Probabilistic computing with p-bits

Jan Kaiser and Supriyo Datta
Elmore Family School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, 47906 USA

(Dated: 14 October 2021)

Digital computers store information in the form of bits that can take on one of two values 0 and 1, while quantum computers are based on qubits that are described by a complex wavefunction whose squared magnitude gives the probability of measuring either a 0 or a 1. Here we make the case for a probabilistic computer based on p-bits which take on values 0 and 1 with controlled probabilities and can be implemented with specialized compact energy-efficient hardware. We propose a generic architecture for such p-computers and emulate systems with thousands of p-bits to show that they can significantly accelerate randomized algorithms used in a wide variety of applications including but not limited to Bayesian networks, optimization, Ising models and quantum Monte Carlo.

I. INTRODUCTION

Feynman famously remarked “Nature isn’t classical, dammit, and if you want to make a simulation of nature, you’d better make it quantum mechanical”. In the same spirit we could say “Many real life problems are not deterministic, and if you want to simulate them, you’d better make it probabilistic”. But there is a difference. Quantum algorithms require quantum hardware and this has motivated a worldwide effort to develop a new appropriate technology. By contrast probabilistic algorithms can be and are implemented on existing deterministic hardware using pseudo RNG’s (random number generators). Monte Carlo algorithms represent one of the top ten algorithms of the 20th century and are used in a broad range of problems including Bayesian learning, protein folding, optimization, stock option pricing, cryptography just to name a few. So why do we need a p-computer?

A key element in a Monte Carlo algorithm is the RNG which requires thousands of transistors to implement with deterministic elements, thus encouraging the use of architectures that time share a few RNG’s. Our work has shown the possibility of high quality true RNG’s using just three transistor [9], prompting us to explore a different architecture that makes use of large numbers of controlled-RNG’s or p-bits. Fig. 1(a) shows a generic vision for a probabilistic or a p-computer having two primary components: an N-bit random number generator (RNG) that generates N-bit samples and a Kernel that performs deterministic operations on them. Note that each RNG-Kernel unit could include multiple RNG-Kernel sub-units (not shown) for problems that can benefit from it. These sub-units could be connected in series as in Bayesian networks (Fig. 2(a) or in parallel as done in parallel tempering or for problems that allow graph coloring. The parallel RNG-Kernel units shown in Fig. 1(a) are intended to perform easily parallelizable operations like ensemble sums using a data collector unit to combine all outputs into a single consolidated output.

Ideally the Kernel and data collector are pipelined so that they can continually accept new random numbers from the RNG, which is assumed to be fast and available in numerous numbers. The p-computer can then provide \(N_p f_c\) samples per second, \(N_p\) being the number of parallel units and \(f_c\) the clock frequency. We argue that when \(N_p = 1\), this throughput is well in excess of what is achieved with standard implementations on either CPU (central processing unit) or GPU (graphics processing unit) for a broad range of applications and algorithms including but not limited to those targeted by modern digital annealers or Ising solvers. Interestingly, a p-computer also provides a conceptual bridge to quantum computing, sharing many characteristics that we associate with the latter. Indeed it can implement algorithms intended for quantum computers, though the effectiveness of quantum Monte Carlo depends strongly on the extent of the so-called sign problem specific to the algorithm and our ability to ‘tame’ it.

II. IMPLEMENTATION

Of the three elements in Fig. 1 two are deterministic. The Kernel is problem-specific ranging from simple operations like addition or multiplication to more elaborate operations that could justify special purpose chiplet. Matrix multiplication for example could be implemented using analog options like resistive crossbars. The data collector typically involves addition and could be implemented with adder trees. The third element is probabilistic, namely the N-bit RNG which is a collection of N 1-bit RNG’s or p-bits. The behavior of each p-bit can be described by

\[ s_i = \Theta [ \sigma (I_i - r) ] \]

where \(s_i\) is the binary p-bit output, \(\Theta\) is the step function, \(\sigma\) is the sigmoid function, \(I_i\) is the input to the p-bit and \(r\) is a uniform random number between 0 and 1. Eq. 1 is illustrated in Fig. 1(b). While the p-bit output is always binary, the p-bit input \(I_i\) influences the mean of the output sequence. With \(I_i = 0\), the output...
is distributed 50–50 between 0 and 1 and this may be adequate for many algorithms. But in general a non-zero $I_1$ determined by the current sample is necessary to generate desired probability distributions from the $N$-bit RNG-block.

One promising implementation of a p-bit is based on a stochastic magnetic tunnel junction (s-MTJ) as shown in Fig. 1(b) whose resistance state fluctuates due to thermal noise. It is placed in series with a transistor, and the drain voltage is thresholded by an inverter to obtain a random binary output bit whose average value can be tuned through the gate voltage $V_{IN}$. It has been shown both theoretically and experimentally that s-MTJ-based p-bits can be designed to generate new random numbers in times ~ nanoseconds. The same circuit could also be used with other fluctuating resistors but one advantage of s-MTJ’s is that they can be built by modifying magnetoresistive random access memory (MRAM) technology that has already reached gigabit levels of integration.

Note, however, that the examples presented here all use p-bits implemented with deterministic CMOS elements or pseudo-RNG’s using linear feedback shift registers (LFSR’s) combined with lookup tables (LUT’s) and thresholding element as shown in Fig. 1(b). Such random numbers are not truly random, but have a period that is longer than the time range of interest. The longer the period, the more registers are needed to implement that is longer than the time range of interest. The longer dom numbers are not truly random, but have a period around four times as many. Physics-based approaches, like s-MTJ’s, naturally generate true random numbers with infinite repetition period and ideally require only 3 transistors and 1 MTJ.

A simple performance metric for $p$-computers is the ideal sampling rate $N_p f_c$ mentioned above. The results presented here were all obtained with an field-programmable gate array (FPGA) running on a 125 MHz clock, for which $1/f_c = 8$ ns, which could be significantly shorter (even ~ 0.1 ns) if implemented with s-MTJ’s. Furthermore, s-MTJ’s are compact and energy-efficient, allowing up to a factor of 100 larger $N_p$ for a given area and power budget. With an increase of $f_c$ and $N_p$, a performance improvement by 2–3 orders of magnitude over the numbers presented here may be possible with s-MTJ’s or other physics-based hardware.

We should point out that such compact p-bit implementations are still in their infancy and many questions remain. First is the inevitable variation in RNG characteristics that can be expected. Initial studies suggest that it may be possible to train the Kernel to compensate for at least some of these variations. Second is the quality of randomness, as measured by statistical quality tests which may require additional circuitry as discussed for example in Ref. Certain applications like simple integration (Section III A) may not need high quality random numbers, while others like Bayesian correlations (Section III B) or Metropolis-Hastings methods that require a proposal distribution (Section III C) may have more stringent requirements. Third is the possible difficulty associated with reading sub-nanosecond fluctuations in the output and communicating them faithfully. Finally, we note that the input to a p-bit is an analog quantity requiring Digital-to-Analog Converters (DAC’s)
unless the kernel itself is implemented with analog components.

III. APPLICATIONS

A. Simple integration

A variety of problems such as high dimensional integration can be viewed as the evaluation of a sum over a very large number $N$ of terms. The basic idea of the Monte Carlo method is to estimate the desired sum from a limited number $N_s$ of samples drawn from configurations $\alpha$ generated with probability $q_\alpha$:

$$M = \sum_{\alpha=1}^{N} m_\alpha \approx \frac{1}{N_s} \sum_{\alpha=1}^{N_s} m_\alpha$$

The distribution $\{q\}$ can be uniform or could be cleverly chosen to minimize the standard deviation of the estimate. In any case the standard deviation goes down as $1/\sqrt{N_s}$ and all such applications could benefit from a $p$-computer to accelerate the collection of samples.

B. Bayesian Network

A little more complicated application of a $p$-computer is to problems where random numbers are generated not according to a fixed distribution, but by a distribution determined by the outputs from a previous set of RNG’s. Consider for example the question of genetic relatedness in a family tree, with each layer representing one generation. Each generation in the network in Fig. 2 (a) with $N$ nodes can be mapped to a $N$-bit RNG-block feeding into a Kernel which stores the conditional probability table (CPT) relating it to the next generation. The correlation between different nodes in the network can be directly measured and an average over the samples computed to yield the correct genetic correlation as shown in Fig. 2 (b). Nodes separated by $p$ generations have a correlation of $1/2^p$. The measured absolute correlation between strangers goes down to zero as $1/\sqrt{N_s}$.

This is characteristic of Monte Carlo algorithms, namely, to obtain results with accuracy $\varepsilon$ we need $N_s = 1/\varepsilon^2$ samples. The $p$-computer allows us to collect samples at the rate of $N_p f_c = 125 \text{ MSamples per second}$ if $N_p = 1$ and $f_c = 125$ MHz. This is about two orders of magnitude faster than what we get running the same algorithm on a Intel Xeon CPU.

How does it compare to deterministic algorithms run on CPU? As Feynman noted in his seminal paper, deterministic algorithms for problems of this type are very inefficient compared to probabilistic ones because of the need to integrate over all the unobserved nodes $\{x_B\}$ in order to calculate a property related to nodes $\{x_A\}$:

$$P_A(x_A) = \int dx_B P(x_A, x_B)$$

By contrast, a $p$-computer can ignore all the irrelevant nodes $\{x_b\}$ and simply look at the relevant nodes $\{x_A\}$. We used the example of genetic correlations because it is easy to relate to. But it is representative of a wide class of everyday problems involving nodes with one-way causal relationships extending from ‘parent’ nodes to ‘child’ nodes, all of which could benefit from a $p$-computer.
C. Knapsack Problem

Let us now look at a problem which requires random numbers to be generated with a probability determined by the outcome from the last sample generated by the same RNG. Every RNG then requires feedback from the very Kernel that processes its output. This belongs to the broad class of problems that are labeled as Markov Chain Monte Carlo (MCMC). An excellent summary and evaluation of MCMC sampling techniques we refer the reader to Ref. (b) Time to solution

FIG. 3. Example of MCMC - Knapsack problem: (a) Mapping to the general p-computer framework. (b) Performance of the p-computer compared to CPU implementation of the same probabilistic algorithm and along with two well-known deterministic approaches. A deterministic algorithm like Pisinger (40) which is optimized specifically for the Knapsack problem can outperform MCMC. But for a given MCMC algorithm, the p-computer provides orders of magnitude improvement over the standard CPU implementation.

The knapsack is a textbook optimization problem described in terms of a set of items, \( m = 1, \ldots, N \), each containing a value \( v_m \) and weighing \( w_m \). The problem is to figure out which items to take \((s_m = 1)\) and which to leave behind \((s_m = 0)\) such that the total value \( V = \sum_m v_m s_m \) is a maximum, while keeping the total weight \( W = \sum_m w_m s_m \) below a capacity \( C \). We could straightforwardly map it to the p-computer architecture (Fig. 1), using the RNG to propose solutions \( \{s\} \) at random, and the Kernel to evaluate \( V, W \) and decide to accept or reject. But this approach would take us toward the solution far too slowly. It is better to propose solutions intelligently looking at the previous accepted proposal, and making only a small change to it. For our examples we proposed a change of only two items each time.

This intelligent proposal, however, requires feedback from the kernel which can take multiple clock cycles. One could wait between proposals, but the solution is faster if instead we continue to make proposals every clock cycle in the spirit of what is referred to as multiple-try Metropolis (42). The results are shown in Fig. 3 and compared with CPU (Intel Xeon @ 2.3GHz) and GPU (Tesla T4 @ 1.59GHz) implementations, using the probabilistic algorithm. Also shown are two efficient deterministic algorithms, one based on dynamic programming (DP), and one due to Pisinger et al (40,43).

Note that the probabilistic algorithm (MCMC) gives solutions that are within 1% of the correct solution, while the deterministic algorithms give the correct solution. For the Knapsack problem getting a solution that is 99% accurate should be sufficient for most real world applications. The p-computer provides orders of magnitude improvement over CPU implementation of the same MCMC algorithm. It is outperformed by the algorithm developed by Pisinger et al (40,43) which is specifically optimized for the Knapsack problem. However, we note that the p-computer projection in Fig. 3 (b) is based on utilizing better hardware like s-MTJ’s but there is also significant room for improvement of the p-computer by optimizing the Metropolis algorithm used here and/or by adding parallel tempering (44).

D. Ising model

Another widely used model for optimization within MCMC is based on the concept of Boltzmann machines (BM) defined by an energy function \( E \) from which one can calculate the synaptic function \( I_i \),

\[
I_i = \beta (E(s_i = 0) - E(s_i = 1)),
\]

that can be used to guide the sample generation from each RNG ‘i’ in sequence (44) according to Eq. (1). Alternatively the sample generation from each RNG
can be fixed and the synaptic function used to decide whether to accept or reject it within a Metropolis-Hastings framework. Either way, samples will be generated with probabilities \( P_\alpha \sim \exp(-\beta E_\alpha) \). We can solve optimization problems by identifying \( E \) with the negative of the cost function that we are seeking to minimize. Using a large \( \beta \) we can ensure that the probability is nearly 1 for the configuration with the minimum value of \( E \).

In principle, the energy function is arbitrary, but much of the work is based on quadratic energy functions defined by a connection matrix \( W_{ij} \) and a bias vector \( h_i \) (see for example:\cite{8,10–17}):

\[
E = -\sum_{ij} W_{ij} s_i s_j - \sum_i h_i s_i,
\]

For this quadratic energy function, Eq. (4) gives \( I_i = \beta(\sum_j W_{ij} s_j + h_i) \), so that the Kernel has to perform a multiply and accumulate operation as shown in Fig. 4 (a). We refer the reader to Sutton et al.\cite{8} for an example of the max-cut optimization problem on a two-dimensional 90 \( \times \) 90 array implemented with a \( p \)-computer.

Eq. (4), however, is more generally applicable even if the energy expression is more complicated, or given by a table. The Kernel can be modified accordingly. For an example of a energy function with fourth order terms implemented on an eight bit \( p \)-computer, we refer the reader to Borders et al.\cite{10}.

A wide variety of problems can be mapped onto the BM with an appropriate choice of the energy function. For example, we could generate samples from a desired probability distribution \( P_\prime \), by choosing \( \beta E = -\ln P_\prime \). Another example is the implementation of logic gates by defining \( E \) to be zero for all \( \{s\} \) that belong to the truth table, and have some positive value for those that do not\cite{22}. Unlike standard digital logic, such a BM-based implementation would provide invertible logic that not only provides the output for a given input, but also generates all possible inputs corresponding to a specified output\cite{24,46,47}.

E. Quantum Monte Carlo

Finally let us briefly describe the feasibility of using \( p \)-computers to emulate quantum or \( q \)-computers. A \( q \)-computer is based on qubits that are neither 0 or 1, but are described by a complex wavefunction whose squared magnitude gives the probability of measuring either a 0 or a 1. The state of an \( n \)-qubit computer is described by a wavefunction \( \{\psi\} \) with \( 2^n \) complex components, one for each possible configuration of the \( n \) qubits.

In gate-based quantum computing (GQC) a set of qubits is placed in a known state at time \( t \), operated on with \( d \) quantum gates to manipulate the wavefunction through unitary transformations [\( U^{(i)} \)]

\[
\{\psi(t+d)\} = [U^{(d)}] \cdots [U^{(1)}] \{\psi(t)\} \quad \text{(GQC)}
\]

and measurements are made to obtain results with probabilities given by the squared magnitudes of the final wavefunctions. From the rules of matrix multiplication, the final wavefunction can be written as a sum over a very large number of terms:

\[
\psi_m(t+d) = \sum_{i\cdots j,k} U^{(d)}_{m,i} \cdots U^{(1)}_{j,k} \psi_k(t)
\]

Conceptually we could represent a system of \( n \) qubits and \( d \) gates with a system of \( (n \times d) \) \( p \)-bits with \( 2^{nd} \) states which label the \( 2^{nd} \) terms in the summation in Eq. 7. Each of these terms is often referred to as a Feynman path and what we want is the sum of the amplitudes of

---

**FIG. 4. Example of Quantum Monte Carlo (QMC) - Transverse Field Ising model:** (a) Mapping to the general \( p \)-computer framework. (b) Solving the transverse Ising model for quantum annealing. Subfigure (b) is adapted from B. Sutton, R. Faria, L. A. Ghantasala, R. Jaiswal, K. Y. Camsari and S. Datta, IEEE Access, vol. 8, pp. 157238-157252, 2020 licensed under a Creative Commons Attribution (CC BY) license.
all such paths:

\[ \psi_m(t + d) = \sum_{\alpha=1}^{2^m} A_m^{(\alpha)} \]  

The essential idea of quantum Monte Carlo is to estimate this enormous sum from a few suitably chosen samples, not unlike the simple Monte Carlo stated earlier in Eq. (2). What makes it more difficult, however, is the so-called sign problem \[ e^{-\frac{\beta H}{r}} \] which can be understood intuitively as follows. If all the quantities \( A_m^{(\alpha)} \) are positive then it is relatively easy to estimate the sum from a few samples. But if some are positive while some are negative with lots of cancellations, then many more samples will be required. The same is true if the quantities are complex quantities that cancel each other.

The matrices \( U \) that appear in GQC are unitary with complex elements which often leads to significant cancellation of Feynman paths, except in special cases when there may be complete constructive interference. In general this could make it necessary to use large numbers of samples for accurate estimation. A noiseless quantum computer would not have this problem, since qubits intuitively perform the entire sum exactly and yield samples according to the squared magnitude of the resulting wavefunction. However, real world quantum computers have noise and \( p \)-computers could be competitive for many problems.

Adiabatic quantum computing (AQC) operates on very different physical principles but its mathematical description can also be viewed as summing the Feynman paths representing the multiplication of \( r \) matrices:

\[ [e^{-\beta H_1/r}] \cdots [e^{-\beta H_r/r}] \quad (AQC) \]  

This is based on the Suzuki-Trotter method described in Camsari et al.\[8\], where the number of replicas, \( r \), is chosen large enough to ensure that if \( H = H_1 + H_2 \), one can approximately write \( e^{-\beta H/r} \approx e^{-\beta H_1/r} e^{-\beta H_2/r} \). The matrices \( e^{-\beta H/r} \) in AQC are Hermitian, and their elements can be all positive.\[50\] A special class of Hamiltonians \( H \) having this property is called stoquastic and there is no sign problem since the amplitudes \( A_m^{(\alpha)} \) in the Feynman sum in Eq. (8) all have the same sign.

An example of such a stoquastic Hamiltonian is the transverse field Ising model (TFIM) commonly used for quantum annealing where a transverse field which is quantum in nature is introduced and slowly reduced to zero to recover the original classical problem. Fig. 3 adapted from Sutton et al.\[8\] shows a \( n = 250 \) qubit problem mapped to a 2-D lattice of 250 \( \times 10 = 2500 \) \( p \)-bits using \( r = 10 \) replicas to calculated average correlations between the \( z \)-directed spins on lattice sites separated by \( L \). Very accurate results are obtained using \( N_s = 10^5 \) samples. However, these samples were suitably spaced to ensure their independence, which is an important concern in problems involving feedback.

Finally we note that quantum Monte Carlo methods, both GQC and AQC, involve selective summing of Feynman paths to evaluate matrix products. As such we might expect conceptual overlap with the very active field of randomized algorithms for linear algebra\[51,52\] though the two fields seem very distinct at this time.

IV. CONCLUDING REMARKS

In summary, we have presented a generic architecture for a \( p \)-computer based on \( p \)-bits which take on values 0 and 1 with controlled probabilities, and can be implemented with specialized compact energy-efficient hardware. We emulate systems with thousands of \( p \)-bits to show that they can significantly accelerate the implementation of randomized algorithms that are widely used for many applications.\[53\] A few prototypical examples are presented such as Bayesian networks, optimization, Ising models and quantum Monte Carlo.

ACKNOWLEDGMENTS

The authors are grateful to Behtash Behin-Aein for helpful discussions and advice. We also thank Kerem Camsari and Shuvro Chowdhury for their feedback on the manuscript. The contents are based on the work done over the last 5-10 years in our group, some of which has been cited here, and it is a pleasure to acknowledge all who have contributed to our understanding. This work was supported in part by ASCENT, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

DATA AVAILABILITY

The data that support the findings of this study are available from the corresponding author upon reasonable request.

AUTHOR DECLARATIONS

Conflict of interests

One of the authors (SD) has a financial interest in Ludwig Computing.

1R. P. Feynman, “Simulating physics with computers,” Int. J. Theor. Phys 21 (1982).
2B. A. Cipra, “The Best of the 20th Century: Editors Name Top 10 Algorithms,” SIAM News 33, 1–2 (2000).
3K. Y. Camsari, S. Salahuddin, and S. Datta, “Implementing \( p \)-bits with Embedded MTJ,” IEEE Electron Device Letters 38, 1767–1770 (2017).
4J. Kaiser, R. Jaiswal, B. Behin-Aein, and S. Datta, “Benchmarking a Probabilistic Coprocessor,” arXiv:2109.14801 [cond-mat] (2021), arXiv:2109.14801 [cond-mat].
5R. Swendsen and J.-S. Wang, “Replica Monte Carlo Simulation of Spin-Glasses,” Physical review letters 57, 2607–2609 (1986).
45. W. K. Hastings, “Monte Carlo sampling methods using Markov chains and their applications,” Biometrika 57, 97–109 (1970).
46. Y. Lv, R. P. Bloom, and J.-P. Wang, “Experimental Demonstration of Probabilistic Spin Logic by Magnetic Tunnel Junctions,” IEEE Magnetics Letters 10, 1–5 (2019).
47. N. A. Aadit, A. Grimaldi, M. Carpentieri, L. Theogarajan, G. Finocchio, and K. Y. Camsari, “Computing with Invertible Logic: Combinatorial Optimization with Probabilistic Bits,” IEEE International Electron Devices Meeting (IEDM) (To appear) (2021).
48. S. Chowdhury, K. Y. Camsari, and S. Datta, “Emulating Quantum Interference with Generalized Ising Machines,” arXiv:2007.07379 [cond-mat, physics:quant-ph] (2020), arXiv:2007.07379 [cond-mat, physics:quant-ph].
49. K. Y. Camsari, S. Chowdhury, and S. Datta, “Scalable Emulation of Sign-Problem–Free Hamiltonians with Room-Temperature $p\hbar$-bits,” Phys. Rev. Applied 12, 034061 (2019).
50. S. Bravyi, D. P. Divincenzo, R. Oliveira, and B. M. Terhal, “The complexity of stoquastic local Hamiltonian problems,” Quantum Information & Computation 8, 361–385 (2008).
51. P. Drineas, R. Kannan, and M. W. Mahoney, “Fast Monte Carlo Algorithms for Matrices I: Approximating Matrix Multiplication,” SIAM Journal on Computing 36, 132–157 (2006).
52. P. Drineas, R. Kannan, and M. W. Mahoney, “Fast Monte Carlo Algorithms for Matrices II: Computing a Low-Rank Approximation to a Matrix,” SIAM Journal on Computing 36, 158–183 (2006).
53. A. Buluc, T. G. Kolda, S. M. Wild, M. Anitescu, A. DeGennaro, J. Jakeman, C. Kamath, Ramakrishnan, Kannan, M. E. Lopes, P.-G. Martinsson, K. Myers, J. Nelson, J. M. Restrepo, C. Seshadhri, D. Vrabie, B. Wohlberg, S. J. Wright, C. Yang, and P. Zwart, “Randomized Algorithms for Scientific Computing (RASC),” arXiv:2104.11079 [cs] (2021), arXiv:2104.11079 [cs].