Computational Proof as Experiment: Probabilistic Algorithms from a Thermodynamic Perspective[⋆]

Krishna V. Palem

Center for Research on Embedded Systems and Technology,
School of Electrical and Computer Engineering,
Georgia Institute of Technology, Atlanta GA 30332, USA.
palem@ece.gatech.edu

Abstract. A novel framework for the design and analysis of *energy-aware* algorithms is presented, centered around a deterministic *Bit-level (Boltzmann) Random Access Machine* or BRAM model of computing, as well its probabilistic counterpart, the RABRAM . Using this framework, it is shown for the first time that *probabilistic algorithms can asymptotically yield savings in the energy consumed, over their deterministic counterparts*. Concretely, we show that the *expected energy savings* derived from a probabilistic RABRAM algorithm for solving the *distinct vector problem* introduced here, over *any* deterministic BRAM algorithm grows as $\Theta\left(n \ln\left(\frac{n}{n-\varepsilon\log(n)}\right)\right)$, even though the deterministic and probabilistic algorithms have the same (asymptotic) time-complexity. The probabilistic algorithm is guaranteed to be correct with a probability $\mathbf{p} \geq (1-\frac{1}{n^{\mathbf{c}}})$ (for a constant $\mathbf{c}$ chosen as a design parameter). As usual n denotes the length of the input instance of the DVP measured in the number of bits. These results are derived in the context of a technology-independent complexity measure for energy consumption introduced here, referred to as *logical work*. In keeping with the theme of the symposium, the introduction to this work is presented in the context of "computational proof" (algorithm) and the "work done" to achieve it (its energy consumption).

1 Introduction

The word "fact" conjures up images of a sense of definitiveness in that there is a belief in its absolute *truth*. This notion is the very essence of modern mathematical theories, with their foundational framework based on (formal) languages such as the *predicate calculus*. Thus, following Russell and Whitehead's seminal formalization of mathematical reasoning embodied in their Principia [31], the very notion of the consistency of an axiomatic theory disallows even a hint of a doubt about a fact, often referred to as a *theorem* (or its subsidiary *lemma*) in modern as well as ancient mathematical thought. The modern foundations of verification as proof, with emphasis on its automatic or mechanized form, applied to problems motivated in large part from within the disciplines of

⋆ This work is supported in part by DARPA under seedling contract #F30602-02-2-0124.

computer science and electrical engineering (see Manna for example [13, 14]) are also bound in essential ways to this notion of an absolute or *deterministic* truth.

Concomitant to this absolute notion of truth, and a significant contribution of the mathematical theory of computing (referred to in popular terms as theoretical computer science) is the notion of the *complexity* or equivalently, the "degree of difficulty" of such a proof. Thus, starting with Rabin's [23] work as a harbinger with further contributions by Blum [1], the notion of a machine independent measure of *complexity* led to the widely used formulations of Hartmanis and Stearns [7]—essentially within the context of a deterministic mechanistic approach to proof. Here, a deterministic algorithm—equivalently, any execution of a Turing machine's program [28]—upon halting, is viewed as proving a theorem or fact, stated as a decision problem. For example, determining the outcome of the celebrated *halting* problem [14, 28] would constitute proving such a theorem in the context of a given instance, where an answer of a *yes* would imply that the Turing machine program given as the input would halt with certainty.

Both this notion of absolute truth as well as the deterministic (Turing machine based) approach to arriving at it mechanically are subject to philosophically significant revision if one considers alternate approaches that are *not* deterministic. A critical first step involves non-deterministic approaches with the foundations laid by Rabin and Scott [25]. Based on these foundations, Cook's [4] (and Levin's [12]) characterizations of NP as a resource bounded class of proofs, whose remarkable richness was demonstrated by Karp [9], elevated NP to a complexity class of great importance, and the accompanying P=?NP question to its exalted status. Here, while the approach to proving is not based on the traditional deterministic transition of a Turing machine, the meaning of truth one associates with the final outcome—*accept* or *reject*—continues to be definite or deterministic.

Moving beyond nondeterminism, the early use of statistical methods with emphasis on probability can be found in Karp's [10] introduction of *average case analysis*. Compelled by the need to better understand the gap between the empirical behavior and the results of pessimal (mathematical) analysis of algorithms (or a determination of lengths of proofs in our sense), in Karp's approach, the input is associated with a probability distribution. Thus, while the proof itself is deterministic, its difficulty, length, or more precisely its *expected time complexity* is determined by averaging over all possible inputs.

A striking shift in the notion of proof as well as the truth associated with it emanated from the innovation of *probabilistic* methods and algorithms. In this context, both the method or "primitive" proof-step (of the underlying program) as well as the certainty associated with the proof undergo profound revision. Schwartz [26] anticipated the eventual impact of the role of probability in the context of these influential developments best: "The startling success of the Rabin-Strassen-Solovay (see Rabin [24]) algorithm, together with the intriguing foundational possibility that axioms of randomness may constitute a useful fundamental source of mathematical truth independent of, but supplementary to, the standard axiomatic structure of mathematics (see Chaitin and Schwartz [3]), suggests that probabilistic algorithms ought to be sought vigorously." Thus, *in this probabilistic context, both the deduction step as well as the meaning of* *truth are both associated with probabilities as opposed to certainties*. For convenience, let us refer to these as probabilistic proofs (or algorithms when convenient).

With this as background, we now consider the long and fruitful relationship between the notions of proof in the domain of mathematics and its remarkable use in the physical sciences over the past several centuries. Historically, mathematical theories have served remarkably well in characterizing and deducing truths about the universe in a variety of domains, with notable successes in mechanics (classical and quantum), relativity and cosmology, and physical chemistry to name a few areas—see von Neumann's [30] development of quantum mechanics as a notable example. In this role, knowledge about the physical world is derived from mathematical frameworks, methods, and proofs, which could include the above mentioned algorithmic form of proof as well. Thus, in all of the above endeavors, the *direction* is *from* (applying) mathematics *to* (creating knowledge about) physical reality. By contrast, in this work, we are concerned with the opposite direction—*from* using computational devices rooted in the reality of the physical universe such as transistors, *to* establishing (computationally derived) mathematical facts or theories. Let us, for convenience (and without a careful and scholarly study of the possible use of this concept by philosophers earlier on), refer to this opposing perspective as a *reversal of ontological direction, wherein the physical universe and its empirical laws form the basis for all deduction of mathematical facts through computational proof.* To clarify, the reversal in "ontological direction" which this work (and earlier publications of this author on which it is based [19, 20]) explore, refers to the fact that the physical universe and its laws as embodied in computing devices, form the basis for (algorithmically) generating mathematical knowledge, by contrast with the traditional and *opposite* direction wherein mathematical methods produce knowledge about the physical world.

To reiterate, in all of this work, the meaning we associate with proof will be that associated with the execution of a Turing machine program, and we will be interested in the "complexity" of realizing such a (mechanized proof) in the physical universe. Thus, to reiterate, we will consider a concrete and physically realizable form of a proof—such as that generated by a theorem-prover executing on a conventional microprocessor, or perhaps its Archimedian predecessor—as a physical counterpart of Putnam's [22] "verificationist" approach by contrast with (as observed by him [22]) the "Platonic" approach with "evidence that the mind has mysterious faculties of grasping concepts" (or "perceiving mathematical objects...").

Continuing, a first and important observation about the universe of physical objects such as modern microprocessors is that their inherent behavior is best described statistically. Thus, all notions of *determinism* are "approximations" *in that they are only true with sufficiently high probability*. (See Meindl [15] and Stein [27] for a deterministic interpretation of the values 0 and 1 within the context of switching based computing through electrical devices, to better understand this point.) Building on this observation, the work described in this paper characterizes the (somewhat oversimplified in this introduction) fact that the process of computational proof entails physical "work", *which in turn consumes energy* described in its most elegant form through statistical thermodynamics. *The crux of our thesis is that since nature at its very heart, or our perception of it, as we understand it today, is statistical at a (sufficiently) small, albeit* *classical scale—side-stepping the debate whether "God does or does not play dice" (attributed to Einstein to whom a statistical foundation to physical reality was a source of considerable concern)—the most natural physical models for algorithmic proof or verification using fine-grained physical devices such as increasingly small transistors, are essentially probabilistic, and their energy consumption is a crucial figure of merit!* Thus, any deterministic form of computational proof based on using modern computing devices are essentially approximations derived by investing (sufficiently) large amount of energy to make the probability of error small [15]. For completeness, we reiterate here that following the principle of reversal of ontological direction, we are only concerned with the discovery of mathematical knowledge via computational proofs realized through the dynamics of a physical computing device, such as the repeated switching of semiconductor devices in a microprocessor.

Now, considering the specific technical contributions of this work, first, in order to describe and analyze these physically realized proofs or algorithms, we introduce (Section 2) a simple *energy-aware* model for computing: the *Bit-level (Boltzmann) Random Access Machine* or BRAM , as well as its probabilistic variant, the RABRAM (in Section 2.4). Specifically, each primitive step or transition of these models involves a change of state—realized in a canonical way through a transition function associated with a finite state control as in Turing machines [28]—that mirrors a corresponding and explicit change in some physically realizable device. One variant of such a realization is through the notion of a *switching step* [15, 20] whereas an earlier more abstract variation is through the notion of an *emulation* [19] of the transition in the physical universe.

Any computational proof (or equivalently algorithm) described in such a model has an associated technology-independent *energy complexity*, introduced as *logical work* in Section 3 for the deterministic as well as the probabilistic cases. Historically, the interest and subsequently the success of probabilistic algorithms within the context of algorithm design, was to derive (asymptotically) faster algorithms. Assuming that all steps take (about) the same amount of energy, traditional analysis of time-complexity will trivially imply that a probabilistic algorithm might consume less energy, because it computes and solves problems faster—shorter running time implies lesser switching energy. In contrast to these obvious advantages, we show in Section 4 that the energy advantages offered by probabilistic algorithms can be more subtle and varied. Concretely, we prove that for the *distinct vector problem* or DVP , a probabilistic algorithm and its deterministic counterpart take the same number of (time) steps asymptotically, whereas the probabilistic approach yields *energy savings* that grow as $n \longrightarrow \infty$.

Solving the DVP involves computationally (in the BRAM or RABRAM model) proving that a given $n-$ tuple defined on the set of symbols $\{0,1\}$ has the symbol 1 in all of its n positions; the answer to this decision question (or theorem) is YES if indeed all positions of the input $n-$ tuple have the symbol 1 and the answer is NO otherwise. In this paper, we are interested in the following *dense* variant of the DVP : the input $n-$ tuple either has no 0 symbol in it, or if it does have a 0 symbol, it has $\log(n)$ such symbols. For this (dense) version of the DVP problem, which for convenience will be referred to as the DVP problem throughout (defined in Section 4.1), we prove that a novel *probabilistic value amplification* algorithm, proves the (algorithmic) theorem, or resolves the associated decision question with an error probability bound above by $\frac{1}{n^c}$ (for a constant

c chosen as a design parameter) using an *expected* $(2n+\log^k(n))\kappa T \ln\left(2\left[1-\frac{\varepsilon \log n}{n}\right]\right)$ Joules, where $0<\varepsilon<1$ and $k>2$ are constants. The algorithm and its associated analysis are outlined in Section 4.

In an earlier publication, this author proved [19] that any deterministic BRAM algorithm for solving the DVP consumes at least $(2n-\log(n)+1)\kappa T \ln 2$ Joules; this is a lowerbound. By combining these two facts, we show that through the use of the probabilistic algorithm introduced here, the expected savings in energy measured in Joules grows as $\Theta\left(n\log\left(\frac{n}{n-\varepsilon\log(n)}\right)\right)$, for a constant $0<\varepsilon<1$, and for an n bit input to the DVP . Thus both the savings as well as the error probability are respectively monotone increasing and decreasing functions of n. *To the best of our knowledge, this result is the first of its kind that establishes an asymptotic improvement in the energy consumed.*

These models and analysis methodology build on the following results (from [18, 20]) that bridge computational complexity and statistical thermodynamics for the first time: *a single deterministic computation step, which corresponds to a switching step, consumes at least* $\kappa T ln(2)$ *Joules, and this is a lowerbound. Furthermore, using probabilistic computational steps (or switching), the energy consumed by each step can be shown to be as low as* $\kappa T \ln(2p)$ *Joules, where* $p\geq\frac{1}{2}$ *is the probability that the transition is correct;* $(1-p)$ *is the per-step error probability.* Also, κ is the well-known Boltzmann's constant, T is the temperature of the thermodynamic system, and ln is the natural logarithm. In all of this work, the physical models are based on the statistical and hence probabilistic generalizations of switches formulated originally by Szilard [11] within the context of clarifying the celebrated Maxwell's demon paradox [11, 29]. A detailed comparison and bibliography of relevant work from the related field referred to as the Thermodynamics of Computing can be found in [20]. Additionally, Feynman [5] provides a simple and lucid introduction to the interplay between thermodynamically based physical models of computing, mathematical models, and abstractions such as Turing machines.

2 The Bit-Level (Boltzmann) Random Access Machine - BRAM

In this section, we will introduce our machine model for computing, exclusively operating in the *logical* domain. However, to reiterate, a fundamental theorem of this work is that each of its *state transitions*—explained below—will be associated with definite amounts of energy expenditure. Furthermore, this energy consumption will be precisely related to the inherent amount of energy needed to compute, using this model. Significantly, a BRAM model will allow us to abstract away all aspects of the underlying physics and characterize energy purely in the world in which models of computation such as Turing machines are realized. We anticipate this as being very helpful from the perspective of algorithm analysis and design—an exercise which, in a BRAM , can be decoupled from the specificities of physical implementations.

The BRAM however does provide a bridge to the physical world through the energy costs associated with the transitions of its *finite state control* (defined below). This bridge to the world of implementation and energy allows us to define the novel complexity measure of *logical work* as detailed in Section 3, which characterizes the "energy complexity" of the algorithm being designed.

2.1 Informal Introduction to a BRAM

Informally, a BRAM (a *bit-level random access machine*[1]) has a *program* with a finite number of *states*. The transition from a current state to the next involves evaluating the associated *transition function* leading to the “reading” of one or more bits of an input from a specific memory location, transitioning to a new state and writing a new value in a designated memory location. The number of bits read is dependent of the size of the *alphabet*, to be defined below. Every execution starts in a unique START state, and halts upon reaching a unique STOP state.

To extend such models to be able to account for the energy consumed, we define a BRAM (somewhat) formally. For a computer scientist, defining a BRAM based on well-understood elements of a random access machine (or RAM) is elementary; however, we define it here for completeness. The textbook by Papadimitriou [21] provides a rigorous and complete introduction to models such as Turing machines and random access machines including definitions of conventional measures of complexity for representing time and space. This book also provides a comprehensive introduction to the numerous well-understood interrelationships between classes of (time and space) complexity, and can serve as an excellent guide to the topic of defining models of computation in classical contexts, not concerned with energy.

2.2 Defining a BRAM

A BRAM consists of several components, which will be introduced in the rest of this section.

The BRAM Program Following convention, the *program* P is represented as a five-tuple $\{PC, \Sigma, R, \delta, Q\}$. Note that conventionally, variants of the program are referred to as the *finite state control*.

The Set of States - PC is the set of states. Each state $pc_i \in PC$ has designated locations in *memory*, defined below, that serve respectively as its input and output. Without loss of generality, let the states be labeled $1, 2, 3, \ldots, |PC|$. The set Q consists of three special states, START , STOP and UNDEFINED-STATE not in PC.

The Alphabet of the BRAM - Σ is a finite alphabet, and without loss of generality, we will use the set $\{1, 2, \ldots |\Sigma|\}$, which includes the empty symbol ϕ to denote this alphabet. From the standpoint of algorithm design, in most cases, it suffices to work with an alphabet drawn from the set $\Sigma = \{0, 1\}$. However, as we will discuss in this paper, the size of the alphabet $|\Sigma|$ has important consequences to the precise energy behavior of the associated state transitions. Therefore, the contexts wherein the more restricted alphabet is used need to be distinguished from those contexts in which the more general alphabet of size $|\Sigma| > 2$ is used.

The Address Registers of the States in PC - These registers are places where the input and output addresses of a state are stored. In conventional computer science and engineering parlance, a BRAM uses a form of accessing memory that is referred to

[1] Given a BRAM’s eventual connection with energy and its statistical interpretation, one can also interpret the acronym to mean a Boltzmann random access machine.

as *indirect addressing*. We shall return to a discussion of the role of these registers in Section 2.3. The *address registers*, represented by the set R is partitioned into two classes R^{in} and R^{out}; these are both sets (of registers) where each register $\rho_j^{in} \in R^{in}$ ($\rho_j^{out} \in R^{out}$) is a (potentially unbounded) linearly ordered set of elements referred to as *cells* $< s_{j,1}, s_{j,2}, \ldots, s_{j,k} >$ $(< t_{j,1}, t_{j,2}, \ldots, t_{j,k} >)$. Each of the cells s_l (t_l) is *associated* with a value from the set $\{0, 1, \phi\}$. We note that even though the overall alphabet may be of size $|\Sigma| > 2$, each cell in the registers either stores a single bit, or is empty. Furthermore, if the value associated with such an element is ϕ (not defined) for some value of $k' \leq k$, then the value associated with all $s_{j,k''}$ $(t_{j,k''})$ is ϕ for all $k' \leq k'' \leq k$; thus, in the general case, the values stored in any of the address registers are a continuous "run" of values from the set $\{0, 1\}$ followed by a run, possibly of length zero, of the symbol ϕ.

We associate the pair $\rho_j^{in} \in R^{in}$ and $\rho_j^{out} \in R^{out}$ uniquely with the state pc_j. For a given state, intuitively, these pair of registers yield the addresses from where the input σ is to be read, and to where the output σ' (if any) is to be "written" respectively. It is important to note that these addresses can in fact be the registers themselves. The potentially unbounded lengths of the registers denote the fact that the range of addresses being accessed (corresponding to the length of a Turing machine's tape for example) could be unbounded[2].

The Transition Function - We are now ready to define the transition function δ, which will play a central role in characterizing the energy behavior of computations. In its most general form, a transition function is based on an alphabet of size $|\Sigma| \geq 2$.

Syntactically, $\delta : (PC \cup \{\text{START}\}) \times \Sigma \rightarrow (PC \cup Q - \{\text{START}\}) \times \Sigma$ is the transition function. Whenever $\delta(pc_i, \sigma \in \Sigma) = (pc_j, \sigma' \in \Sigma)$, we say that δ transitions from pc_i to the *next-state* pc_j with σ as input and σ' as the output.

Some useful remarks about the transition function follow. First, we note that the state UNDEFINED-STATE is in the range of δ. Given a state pc_i, let ν_i denote the number of symbols from Σ for which δ transitions into a state in $PC \cup \{\text{STOP}\}$, as opposed into the UNDEFINED-STATE . For the remaining $(|\Sigma| - \nu_i)$ symbols, δ transitions into UNDEFINED-STATE . (This is one way of defining transitions of varying "arity" ν_i associated with state pc_i, thus allowing states with varying number of successors with an alphabet of fixed size). In this setting, it is trivial to verify that there is no loss of generality in defining δ such that the first ν_i symbols from the linearly ordered set Σ represent defined transitions whereas symbols $\nu_{i+1}, \nu_{i+2}, \ldots |\Sigma|$ represent undefined transitions. These notions are illustrated in Figure 1. In the sequel, we will (mostly) be concerned with BRAM programs whose transition function has a maximum arity of two. (It is trivial to verify that any BRAM program with transition function of arity more than two can be replaced with a BRAM program with transition function whose maximum arity is two although its energy behavior need not be preserved). Furthermore, any transition with an arity of two will henceforth be referred to as the BRANCH instruction.

For convenience, drawing upon graph theoretic terminology, let us refer to ν_i as the *fanout* of pc_i and furthermore, refer to state pc_j as being *a successor* of pc_i if and only if there exists a symbol $\sigma \in \Sigma$ such that $\delta(pc_i, \sigma)$ yields pc_j as the next state. Let *successors*$_i$ denote the set of *all* successors of state pc_i from PC.

[2] In any terminating computation, there will be a limit on this bound, typically specified as a function of the length of the input [21].

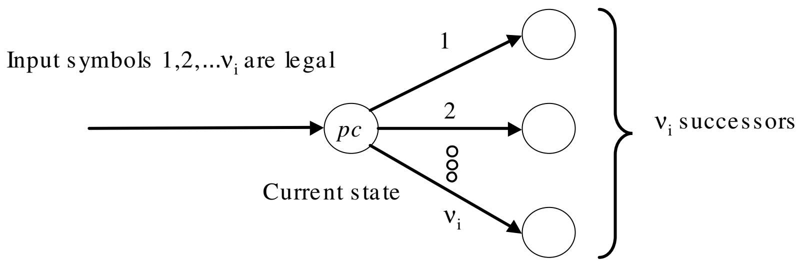
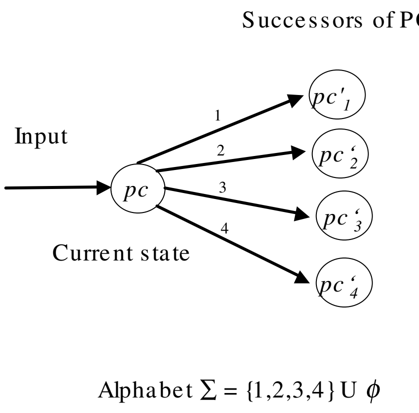


Fig. 1. Illustrating the legal and illegal cases of a transition function with an alphabet of size $|\Sigma| \geq \nu_i$



Input	Transition to
1	pc'$_1$
2	pc'$_2$
3	pc'$_3$
4	pc'$_4$

Fig. 2. A state, its successors and related transitions

The Memory Each BRAM has a (potentially unbounded) MEMORY denoted as the set of $L = (2|PC|+1)$ linearly ordered sets or *banks*, each potentially unbounded. As shown in Figure 3, elements I and $(I+1)$ in MEMORY are denoted M_I and M_{I+1} where $1 \leq I \leq 2|PC|$ are respectively used as registers $\rho_i^{in} \in R^{in}$ and $\rho_i^{out} \in R^{out}$, where $i = \lceil \frac{I}{2} \rceil$. Additionally, the last set M_L of MEMORY, denoted $\mathbf{M}$ is a potentially unbounded set $\mathbf{M} < m_1, m_2, \ldots, m_k >$. Each cell m_j of memory is associated with an element from the set $\{0, 1, \phi\}$. Informally, $\mathbf{M}$ is the set of locations where the inputs and outputs values being computed by the BRAM "program" are stored—it is the workspace.

Recall that the input arguments to the transition function δ are the current state pc and the input value from the alphabet Σ. Since the input can only be a symbol from Σ, a maximum of $\log(|\Sigma|)$ bits are needed to store this value [3]. Therefore, for convenience,

[3] Unless specified otherwise, all logarithms written as log are to the base two

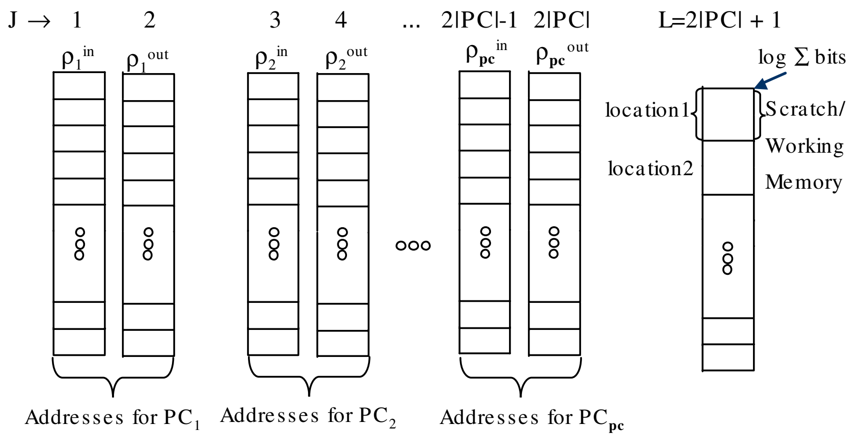

J → 1 2 3 4 ... 2|PC|-1 2|PC| L=2|PC| + 1
ρ_1^{in} ρ_1^{out} ρ_2^{in} ρ_2^{out} ρ_{pc}^{in} ρ_{pc}^{out}
log Σ bits
location1
location2
Scratch/ Working Memory
Addresses for PC1
Addresses for PC2
Addresses for PCpc

Fig. 3. The Memory Structure of the BRAM

each M_I will be partitioned into "locations" where *location* L_J for $J \geq 1$ is made up of $\log(|\Sigma|)$ *constituent* cells; let $s = (\log|\Sigma|(J-1))$. Then $L_J =< m_{s+1} \ldots m_{(s+\log|\Sigma|)} >$

The Memory Access Unit The *value at a location* L_J is the concatenation of the values in its constituent cells. Since the value of a location, when defined, is a natural number from the range $\{1, 2, \ldots |\Sigma|\}$, it is determined by a binary interpretation, of the concatenation of symbols from the set $\{0, 1\}$. If one of the values associated with any of the cells in L_j is ϕ, then the value of this location is undefined.

A VALUE in MEMORY is a function from $(N^+ \times N^+)$ into the set $\Sigma \cup \{\phi\}$ defined as follows:

1. If $1 \leq I \leq 2|PC|$ namely if index I corresponds to a register, then VALUE $(I, J) = \S$ where § is the value at the J^{th} location of M_I.
2. If $I = M_L$, VALUE $(M_L, J) = \S$ is the value at the J^{th} location (L_J) of **M**.

The function VALUE that is implemented through the *memory access unit* of a BRAM yields the value associated with the J^{th} location in one of the registers in R or at the location L_J from **M** depending on the value of I.

The *address* in register ρ_i^{in} (or ρ_i^{out}) is the unique non-negative integer whose value is u, where the *address* is represented in unary. The MAU is a function that uses these (pair of) addresses as an argument. *Throughout the rest of this paper we will consider an alphabet where* $|\Sigma| = 2$*, and this unary representation will across locations L be used to analyze the energy advantages of probabilistic computing. Alternate alphabet sizes as well as binary representations will be the topic of future study as discussed briefly in Section 6.*

We define functions *read* and *write* with addresses as their domain. Thus, using conventions inspired by Turing machines as originally defined [28], *read* $(I,$ LOCATION $)$

and *write* $(\Sigma, I, \text{LOCATION})$ are respectively used to read the value or (over)write the values associated with the constituent cells of location L in M_I. The MAU is the union of the *read* and *write* functions. It will be used to evaluate the transition function as explained in Section 2.3 below.

2.3 The Computation of a BRAM

Building on the elements introduced above, we will now introduce the operational behavior of a BRAM . Given an arbitrary BRAM program P, initially, all computations start in the START state. All the registers and the memory cells are initialized from the set $\{0, 1, \phi\}$. It is convenient to define the operation of the BRAM inductively as follows. The START state transitions to, without loss of generality, state pc_1 at which point the computation starts where the concatenation of the cells in M_L is interpreted as a number in unary representation and is referred to as the input I to P. Now, state pc_1 is said to be the *current state*. More generally, let pc_l be the current state. In state $pc_l \in PC$, the transition function is evaluated.

The input to the transition function is a symbol from Σ, which is accessed using $\sigma = \mathit{read}(\mathbf{M}, \text{LOCATION})$, where LOCATION is the address stored in unary in ρ_l^{in}. These notions are illustrated in Figure 4.

Continuing with the evaluation of the transition function $\delta(pc_l, \sigma)$ yields the *next state* $pc_{l'}$ which then becomes the current state. Furthermore, the output symbol σ' is written (using *write*) into the LOCATION whose address is stored in register ρ_l^{out}. The computation *halts* whenever $pc_{l'} \equiv \text{STOP}$.

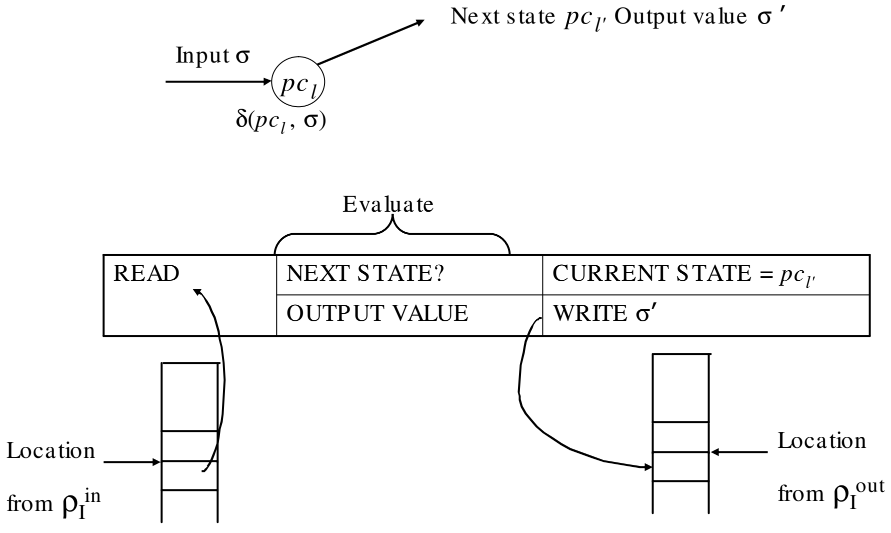

Next state $pc_{l'}$ Output value σ'
Input σ
pc_l
$\delta(pc_l, \sigma)$
Evaluate
READ
NEXT STATE?
CURRENT STATE = $pc_{l'}$
OUTPUT VALUE
WRITE σ'
Location from ρ_I^{in}
Location from ρ_I^{out}

Fig. 4. Illustrating the Evaluation of the Transition Function

More generally, a *computation* C is the *sequence* of state transitions $s_1 \equiv \text{START} \rightarrow s_2 \rightarrow \ldots \rightarrow \ldots s_\pi \equiv \text{STOP}$ where $\pi \in N^+$. Given $s_i \equiv pc_l$ to be the current state during the evaluation of the transition, $s_{i+1} \equiv pc_{l'}$ is the next state. A computation is *legal* if and only if, during the evaluation of the transition function with $s_i \equiv pc_l$ as the current state, the addresses in ρ_{in}^j and ρ_{out}^j as well as the input determined by evaluating *read*, are all defined. The computation is *illegal* otherwise. In the sequel, we will be concerned with legal computations only and will, for convenience, refer to them simply as computations.

Fact 1. All computations of a BRAM program P with input I are the identical. Formally, given any two computations $C_q = s_1 \equiv \text{START} \rightarrow s_2 \rightarrow \ldots \rightarrow \ldots s_\pi \equiv \text{STOP}$ and $C_r = \hat{s}_1 \equiv \text{START} \rightarrow \hat{s}_2 \rightarrow \ldots \rightarrow \ldots \hat{s}_\pi \equiv \text{STOP}$ generated by program P with inputs I and $\hat{I}$, $s_j = \hat{s}_j$ whenever $I \equiv \hat{I}$

2.4 The Randomized BRAM or RABRAM

A RABRAM is identical to a BRAM in all aspects except that the transition from the current state to the next state occurs probabilistically. There are alternate forms of defining the particular approach through which this probabilistic transition is introduced into the formulation of a RABRAM . In our formulation, the transition function δ (representing a BRANCH) from Section 2 is extended to a transition function δ_r. Let P be the open interval $(\frac{1}{2}, 1)$. Now, δ_r:$((PC \cup \{\text{START }\}) \times \Sigma) \rightarrow (PC \cup Q - \{\text{START }\}) \times \Sigma \times P$ is the transition function. Let pc_j and pc_k be the possible successors of pc_i, where $j = k$ is allowed. For $\frac{1}{2} \leq p_i \in P \leq 1$, whenever $\delta_r(pc_i, \sigma = 0) = (pc_j, \sigma' \in \Sigma, p_i \in P)$, δ transitions from pc_i into the *next-state* pc_j with $\sigma = 0$ as input, and with σ' as output, with probability p_i, and to state pc_k with $\bar{\sigma}'$ as output with probability $(1 - p_i)$. Note that $\bar{\sigma}'$ is the symbol from Σ that is output whenever δ_r yields a transition from state pc_i to nest state pc_k. The transition function with $\sigma = 1$ can be defined accordingly. Let us refer to this branch instruction as a randomized BRANCH with probability parameter p_i.

Two clarifications are in order here. First, in the current definition of the RABRAM, for simplicity, the probability parameter is defined to be independent of the input symbol. This is consistent with the definition of randomized algorithms where the source of the random bits is *not* biased based on the input. However, the definition of the RABRAM does allow for different probability parameters for different states in *PC*. Secondly, the probability parameter p_i only ranges from $\frac{1}{2}$ to 1 because any transition of the form $\delta_r(pc_i, \sigma = 0) = (pc_j, \sigma' \in \Sigma, p_i < \frac{1}{2})$ can be rewritten as $\delta_r(pc_i, \sigma = 0) = (pc_k, \bar{\sigma}' \in \Sigma, \frac{1}{2} \leq p_i' \leq 1)$ where $p_1' = 1 - p_i$.

While this is a formal model, an equivalent representation can couple the deterministic transition function with a coin-toss and base the outcome on the input symbol and the outcome of the coin-toss based on a previously specified probability distribution on the set of successors of pc_i. This notion of a randomized transition function is shown in Figure 5.

Input Symbol	Current State	Deterministic Transition	Randomized Transition
0	pc_i	pc_j	pc_j with probability p_i pc_k with probability $(1 - p_i)$
1	pc_i	pc_k	pc_k with probability p_i pc_j with probability $(1 - p_i)$

Fig. 5. The Transition Function of a Randomized BRANCH where the output symbols are uniquely associated with the next state

3 Logical Work as a Measure of Complexity

Recall that a computation of the BRAM program P with an input I, is the sequence of state transitions $C \equiv s_1 \equiv \text{START} \longrightarrow s_2 \longrightarrow \ldots \longrightarrow \ldots s_\pi \equiv \text{STOP}$, as defined in Section 2.3; recall that $s_i \equiv pc_l \in PC$ for some index l. The *deterministic logical work* D done by computation C is

$$D(C) = \prod_{i=1}^{\pi} F(s_i)$$

where $F(s_i)$ is the fanout of state s_i. State s_i represents a BRANCH (or where appropriate, a randomized BRANCH) instruction whenever $F(s_i) > 1$. Let $\mathbf{I}_n$ denote the set of all inputs to P of length n bits. The *logical work* done by the BRAM program P with length n is

$$L(P, n) \equiv MAX(D(C, I | I \in \mathbf{I}_n))$$

In earlier work [18, 20], this author established the following theorem

Theorem 1. *The energy consumed in evaluating the transition function in the context a state pc_i of any* BRAM *program P is at least $\kappa T \ln(F(s_i))$ Joules.*

It follows that

Corollary 1. *For a deterministic* BRAM *computation, the energy consumed by a program P with inputs of length* n *is no less than $\kappa T \ln(L(P, n))$ Joules.*

Proof. Immediate from Theorem 1, the fact that energy is additive and the identity

$$\ln\left(\prod_{i=1}^{\pi} F(s_i)\right) = \sum_{i=1}^{\pi} \ln(F(s_i))$$

□

Also, we recall (from [18, 20]) that

Theorem 2. *The energy consumed in evaluating the transition function in the context a state pc_i of any* RABRAM *program P_R can be as low as $\kappa T \ln(F(s_i)p)$ Joules, where p is the probability parameter.*

We note that in the context of computations realized through a RABRAM each state transition is probabilistic. Therefore, since Fact 1 does not hold in this case, different computations are possible with the same input. Thus, given a fixed input I there exits a *set of computations* $\mathbf{C}$ where each computation $C \in \mathbf{C}$ has a probability q_c of occurrence. Thus, in this case, the notion of logical work has to be modified where the *MAX* function is applied over the *expected* length of the probabilistic computations from $\mathbf{C}$. We define the *expected logical work* for an input I to be the expected sum of the fan-outs of the states visited by the family of computations corresponding to I.

$$R(\mathbf{C}, I) = \sum_{\forall C \in \mathbf{C}} q_c D(C) p^{\pi}$$

Corollary 2. *For a* RABRAM *program P_R, the expected energy consumed with input I can be as low as*

$$\kappa T \sum_{\forall C \in \mathbf{C}} q_c \ln(D(C) p^{\pi})$$

The *Probabilistic logical work* of a RABRAM program P_R is defined to be

$$LR(P_R, n) = MAX[R(\mathbf{C}, I \,|\, I \in \mathbf{I}_n)]$$

As before, $\mathbf{I}_n$ is the set of all valid inputs of length n.

In the general case, the *logical work* of an algorithm L might consist of deterministic as well as probabilistic logical work components. Given the nature of asymptotic analysis and in this hybrid case, it will be sufficient only to consider the dominant term.

4 Design and Analysis of Algorithms in the BRAM and the RABRAM

In this Section, we will demonstrate the use of the models described in the previous sections, in the context of analyzing energy savings using probabilistic algorithms. Our problem of choice will be the *distinct vector problem* or DVP for short, introduced in Section 4.1. In Section 4.2, we will outline a probabilistic algorithm for solving this problem, whose energy consumed is provably better than *any* deterministic algorithm for solving the DVP in the BRAM model. To establish this result, we have to prove a lowerbound on the deterministic logical work needed to solve this problem using *any* (deterministic) BRAM algorithm (claimed in Section 5 and proved in this author's earlier work [19]). While the central ideas and some of the details are presented in the sequel, complete proofs and other implementation specifics that are easy to verify, will be included in a full-version of the paper. Thus, this paper should be viewed as an extended abstract. For notational succinctness, we will use the symbols i, j, k, l, m and n in a new context throughout the rest of this paper, where this reuse will not cause any ambiguity.

4.1 The Distinct Vector Problem DVP

Informally, the DVP is defined to be the problem of determining whether a given $n-$ tuple, defined on the set of symbols $\{0,1\}$, is distinct from an $n-$ tuple which has the symbol 1 in all of its positions. Formally,

Input: a *vector* $T \equiv < t_1, t_2, \dots, t_n >$ where $T \in \{0,1\}^n$ such that

1. $t_i = 1$ in all n positions or,
2. $t_i = 0$ for $\log n$ values of i where $1 \leq i \leq n$.

Also, a string of length $COUNT$ is given where the length is a design parameter which will determine the probability of correctness. In our case $COUNT = c \log n$ for an appropriately chosen constant c.

Question: Is $t_i = 1$ for $1 \leq i \leq n$?

Let **T** denote the set of all possible inputs to the DVP. A RABRAM program P_R solves the DVP with probability **p** provided, given an input as defined above, it halts with a symbol 1 denoting an answer of "yes" to the above question, in a designated *output* cell in memory whenever $t_i = 1$ for $1 \leq i \leq n$ in T, and with the symbol 0 otherwise, denoting the answer "no" with probability **p**. For convenience, we will refer to the value in the cell *output* to be the *output bit.*

4.2 A Probabilistic Algorithm PROBDVP

The proposed (probabilistic) RABRAM algorithm for solving the DVP , PROBDVP is now described below and shown in Figure 6. This algorithm revolves around a single crucial step, described and analyzed as the *value-amplification technique* below; it is a probabilistic "test" for detecting whether or not a given value t_i is 1 (or equivalently 0). For convenience, all of the algorithms in this section are specified as a type of "pseudo-code" and not in the more detailed RABRAM (or BRAM) notation. To the extent necessary, we will specify the extensions for converting these specifications into full fledged RABRAM programs.

Value Amplification and Voting Without loss of generality, let n be a power of 2. Value amplification is the following simple algorithm (Figure 7) performed on each of the positions of T using an auxiliary two-dimensional array $X[i, j]$ of size $n \times (c \log(n))$ where $1 \leq i \leq n$, initialized to zero. Here c is a suitably chosen constant. Throughout, let $p = 1 - \frac{\varepsilon \log n}{n}$, where $\varepsilon < \frac{1}{c}$.

Let $1 \leq i \leq n$ and recall that in a RABRAM , each step has an associated probability parameter p, the probability that the outcome of a probabilistic BRANCH is correct, for example in the comparison in Step-**1** (Figure 7. This comparison is implemented as a BRANCH with fan-out two and with probability parameter p as stated before. In this

```
Algorithm: PROBDVP

1.For $i = 1$ to $n$ Do
2.    If $t_i'$==1 Then/*probability parameter $(p)$*/
3.        continue
4.    Else    /* $t_i' \neq 1$ */
5.        VALAMP
6.        MAJORITY
7.        If $output = 0$
8.            halt
9.        Else
10.           continue
11.       End If
12.   End If
13.End For
```

Fig. 6. The probabilistic DVP algorithm based on value amplification

case, the following basic fact about the probability of correctness of value amplification can be derived.

Let us now consider applying value amplification using the value of element t_i, which by definition is either 0 or 1. Also, upon completion of value amplification with value t_i its outputs or "amplified values" are recorded in locations or memory cells $X_i \equiv X[i,1], X[i,2], \cdots, X[i, c \log n]$. Let x_i denote the number of these locations with value identical to t_i. For example if $t_i = 0$, x_i denotes the number of elements or cells in X_i with value zero upon completion of the value amplification step with t_i as input, and vice-versa.

```
Algorithm: VALAMP

1. While $j < COUNT \equiv c\log(n)$
2.     If $t_i = 0$
3.         Then set $X[i,j] = 1$;
4.     End If
5.     $j = j + 1$
6. End While
```

Fig. 7. Algorithm to perform value amplification

We will first state two useful facts

Fact 2. (Chernoff bound [16]) Let $Y_1, Y_2, \cdots, Y_l$ be independent Poisson trials such that, for $1 \le i \le l$, $Pr[Y_i = 1] = p_i$, where $0 < p_i < 1$. Then for $Y = \sum_{i=1}^{l} Y_i$, $\mu = E[Y] = \sum_{i=1}^{l} p_i$ and any $\delta > 0$,

$$Pr[Y > (1+\delta)\mu] < \left[\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right]^{\mu}$$

As above, let $Pr[Y > m]$ denote the probability that event Y occurs at least m times out of l independent trials, where p denotes the probability of occurrence of Y in any one trial. That is $p_i = p_j$ for $1 \le i \le l$. Similarly we define $Pr[Y' > m]$ where Y' is associated with probability p'. The following useful fact is immediate.

Fact 3. $Pr[Y > m] > Pr[Y' > m]$ whenever $p > p'$ for all $l > 0$.
Using these facts, we can now prove

Lemma 1. *In any single invocation of algorithm* VALAMP *with input value t_i the (error) probability that the number of elements x_i is less than $\frac{c}{2}\log n$,*

$$Pr[x_i < \frac{c}{2}\log n] \le \frac{1}{n^{\hat{c}}}$$

for all $n \ge 2$ and constants $\hat{c}$ and ε that are design parameters.

Proof. Let $\hat{p} = 1 - \frac{1}{c'}$ for c' a constant, and also let $y_i = (c\log n - x_i)$. The expected value μ of $y_i = \frac{c}{c'}\log n$. Now,

$$Pr[y_i > \frac{c}{2}\log n] = Pr[y_i > (1+\delta)\frac{c}{c'}\log n] \tag{1}$$

$$\text{for } (1+\delta) = \frac{c'}{2}$$

From Fact 2

$$Pr[y_i > (1+\delta)\log n] \le \left[\frac{e^{\frac{c'}{2}-1}}{\left(\frac{c'}{2}\right)^{\left(\frac{c'}{2}\right)}}\right]^{\frac{c}{c'}\log n} \tag{2}$$

$$\le \frac{1}{n^{\hat{c}}} \tag{3}$$

for a constant $\hat{c} > 1$, with an appropriate choice of c and c'.

Now, the specified probability of error $(1-p) = \frac{\varepsilon \log n}{n} < \frac{1}{c'}$ for any $n \ge 2$ and $\varepsilon < \frac{2}{c'}$ for $n = 2$, and hence for any $n > 2$ since $\frac{n}{\log n}$ is an increasing function of n. With this observation we are done from Fact 3 since the bound in inequality (2) will serve as an underestimate to the error probability. □

Intuitively, the basic idea behind value amplification is that whenever the value at a position t_i is probabilistically tested and found it be 0, its value is “suspect”. In this case, algorithm VALAMP performs repeated independent tests on the same bit, and sets the *output* based on the majority of the tests. The result of an individual test will henceforth be referred to as a *vote*. The *democratic-voting* algorithm entitled MAJORITY (Figure 8) accomplishes the goal of counting the total number of votes and setting the *output* bit appropriately, based on a simple majority.

Algorithm: MAJORITY

```
1. If the number of entries with the symbol 1
in $X[i,j]$: $1 \leq j \leq c\log(n)$ is greater than $\frac{c}{2}\log(n)$
2.         Then set output to 0 and terminate.
3. Else
4.         set output to 1 and terminate.
5. End If
```

Fig. 8. Algorithm to count majority vote

Based on analysis identical to that developed in the proof of Lemma 1 above, it is a simple exercise to verify that

Corollary 3. *The probability that upon termination, the democratic-voting algorithm* MAJORITY *is correct is* $p_m \geq \left(1 - \frac{1}{n^{\hat{c}}}\right)$.

Implementation-Issues While a RABRAM implementation of algorithm VALAMP is immediate—we will defer the discussion of implementing the iteration control to the end of this section—it is interesting to consider an implementation of algorithm MAJORITY in some detail. Specifically, we will propose two alternate approaches for completeness.

The first approach “combines” an implementation of algorithm MAJORITY with that of algorithm VALAMP. In this case, each primitive BRANCH instruction will serve two purposes, as shown in Figure 9. First it is used to test the bit in the input array T. Second, whenever the input bit from T is 0 and hence algorithm VALAMP is invoked (by the overall algorithm referred to as algorithm PROBDVP), it is used to increment the *POINTER* value that points to the location where the “next bit” is to be written in array $X[i,j]$. The input value *COUNT* is used as a unary counter, realized as a single BRANCH instruction which will terminate the iteration (Figure 9) when a zero is detected in *COUNT* after $c\log n$ iterations.

Using this algorithm, it is easily verified inductively that for any value of i, there always exists a $\hat{j} \geq 0$ such that $X[i, j' \leq \hat{j}] \equiv 1$ and $X[i, j'' > \hat{j}] \equiv 0$. Informally, all the

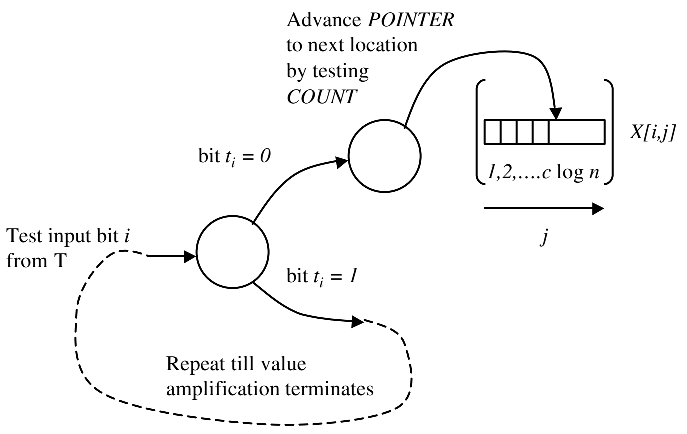


Fig. 9. A combined RABRAM implementation of algorithm VALAMP and algorithm MAJORITY

"1 entries" in row i of $X[i,j]$ will be contiguous as shown in Figure 10. Thus, in this implementation, the analysis from Lemma 1 immediately implies that upon completion of value amplification, the "majority" test can be replaced by testing whether the location $X[i, 1 + \left(\frac{c}{2\log n}\right)]$ has value 1,

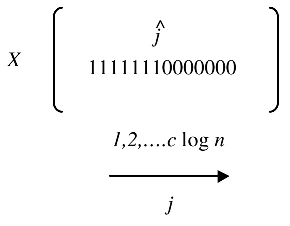


Fig. 10. The structure of any row i in X

Corollary 4. *When considering input t_i and after applying algorithm* VALAMP, *algorithm* MAJORITY *sets output to* 1 *if and only if* $t_i = 1$ *with probability* $p_v \geq \left(1 - \frac{1}{n^{\hat{c}}}\right)$ *for some constant $\hat{c}$, a design parameter.*

For convenience, this "one bit" test as well as the unary counter implementation shown in Figure 9 above are presented to be deterministic BRAM implementations. They can however be realized as probabilistic BRANCH instructions in a RABRAM by "bootstrapping" on the value amplification notion.

The second approach to realizing algorithms VALAMP and MAJORITY is to consider separate implementations. Again, considering algorithm MAJORITY, we note that a straightforward approach to determining the majority will involve a tree-structured computation with $c\log n$ leaves, where the output of the root is a value of 1 if and only if the number of entries in row i of array X designated X_i is greater than $\frac{c}{2}\log n$. Each "node" of the tree represents some constant number, say r, of tests, such that a value of 1 is recorded iff both of its children are associated with a value of 1. The value of r is chosen so that with $(1-p) = \frac{\varepsilon\log n}{n}$, we have an overall error probability bounded above by $\frac{1}{n^{\hat{c}}}$ as in Corollary 3.

5 Analysis of Algorithm PROBDVP

Considering Algorithm PROBDVP from Section 4.2 above, we note that the counter that controls the iteration over the input vector can be implemented through a single branch for each position till the "end-of-array" symbol is detected in the input vector T to determine termination of the entire algorithm. To reiterate, for convenience, we assume that the branch instruction associated with this test is deterministic. We again note in passing that the value amplification algorithm used above can in fact be used to replace this deterministic test by a probabilistic test. For completeness we recall that each test in the "for" loop specified as Step-**2** in algorithm PROBDVP (Figure 6) is implemented using a probabilistic branch, with an associated error probability of $p = \frac{\varepsilon\log(n)}{n}$.

With this as background, let α and β denote the number of times that that the branch used to realize Step-**2** is executed incorrectly—α denotes the number of times that an event of type **A**, wherein the input vector has a value 0 and the branch determined it to be erroneously 1 occurs, whereas β denotes the number of times that an event of type **B** wherein an input value of 1 is erroneously determined to be 0. Similarly, λ denotes the number of times this step is executed correctly, and the corresponding event is said to be of type Λ. Λ_0 denotes an event of type Λ with an input symbol of 0, with Λ_1 defined similarly.

Using the Chernoff bound (from Fact 2), once again, and using analysis similar that that used in the proof of Lemma 1, we can show that

Lemma 2. *The probability that α or β is greater than $\varepsilon c'\log n$ where $\varepsilon \le \frac{1}{c}$ and $1 \le c' \le c$, is bound above by $\frac{1}{n^{\hat{c}}}$, for constants c, c' and ε which are design parameters.*

The Expected Logical Work Done by Algorithm PROBDVP Using the above development as background, we are now ready to analyze the expected logical work LR done by algorithm PROBDVP.

Fact 4. The value amplification in Step-**5** of Algorithm PROBDVP is invoked iff events of type **B** or Λ_0 occur in Step-**2**.

From the above fact, we have

Theorem 3. *The expected logical work LR (PROBDVP$, n)$ is $(2n + \log^k n)$ for some positive constant $k > 2$.*

Proof. The logical work during each invocation of value amplification is trivially $c \log n$, for some constant c. Furthermore, the number of such invocations due to events of type Λ_0 is bound above by $\log n$ from the definition of the input to the DVP. From Lemma 2 the probability that the number of invocations of value amplification caused by events of type **B** will exceed $\varepsilon c' \log n$ is no more than $\frac{1}{n^{\hat{c}}}$. This in turn implies an expected logical work of $c \log^2 n$ from events of type **B**. Also the number of steps (BRANCH) and hence logical work that can be caused by events of type Λ_1 as well as type **A** is cumulatively bound above by n. By noting that a unary implementation of a counter in Step-1 of the algorithm can be realized using n branches, we have $LR\,(\text{PROBDVP}, n) \leq 2n + \log^k n$ with a suitable constant k. □

Expected Energy Savings Using Algorithm PROBDVP In earlier work [19], we have shown that the $L(P,n)$ of *any deterministic* BRAM algorithm P for solving the DVP problem is bound below by $2n - \log n + 1$. From this lowerbound, from Theorem 3, from Corollaries 1 and 2, we can claim that

Theorem 4. *The expected savings in energy in Joules using Algorithm* PROBDVP *over any deterministic algorithm for solving the* DVP *grows as* $\Theta\left(n \log\left(\frac{n}{n - \varepsilon \log n}\right)\right)$ *Joules, for constant* $0 < \varepsilon < 1$*, and therefore is monotone increasing in n.*

5.1 Probability of Error of Algorithm PROBDVP

We note that errors can occur either due to events of type **A** or of type **B**. For these types of events, we have:

Theorem 5. *The probability that Algorithm* PROBDVP *will terminate correctly is at least* $\mathbf{p} = (1 - \frac{1}{n^{\mathbf{c}}})$ *for* $\mathbf{c}$ *a constant and a design parameter.*

Proof. We note that an incorrect termination occurs if and only if the input vector had the value 1 in all of its positions and upon termination, the value of *output* was 0 (events of type **B**), which we will refer to as *Case-1*, and vice-versa, (events of type **A**) which we will refer to as *Case-2*.

Case-1: From algorithm PROBDVP, let us consider events from set **B**. From Lemma 1, the probability of any one of these events setting $output = 0, \leq \frac{1}{n^{\hat{c}}}$. Since there are a total of n positions and hence a maximum of n such events, this probability is trivially bound above by $\frac{1}{n^{\hat{c}-1}}$ and therefore we are done with $\mathbf{c} = \hat{c} - 1$. (We note that using the bound on β from Lemma 2 will yield a better estimate of $\mathbf{c}$.)

Case-2: Since in this case there exists a $t_i = 0$, by the definition of the DVP , there exist $\log n$ positions such that $t_j = 0$ at every one of these positions; let ξ_0 denote the set of all such indices j. Considering events of type **A**, we note from Lemma 2 that the probability that $\alpha < \log n$ is bound below by $\left(1 - \frac{1}{n^{\hat{c}}}\right)$. Therefore, there exists one index $j' \in \xi_0$ such that upon execution of Step-**2** of algorithm PROBDVP with $t_{j'}$ as input, the resulting event is not of type **A** with probability $p' \geq 1 - \frac{1}{n^{\hat{c}}}$. Therefore with $t_{j'}$ as input, Step-**5** and Step-**6** of algorithm PROBDVP would have been executed with probability p' and we are done from Lemma 1. $\square$

6 Remarks and Conclusions

With the ever increasing emphasis on the energy consumed by computers and the need to minimize it, our goal here is to develop a framework and a supporting complexity (theory) that is technology independent, intending to parallel classical computational complexity theory developed in the context of running-time and space (see Papadimitriou [21] for details concerning the classical theory of computational complexity). The work presented here is one approach towards accomplishing this goal wherein energy is the figure of merit—as opposed to traditional time or space. In this context, the measure of complexity introduced here and referred to as logical work serves to provide an abstract, albeit representative measure of the energy consumed. Thus, while analyzing an algorithm as demonstrated in the context of the DVP for example, the logical work can be the figure of merit that one seeks to improve, which is then easily "translated" to deduce energy gains, as demonstrated in Section 4.

The particular formulation presented here affords a clear separation of concerns between the energy behavior of an algorithm across the logical and physical levels, by introducing an "abstract" estimate of energy-consuming behavior through logical work, which is independent of particular physical implementations. Subsequently, we provide specific translations from the domain of logical work, through idealized physical models of computing as summarized in Section 3, into the domain of energy.

Thus, using this framework, the specific *physical* devices that implement the computing elements can be changed, without perturbing the algorithm framework affecting the design and analysis where the latter constitute the *logical* components of our framework. Furthermore, our particular choice of an idealized physical device abstracts away dependencies on specific technologies, but nevertheless exposes the logical components of the framework to the inherent limits to energy consumed—specifically the idealized physical devices used here are based on statistical thermodynamics building on the historic work of Maxwell [29], Boltzmann [2], and Gibbs [6], rather than being based in a specific physical domain such as transistors of a particular feature size for example. Furthermore, these idealized devices consume energy as they compute and once energy is consumed, the complexity measure of logical work *irreversibly* charges for this expenditure; this is in contrast with the *reversible* style of computing (see Feynman [5] for a survey) which allows energy consumed to be recovered allowing, in theory, computations to be realized with zero energy consumption.

From a utilitarian perspective of course, any framework such as that introduced in this paper is "only as useful as the results that it can help achieve." In this context, the central thesis established in this paper that is used to validate the value of this framework is: probabilistic techniques and algorithms—or, as referred to in the introduction and in keeping with the theme of this symposium, "probabilistic proofs"—yield expected energy savings, when compared to their deterministic counterparts.

Several directions of inquiry suggest themselves, given that the energy behavior of algorithms in general and probabilistic algorithms in particular remains a largely unchartered domain. While deferring the cataloging of such "open questions" — computations on finite-fields suggest themselves immediately as candidates for study— including those aimed at developing an energy-based complexity theory to a future publication, we will briefly comment on some of the more immediate questions here.

An obvious first step is to consider other interesting as well as more meaningful candidate problems for demonstrating possible energy savings achieved through probabilistic algorithms or proofs. In this regard, results similar to those presented for the DVP in Section 4 have been derived by this author for *string matching*. The classical probabilistic algorithm for solving this problem based on *fingerprinting* is due to Karp and Rabin [8]. Commenting on the specifics briefly, our energy savings are derived by extending the notion of value amplification (from Section 4.2) rather than through the use of the Karp-Rabin fingerprints. It will be of interest to analyze fingerprinting from an energy perspective using the framework provided by the BRAM and the RABRAM models, and to systematically compare the power and scope of this technique with that of value amplification. Specifically, the error probability of value amplification is higher than the error probability achieved through fingerprinting. The first interesting question is to determine whether value amplification can yield the same error probability as fingerprinting does. Assuming that the probabilities of error are different, it will be interesting to determine whether energy can be used to separate the complexity of fingerprinting from value amplification, even though both of then would yield algorithms that run in $O(n)$.

All of the results presented in this paper were using unary representations of numbers, as opposed to the more natural binary representation. This choice was deliberate in that in a model such as a BRAM , the particular choice of representation has an impact on the asymptotic energy behavior, and our interest in this (first) work is to understand the energy behavior at the most elementary level possible. A basic question to consider in this regard is that of implementing a binary *counter* and its accompanying arithmetic, and comparing it to the unary design used to implement iteration in realizing Algorithm PROBDVP for example.

A direction of inquiry that is only hinted at here but not elaborated upon, is the implication of this work to novel physical computing devices that are probabilistic. As the analysis in Section 4 demonstrated, such implicit randomization in the (abstract) device can lead to energy improvements, even asymptotically. To reiterate, these improvements are not due to faster running times that probabilistic algorithms might yield, but follow from the following fundamental reason: using the idealized physical devices (from [18, 20]) referred to above, a physical interpretation of randomization allows computation to be realized with higher thermodynamic-entropy (or Boltzmann-entropy) which is a physical quantity, thus yielding energy savings. Pursuing realizations of such devices and validating them in the context of implementing probabilistic algorithms promises to be a particularly interesting direction for inquiry, which is being collaboratively pursued [17]. Intuitively, a physical interpretation of probabilistic computing can be viewed as "merely" riding the wave of naturally occurring thermodynamic phenomena, which are best characterized statistically.

Acknowledgments

This work is supported in part by DARPA under seedling contract #F30602-02-2-0124

References

[1] M. Blum. A machine-independent theory of the complexity of recursive functions. *Journal of the ACM*, 14(2):322–326, 1967.

[2] L. Boltzmann. Further studies on the equilibrium distribution of heat energy among gas molecules. *Viennese Reports*, Oct. 1872.

[3] G. J. Chaitin and J. T. Schwartz. A note on monte carlo primality tests and algorithmic information theory. *Communications on Pure and Applied Mathematics*, 31:521–527, 1978.

[4] S. A. Cook. The complexity of theorem proving procedures. *The Third Annual ACM Symposium on the Theory of Computing*, pages 151–158, 1971.

[5] R. Feynman. *Feynman Lectures on Computation*. Addison-Wesley Publishing Company, 1996.

[6] J. W. Gibbs. On the equilibrium of heterogeneous substances. *Transactions of the Connecticut Academy*, 2:108–248, 1876.

[7] J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117, 1965.

[8] R. Karp and M. Rabin. Efficient randomized pattern matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, 1987.

[9] R. M. Karp. *Reducibility among combinatorial problems*. Plenum Press New York, 1972.

[10] R. M. Karp. Probabilistic analysis of partitioning algorithms for the traveling-salesman problem in the plane. *Mathematics of Operations Research,(USA)*, 2(3):209–224, Aug. 1977.

[11] H. Leff and A. F. Rex. *Maxwell's demon: Entropy, information, computing.* Princeton University Press, Princeton, N. J., 1990.

[12] L. A. Levin. Universal sorting problems. *Problems of Information Transmission*, 9:265–266, 1973.

[13] Z. Manna. Properties of programs and the first-order predicate calculus. *Journal of the ACM*, 16(2):244–255, 1969.

[14] Z. Manna. *Mathematical theory of computation*. McGraw-Hill, 1974.

[15] J. D. Meindl. Low power microelectronics: Retrospect and prospect. *Proceedings of IEEE*, pages 619–635, Apr. 1995.

[16] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[17] K. Palem, S. Cheemalavagu, and P. Korkmaz. The physical representation of probabilistic bits (pbits) and the energy consumption of randomized switching. *CREST Technical report*, June 2003.

[18] K. V. Palem. Thermodynamics of randomized computing: A discipline for energy aware algorithm design and analysis. Technical Report GIT-CC-02-56, Georgia Institute of Technology, Nov. 2002.

[19] K. V. Palem. Energy aware computation: From algorithms and thermodynamics to randomized (semiconductor) devices. Technical Report GIT-CC-03-10, Georgia Institute of Technology, Feb. 2003.

[20] K. V. Palem. Energy aware computing through randomized switching. Technical Report GIT-CC-03-16, Georgia Institute of Technology, May 2003.

[21] C. Papadimitriou. *Computational Complexity*. Addison-Wesley Publishing Company, 1994.

[22] H. Putnam. Models and reality. *Journal of Symbolic Logic*, XLV:464–482, 1980.

[23] M. O. Rabin. Degree of difficulty of computing a function and a partial ordering of recursive sets. Technical Report 2, Hebrew University, Israel, 1960.

[24] M. O. Rabin. Probabilistic algorithm for testing primality. *Journal of Number Theory*, 12:128–138, 1980.

[25] M. O. Rabin and D. S. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):115–125, 1959.

[26] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27:701–717, 1980.

[27] K.-U. Stein. Noise-induced error rate as limiting factor for energy per operation in digital ics. *IEEE Journal of Solid-State Circuits*, SC-31(5), 1977.

[28] A. Turing. On computable numbers, with an application to the entscheidungsproblem. In *Proceedings of the London Mathematics Society*, number 42 in 2, 1936.

[29] H. von Baeyer. *Maxwell's Demon: Why warmth disperses and time passes*. Random House, 1998.

[30] von Neumann J. *Mathematical foundations of quantum mechanics*. Princeton University Press, Princeton, N. J., 1955.

[31] A. Whitehead and B. Russell. *Principia Mathematica*. Cambridge University Press, 1913.