Computer Science and Artificial Intelligence Laboratory
Technical Report

MIT-CSAIL-TR-2008-069 November 24, 2008

Stochastic Digital Circuits for Probabilistic Inference

Vikash K. Mansinghka, Eric M. Jonas, and Joshua B. Tenenbaum

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

Stochastic Digital Circuits for Probabilistic Inference

Vikash Mansinghka*(vkm@mit.edu) Eric Jonas*(jonas@mit.edu)
Josh Tenenbaum (jbt@mit.edu)

November 23, 2008

Abstract

We introduce *combinational stochastic logic*, an abstraction that generalizes deterministic digital circuit design (based on Boolean logic gates) to the probabilistic setting. We show how this logic can be combined with techniques from contemporary digital design to generate stateless and stateful circuits for exact and approximate sampling from a range of probability distributions. We focus on Markov chain Monte Carlo algorithms for Markov random fields, using massively parallel circuits. We implement these circuits on commodity reconfigurable logic and estimate the resulting performance in time, space and price. Using our approach, these simple and general algorithms could be affordably run for thousands of iterations on models with hundreds of thousands of variables in real time.

1 Introduction

Structured stochastic processes play a central role in the design of approximation algorithms for probabilistic inference and nonlinear optimization. Markov chain [1, 2] and sequential [3] Monte Carlo methods are classic examples. However, these widely used algorithms - and approximate Bayesian inference in general - can seem unacceptably inefficient when simulated on current general-purpose computers.

This high apparent cost should not be surprising. Computers are based on deterministic Boolean circuits that simulate propositional deduction according to the Boolean algebra [4, 5], while problems of inference under uncertainty - and many stochastic algorithms for solving these problems - are best described in terms of the probability algebra [6]. To perform probabilistic inference on computers based on all-or-none, deterministic logic circuits, one typically rewrites algorithms in terms of generic real-number arithmetic, which is then approximated by general-purpose Boolean circuits for floating-point arithmetic [7]. This indirection has many disadvantages: it obscures fine-grained parallelism, complicates algorithm analysis, and is needlessly costly in both time and space.

In this paper, we explore an alternative approach which directly models probability in digital hardware. We base our approach on a novel abstraction called *combinational stochastic logic*, which stands in relation to the probability algebra as Boolean gates do to the Boolean algebra. We make three contributions. First, we show how combinational stochastic logic circuits generalize Boolean logic, allowing construction of arbitrary propositional probabilistic models. Second, we combine our stochastic logic gates with ideas from contemporary digital design, showing how to build stochastic finite state machines that implement useful sampling algorithms. In particular, we show how to directly implement MCMC algorithms for arbitrary Markov random fields in hardware in a massively parallel fashion. Finally, we estimate the performance of our approach when implemented on commodity reconfigurable logic, finding substantial improvements in time efficiency, space efficiency and price. We also show that stochastic logic circuits can perform robustly in the presence of a range of transient and persistent faults, suggesting interesting possibilities for distributed computing on unreliable substrates.

*These authors contributed equally to this work

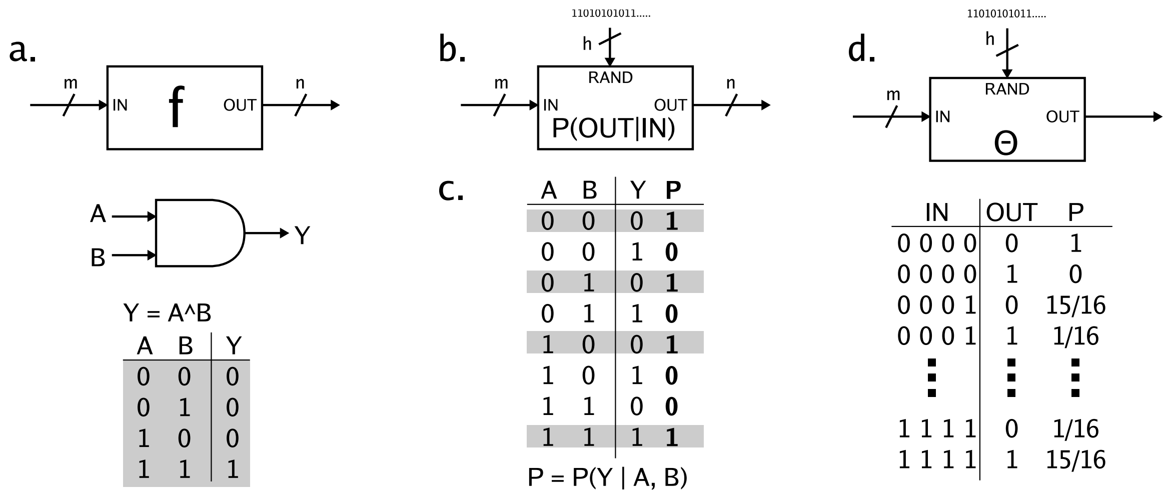

Figure 1: **Combinational stochastic logic.** (a) The combinational Boolean logic abstraction, and one example: the `AND` gate and its associated truth table. (b) The *combinational stochastic logic* abstraction. On each work cycle, samples are drawn on `OUT` from $P(\texttt{OUT}|\texttt{IN})$, consuming h random bits on `RAND` to generate nondeterminism. (c) An `AND` gate can be viewed as a combinational stochastic logic gate that happens to be deterministic. (d) The conditional probability table and schematic for a Θ gate, which flips a coin whose weight was specified on `IN` as a binary number (e.g. for `IN` = `0111`, $P(\texttt{OUT} = 1|\texttt{IN})$ = 7/16). Θ gates can be implemented by a comparator that outputs 1 if `RAND` $\leq$ `IN`.

2 Stochastic Logic Circuits

Our central abstraction, *combinational stochastic logic*, generalizes combinational – or stateless – Boolean circuits to the stochastic setting, recovering Boolean gates and composition laws in the deterministic limit. A Boolean gate has input bit lines and output bit lines, and puts out a Boolean function of its inputs on each work cycle. Each gate is representable by a set of truth tables, one for each output bit; the abstraction and an `AND` gate example are show in in Figure 1a. Figure 1b shows a combinational stochastic logic gate, which adds random bit lines. On each cycle, the gate puts a sample from $P(\texttt{OUT}|\texttt{IN})$ on its output lines, using the random bits – which must each be flips of a fair coin – to provide the nondeterminism. Just as Boolean gates can be represented by families of truth tables, individual stochastic gates can be represented by conditional probability tables (CPTs), where all the probabilities are rational with finite expansions in base 2.

By explicitly representing the bitwidths of values and the entropy requirements per sample from each CPT, we can directly map stochastic gates onto discrete, physical machines for performing computation. Figure 1c shows how to recover deterministic Boolean logic gates by zero-entropy CPTs, using the `AND` gate as an example. Figure 1d shows the conditional probability table and schematic for a unit called Θ, which generates flips of a weighted coin whose weight is specifed on its `IN` lines.

Boolean gates support expressive composition and abstraction laws, shown in Figure 2. By repeatedly composing and abstracting Boolean functions, digital designers build up adders, multipliers, and complex switching networks out of basic Boolean logic operations. Adding stateful elements then leads to flip-flops, accumulators and vector memories, ultimately leading to Turing-universal processors.

We have developed stochastic generalizations of these basic laws. Feeding the output of one gate into the input of another results in samples from the joint distribution of the two elements, allowing construction of samplers for complex distributions from simpler pieces. Furthermore, one can abstract away the details of a complex stochastic circuit, viewing it as a single combinational stochastic gate that simply generates samples from the *marginal* distribution of the original output gate's value given the original input gate's input value. Taken together, these laws support the construction of arbitrarily complex probabilistic (and Boolean) systems out of reusable components. In this paper,

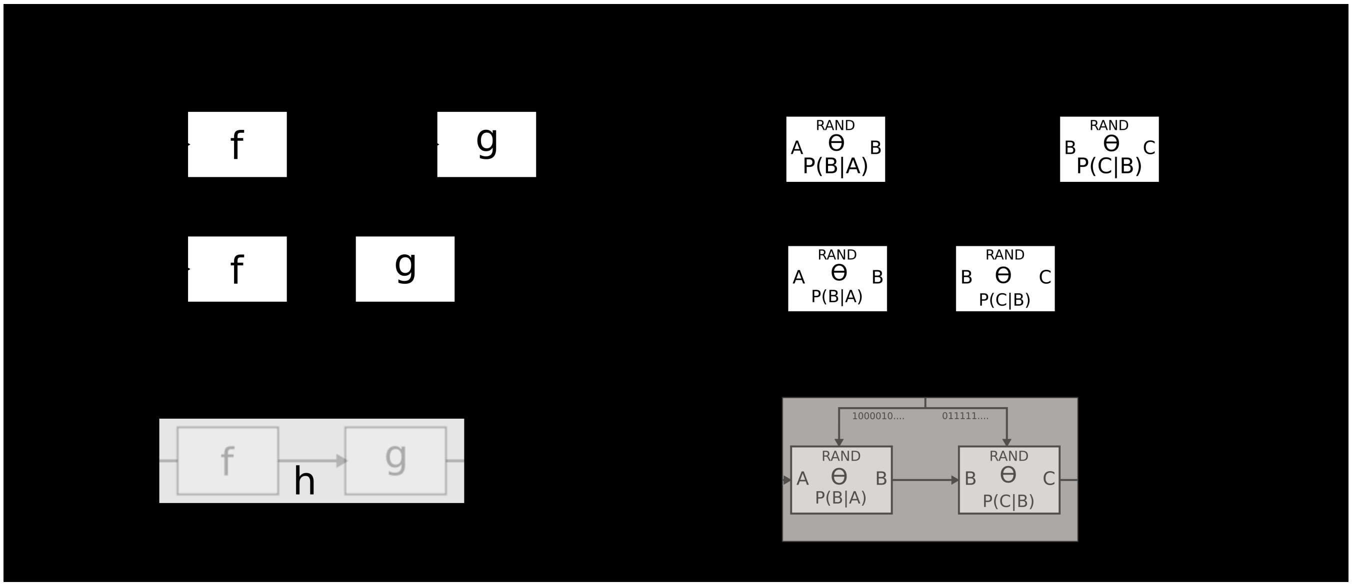

Figure 2: **Composition and abstraction laws.** (a) Boolean gates support expressive composition and abstraction laws. The law of composition states that any two Boolean gates f and g with compatible bitwidths can be composed to produce a new Boolean gate h. The law of abstraction states that h can now be used in designing further Boolean circuits without reference to the components f and g that it is composed of. (b) The analogous laws for combinational stochastic logic: one can sample from joint distributions built out of pieces, or view a complex circuit abstractly as a primitive that samples from a marginal distribution. Note that in this case the input entropy is divided among the two internal elements; the abstraction preserves the notion of a single incoming source of randomness, even though internally the elements receive effectively distinct streams.

we start with stateless Θ gates that flip weighted coins, and build up to circuits for general purpose, MCMC based approximate inference in factor graphs.

2.1 Probabilistic Completeness and Finite Precision

An important aspect of Boolean logic circuits is their logical completeness: given only `AND` and `NOT` gates, one can compute any Boolean function with finite inputs and outputs [5]. Assuming one can implement Θ gates with arbitrarily precise probabilities, it is easy to show the analogous probabilistic completeness properties of our stochastic logic circuits, building on the completeness of the product rule ($P(AB|C) = P(A|BC)P(B|C)$) and the negation form of the sum rule ($P(\bar{A}|C) = 1 - P(A|C)$) shown in [6].

While there are straightforward ways to implement arbitrarily precise Θ gates in either bounded time but unbounded space or bounded space but unbounded time, in practice - as with current digital computers - we will typically approximate our probabilities to some specific, sufficient precision. We expect the actual probability precision needed for exact and approximate sampling algorithms in typical machine learning applications will be low for four reasons. First, fixed parameters in probability models are known only to finite precision, and often only to within an order of magnitude. Second, sampling variance masks small differences in sampling probabilities; approximation error will often be literally in the noise for approximate inference algorithms. Third, most approximate sampling algorithms (e.g. sequential and Markov chain Monte Carlo) depend on ratios of probabilities and weights to obtain the probabilities sampled from during simulation, further pushing dependence up to high order bits. Fourth, expected utility decisions only depend on the low order bits of estimated expectations as the outcomes become increasingly indistinguishable. We recognize that some applications (such as satellite tracking or ab initio physical calcuations) may require substantially higher precision. Obtaining tight analytical bounds on the precision requirements of stochastic

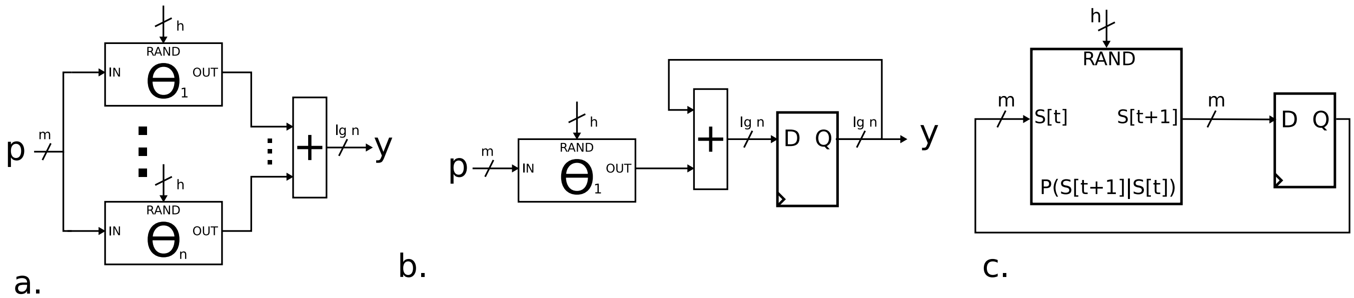

Figure 3: **Example stochastic circuit designs.** (a) and (b) show two circuits for sampling from a binomial distribution on n flips of a coin of weight p, consuming nh total bits of entropy. (a) shows a circuit where the coins are flipped in parallel and then summed, costing $O(n)$ space and $O(log(n))$ time per sample. (b) shows a serial ciruit for the same problem, using $O(log(n))$ space (for the accumulator) and $O(n)$ time. Clocked registers are shown as units with inputs and outputs labeled `D` and `Q`. (c) shows a stochastic finite state machine (or finite-state Markov chain), with a state register connected to a combinational state transition block.

circuits for exact and especially approximate sampling is an important open challenge, recovering the classic Boolean circuit minimization problem in the deterministic limit.

2.2 Stochastic Circuit Design Patterns

We now consider some recurring patterns in probability and stochastic processes, directly translating them to useful designs for stochastic circuits. Our first example is a binomial distribution; we use the conditional independencies in the process to show time and space tradeoffs. For example, Figure 3a shows a circuit for sampling from a binomial distribution on n coin flips using $O(log(n))$ time and $O(n)$ space by both sampling and adding in parallel (via a logarithmic adder tree implementation of +). Figure 3b shows another circuit for the same problem using $O(log(n))$ space and $O(n)$ time, operating by serial accumulation.

Many more interesting circuits become possible when we combine stateless circuits with standard tools from contemporary digital design for maintaing state. For example, we can implement rejection sampling by combining a purely combinational proposal sampler circuit with a Boolean predicate as part of a state machine that loops until the predicate is satisfied. This circuit uses bounded space but possibly unbounded (and in general exponential) time. To implement MCMC, an approximate sampling method, we can combine a combinational circuit that samples from the MCMC transition kernel with a register that stores the current state of the chain. Figure 3c shows the circuit structure of a generic stochastic finite state machine (FSM) applicable to these sampling algorithms. The FSM could be implemented by storing the current state in a clocked register, with a stateless block consuming random bits to sample one next state via a stochastic transition model on each cycle.

2.3 Approximate Inference using MCMC

We focus on tempered Gibbs sampling algorithms for Markov random fields (MRFs) because they are simple and general but usually considered inefficient. To implement a Gibbs MCMC kernel for a given variable, we must score each possible setting given its neighbors under the joint density of the MRF, temper those scores, compute the (log) normalizing constant, normalize the energies, convert them to probabilities, and generate a sample. This pipeline is shown in Figure 4a, and can be implemented in linear time in the size of the variable by standard techniques combined with a simple stochastic accumulator for sampling. However, when the CPT for the variable has sufficiently small size, we can do better: by precomputing CPTs and using the design in 4b, we can obtain samples in constant time. This will be tractable whenever the energy precision requirements are not too high and the degree of the MRF is not too large.

We can then combine Gibbs kernels into massively parallel Gibbs samplers by exploiting conditional independencies in the MRF. Specifically, given a coloring of the MRF (an assignment of colors to nodes so no two adjacent nodes have the same color), all nodes of each color are conditionally independent of each other given all other colors, and thus can be sampled in parallel. This was first observed in [2] for square-lattice MRFs. Figures 4c and 4d show two example colorings. The degree of parallelism depends inversely on the number of colors, and the communication cost between Gibbs units is determined by the total bits crossing coloring boundaries in the MRF. Figure 4e shows a massively parallel circuit built out of Gibbs units exploiting a graph coloring, clocked in a distributed fashion to implement the two-phase structure of a parallel cycle of single site Gibbs kernels. For very large models this pattern can be tiled arbitrarily, preserving constant time Gibbs scans independent of lattice size at linear space cost in lattice area.

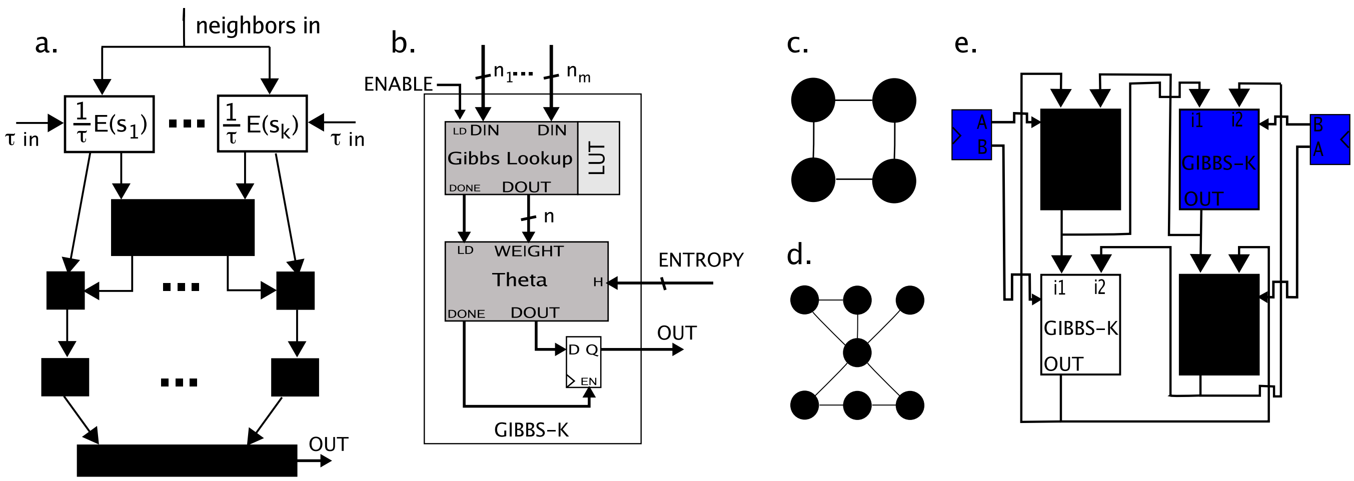

Figure 4: **Designs for Gibbs samplers.** (a) shows a schematic Gibbs pipline, outlining the operations needed to numerically sample a single arbitrary-size variable. (b) shows a condensed implementation of a Gibbsable binary variable unit, which internally stores both a current state setting and a precomputed CPT lookup table (LUT), and runs in 3 clock cycles. (c) and (d) show colored MRFs, where all variables of each color can be sampled in parallel. (e) shows a distributed circuit that implements Gibbs sampling on the MRF in (c) using the Gibbs units from (b).

2.4 Implementation via commodity FPGAs

To estimate the performance of stochastic circuit designs on current physical substrates for computation and compare to widely available general purpose processors, we implemented our stochastic circuits on Xilinx Spartan 3 family Field Programmable Gate Arrays (FPGAs). FPGAs provide digitial logic elements that can be reprogrammed arbitrarily to directly model any digital circuit. They are widely used in consumer electronics and signal processing because they offer lower development cost compared to traditional application specific integrated circuits (ASICs).

Stochastic circuits nominally require large quantities of truly random bits. However, almost all Monte Carlo simulations use high-quality pseudorandom numbers, which can be produced by well-known methods. For our FPGA implementation, we use the 128-bit XOR-SHIFT pRNG from Marsaglia [8], which has a period of $2^{128} - 1$, directly implementing one pRNG in digital logic per stochastic element in our circuit (each initially seeded differently).

Since logic utilization influences circuit cost, energy efficiency, and speed, we briefly mention some techniques we use to compress the logic in our circuits. We can use these ideas whenever the analytical relationships and conditional independencies that directly lead to exact, compact sampling circuits are unavailable, as is often the case in MCMC and SMC proposal generation. The key is to represent *state values*, *energies* (i.e. unnormalized log probabilities), and *probabilities* in a fixed-point form, with m bits for the integer parts of energies, n bits for the decimal part, and $1 + m + n$ total bits for probability values. We then compactly approximate the `logsumexp(`e_1`,`e_2`)` function (to add and normalize energies) and the $\texttt{exp}(e_1)$ function (to convert energies to probabilities), and sample by exact accumulation. We note that numerically tempering a distribution - exponentiating it to some $\frac{1}{\tau}$ - can be parsimoniously implemented as energy bit shifting, for dyadic τ. Heating a distribution by 2^k causes k low-order bits of energy to be lost, while cooling a distribution causes low-order bits to become arbitrarily significant.

2.5 Automatic Translation

We have also built a compiler that produces finite-precision parallel automata for sampling-based inference in factor graphs that are bit-accurate representations of our circuits, substantially reducing development time. Our compiler operates by first coloring the factor graph (using an approximation algorithm) to identify opportunities for parallelism and then constructing a graphical representation of the abstract stochastic automaton that is the desired Gibbs sampler (using the State-Density-Kernel language from [9]). Currently our compiler can target x86 assembly for simulation purposes as well as commodity reconfigurable digital logic as indicated above.

Extending the range of models and inference strategies supported by the compiler (and minimizing human effort in the pathway from its output to synthesizable stochastic logic) looms large, and will be described in future work. We are also actively pursuing attempts to compile to circuit substrates which look less like deterministic digital logic, exploiting (instead of avoiding) some of the "noisy" properties of analog design for more efficient sampling.

2.6 Related Work

The pioneering work of von Neumann [10] and Gaines [11] addressed the reliable implementation of Boolean algebra and arbitrary real arithmetic using stochastic components. Our work is different in motivation and in application: we have introduced methods for engineering large-scale, probabilistic systems, without indirection through generic real arithmetic, which can be used with both deterministic and noisy substrates.

Adaptive or noise-resistant circuit implementations made from stochastic elements have arisen in analog VLSI [12] [13], ultra low power digital logic (via "probabilistic CMOS" [14]), and self-assembled nanoscale circuit fabrication [15]. Our work is at a different level of abstraction, providing complete, compositional specifications of stochastic yet digital circuits for probabilistic arguments and circuit patterns for sampling algorithms. However, our stochastic circuits could be implemented on these substrates, potentially yielding cheaper, more efficient circuits than is possible with standard digital semiconductor techniques. Finally, in mainstream digital design, various application specific accelerators for particle filtering have been explored; see [16] for one detailed example. These efforts have focused on efficient parallel architectures for particle advancement and resampling, using classical methods – not direct rendering of probabilistic arguments in digital circuits – to simulate from forward models and compute weights.

3 Performance Estimates

We synthesized parallel Gibbs circuits on Spartan 3 family FPGAs, measuring the clock rate achieved, clock cycles per sample, and circuit space costs. Figures 5b and 5c show our results for a circuit for Gibbs sampling on a binary, square-lattice Markov Random Field, using the Gibbs lookup table design. We show estimated price/performance curves for 16 bit samplers. In many applications, quantization error due to 16-bit truncation of probabilities for binary variables will be washed away by noise due to Markov chain convergence, Monte Carlo variance, and model uncertainty. Each horizontal step corresponds to a range of problem sizes that fit on the same number of FPGAs; this tiling, and therefore our price/performance results, should actually be achievable in large-scale practice (with only a 3 clock-cycle overhead for FPGA to FPGA communication).

We include performance estimates for parallel sampling on microprocessors, to highlight the gains coming from *combining* parallel sampling with direct simulation in hardware. These estimates generously assume zero cost in time and dollars for interprocessor communication, and 20 clock cycles per sample, ignoring the serial costs due to memory accesses, branch prediction, cycle cost of floating-point operations, and random bit generation. In our experience the actual performance in practice of even highly efficiently programmed parallel Gibbs samplers on conventional CPUs is lacking, with these costs playing a substantial role. Since a conservative MCMC run typically entails hundreds of

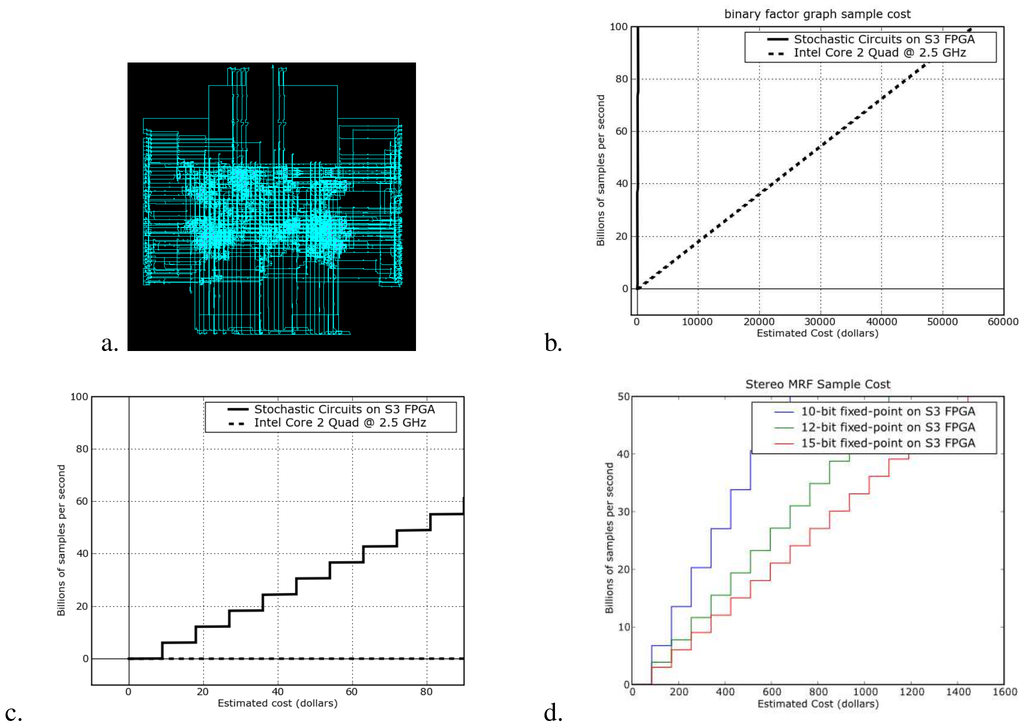

Figure 5: **FPGA price/performance results.** (a) shows an example synthesized FPGA layout for Gibbs sampling on a 9x9 lattice. (b) compares the price/performance ratio of a stochastic circuit to an optimistic estimate for conventional CPUs, in billions of single-site samples per second per dollar, for a binary square-lattice MRF. (c) shows a zoomed-in comparison. (d) shows price/performance results for stereovision. Approximately 10 billion samples per second are needed for real-time (24 FPS) performance at moderate (320x240 pixels) resolution.

thousands of complete Gibbs scans, our circuits should make it possible to affordably obtain reasonable solutions to models with hundreds of thousands of variables in real time.

Figure 5d shows price/performance estimates for a realistic application: annealed Gibbs sampling on an MRF for stereovision [17]. The model has 32 distinguishable disparities per pixel. Our gains should allow generic, annealed Gibbs sampling for discrete variable MRFs to support affordable, dense, real-time stereovision (using standard digital techniques to stream calibrated images into the registers storing MRF potentials on an FPGA). For example, a 320x240 image requires ∼380 million single-site samples per frame, assuming 5000 full Gibbs sweeps of the image for MCMC convergence. With ∼\$300 of hardware, we should be able to solve this problem at 24 frames per second in 15-bit precision. For different models, the time per sample increases roughly linearly with MRF clique size and with graph coloring number, as individual sites take longer to sample and fewer sites can be sampled in parallel.

We have endeavored to make our price-performance comparison as meaningful as possible, comparing the off-the-shelf price for both commodity x86 hardware and commodity FPGAs. This does not take into account the (nontrival) cost of support hardware, such as PCBs, power, and cooling. It is not clear that this is an advantage on either side – while commodity x86 motherboards are cheaper than low-quantity custom-designed FPGA PCBs, it is also much easier to add dozens of FPGAs to a single PCB, which no commodity x86 motherboards support.

3.1 Robustness

Classical digital logic yields state machines that are unstable to even transient faults: a one-bit error in a state representation can yield a next state that is arbitrarily far from the desired next state in Hamming distance. This instability

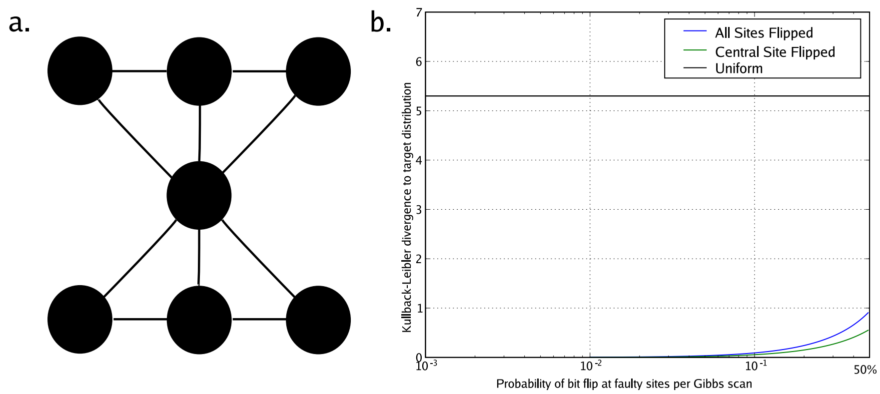

Figure 6: **Robustness to faults.** (a) shows a butterfly-structured binary Markov random field with attractive potentials. (b) shows robustness results to transient single site faults (assessed by deviation of the equilibrium distribution of a Gibbs sampler circuit from the target).

is inherited from the brittleness and determinism of combinational Boolean circuits.

We expect stochastic circuits to be robust for three reasons. First, although a one-bit error might result in a particular sample being sampled from an incorrect distribution, the computation is specified in terms of a *distribution* on outputs. This specification space has a continuous topology, unlike deterministic circuits, whose specification space has a discrete topology. It is therefore possible for specifications of our digital circuits to be meaningfully violated by degrees, allowing for errors to only distort the resulting answer. Second, because our circuits are naturally distributed (exploiting conditional independence), even persistent local faults are much less likely to result in global failures. Third, Markovian fixed-point iteration (as in ergodic convergence of MCMC) yields a "restoring force" that reduces the impact of transient faults over time and encourages local consistency despite persistent faults.

We explore this question on a small but illustrative example shown in Figure 6, using numerical techniques to exactly compute the equilibrium distribution on states for a parallel Gibbs circuit with stochastic transient faults, where the state value for a given site is flipped with some probability. The circuit performs inference in a butterfly-structured attractive binary Markov random field, and thus has a potential central point of failure. Deterministic digital design normally requires Boolean primitives to be extremely reliable, e.g. fault probabilities around 10^{-8}. Here we see that with fault probabilities of 10^{-2}, the equilibrium distribution of our Gibbs circuit is very close to the target, and is still reasonable even with 50% fault probability per site per cycle. If faults are restricted only to the critical central site, performance only degrades slightly, due to the Gibbs sampler's pressure for local consistency. Detailed empirical and theoretical characterization of the robustness properties of large circuits - ideally by comparison to exact calculation or analytical bounds from the theory of Markov chains - remains an open challenge.

4 Discussion and Future Work

We introduced *combinational stochastic logic*, a complete set of stochastic gates for the probability algebra, naturally generalizing the deterministic, Boolean case. We have shown how to construct massively parallel, fault-resistant stochastic state machines for Monte Carlo algorithms, using designs quite unlike the real-valued, vector-arithmetic structures underlying current computers. Instead, we directly model the probability algebra in digital hardware, exploiting the resulting time and space savings to structure our circuits to match our stochastic algorithms. When implemented on tiled arrays of commodity FPGAs, our circuits should support low-cost, real-time approximate inference on models with hundreds of thousands of variables. Much work remains to be done.

First, we should explore different implementation substrates. For example, we could use Gaines-style circuits built via analog VLSI to cheaply implement our Gibbs pipeline elements, combining the speed and energy efficiency of analog computation with the arbitrary composability of digital machines. We could also build reliable stochastic circuits out of nanoscale substrates, exploiting the robustness of our approach. Second, we should explore hypotheses in computational neuroscience based on stochastic circuits implementing approximate inference. One starting point is the observation that time-averaging a wire in a stochastic circuit yields a "rate code" that approximately reports the wire's "instantaneous" marginal probability. Third, we should develop mathematical connections between the finite-size time, space and entropy requirements of stochastic circuits and asymptotic complexity results from randomized algorithms.

We should also construct more sophisticated circuits. We can start by building circuits for approximate inference in nonparametric and hierarchical Bayesian models by combining stochastic samplers with stack-structured memories (for growing statespaces) and content-addressible memories (for e.g. sufficient statistics). We can also directly use the pieces from our Gibbs pipeline to implement more sophisticated algorithms, including sequential Monte Carlo methods and cluster techniques like Swendsen-Wang. We are also exploring novel reprogrammable computer architectures better suited to probabilistic inference than traditional stored-program machines. For example, we have begun development of the *IID*, an FPGA-hosted, stochastic version of the Connection Machine architecture [18], which will be programmed by specifying a probability model and relying on a compiler to automatically produce an appropriate inference circuit.

The apparent intractability of inference has hindered the use of Bayesian methods in the design of intelligent systems and in the explanation of computation in the mind and brain. We hope stochastic logic circuits help to address this concern, by providing new tools for mapping probabilistic inference onto existing digital computing machinery, and suggesting a new class of natively stochastic digital computing machines.

Acknowledgements

We are grateful to Gerry Sussman, Tom Knight and Hal Abelson for helpful discussions and encouragement, and Danny Hillis for writing the dissertation that inspired our approach. VM and EJ would also like to acknowledge their academic advisors, Josh Tenenbaum and Matt Wilson, for support throughout this unusual and extended excursion into computer architecture.

References

[1] N. Metropolis, AW Rosenbluth, MN Rosenbluth, AH Teller, and E. Teller. Equations of State Calculations by Fast Computing Machines. *Journal of Chemical Physics*, 1953.

[2] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images. *Readings in Computer Vision: Issues, Problems, Principles, and Paradigms*, pages 564–584, 1987.

[3] A. Doucet, N. de Freitas, and N. Gordon. Sequential Monte Carlo in Practice, 2001.

[4] George Boole. *An Investigation of the Laws of Thought*. 1854.

[5] Claude Shannon. *A Symbolic Analysis of Relay and Switching Circuits*. PhD thesis, 1940.

[6] E. T. Jaynes. *Probability Theory : The Logic of Science*. Cambridge University Press, April 2003.

[7] W. Kahan and J. Palmer. On a proposed floating-point standard. *ACM SIGNUM Newsletter*, 14:13–21, 1979.

[8] G. Marsaglia. Xorshift RNGs. *Journal of Statistical Software*, 2003.

[9] Keith Bonawitz. *Composable Probabilistic Inference with BLAISE.* PhD thesis, 2008.

[10] J. von Neumann. Probabilistic logics and synthesis of reliable organisms from unreliable components. In C. Shannon and J. McCarthy, editors, *Automata Studies*, pages 43–98. Princeton University Press, 1956.

[11] B. R. Gaines. Stochastic Computing Systems. *Advances in Information Systems Science*, 2, 1969.

[12] R. Genov and G. Cauwenberghs. Stochastic Mixed-Signal VLSI Architecture for High-Dimensional Kernel Machines. *Advances in Neural Information Processing Systems*, 14.

[13] AF Murray, D. Del Corso, and L. Tarassenko. Pulse-stream VLSI neural networks mixing analog and digital-techniques. *Neural Networks, IEEE Transactions on*, 2(2):193–204, 1991.

[14] S. Cheemalavagu, P. Korkmaz, and K.V. Palem. Ultra low-energy computing via probabilistic algorithms and devices: CMOS device primitives and the energy-probability relationship. *Proc. of The 2004 International Conference on Solid State Devices and Materials*, pages 402–403.

[15] W. Qian, J. Backes, and M. Riedel. The Synthesis of Stochastic Circuits for Nanoscale Computation.

[16] M. Bolic. Architectures for Efficient Implementation of Particle Filters. *USA: State University of New York at Stony Brook*, 2004.

[17] MF Tappen and WT Freeman. Comparison of graph cuts with belief propagation for stereo, using identical MRF parameters. *Proc. ICCV*.

[18] W.D. Hillis. *The Connection Machine*. MIT Press, 1989.