input vector is:

\[
s_{j} = \sum_{i} w_{ij} x_{i}
\]

\[
\bar{y}_{j} = f_{j}(s_{j} - \theta_{j})
\]

\( f() \) is a step function \( f(x) = 1 \) if \( x \geq 0 \).

\( \Theta \) is usually adjusted so that \( k \) nodes are above threshold.
An Associative Memory Creates A Voronoi Tessellation Of the Input Space
A Common Operating Principle: Energy Minimization

- We can define an “energy” based on the state of the network, where a network converges by trying to minimize its energy state.

- Energy cannot be decrease forever; there is a definite minimum.

- Therefore operation must eventually terminate in a final State.

- For asynchronous (basic) behavior, a unique final state is not guaranteed: it could be a local minimum.
Attractor Dynamics

- Attractor state or basin

- Bidirectional excitation caused network to settle into a particular stable state over time: the attractor

- Everything can be understood as an attractor!
Hardware For Associative Processing

What are the hardware requirements for typical associative networks?

- They require minimal precision
  - 1 bit weights, 8-10 bits computation precision (accumulation)
- They are asynchronous
- They are relatively fault tolerant
  - We have found that robustness is not effected with up to 10% connection faults (s@1 or s@0)
- Robust in the presence of noise, at input and during computation
- Throwing dirt cheap, crude neurons at the problem vs. precision
- Limited scalability – with tens of thousand of nodes they work very well, but with hundreds of thousands they do not – not a model of cortex, but maybe of a single column …
- \( k \)-WTA is the compute intensive operation

- It typically requires a sort to find the \( k \) largest accumulated sums

- It is expensive in digital and in analog

- One simplification is to use Lansner’s \( k \) 1-WTA regions – this appears to be more biological at the column level
Palm / Wilshaw Association Uses Sparse Distributed Representations For Efficient Bayesian Inference

- From “Approximate Bayesian Classification in Associative Memory,” Shaojuan Zhu and Dan Hammerstrom:
  - The binary Palm / Willshaw model operates by partitioning the input space into Voronoi regions, with each region centered at a training vector.
  - Under certain convergent conditions, if all the training vectors are equally distributed, then the network implements Bayesian classification with naïve or “equal” priors.
  - Implements the VQ codebook for input vectors.
This result also holds for dynamic learning, where the training vectors have greater resolution, with different lengths representing different prior probabilities - arbitrary vector length allows for weighted Voronoi regions.

If the vector lengths are roughly proportional to the prior probabilities, then the network retrieves an output vector that has approximately the maximum posterior probability.

And the output is sparse and distributed and for hetero-associative networks the output can be a different vector space, in fact, ....

- Using a “BAM – Bidirectional Associative Network” allows for input and output to be in different vector spaces.
So far I’ve only dealt with spatial information

Neural systems represent, and do inference over space and time

However, adding temporal information is very difficult - one way to do this is with spiking models

We believe that spiking based models are not only a more efficient mapping to hardware, but are also an efficient way to extend association through time

- Pulsed Neural Networks, W. Maass and C. M. Bishop, 1999, MIT Press, Cambridge, MA, USA
Spiking Associative Memory

- One approach to adding time is to use more biologically plausible spiking models which inherently operate in the time domain.

- Several groups are now looking at spiking associative memory.

- Though more complex to use, spiking or pulse based models actually present an important opportunity at the hardware level:
  
  - *Computation can proceed by incremental change in response to spikes to a base-line state, where data are represented by the inter-pulse timing.*
Putting It All Together …

Our Hypothesis:

- A reasonable building block for intelligent computing is the Bayesian Memory (BM), which roughly corresponds to a cortical column
  - The BM is approximated by a k-WTA Auto-Associative network
  - The BMs are then connected into 2D sheets with sparse inter-BM connectivity
  - The sheets are then connected hierarchically with feedforward and feedback paths
  - A Pearl-like message passing system is used by the system to perform inference
  - Many smaller BMs are more efficient than fewer, larger BMs

- Our goal is to create a system that efficiently captures higher-order, highly structured probabilities, and performs efficient inference over those probability structures
So Where Are We?

- Bayesian methods appear to be a major part of cognition
- Distributed representations have promise as a way of spreading the computation over a larger number of smaller processes – and it appears to be a technique used by nature
- Distributed representation is a kind of “factorization”
- We have formulated a “Bayesian Memory” (BM) which embodies basic Bayesian principles
- We have proposed auto-associative memory using distributed representations as a promising way to implement each BM
- We have suggested that cortex can be crudely approximated by a 2D network of moderate sized, sparsely connected BMs
- But there is much more to it than can be covered here
Some Interesting Implementation Opportunities
2007 ITRS “Beyond CMOS”

Multiple gate MOSFETs
Channel Replacement Materials
Low Dimensional Materials Channels

“More Moore”

Computing and Data Storage Beyond CMOS

Source: Emerging Research Device Working Group

2008 ITRS Spring Meeting – Koenigswinter, Germany – 3 – 4 April 2008
New Computer Architectures

- The semiconductor industry has been following Moore's law for over 40 years
  - The industry is now designing and manufacturing transistors that are in the 45 nm range, in chips that have over 1 billion transistors

- When will it ever end? The ITRS (roadmap) indicates 8 nm, some believe that it may even be 22 nm

- However, you can do a quick calculation and any model that even approaches biological realism will be too expensive in silicon real-estate even in 16 nm CMOS
An Example Of A Promising “Hybrid”: Nanogrids

- Simplistically: a nanogrid consists of
  - A roughly horizontal group of nanowires
  - A layer of some specialized chemical capable of bistable resistance
  - Another roughly vertical group of nanowires
  - Connections of both groups of nanowires to CMOS metal lines

- Currently researchers are making wires out of silicon and other materials that are ~15 nm in diameter, eventually going to < 10 nm, with lengths up to 10 µm

- These are itsy bitsy wires and they have very high resistance, severely limiting their speed, but oh that density …
- A real nano-grid (approximately 15nm wires and 30nm pitch)
CMOL

- Fortunately, just in the nick of time, CMOL comes riding into town …
- Originally formulated by Prof. Konstantin Likharev at SUNY Stony Brook
- CMOL = Cmos / MOLECular, meant a specific implementation structure
- Now it has come to mean a more general nano-grid system fabricated (generally by nano-imprint lithography) on top of traditional CMOS
- HP Labs on similar nanogrid technology
CMOL
CMOL

\[ 2\beta F_{\text{CMOS}} \]

\[ 2\beta F_{\text{CMOS}} \times 2(r - 1) \]
Some CMOL applications: Long-term prospects

- “CMOL Technology Development Roadmap,” Konstantin K. Likharev and Dmitri B. Strukov

<table>
<thead>
<tr>
<th>Metrics (units)</th>
<th>2016</th>
<th>2019</th>
<th>2022</th>
<th>2025</th>
<th>2028</th>
</tr>
</thead>
<tbody>
<tr>
<td>Half-pitch FCMOS (nm)</td>
<td>30</td>
<td>28</td>
<td>26</td>
<td>24</td>
<td>22</td>
</tr>
<tr>
<td>Half-pitch Fnano (nm)</td>
<td>10</td>
<td>6</td>
<td>4</td>
<td>3.5</td>
<td>3</td>
</tr>
<tr>
<td>CMOS memories (Gbits/cm²)</td>
<td>18</td>
<td>21</td>
<td>25</td>
<td>29</td>
<td>35</td>
</tr>
<tr>
<td>CMOL memories (Gbits/cm²)</td>
<td>100</td>
<td>350</td>
<td>900</td>
<td>1,200</td>
<td>1,700</td>
</tr>
<tr>
<td>CMOS logic (Mgates/cm²)</td>
<td>1.1</td>
<td>1.3</td>
<td>1.5</td>
<td>1.7</td>
<td>2.1</td>
</tr>
</tbody>
</table>
Some Speculation: Nano-CrossBar Implementation

- Synapse footprint: \( \sim 500 \text{ nm}^2 \)
- Synapse density: \( \sim 2 \times 10^{11} \text{ cm}^{-2} \)
- Neural cell density: \( \sim 5 \times 10^7 \text{ cm}^{-2} \)

Note, the lack of a rectifying synapse is a potentially serious problem.
Each Square is a single associative BM module.

Nano-grid implements weight and local / non-local connection indices.

CMOS provides sparse inter-module grid routing structure, I/O, signal amplification.
### Assumed Values

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Range</th>
<th>Typical Value I</th>
<th>Typical Value II</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hypercolumn node size</td>
<td>128 ~ 128 K</td>
<td>1 K</td>
<td>16 K</td>
</tr>
<tr>
<td>Weight matrix size (single-weight-bit)</td>
<td>$2^{14} \sim 2^{34}$ bits</td>
<td>$2^{20}$ bits</td>
<td>$2^{28}$ bits</td>
</tr>
<tr>
<td>Weight matrix size (multi-weight-bit)</td>
<td>$2^{18} \sim 2^{51}$ bits</td>
<td>$2^{30}$ bits</td>
<td>$2^{42}$ bits</td>
</tr>
<tr>
<td>Multi-weight-bits</td>
<td>7 ~ 17 bits</td>
<td>1 bit</td>
<td>8 bits</td>
</tr>
<tr>
<td># Active nodes in hypercolumn</td>
<td>7 ~ 17</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Inner-product result bits (single-weight-bit)</td>
<td>11 ~ 21 bits</td>
<td>14 bits</td>
<td>18 bits</td>
</tr>
</tbody>
</table>
BM (As Associative Memory) Configurations Studied

An exploration of the **Virtualization Spectrum**

The Four Major Configurations studied:

(a) all digital CMOS design
(b) mixed-signal CMOS (2 configurations)
(c) all digital hybrid CMOS/CMOL design
(d) mixed-signal hybrid CMOS/CMOL design

The different computation tasks are partitioned onto different hardware structures.
<table>
<thead>
<tr>
<th>Design</th>
<th># Column Processors</th>
<th>Power (W)</th>
<th>Update Rate (Giga nodes/sec)</th>
<th>memory %</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMOS All Digital</td>
<td>1-bit eDRAM</td>
<td>6.6K</td>
<td>528</td>
<td>3K</td>
</tr>
<tr>
<td>CMOS Mixed-Signal</td>
<td>1-bit eDRAM</td>
<td>19.5K</td>
<td>487</td>
<td>22K</td>
</tr>
<tr>
<td>CMOL All Digital</td>
<td>1-bit CMOL Mm</td>
<td>4M</td>
<td>317</td>
<td>4.5K</td>
</tr>
<tr>
<td>CMOL MS</td>
<td>1-bit CoNets</td>
<td>10M</td>
<td>165</td>
<td>11K</td>
</tr>
</tbody>
</table>
Analysis

- Performance/price comparisons based on a single 858 mm$^2$ (~3 cm on a side) die size

- A Mixed Signal (combined analog/digital) CMOL BM processor with 1-bit weight precision:
  - 10K nodes in each BM associative memory
  - Density: 10K BMs on a single chip (10$^8$ connections per BM), ~ 10$^{12}$ total connections/chip
  - Update rate: 10$^{15}$ operations / sec, i.e., the entire network every 1 ms
Such devices adapt, rather than being programmed, to perform real-time, adaptive Bayesian inference over very complex spatial and temporal knowledge structures.

Implements arrays of self-organizing Bayesian Memories (FABA – Field Adaptable Bayesian memory Arrays)

Conclusion

- A wide range of applications for this type of device in robotics, automobiles, safety, security, the reduction and compression of widely distributed sensor data, power and energy management.

- I believe that success in this area will revolutionize computing – this type of computer has the potential to be the equivalent of the microprocessor for the 21st century!
We do not claim that by creating generic implementations of very large BMs duplicates the brain or, in and of itself, implements cognition, but it should take us a ways to improving the quality of “intelligent computation” for a broad range of applications.

Distributed BMs have characteristics that should make it an extremely useful capability:

- Scaling
- Efficient inference on very large scale models
- Adapts rather than being programmed
- Complex, hierarchical data representations that are formed automatically
Architecture Issues

- Asynchronous logic, time representation
- AER (Address Event Representation) packet grids
- Map – Reduce, communication versus computational efficiency
- Fault tolerance
- Fixed or variable algorithms
- Data representation, resolution, and computation
- Increased activity:
  - Seagate Research “non-linear stochastic processor”
  - MITRE Neuromorphic computing workshop
  - Numenta, Intelisis
  - The DARPA SyNAPSE Program