Architecting for Causal Intelligence at Nanoscale

Santosh Khasanvis, Mingyu Li, Mostafizur Rahman, Ayan K. Biswas, Mohammad Salehi-Fashami, Jayasimha Atulasimha, Supriyo Bandyopadhyay, and Csaba Andras Moritz*
Email: andras@ecs.umass.edu*

***Abstract*–Machine-learning frameworks such as Bayesian networks are widely acknowledged for their capability to reason under uncertainty. However their massive computational requirement, when implemented on conventional computers, hinders their usefulness in critical problem areas. We propose a non von Neumann machine paradigm purposefully architected with physical equivalence across all layers for solving these problems efficiently. It uses emerging magneto-electric nanoscale devices in a novel mixed-signal circuit framework operating directly on probabilities, without segregation between memory and computation. Based on bottom-up simulations, we show four orders of magnitude performance improvement vs. best-of-breed microprocessors with 100 cores, for Bayesian inference involving a million variables. Smaller problem sizes in the order of a 100 variables can be realized at 12mW power consumption and very low area of about a tenth of a mm^2. Our vision is to enable solving complex Bayesian problems in real time, while incorporating intelligence capabilities at smaller scales everywhere.**

Keywords – B.7.1.a Advanced technologies; C.0.a Emerging technologies.

Introduction

Today, all computation occurs on microprocessors based on stored-program von Neumann architecture. However, computers are fast number-crunching machines and, while very efficient for solving problems requiring high precision arithmetic, they are inefficient for supporting intelligence in machines. Many cognitive computing paradigms have emerged such as Bayesian networks for reasoning under uncertainty [1], sparse distributed memory focusing on neural encoding for modeling human associative memory [2], neural networks inspired by neurosynaptic organization of the brain [3][4] etc. These paradigms exhibit high computational complexity and require distributed storage and processing capabilities. Implementing them on conventional von Neumann processors is inefficient in terms of performance, power and area. This inefficiency is due to the use of abstraction at every layer, from Boolean digital logic used to emulate computation to the overall microarchitecture that uses segregation of memory and computation.

In this article, we propose to architect for machine intelligence using a mindset of *physical equivalence*, which we define as a direct mapping of the conceptual computational framework to the physical layer without abstraction, by leveraging emerging nanotechnology. We illustrate our mindset using Bayesian network framework for reasoning as an example, but the ideas presented here may be extended to other cognitive frameworks as well. Bayesian networks are widely successful probabilistic formalisms for machine intelligence that model causal relationships between variables in an application domain. We identify physical equivalence for reasoning with Bayesian networks at all layers to the extent possible, spanning probability representation, a novel nanoscale mixed-domain circuit technology for probability arithmetic without emulation, and a reconfigurable nanoscale Bayesian Cell architecture that can map any Bayesian network structure directly in hardware. We use extensive bottom-up simulations for evaluating our physical equivalence approach, and present a methodology to estimate the benefits for Bayesian reasoning compared to state-of-the-art 100-core microprocessors. Our evaluation shows that for a computational resolution of 0.1 it can yield orders of magnitude improvement in Bayesian inference runtime compared to 100-core microprocessors. This could enable solving complex Bayesian problems involving large number of variables in real time. Bayesian reasoning and learning in smaller networks can be achieved with ultra low power consumption and area using our approach, which can enable incorporating machine intelligence capabilities in embedded systems everywhere.

On a side note, there are recent trends that explore paradigms such as stochastic computing [5][6] and approximate computing [7]. These are primarily motivated by applications (e.g. image processing) where implementation cost benefits such as small size, low power and error-tolerance are more desirable at the expense of less-than-perfect computation (approximate results) and speed. However, these paradigms are conceptually different and do not specifically address the goal of realizing cognitive computing. While we present our nanoscale circuit framework in this article in the context of reasoning with Bayesian networks, the circuit framework may be relevant to these domains as well through further research.

Overview of Reasoning with Bayesian Networks

Bayesian networks use probabilities as the basis of representing uncertainty in knowledge for a given domain, and require probability computations for reasoning and learning. The structure of a Bayesian network is a directed acyclic graph, where every node represents a variable and every edge represents dependency between connected variables. Its parameters are conditional probability tables (CPTs) that quantify the strength of this dependency between variables.

Bayesian networks can be used for expressing the belief (probability of a hypothesis) in the state of a system given some observations on its environment (evidence). Given a parameterized Bayesian network, reasoning is performed through inference operation to calculate beliefs of unobserved variables, triggered by a change in the state of evidence variables. Belief propagation algorithm [1] implements inference in trees and poly-trees using distributed local computations at every node and message propagation. This requires frequent arithmetic on probabilities such as multiplication and addition, and distributed storage (see Supplementary Document Figure S-1 for details). A key requirement for scalable Bayesian hardware is an efficient and parallel implementation of these probability computations.

Many problems can be mapped into this formalism. For example, gene expression networks are being studied in order to understand the genetic basis of diseases [8]. Unfortunately the resulting networks are generally very complex owing to gene-gene and gene-environment interactions. Other applications include image classification [9], medical diagnosis [10], cyber-security [11], etc. Inference and learning operations for these applications result in high computational complexity when implemented on stored-program computers. Additionally, cost and power efficiency aspects make adding reasoning capabilities infeasible in embedded systems.

Architecting for Bayesian Reasoning with Physical Equivalence at Nanoscale

Our objective is to architect an efficient machine for Bayesian reasoning enabled by emerging nanotechnology. Therefore, we identify representations resulting in *physical equivalence* with the probabilistic framework spanning data representation to circuit and architecture layers.

The first element is the data representation. Since Bayesian networks operate on probabilities, we represent data as non-Boolean flat probability vectors tied to the physical layer. We define *n* spatially distributed digits $p_1, p_2, \ldots, p_n$ (Figure 1). Each digit p_i can take any one of *k* values, where *k* is the number of states supported by the physical device (e.g., for devices with 2-states, $k = 2$ and $p_i \in \{0, 1\}$). The value of the encoded probability P is given by:

$$P = \frac{\sum_{i=1}^{n} p_i}{n(k-1)}.$$

This representation yields fault resilience through graceful degradation in case of faults. Using *n* digits with *k* values each gives us a resolution of $1/[n(k\text{-}1)]$ (resolution is the minimum non-zero probability that can be encoded).

Aiming to get the physical implementation close to this representation, we have embraced an unconventional nanoscale device technology and circuit model. We use strain-switched magneto-tunneling junctions (S-MTJs), in addition to transistors, to build the hardware. S-MTJs are attractive due to low switching energy [12] and can support non-volatility (persistence in device state even after voltage is removed) with the same technology. An S-MTJ is a four-terminal device, where a pair of *input* digital voltages changes the resistance between *reference* and *output* terminals (Figure 1A-B). We refer the reader to references [13]-[14] for details on S-MTJ device structure and operation.

Following the mindset of *physical equivalence*, each digit in the probability representation is mapped directly in the physical layer to S-MTJ resistance, and has equivalent digital voltage representation (Figure 1C-D). In this work, we focus on binary S-MTJ devices, but the approach is applicable to other devices that may exhibit more than two states. While S-MTJs are energy

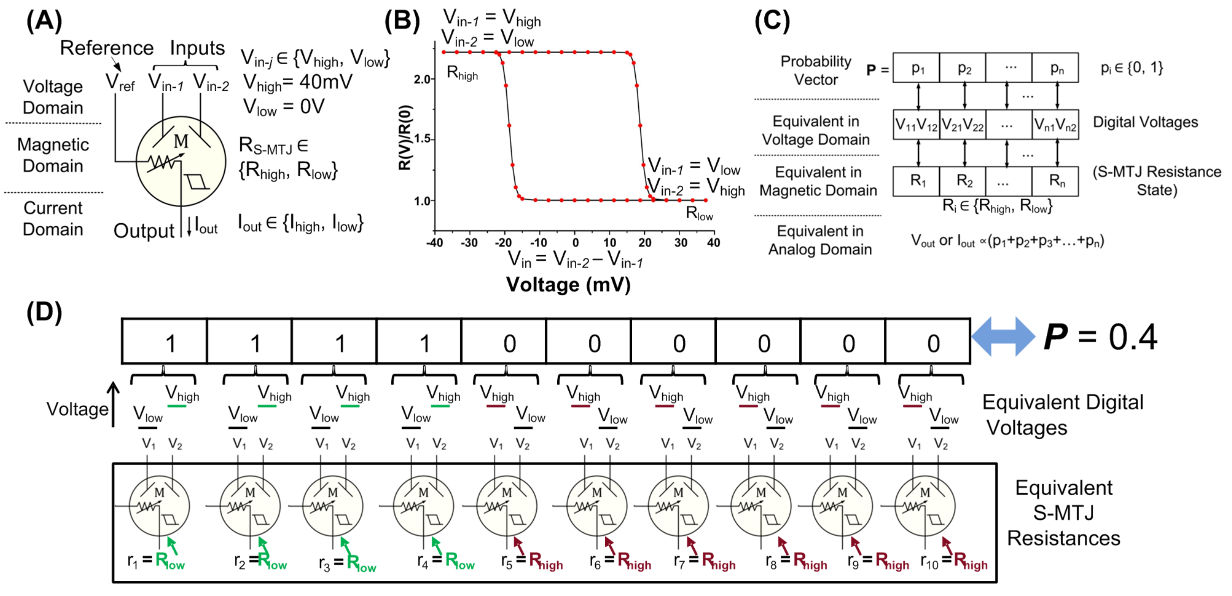
(A)
Reference Inputs
Voltage Domain
Magnetic Domain
Current Domain
Output
$V_{in\text{-}j} \in \{V_{high}, V_{low}\}$
V_{high} = 40mV
V_{low} = 0V
$R_{S\text{-}MTJ} \in \{R_{high}, R_{low}\}$
$I_{out} \in \{I_{high}, I_{low}\}$
(B)
$V_{in\text{-}1} = V_{high}$, $V_{in\text{-}2} = V_{low}$
$V_{in\text{-}1} = V_{low}$, $V_{in\text{-}2} = V_{high}$
R(V)/R(0)
$V_{in} = V_{in\text{-}2} - V_{in\text{-}1}$
Voltage (mV)
(C)
Probability Vector
Equivalent in Voltage Domain
Equivalent in Magnetic Domain
Equivalent in Analog Domain
$p_i \in \{0, 1\}$
Digital Voltages
(S-MTJ Resistance State)
$R_i \in \{R_{high}, R_{low}\}$
V_{out} or $I_{out} \propto (p_1+p_2+p_3+\ldots+p_n)$
(D)
1 1 1 1 0 0 0 0 0 0
P = 0.4
Equivalent Digital Voltages
Equivalent S-MTJ Resistances
$r_1 = R_{low}$ … $r_{10} = R_{high}$

Figure 1| Strain-switched magneto-tunneling junction (S-MTJ) device and probability encoding. (A) Circuit symbol for S-MTJ showing *inputs*, *reference* and *output* terminals. Here *V* represents voltage, *I* represents current and *R* is resistance. (B) Simulated S-MTJ device characteristics [14]. Hysteresis in resistance vs. voltage characteristics indicates non-volatility. (C) Probability data encoding using spatially distributed digits, and physically equivalent representations in voltage/current and resistance domains using two-state S-MTJs. Resolution is determined by the number of digits *n*. (D) Example showing encoding of probability value P = 0.4 with a resolution of 0.1 (10 digits).

efficient, there is a finite probability of a switching error (~2x10^{-6} [13]) due to random thermal fluctuations during switching, and can lead to random bit flips. The choice of using flat digital vectors for probability representation takes this into account. It can alleviate the impact of numerical errors due to faulty switching and allows graceful degradation; a single switching fault results in an error of 1/*n* for binary S-MTJs, where as in weighted representations the error would be dependent on the position of the digit and can be as high as 2^{n-1}. Research efforts on mitigating S-MTJ switching errors are currently ongoing.

Our circuit model is unconventional in that it incorporates S-MTJs and transistors in a mixed-signal magneto-electric circuit style, without segregation between memory and computation (S-MTJs store data and participate in computation). It operates directly on probabilities that have physical representations (Figure 1C). The control lever is in voltage-magnetic domains; by changing the magnetization with voltage we can persistently

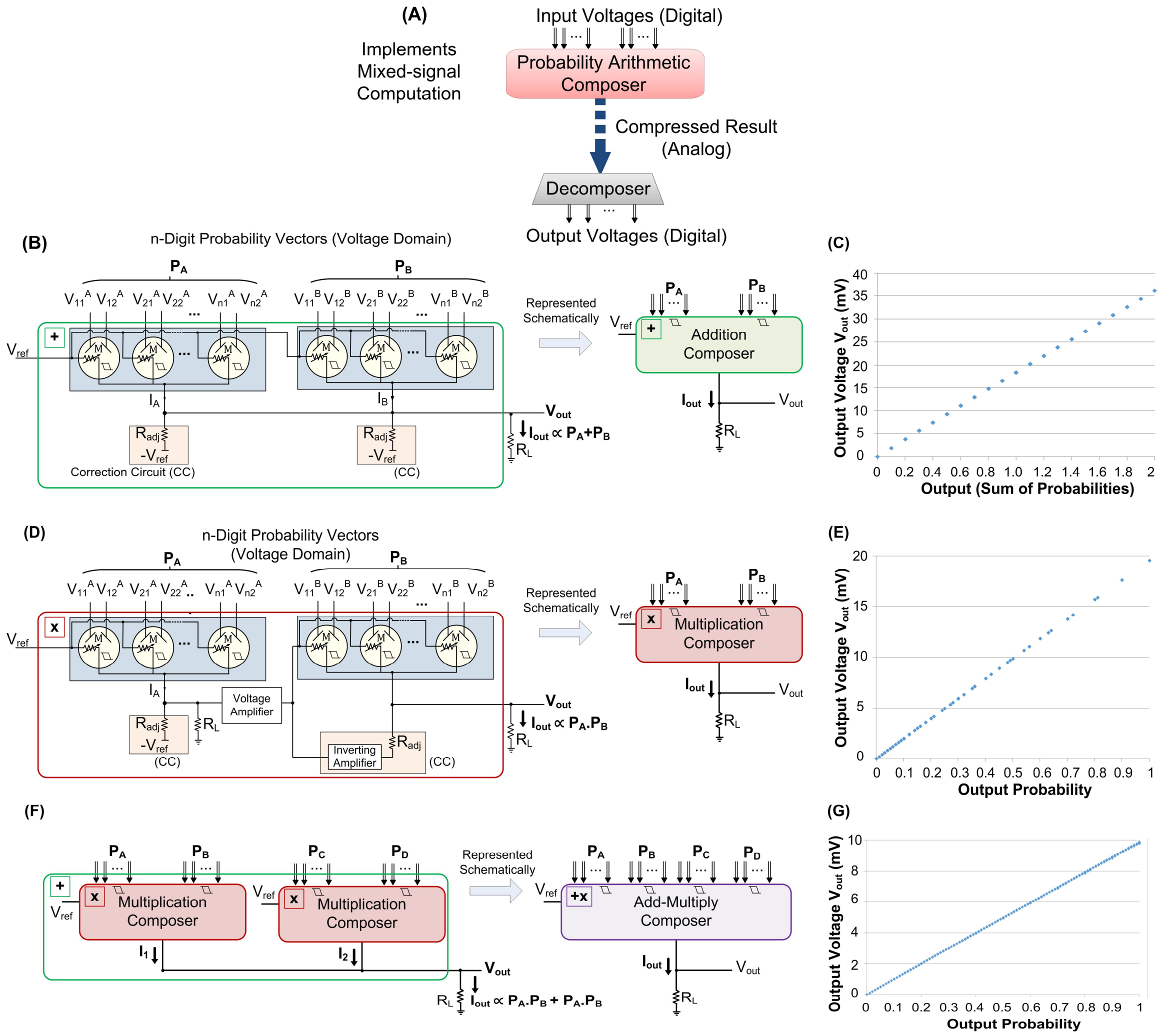

Figure 2| Magneto-electric circuit framework. (A) Probability Arithmetic Composer framework. (B) Elementary Addition Composer, and (C) simulated output characteristics using HSPICE circuit simulator. (D) Elementary Multiplication Composer, and (E) simulated output characteristics using HSPICE. Support circuits such as amplifiers can be implemented with CMOS operational amplifiers. (F) Example of composing Add-Multiply operation with Multiplication Composers arranged in topology of Addition Composer. This is used extensively in Bayesian inference. (G) Simulated output characteristics using HSPICE for Add-Multiply Composer. Here, V_{ref} = 1V, $R_{S\text{-}MTJ\text{-}High}$ = 40MΩ, R_L = 100KΩ, R_{adj} = 4MΩ.

change the S-MTJ resistance, which in turn changes the output analog current/voltage representing resulting probabilities. We call this circuit style the Probability Arithmetic Composer [14] (Figure 2A), since it is operating intrinsically on probabilities and Bayesian computations for reasoning are composed hierarchically using analog arithmetic functions as elementary building blocks.

Elementary Arithmetic Composers for probability Addition and Multiplication (Figure 2B,D) are at the core of the recursive building of Bayesian functions. *Physical equivalence* stems from the use of underlying circuit physics for computation, rather than abstraction-based Boolean logic, and hence these operations are significantly simplified compared to their digital Boolean counterparts. Computations required for Bayesian reasoning can be composed by instantiating Elementary Arithmetic Composers recursively (Figure 2F), leading to *self-similar* fractal-like circuits. Decomposers [14] are used to convert analog output from Composers back to spatial probability representation.

Building on this framework, we define Physically Equivalent Architecture for Reasoning (PEAR) that intrinsically supports Bayesian networks [15] (Figure 3). A departure from von Neumann mindset, it uses a distributed Bayesian Cell architecture where each Bayesian Cell maps a Bayesian variable in hardware for *physical equivalence*. A Bayesian Cell's architectural state includes CPTs, likelihood vectors (**λ**), belief vectors (**BEL**) and prior vectors (**π**). It incorporates Probability Arithmetic Composers (Supplementary Document Figures S-2, S-3, S-4) that locally store these quantities persistently and perform computations on them for inference as per belief propagation algorithm [1], obviating the need for external memory. Bayesian Cells are interconnected through metal routing layers, typically used in integrated circuits, for message propagation. This connectivity is made programmable through reconfigurable Switch Boxes (Supplementary Document Figure S-5), and can support mapping arbitrary graph structures.

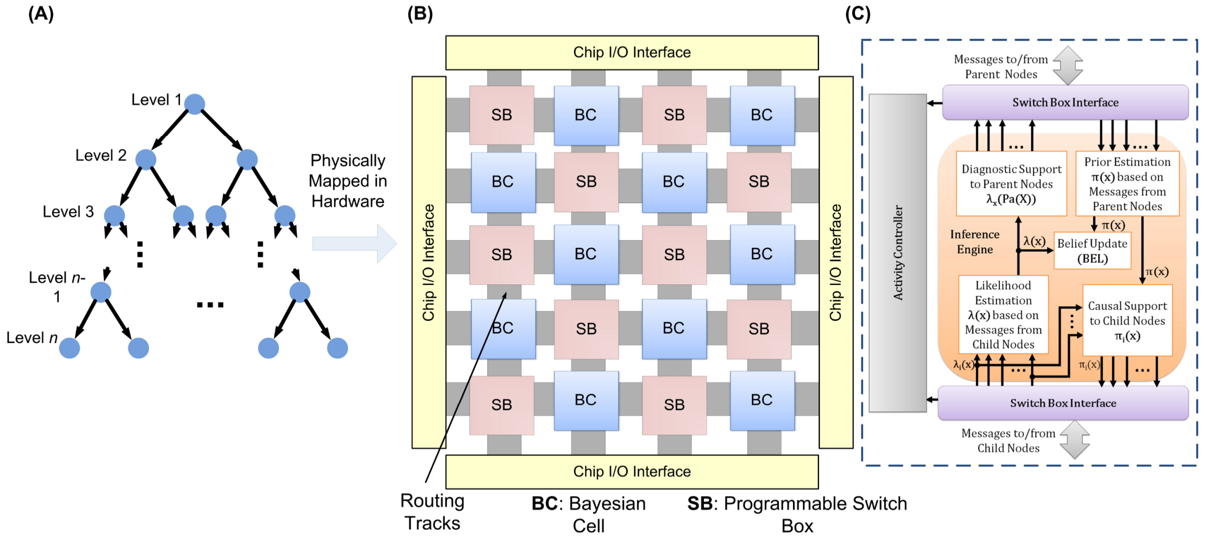
(A)
Level 1
Level 2
Level 3
Level n-1
Level n
Physically Mapped in Hardware
(B)
Chip I/O Interface
SB
BC
Routing Tracks
BC: Bayesian Cell
SB: Programmable Switch Box
(C)
Messages to/from Parent Nodes
Switch Box Interface
Activity Controller
Diagnostic Support to Parent Nodes $\lambda_x(Pa(X))$
Prior Estimation $\pi(x)$ based on Messages from Parent Nodes
Inference Engine
$\lambda(x)$
$\pi(x)$
Belief Update (BEL)
Likelihood Estimation $\lambda(x)$ based on Messages from Child Nodes
Causal Support to Child Nodes $\pi_i(x)$
$\lambda_i(x)$
$\pi_i(x)$
Messages to/from Child Nodes

Figure 3 | Physically Equivalent Architecture for Reasoning (PEAR) and Evaluation. (A) Example binary tree Bayesian Network (BN). (B) PEAR: Reconfigurable Bayesian Cell (BC) framework mapping every node in BN graph to a BC. The directed links in BN are implemented with metal routing layers typically used in integrated circuits, made reconfigurable using Switch Boxes (similar to Field-Programmable Gate Arrays). This allows mapping any BN structure to PEAR. (C) Schematic of computation modules in each BC, implemented with Composers.

An activity controller may be used to switch off Bayesian Cells when idle.

Methodology and Evaluation

We use extensive bottom-up evaluation methodology (Figure 4), to evaluate PEAR in terms of runtime, power and area for Bayesian inference. Using binary tree Bayesian network with each variable supporting up to four states, we scale the number of variables, in the order of 100 to a million, for evaluation. We compare it with implementation on 100-core microprocessors [17][18], which represent best-in-breed von Neumann machines designed to leverage the inherent parallelism in such applications.

A. Bayesian Inference Runtime Modeling on Multicore Processors

Our multi-core processor runtime model is highly optimistic, and while unattainable (underestimates significantly what runtime could be achieved in practice), it can be used as a baseline that allows exploring Bayesian networks with very large problem sizes up to a million nodes. This model assumes that all processors achieve full utilization in every cycle (as if the instruction level parallelism in software would be always available to max out execution to all functional units), all computations occur in one cycle (ignores that computations would often require multiple cycles with glue logic in-between), and cache/memory performance is idealized. The model takes into account the overhead of data communication with memory [16] but ignores resource contention to access DRAM and on-chip network contention. The performance for multicore processors mentioned here is therefore unattainable in practice. Hardware parameters considered (Table 1) for multi-core processors are based on Tilera 100-core processor specifications [17][18].

Bayesian inference using belief propagation algorithm proceeds in an event-driven manner across multiple time-steps. At a given time-step a set of nodes in a binary tree are activated when

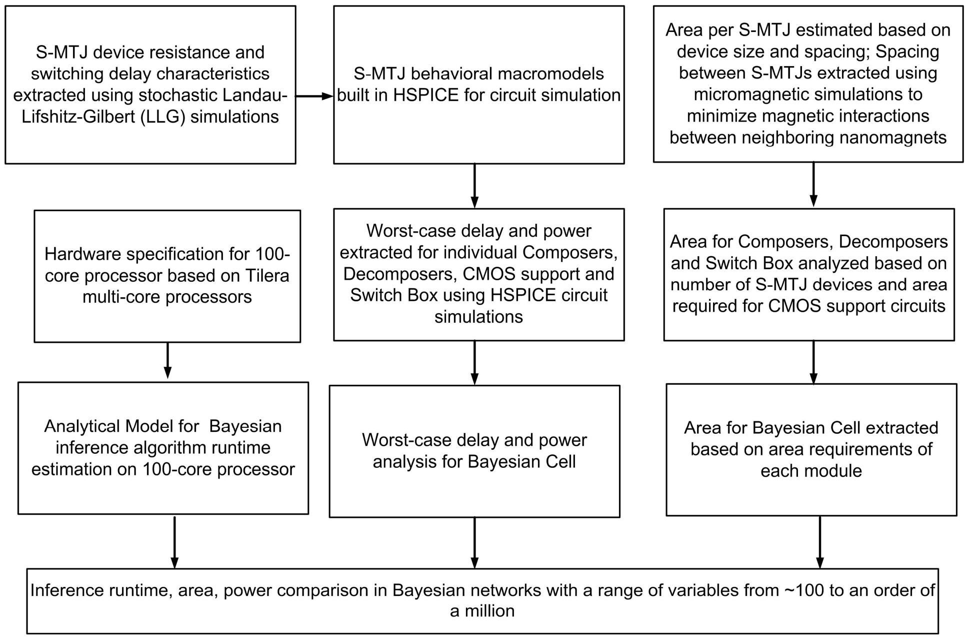
S-MTJ device resistance and switching delay characteristics extracted using stochastic Landau-Lifshitz-Gilbert (LLG) simulations
S-MTJ behavioral macromodels built in HSPICE for circuit simulation
Area per S-MTJ estimated based on device size and spacing; Spacing between S-MTJs extracted using micromagnetic simulations to minimize magnetic interactions between neighboring nanomagnets
Hardware specification for 100-core processor based on Tilera multi-core processors
Worst-case delay and power extracted for individual Composers, Decomposers, CMOS support and Switch Box using HSPICE circuit simulations
Area for Composers, Decomposers and Switch Box analyzed based on number of S-MTJ devices and area required for CMOS support circuits
Analytical Model for Bayesian inference algorithm runtime estimation on 100-core processor
Worst-case delay and power analysis for Bayesian Cell
Area for Bayesian Cell extracted based on area requirements of each module
Inference runtime, area, power comparison in Bayesian networks with a range of variables from ~100 to an order of a million

Figure 4 | Methodology used for evaluation of physically equivalent approach across all layers from device, circuit to architecture, and comparison with Bayesian network inference implementation on 100-core microprocessors.

they receive new messages propagated from evidence nodes. Active nodes execute probability computations (multiplication and addition) to update their belief values, and then propagate new messages to neighboring nodes. These neighboring nodes are marked as active in the succeeding time-step and the operations are repeated. All computations among active nodes at a given time-step can be performed in parallel. The total number of time-steps required by the algorithm is determined by the diameter of the network [1]. Assuming that operations are scheduled such that maximum instruction level parallelism is achieved, the arithmetic execution time for a given time-step *l* is given by,

$$T_{arith}^{l} = \frac{x.N_l}{C.p} \times T_{clock}.$$

Here, N_l is the number of active nodes in time-step *l*, and *x* is the number of operations per node. *C* and *p* are hardware characteristics denoting the number of cores and arithmetic pipelines respectively. We enumerate active nodes at every step of the algorithm, and the total arithmetic execution time is given by adding the runtime for each of these steps.

Table 1. Multi-core Processor Hardware Characteristics Employed [17][18].

Notation Used	Parameter Values
C: No. of cores	100
T_{clock}: Clock period	0.67ns
p: No. of arithmetic pipelines per core	2
S: Size of L2 cache line	64 Bytes
B: DRAM bus-width	72 bits
R: DRAM data rate	136.5 Gbps
k: No. of DRAM ports	4
L: Latency of DRAM access for cache miss	80 clock cycles
n: No. of levels in binary tree Bayesian network	7 to 20
E_{CPT}, E_{BEL}, E_λ, E_π: No. of entries for memory	16 for CPT, 4 for others (supporting up to 4 states per node)
M_{CPT}, M_{BEL}, M_λ, M_π: Parameter memory size	2 Bytes

In order to execute inference operation, each node requires access to corresponding data (probabilities in CPTs, belief vector ***BEL***, likelihood $\boldsymbol{\lambda}$ and prior $\boldsymbol{\pi}$ vectors). Data memory requirement per node, *M* (bytes), is given by,

$$M = E_{CPT} \times M_{CPT} + E_{BEL} \times M_{BEL} + E_\lambda \times M_\lambda + E_\pi \times M_\pi.$$

Here, E_i denotes number of entries and M_i denotes memory size (bytes) per entry for parameter *i*. We determine the data memory requirements for computations occurring in every time-step. This data has to be retrieved from the main memory (DRAM) and the overhead of this communication is estimated as follows. If cache-line size is *S* bytes, DRAM latency is *L* clock cycles, DRAM bus-width is *B* bytes and data rate (minimum of DRAM data rate and on-chip network data rate) is *R* bytes per second, the time to service a cache miss is given by the following equation.

$$T_{miss} = L \times T_{clock} + \frac{(S - B)}{R}$$

If *k* cores can be serviced by the main memory in parallel, the total time to service memory requests for a given time-step is estimated as follows.

$$T_{mem}^{l} = \frac{1}{k} \times (No.of\ cache\ misses) \times T_{miss}$$
$$= \frac{1}{k} \times \frac{N_l \times M}{S} \times T_{miss}$$

Here N_l is the number of active nodes in a given time-step. Thus inference runtime for a binary tree Bayesian network with *n* levels (2*n*-1 time-steps) is given by,

$$T_{Exec} = \sum_{l=1}^{2n-1} (T_{arith}^{l} + T_{mem}^{l}).$$

B. Bayesian Inference Runtime Analysis on Physically Equivalent Architecture for Reasoning (PEAR)

We implement Bayesian networks directly in hardware on PEAR. Here, Bayesian Cells use non-volatile Composers for inference, which support memory-in-computation by storing the required data internally. Thus there is no interfacing with external memory required, which mitigates memory latency overhead. Inference runtime analysis is based on critical-path delay in each Bayesian Cell and switch box, which are extracted using HSPICE circuit simulations (Figure 5 shows an overview of S-MTJ HSPICE behavioral device macromodel). Area required is estimated based on total number of Bayesian Cells and switch boxes for a given Bayesian network size. The maximum number of active nodes at a given time-step determines the worst-case power dissipation, and the power dissipated per node is extracted using HSPICE simulations (Table 2).

If the number of levels is n and execution time at a given time-step l is T_l (critical-path delay of a Bayesian Cell), the inference runtime for $2n$-1 time-steps is given by,

$$T_{PEAR} = (2n - 1) \times T_l + T_{comm}.$$

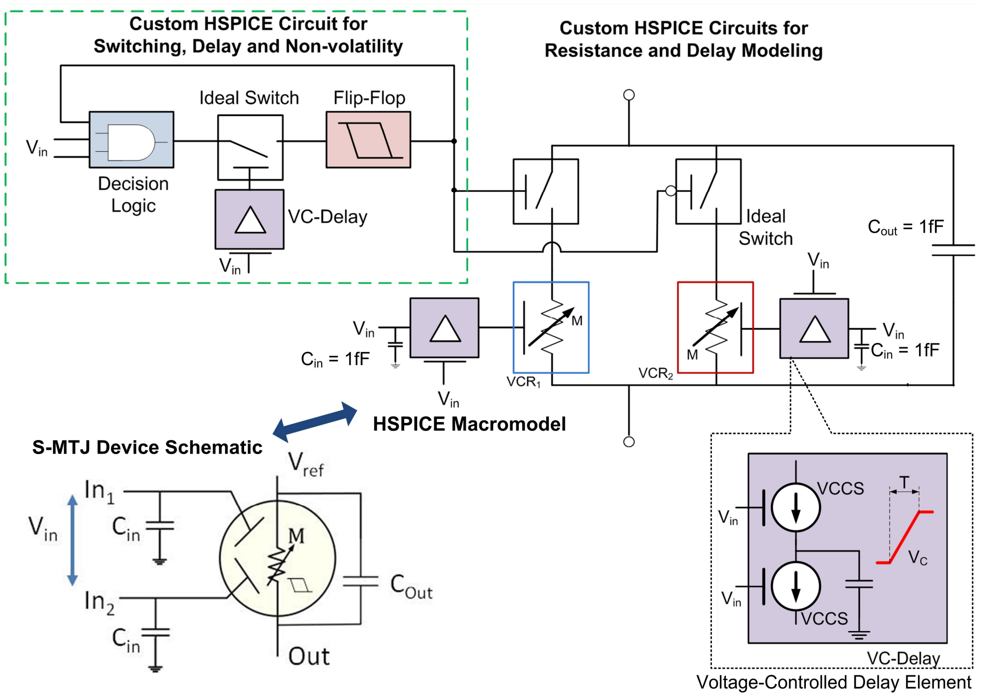

Figure 5 | HSPICE behavioral macromodel overview for non-volatile S-MTJ. Device schematic showing input/output terminals, and parasitic capacitances to be modeled (bottom left). The S-MTJ resistance vs. input voltage characteristics are captured using voltage-controlled resistances (VCR_1, VCR_2). VCR_1 models the low resistance to high resistance switching, and VCR_2 models the high-to-low resistance switching. The appropriate element is selected based on the current state of the S-MTJ, which is stored using a flip-flop [19]. The voltage-controlled delay (VC-Delay) is a custom circuit that models the transient switching delay of the S-MTJ. Finally the Decision Logic block is a behavioral circuit model that accepts as inputs the current applied voltage ($V_{in} = V_{in-2} - V_{in-1}$) as well as the previous state (from flip-flop), and determines if the S-MTJ's current state needs to be switched. Based on the input voltage polarity, as long as the input voltage is above the switching threshold the Decision Logic block causes the state to switch.

Table 2. Evaluation of Composer Circuits for Bayesian Inference (Resolution is 0.1).

Module	**Critical Path Delay (ns)**	**Area (μm^2)**	**Worst-case Power (μW)**
Likelihood Estimation (Multiplication Composers x4)	144	20	4.57
Belief Update (Multiplication Composers x4)	144	20	4.57
Prior Estimation (Add-multiply Composers x4)	137	50	11.24
Diagnostic Support (Add-multiply Composers x4)	137	50	11.24
Prior Support (Multiplication Composers x8)	144	40	9.14
Decomposers (x60)	132.9	240	11.37
CMOS Op-Amps (x176)	100	95.4	89.32
Bayesian Cell	998.2	515.4	141.45
Switch Box	10	398.8	0.85

Here, T_{comm} is the latency of communicating probability messages between nodes, which are near-neighbor voltage communication events. The switch-box delay extracted through HSPICE circuit simulations determines this communication delay, and total number of message propagation events multiplied by this number yields the total communication delay.

Conclusion

Our evaluation (Figure 6) indicates that PEAR can provide about four orders of magnitude runtime speedup over 100-core processors in supporting large Bayesian networks involving about a million variables, for a resolution of 0.1 (studies in image classification [9] and medical diagnosis [10] applications have been reported to have close to optimal accuracy with this resolution). This tremendous performance speedup may enable applications that are computationally infeasible today, particularly in Bayesian network learning which requires repeated inference operations. Furthermore, it is able to support real-time intelligence capabilities at about 12mW power consumption and very low die area cost of $0.1mm^2$ for smaller problem domains (~100 variables). This latter is adequate for many real-world systems such as sensors and automation controllers. Our vision is that every embedded application could incorporate intelligence capability at this problem scale.

Acknowledgements

We acknowledge valuable feedback from coordinating guest-editor Dr. Erik P. DeBenedictis and discussion with Dr. Laura Monroe (Los Alamos National Laboratory). We acknowledge support by National Science Foundation grants CCF-1407906, ECCS-1124714, CCF-1216614 and CCF-1253370.

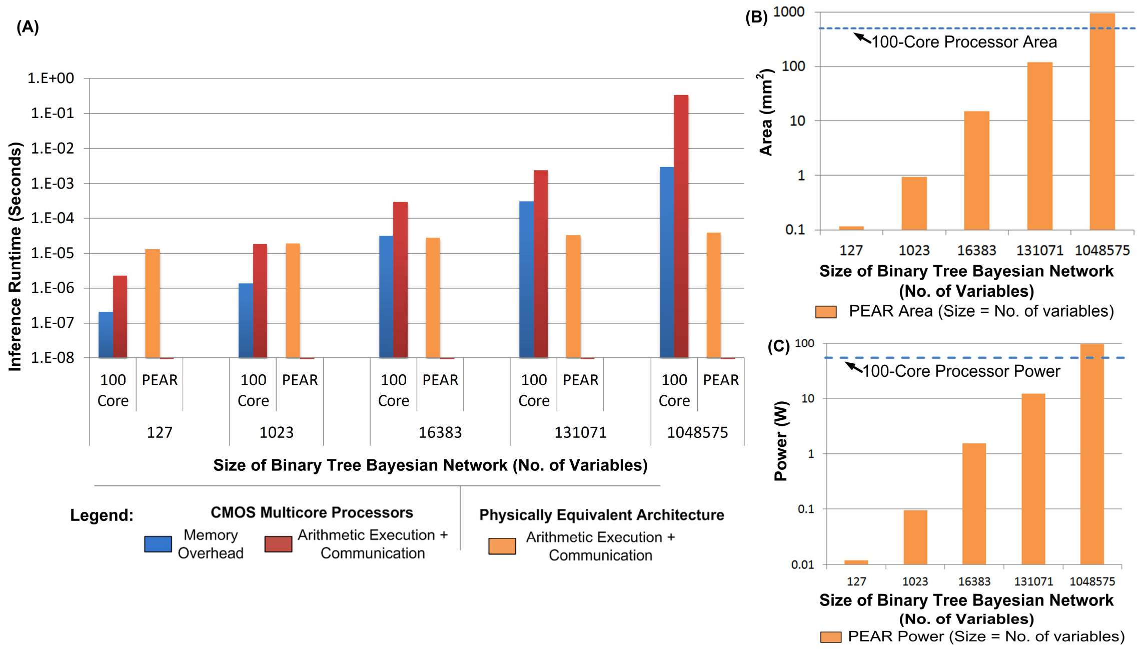

Figure 6 | Evaluation. (A) Estimated runtime for Bayesian inference using a binary tree Bayesian Network. We extract operation time for multi-core processors based on computational and memory requirements, assuming ideal parallelism. Operation time for inference on PEAR is based on worst-case critical-path delay analysis, obtained using HSPICE circuit simulations. Composers support computational resolution of 0.1. (B) Area evaluation. (C) Power evaluation.

References

[1] J. Pearl, *Probabilistic reasoning in intelligent systems: Networks of plausible inference*, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1988.

[2] P. Kanerva, *Sparse distributed memory*, Cambridge, Mass: Bradford/MIT Press, 1988.

[3] W. S. McCulloch, and W. Pitts, "A logical calculus of ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[4] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85-117, 2015.

[5] W.J. Poppelbaump, C. Afuso and J.W. Esch, "Stochastic computing elements and systems," in proceedings of ACM Fall Joint Computer Conference, pp. 635-644, 1967.

[6] A. Alaghi and J.P. Hayes, "Survey of stochastic computing," *ACM Transactions on Embedded Computing Systems*, vol. 12, no. 2s, article 92, pp. 92:1-92:19, 2012.

[7] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in proceedings of 18th IEEE European Test Symposium (ETS), pp.1-6, 2013.

[8] C. Su, A. Andrew, M. R. Karagas, and M. E. Borsuk, "Using Bayesian networks to discover relations between genes, environment, and disease," *BioData Mining*, vol. 6, no. 6, pp. 1–21, 2013.

[9] S. Tschiatschek, and F. Pernkopf, "On Bayesian Network classifiers with reduced precision parameters," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 4, pp.774-785, 2015.

[10] A. Onisko, and M. J. Druzdzela, "Impact of precision of Bayesian network parameters on accuracy of medical diagnostic systems,"*Artificial Intelligence in Medicine*, *Elsevier*, vol. 57, no. 3, pp. 197 – 206, 2013.

[11] Valdes and K. Skinner, "Adaptive, model-based monitoring for cyber attack detection," in *Recent Advances in Intrusion Detection (RAID)*, pp. 80–92, 2000.

[12] J. Atulasimha and S. Bandyopadhyay, "Hybrid spintronics and straintronics: A super energy-efficient computing paradigm based on interacting multiferroic nanomagnets" in *Spintronics in Nanoscale Devices*, Ed. E. R. Hedin, CRC press, pp. 121-154, 2013.

[13] A. K. Biswas, S. Bandyopadhyay and J. Atulasimha, "Energy-efficient magnetoelastic non-volatile memory," *Applied physics Letters*, vol. 104, pp. 232403-1-232403-5, 2014.

[14] S. Khasanvis, M. Li, M. Rahman, M. S.-Fashami, A. K. Biswas, J. Atulasimha, S. Bandyopadhyay, and C. A. Moritz, "Self-similar magneto-electric nanocircuit technology for probabilistic inference engines," *IEEE Transactions on Nanotechnology*, Special Issue on Cognitive Computing with Nanotechnology, in press. [IEEE Early Access Available Online]

[15] S. Khasanvis, M. Li, M. Rahman, M. S.-Fashami, A. K. Biswas, J. Atulasimha, S. Bandyopadhyay, and C. A. Moritz, "Physically equivalent magneto-electric nanoarchitecture for probabilistic reasoning," in proceedings of 11th IEEE/ACM

International Symposium on Nanoscale Architectures, pp. 25-26, 2015.

[16] C. A. Moritz, D. Yeung, and A. Agarwal, “SimpleFit: A framework for analyzing design trade-offs in raw architectures,” *IEEE Transactions on Parallel and Distributed Systems*, vol.12, no.7, pp.730-742, July 2001.

[17] C. Ramey, “TILE-Gx100 manycore processor: Acceleration interfaces and architecture”, Presented at Hot Chips 23, Aug. 2011, Tilera Corporation. Available Online: http://www.hotchips.org/wp-content/uploads/hc_archives/hc23/HC23.18.2-security/HC23.18.220-TILE-GX100-Ramey-Tilera-e.pdf.

[18] J. Mars and R. Hott, “Tilera (RAW) processor,” March 16, 2011. Available Online: http://www.cs.virginia.edu/~skadron/cs8535_s11/Tilera.pdf.

[19] P. Junsangsri, F. Lombardi, and J. Han, "Macromodeling a phase change memory (PCM) cell by HSPICE," in proceedings of IEEE/ACM International Symposium Nanoscale Architectures (NANOARCH), pp.77-84, 2012.

Authors:

Santosh Khasanvis is a Senior Research Scientist at BlueRISC Inc. This work was performed during his PhD. His research interests include unconventional computing architectures, nanoscale computing, and cyber-security. He received his PhD degree in Computer Engineering from University of Massachusetts, Amherst, in 2015. Contact him at santosh@bluerisc.com.

Mingyu Li is a PhD candidate in Electrical and Computer Engineering at University of Massachusetts Amherst, USA. His research interests include 3D integration and nanoscale computing. He received his M.S. degree in Computer Engineering from University of Massachusetts, Amherst, in 2015. Contact him at mingyul@umass.edu.

Mostafizur Rahman is an Assistant Professor of Computer Science and Electrical Engineering at University of Missouri Kansas City, USA. His research interests include 3D fabrics and nanoscale prototyping. He received his PhD degree in Computer Engineering from University of Massachusetts, Amherst, in 2015. Contact him at rahmanmo@umkc.edu.

Ayan K. Biswas is a PhD candidate in Electrical and Computer Engineering at Virginia Commonwealth University, USA. His research interests include hybrid straintronic-spintronic device modeling and simulation. He received his B. Sc. degree in Electrical and Electronic Engineering from Bangladesh University of Engineering and Technology, Dhaka, Bangladesh in 2011. Contact him at biswasak@mymail.vcu.edu.

Mohammad Salehi-Fashami is a Postdoctoral Research Fellow in Physics and Astronomy at University of Delaware, USA. His research interests include magnetostrictive materials and multiferroic nanomagnet-based computing. He received his PhD degree in Mechanical Engineering from Virginia Commonwealth University, USA, in 2014. Contact him at mfashami@udel.edu

Jayasimha Atulasimha is an Associate Professor of Mechanical and Nuclear Engineering, and has a courtesy appointment as Associate Professor of Electrical and Computer Engineering at Virginia Commonwealth University, USA. His research interests include magnetostrictive materials, nanoscale magnetization dynamics, and multiferroic nanomagnet-based computing architectures. He received his PhD degree in Aerospace Engineering from University of Maryland, College Park. Contact him at jatulasimha@vcu.edu.

Supriyo Bandyopadhyay is a Professor of Electrical and Computer Engineering at Virginia Commonwealth University, USA. His research interests include nanoelectronics and spintronics. He received his PhD degree in Electrical Engineering from Purdue University, West Lafayette, Indiana. Contact him at sbandy@vcu.edu.

Csaba Andras Moritz is a Professor of Electrical and Computer Engineering at University of Massachusetts Amherst, and is the founder and Chairman of BlueRISC Inc. His research interests include post-CMOS nanoscale computing, computer architecture, and security. He received his PhD degree in Computer Systems from Royal Institute of Technology, Stockholm, Sweden. Contact him at andras@ecs.umass.edu.