Algorithmic cooling and scalable NMR quantum computers

P. Oscar Boykin*, Tal Mor†‡§, Vwani Roychowdhury*, Farrokh Vatan*¶, and Rutger Vrijen‖

*Electrical Engineering Department, University of California, Los Angeles, CA 90095; †Electrical Engineering Department, College of Judea and Samaria, Ariel 44837, Israel; ‡Computer Science Department, Technion, Technion City, Haifa 32000, Israel; ¶Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, CA 91109; and ‖RAS Computer Analysis Laboratory, Sun Microsystems, Menlo Park, CA 94025

Communicated by N. David Mermin, Cornell University, Ithaca, NY, December 3, 2001 (received for review June 19, 2001)

We present here algorithmic cooling (via polarization heat bath)—a powerful method for obtaining a large number of highly polarized spins in liquid nuclear-spin systems at finite temperature. Given that spin-half states represent (quantum) bits, algorithmic cooling cleans dirty bits beyond the Shannon's bound on data compression, by using a set of rapidly thermal-relaxing bits. Such auxiliary bits could be implemented by using spins that rapidly get into thermal equilibrium with the environment, e.g., electron spins. Interestingly, the interaction with the environment, usually a most undesired interaction, is used here to our benefit, allowing a cooling mechanism. Cooling spins to a very low temperature without cooling the environment could lead to a breakthrough in NMR experiments, and our "spin-refrigerating" method suggests that this is possible. The scaling of NMR ensemble computers is currently one of the main obstacles to building larger-scale quantum computing devices, and our spin-refrigerating method suggests that this problem can be resolved.

1. Introduction

Ensemble computing is based on a model composed of a macroscopic number of computers, where the same set of operations is performed simultaneously on all computers. The concept of ensemble computing became very important recently, because NMR quantum computers (1–4) perform ensemble computing. NMR quantum computing has already succeeded in performing complex operations involving up to 7–8 quantum bits (qubits), and therefore NMR quantum computers are currently the most successful quantum computing devices.

In NMR quantum computing, each computer is represented by a single molecule, and the qubits of the computer are represented by the nuclear spins embedded in a single molecule. A macroscopic number of identical molecules is available in a bulk system, and these molecules act as many computers performing the same computation in parallel. To perform a desired computation, the same sequence of external pulses is applied to all molecules/computers. Finally, a measurement of the state of a single qubit is performed by averaging over all computers/molecules to read out the output on a particular bit on all computers. Because of the use of a macroscopic number of molecules, the output is a noticeable magnetic signal. It has been shown that almost all known quantum algorithms designed for the usual single-computer model can be adapted to be implemented on ensemble computers (5), and in particular, these ensemble computers can perform fast factorization of large numbers (6) and fast database search (7).

Unfortunately, the widespread belief is that even though ensemble quantum computation is a powerful scheme for demonstrating fundamental quantum phenomena, it is not scalable (see, for instance refs. 8–10). In particular, in the current approaches to ensemble computing, identifying the state of the computer requires sensing signals with signal-to-noise ratios that are exponentially small in n, the number of qubits in the system. We refer to this well-known problem as the scaling problem. The origin of the *scaling problem* is explained in the following.

The initial state of each qubit, when averaged over all computers (a macroscopic number), is highly mixed, with only a small bias towards the zero state. At thermal equilibrium, the state is

$$\rho_{\varepsilon_0} = \begin{pmatrix} (1+\varepsilon_0)/2 & 0 \\ 0 & (1-\varepsilon_0)/2 \end{pmatrix}, \qquad [1]$$

where the initial bias, ε_0, is mainly determined by the magnetic field and the temperature but also depends on the structure and electronic configurations of the molecule. For an ideal system, one has $\varepsilon_0 = \varepsilon_{\text{perfect}} = 1$ leading to $\rho_{\varepsilon_{\text{perfect}}} = |0\rangle\langle 0| = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 0 \end{smallmatrix}\right)$, meaning that the state is $|0\rangle$ with probability one, and it is $|1\rangle$ with probability zero. For a totally mixed system, $\varepsilon_0 = 0$, hence the probabilities of $|0\rangle$ and $|1\rangle$ are both equal to half. We also define $\delta_0 = (1 - \varepsilon_0)/2$ to be the initial error probability. Typically, ε_0 is around 10^{-6} for the liquid NMR systems in use (1–4) and can probably be improved (increased) a great deal in the near future. Especially promising directions are the use of liquid crystal NMR for quantum computing (11) and the use of a SWAP operation for the nuclear spin and the electron spin known as the electron nuclear double-resonance (ENDOR) technique (12). These techniques and others (e.g., SWAP with hyperpolarized Xenon, optical pumping, Overhauser effect) might yield much-improved polarization biases and maybe even sufficiently good polarization biases in the far future. It seems that a combination of these strategies with the *algorithmic cooling* presented here might yield optimal results.

The state of an n qubit system in the ideal case is $\rho_{\text{ideal}}^{\{n\}} = |0_n\rangle\langle 0_n|$ with $|0_n\rangle = |0\rangle \otimes |0\rangle \otimes \ldots \otimes |0\rangle$ (a tensor product of n single qubit states). In general, the initial state of an n-qubit liquid NMR system can be represented as a tensor product of states of the individual qubits:

$$\rho_{\text{init}}^{\{n\}} = \rho_{\varepsilon_0} \otimes \rho_{\varepsilon_0} \otimes \cdots \otimes \rho_{\varepsilon_0}. \qquad [2]$$

This state can also be written as $\Sigma_{i=0}^{2^n-1} P_i |i\rangle\langle i|$, a mixture of all states $|i\rangle$—the basis vectors of the system, and i (for n qubits) is an n bit binary string, e.g., for two qubits, $P_{00} = (1 + \varepsilon_0)^2/4$. In fact, the initial bias is not the same on each qubit,** but as long as the differences between these biases are small, we can ignore this fact in our analysis. The analysis we do later on is correct if we replace all these slightly different initial biases by their minimum value and call this value ε_0.

Currently, researchers use the so-called "pseudo-pure state (PPS)" technique (1–4) to perform computations with such highly mixed initial states. In this technique, the initial mixed density matrix is transformed to a state

Abbreviations: PPS, pseudo-pure state; RRTR, rapidly reaching thermal relaxation; BCS, basic compression subroutine; qubit, quantum bit.

§To whom correspondence should be addressed. E-mail: talmo@cs.technion.ac.il.

**Individual addressing of qubits requires a slightly different bias for each one, which is easily achievable in practice.

Downloaded by guest on December 2, 2020

$$\rho^{\{n\}}_{\mathrm{PPS}} \equiv (1-p)\mathcal{I} + p|\psi\rangle\langle\psi|, \tag*{[3]}$$

which is a mixture of the totally mixed state $\mathcal{I} = (1/2^n)I_{2^n}$ (with I_{2^n}, the identity matrix of order 2^n), and a pure state $|\psi\rangle$ initially set to be $|0_n\rangle$ in our case. Such a state is called a PPS. Unitary operations then leave the totally mixed state unchanged but do affect the pure state part, to perform the desired computation via entanglement of the pure part (which we refer to as "pseudo-entanglement"). Finally, the state of the ensemble-computer is measured. If the probability p of the pure state is not too small, then the pure part of the state yields the expectation value for each qubit, an outcome sufficient for performing quantum computing as powerful as standard (nonensemble) quantum computing (5). Unfortunately, in all existing PPS methods,

$$p = \frac{(1+\varepsilon_0)^n - 1}{2^n - 1} < 2\left(\frac{1+\varepsilon_0}{2}\right)^n, \tag*{[4]}$$

and hence, p scales exponentially badly with n (the number of computation qubits), leading to an exponentially small signal-to-noise ratio. As a result, an exponential number of computers (molecules) are required to read the signal. With ε_0 in the range $10^{-6} - 10^{-1}$, one might still hope to obtain a 20-qubit computer, because then p (approximately $10^{-5} - 10^{-6}$) can still lead to an observed signal when an Avogadro number of computers are used. But one cannot hope to go beyond a 50-qubit computer, because then p is approximately $10^{-13} - 10^{-15}$, which is smaller than the standard deviation in reading the result (and, even with perfect devices, the signal cannot be read).

The exponential advantage of quantum computers over classical ones (6) is totally lost in these NMR computing devices, because an exponential number of molecules/computers is required for the computation, and therefore the scaling problem must be resolved to achieve any useful NMR quantum computing. This scaling problem (plus the assumption that quantum computing requires entanglement and cannot rely on pseudo-entanglement) has led several researchers to suggest that current NMR quantum computers are no more than classical simulators of quantum computers (10).[††]

The first important step in resolving the scaling problem is to understand that the scaling problem is not an inherent characteristic of ensemble computers but is an artifact of existing PPS methods. In fact, the original highly mixed state contains a great deal of information, and this can be seen by rotating each qubit separately and finally measuring the qubits. However, the existing methods of transforming the highly mixed state into the PPS cause the scaling problem by losing information on purpose. Furthermore, it is important to mention that for any n, there is a range of bias, ε, not close to zero, where the currently existing methods for creating PPS work just fine. To be in that range, the state of each qubit must be almost pure: $\varepsilon = 1 - 2\delta$, where δ, the error probability, satisfies $\delta \ll 1$ (actually $\delta \approx 0.2$ is already useful). Then p scales well:

$$p = \frac{(1+\varepsilon)^n - 1}{2^n - 1} \approx (1-\delta)^n. \tag*{[5]}$$

As long as $\delta \approx O(1/n)$, the probability p is sufficiently large for all practical purposes, thus much larger ns can still be used. Furthermore, any n can be used if one can control δ as a function of n. The PPS technique, the loss of information, and the scaling problem are presented in *Appendix A*, which is published as supporting information on the PNAS web site, www.pnas.org.

Instead of converting the initial state (Eq. **2**) to a PPS (Eq. **3**), we perform a "purification" transformation that takes a subset, m (with $m < n$), of the qubits to a final state of the form

$$\rho^{\{m\}}_{\mathrm{final}} = \rho_{\varepsilon_{\mathrm{des}}} \otimes \rho_{\varepsilon_{\mathrm{des}}} \otimes \cdots \otimes \rho_{\varepsilon_{\mathrm{des}}}, \tag*{[6]}$$

where $\varepsilon_{\mathrm{des}}$ is some desired bias close enough to 1. This state with a higher bias can then be transformed into a scalable PPS, $\rho^{\{m\}}_{\mathrm{PPS}}$. For example, we shall demonstrate how to achieve (via algorithmic cooling) $\delta \approx 0.2$, which allows m in the range of 20–50 qubits, and $\delta \approx 0.04$, which allows m in the range of 50–200 qubits.

In this paper, we present a purification process that uses concepts from information theory (data compression) and from thermodynamics (heat bath, thermal relaxation) and that resolves the scaling problem. Our "information-theoretic" purification is totally classical, hence the density matrices are treated as classical probability distributions, and no explicit quantum effects are taken into consideration. In an earlier work, Schulman and Vazirani (13) already demonstrated novel compression-based (and not PPS-based) alternative NMR computing, which does not suffer from the scaling problem. Their invention, "(reversible) molecular scale heat engines," is based on information theoretic tools, and it leads to Eq. **6**. However, the Shannon bound on the purification ability of reversible data compression prevents purifying any reasonable fraction of bits for small values of ε_0: $m \approx [(\varepsilon_0^2)/(2\ln 2)]n$ (see Section 2), meaning that thousands of bits are required in order to obtain one or a few purified bits (with a reasonable probability of success). More explicitly, any entropy-preserving purification scheme cannot currently be useful for NMR computation.

We present here the first cooling scheme that goes *beyond the Shannon bound*, an algorithmic cooling via polarization heat-bath, or in short, algorithmic cooling. This cooling scheme, presented in Section 3, purifies a large fraction of the bits initially set in a highly mixed state and hence resolves the scaling problem. Algorithmic cooling can bypass the Shannon bound, because it does not preserve entropy of the system but removes entropy into a heat bath at a temperature β_0. To pump entropy into the polarization heat bath, algorithmic cooling demands the existence and mutual processing of two types of qubits:[‡‡] computation bits and bits that rapidly reach thermal relaxation (RRTR bits). The computation bits are assumed to have a very long relaxation time, $\mathcal{T}_{\mathrm{comput-bits}}$, and they are used for the computation; the RRTR bits are assumed to have a much shorter relaxation time, $\mathcal{T}_{\mathrm{RRTR}}$, hence they rapidly get into thermal equilibrium with the environment (a heat bath) at a temperature of β_0. Because the RRTR bits are defined via their spin (to be 0 or 1), the heat bath is actually a spin-polarization heat bath. In our algorithmic cooling, a standard compression is performed on the computation bits, purifying (cooling) some while concentrating the entropy (heating) of the others, to heat them above β_0. Then the hotter bits are replaced with the RRTR bits, which are at the heat-bath temperature β_0, resulting in an overall cooling of the system. Repeating the process many times via a recursive algorithm, any final close-to-zero "temperature" (that is, any final bias), can in principle be achieved.

Algorithmic cooling provides a new challenge for experimentalists, because such processing of two types of quantum bits (two different spin systems) is highly nontrivial. The currently existing experimental technologies and the new "experimental chal-

[††] Actually, the important contribution of ref. 10 is the result that in some neighborhood of the totally mixed state, all states are separable; hence, some pseudo-entangled state (a state for which the pseudo pure part is entangled) contains no entanglement. But ref. 10 does not prove (and does not claim to prove) that current NMR quantum computers do not perform quantum computation. We, in contrast, conjecture that the PPS technique and the work of ref. 10 form the first step in proving that quantum computing without entanglement is possible.

[‡‡] We refer to these qubits as bits, because no quantum effects are used in the cooling process.

Downloaded by guest on December 2, 2020

lenge" of combining them to perform algorithmic cooling are explained further in *Appendix B*, which is published as supporting information on the PNAS web site. Conclusions and some open questions for further research are provided in Section 4.

2. Information Theory, the Basic Compression Subroutine, and Purification Levels

2.1. Shannon's Bound. Let us briefly describe the purification problem from an information theoretic perspective. There exists a straightforward correspondence between the initial state of our n qubit system and a probability distribution of all n bit binary strings, where the probability of each string i is given by the term P_i, the probability of the state $|i\rangle$ in the mixed state ρ_{init}^{n} described by Eq. **2**. A loss-less compression of a random binary string that is distributed as stated above has been well studied (see any textbook on information theory, e.g., refs. 14 and 15). In an optimal compression scheme, all the randomness (and hence the entropy) of the bit string is transferred to $n - m$ bits, while with extremely high probability leaving m bits in a known deterministic state, say the string 0. The entropy H of the entire system is $H(\text{system}) = nH(\text{single-bit}) = nH(1/2 + \varepsilon_0/2)$ with $H(P) \equiv -P \log_2 P - (1 - P) \log_2(1 - P)$ measured in bits. Any loss-less compression scheme preserves the entropy H of the entire system, hence one can apply Shannon's source coding bound on m to get $m \leq n[1 - H(1/2 + \varepsilon_0/2)]$. Simple leading-order calculation shows that m is bounded by (approximately) $[(\varepsilon_0^2)/(2 \ln 2)]n$ for small values of the initial bias ε_0, and in a practical compression scenario, this can be achieved if a large enough string (large enough n) is used. Schulman and Vazirani (13) were the first to use information theoretic tools for solving the scaling problem, and they also demonstrated how to get very close to the Shannon bound, once n is very large. We consider here a bias of 0.01 and a bias of 0.1, and with these numbers, the Schulman–Vazirani compression cannot be useful in practice and cannot help in achieving NMR computing with more than 20 qubits in the foreseeable future. In fact, any entropy-preserving purification scheme cannot be useful for NMR computation in the near future.

We suggest here an entropy-nonpreserving purification. Our purification, algorithmic cooling, has some common properties with the entropy-preserving purification, such as the basic compression subroutine and the purification levels. These are therefore described in the following.

2.2. Basic Compression Subroutine and Purification Levels. The basic compression subroutine (BCS) is the simplest purification procedure used to convert a mixture with a particular bias ε_j to one with a higher bias ε_{j+1} but fewer bits. We take pairs of bits and check whether they are the same or different. One bit (the "supervisor") retains the information of whether they were the same. If they were the same, then we keep the other bit (the "adjusted" bit), and we say it is purified. In this way, we increase the bias or push the bits to a higher purification level. To realize this operation, we use a Controlled-NOT (CNOT) transformation on a control bit (c) and a target bit (t): $0_c0_t \rightarrow 0_c0_t$, $0_c1_t \rightarrow 0_c1_t$, $1_c0_t \rightarrow 1_c1_t$, $1_c1_t \rightarrow 1_c0_t$. After the transformation, the target bit holds the information regarding the identity of the initial states of the two bits. Hence, without being measured, this target bit can then be used as a supervisor bit for the next steps: If the target bit is 0 after the CNOT operation between a pair of bits, then the pair had the same initial value, and the control bit of the CNOT (the adjusted bit) is retained because it is purified, otherwise they were different and the adjusted bit is thrown away because it got dirtier. In both cases, the supervisor bit has a reduced bias (increased entropy), hence it is thrown away. However, before being thrown away, the supervisor bit is used as a control bit for a SWAP operation: if it has the value "0," then it SWAPs the corresponding adjusted bit at the head of the array (say to the left), and if it is "1," it leaves the corresponding adjusted bit at its current place. In either case, the supervisor bit is then SWAPped to the right of the array. (Note that we use here a hybrid of English and symbol languages to describe an operation such as SWAP.) As a result, at the end of the BCS, all purified bits are at the first locations at the left side of the array, the dirty adjusted bits are at the center, and the supervisor bits are at the right side of the array. Thus the dirty adjusted bits and the supervisor bits can be thrown away (or just ignored).

Starting a particular BCS on an even number n_j of bits with a bias ε_j at the end of the BCS, there are (on average) $n_{j+1} = [(1 + \varepsilon_j^2)/4]\ n_j = (\varepsilon_j/2\varepsilon_{j+1})n_j$ purified bits with a new bias $\varepsilon_{j+1} = 2\varepsilon_j/(1 + \varepsilon_j^2)$. The number of computation steps in one such BCS is $T_{\text{BCS}} < (2n_j)(n_j/2) = n_j^2$. The new length, the new bias, and the number of steps are calculated in detail in the supporting information in *Appendix C* (www.pnas.org).

A full compression scheme (from ε_0 to $\varepsilon_{j_{\text{final}}}$) can be built by repeating the BCS $j_{\text{final}} \equiv j_f$ times, each BCS acting on bits purified by the previous BCS. See *Appendix C* for more details. When only the repeated BCS is performed, the resulting average final length of the string is $m = n_{j_f} = (\varepsilon_{j_f-1}/2\varepsilon_{j_f})n_{j_f-1} = (\varepsilon_0/2^{j_f}\varepsilon_{j_f})n_0$, so that the initial required number of bits, n_0, is huge. Because the final desired bias, for the purpose of obtaining PPS after the cooling, is not very close to 1 (see examples for actual numbers in *Appendix C*, which is published as supporting information in the PNAS web site), then even a ratio (n/m) of $2^{j_f}\varepsilon_{j_f}/\varepsilon_0 = 8 * 6.666 \approx 50$ could lead to $m = 50$ bits via the reversible data compression, with $n \approx 2{,}500$ bits to start with.

By the way, better compression schemes can be designed (13), which approach the Shannon's bound $n = 1.3963m/\varepsilon_0^2$. For our purpose, which is to achieve a "cooling via polarization heat-bath" algorithm, this simplest compression scheme, the BCS, is sufficient, and much larger ratios of m/n can be obtained.

3. Algorithmic Cooling via Polarization Heat-Bath

3.1. Going Beyond Shannon's Bound. To go beyond Shannon's bound, we assume that we have a thermal bath of partially polarized bits with a bias ε_0. As a more realistic way to implement that assumption in a physical system, we assume that we have RRTR bits, in addition to the computation bits. These RRTR bits, by interaction with the environment at some constant temperature β_0, rapidly return to the fixed initial distribution with bias of ε_0 (a reset operation). Hence the environment acts as a polarization heat bath.

In one application of the BCS on bits at a bias of ε_j, some fraction f (satisfying $1/4 \leq f \leq 1/2$) is purified to the next level, ε_{j+1}, whereas the other bits have increased entropy. The supervisor bits are left with a reduced bias of ε_j^2, and the adjusted bits that failed to be purified are changed to a bias $\varepsilon = 0$, that is, they now remain with full entropy.

To make use of the heat bath for removing entropy, we swap a dirtier bit with an RRTR bit at bias ε_0 and do not use this RRTR bit until it thermalizes back to ε_0. We refer to this operation as a single "cooling" step.[§§] In a nearest-neighbor gate array model, which is the appropriate model for NMR quantum computing, we can much improve the efficiency of the cooling by assuming that each computation bit has an RRTR bit as its neighbor (imagine a ladder built of a line of computation bits and a line of RRTR bits). Then k cooling steps (a single "cooling operation") can be done in a single time step by replacing k dirty bits with k RRTR bits in parallel.

[§§] Actually, adjusted bits that failed to purify are always dirtier than the RRTR bits, but supervisor bits are dirtier only as long as $\varepsilon_j^2 < \varepsilon_0$. Therefore the CUT of the adjusted bits that failed to purify, $\mathscr{C}$ (which is explained in the next subsection) is the main "engine" that cools the NMR system at all stages of the protocol.

Downloaded by guest on December 2, 2020

By applying many BCSs and cooling operations in a recursive way, spins can be refrigerated to any temperature, via algorithmic cooling.

3.2. Cooling Algorithm. For the sake of simplicity, we design an algorithm whereby BCSs are always applied to blocks of exactly m bits (thus, m is some prechosen even constant), and which finally (via repeated cooling operations and BCSs) provide m bits at a bias ε_{j_f}. Any BCS is applied onto an array of m bits at a bias ε_j, all purified bits are pushed to the head of the array (say, to the left), all supervisor bits are swapped to the back of the array (say, to the right), and all unpurified adjusted bits (which actually became much dirtier) are kept in their place. After one such BCS, the $m/2$ bits at the right have bias of ε_j^2, the purified bits at the left have a bias ε_{j+1}, and to their right there are bits with a bias zero. Note that the boundary between the purified adjusted bits and the dirtier adjusted bits is known only by its expected value $\langle \mathscr{L}_{j+1} \rangle = [(1 + \varepsilon_j^2)/4]\, m$ (where the number of purified bits, $\mathscr{L}_{j+1}$, with the new bias ε_{j+1} is different on each molecule). By repeating this set of operations ℓ times (as explained in the following paragraphs), with $\ell \geq 4$, an expected value $\langle \mathscr{L}_{j+1}^{\ell} \rangle = [\ell(1 + \varepsilon_j^2)/4]\, m$ of bits is obtained, from which the first m bits are defined as the output bits with ε_{j+1}, and the rest are ignored. If an additional purification is now performed, only these first m bits are considered as the input for that purification. We refer to ℓ as the "cooling depth" of the cooling algorithm.[¶¶]

The algorithm is written recursively with purification steps M_j, where the jth purification step corresponds to purifying an initial array of N_j bits into a set of m bits at a bias level of ε_j, via repeated compression/cooling operations described as follows: In one purification step M_0, we wish to obtain m bits with a bias ε_0. To achieve this, we SWAP m bits with m RRTR bits, which results in m cooling steps performed in parallel (one cooling operation). The number of bits required for M_0 is $N_0 = m$. In one purification step M_{j+1} (with $j \geq 0$), we wish to obtain m bits with a bias ε_{j+1}. To achieve this goal, we apply ℓ purification steps M_j, each followed by a BCS applied to exactly m bits at a bias ε_j. First, M_j is applied onto N_j bits, yielding an output of m bits at a bias ε_j. A BCS is then applied onto these bits, yielding a string of expected length $\langle \mathscr{L}_{j+1}^{1} \rangle = [(1 + \varepsilon_j^2)/4]\, m$ bits purified to a bias ε_{j+1} and pushed all the way to the left. At the end of that BCS, all the $m/2$ supervisor bits are located at positions $m/2 + 1$ until m. Then M_j is applied again onto an array of N_j bits, starting at position $m/2 + 1$. This time, all BCSs within this second application of M_j push the bits to the relative first location of that M_j array, which is the location $m/2 + 1$ of the entire string. (In the case of $j = 0$, of course, there are no BCSs within M_0.) At the end of that second M_j application, a BCS is applied to m bits at a bias ε_j (at locations $m/2 + 1$ till $m/2 + m$), purifying them to ε_{j+1}. The purified bits are pushed all the way to the left, leading to a string of expected length $\langle \mathscr{L}_{j+1}^{2} \rangle = 2[(1 + \varepsilon_j^2)/4]\, m$. At the end of that BCS, all the $m/2$ supervisor bits are located at positions $m + 1$ until $3m/2$. Then M_j is again applied onto an array of N_j bits, starting at position $m + 1$. All BCSs within this third application of M_j push the bits to the relative first location of that M_j array (the location $m + 1$ of the entire string). At the end of that third M_j application, a BCS is applied to m bits at a bias ε_j (at locations $m + 1$ until $m + m$), purifying them to ε_{j+1}, and the purified bits are pushed all the way to the left. This combined M_j and BCS is repeated ℓ times, yielding $\langle \mathscr{L}_{j+1}^{\ell} \rangle = \ell[(1 + \varepsilon_j^2)/4]\, m$ bits purified to ε_{j+1}. For $\ell \geq 4$, we are promised that $\langle \mathscr{L}_{j+1}^{\ell} \rangle > m$, and a CUT operation, $\mathscr{C}_{j+1}$, defines the first m bits to be the output of M_{j+1}.

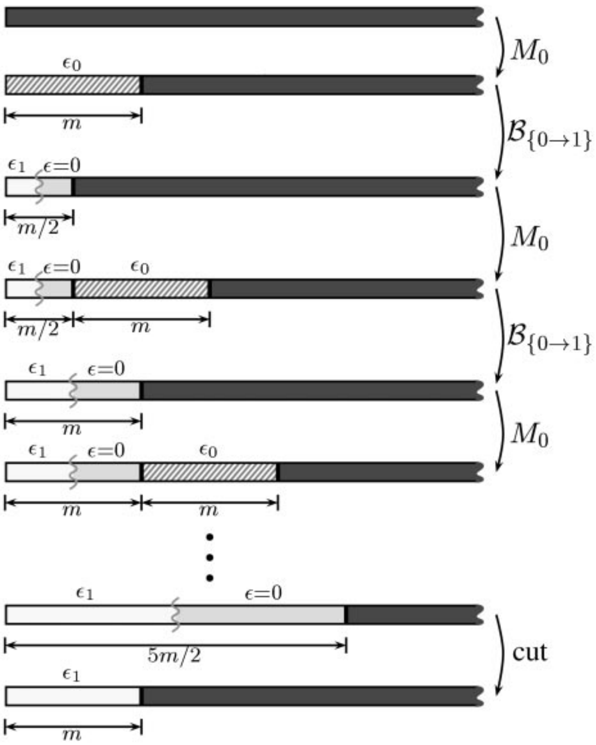

Fig. 1. The purification step M_1.

The total number of bits used in M_{j+1} is $N_{j+1} = (\ell - 1)m/2 + N_j$ bits, where the N_j bits are the ones used at the last M_j step, and the $(\ell - 1)m/2$ bits are the ones previously kept. The output of M_{j+1} is defined as the first m bits, and in case M_{j+2} is to be performed, these m bits are its input. Let the total number of operations applied at the jth purification step, M_j, be represented as T_j. Note that $T_0 = 1$, meaning that m bits are SWAPped with RRTR bits in parallel. Each application of the BCS has a time complexity smaller than m^2 for a near-neighbor connected model, $T_{\mathrm{BCS}} < (2n_j)(n_j/2) = n_j^2$ (see *Appendix C*). When the kth cooling is done (with $k \in \{1, \ldots, \ell\}$), the number of additional steps required to (control-)SWAP the adjusted bit at the top of the array is less than $2(k - 1)m$. Thus we get $T_{j+1} < \Sigma_{k=1}^{\ell}[(2\{k - 1\}m + 2m)(m/2) + T_j]$. Hence, for all j,

$$T_{j+1} < \sum_{k=1}^{\ell} [km^2 + T_j] = \frac{\ell(\ell + 1)}{2} m^2 + \ell T_j. \qquad [7]$$

The purification steps M_1 and M_2 can be obtained by following the general description of M_{j+1}. For clarity, M_1 is described in Fig. 1, M_2 is described in Fig. 2, and both M_1 and M_2 are described in words in *Appendix D*, which is published as supporting information on the PNAS web site. For the entire protocol, we choose j_{final} and perform M_{j_f} starting with $N_{j_f} \equiv n$ bits, and we end up with m bits.

To emphasize the recursive structure of this algorithm, we use the following notations: $[\mathscr{B}_{\{(k-1)\to k\}}]$, the BCS procedure purifying from ε_{k-1} to ε_k (followed by moving the purified bits to the relevant starting point); $[\mathscr{S}]$, SWAP m bits with the RRTR; $[\mathscr{C}_j]$, CUT, keep the first m bits from the starting point of the subarray of the bits with a bias ε_j. Then, $M_0 \equiv \mathscr{S}$, and for $j \in \{1, \ldots, j_f\}$,

[¶¶] An algorithm with ℓ replaced by ℓ_j (different numbers of repetitions, depending on the bias-level j) could have some advantages but will not be as easy to analyze.

Downloaded by guest on December 2, 2020

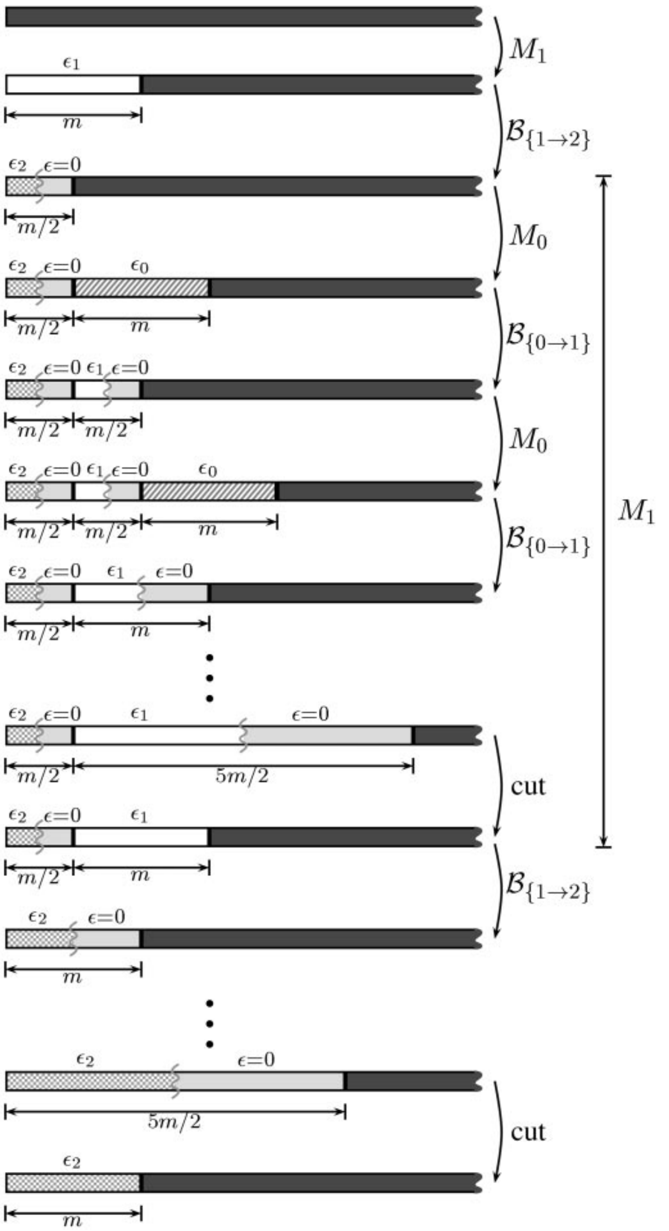
M_1
ϵ_1
m
$\mathcal{B}_{\{1\to 2\}}$
ϵ_2 $\epsilon{=}0$
$m/2$
M_0
ϵ_0
$\mathcal{B}_{\{0\to 1\}}$
ϵ_1
M_0
M_1
$\mathcal{B}_{\{0\to 1\}}$
$5m/2$
cut
$\mathcal{B}_{\{1\to 2\}}$
cut

Fig. 2. The purification step M_2. The details of the operations of M_1 on the second level are shown.

$$M_j = \mathcal{C}_j \underbrace{\mathcal{B}_{\{(j-1)\to j\}} M_{j-1} \cdots \mathcal{B}_{\{(j-1)\to j\}} M_{j-1}}_{\ell \text{ times}} \qquad [8]$$

is the recursive formula describing our algorithm.

A full cooling algorithm is M_{j_f}, performed starting at location $\mu = 0$. A pseudo-code for the complete algorithm is shown in Fig. 3 in *Appendix D* in the supporting information. For any choice of ε_{des}, one can calculate the required (minimal) j_f such that $\varepsilon_{j_f} \geq \varepsilon_{\text{des}}$, and then m bits (cooled as desired) are obtained by calling the procedure COOLING (j_{final}, 1, ℓ, m), where $\ell \geq 4$. We actually use $\ell \geq 5$ in the rest of the paper (although $\ell = 4$ is sufficient when the block's size m is very large) to make sure that the probability of a successful process does not become too small. (The analysis done in ref. 13 considers the case in which m goes to infinity, but the analysis does not consider the probability of success of the purification in the case where m does not go asymptotically to infinity. However, to motivate experiments in this direction, one must consider finite and not too large blocks, with a size that shall potentially be accessible to experimentalists in the near future. In our algorithm, the case of $\ell = 4$ does not provide a reasonable probability of success for the cooling process, but $\ell = 5$ does.)

3.3. Algorithmic Complexity and Error Bound 3.3.1. *Time and space complexity of the algorithm.* We now calculate $N_f = n$, the number of bits we must start with to get m purified bits with bias ε_{j_f}. We have seen that $N_0 = m$ and $N_j = [(\ell - 1)/2]\, m + N_{j-1}$, leading to $N_j = (\{[\ell{-}1]/2\}j + 1)\, m$, and in particular

$$N_{j_f} = \left(\frac{\ell - 1}{2}\, j_{\text{final}} + 1\right) m. \qquad [9]$$

Thus, to obtain m bits, we start with $n = cm$ bits where $c = [(\ell - 1)/2]\, j_{\text{final}} + 1$ is a constant depending on the purity we wish to achieve (that is, on j_{final}) and on the probability of success we wish to achieve (that is, on ℓ). For reasonable choices, j_f in the range 3–7 and ℓ in the range 5–7, we see that c is in the range 7–22. To compare with the Shannon's bound, where the constant goes as $1/\varepsilon_0^2$, one can show that here c is a function of $1/\log \varepsilon_0$.

As we have seen in Section 3.2, the total number of operations applied at the jth purification step, M_j, satisfies $T_j < [\ell(\ell + 1)]/2\; m^2 + \ell T_{j-1}$. Writing $d = m^2[\ell(\ell + 1)]/2$, the recursive formula leads to $T_{j_f} < \ell^{j_f} T_0 + d \sum_{i=0}^{j_f-1} \ell^k = \ell^{j_f} + d[\ell^{j_f} - 1]/[\ell - 1]$. After some manipulations, we get

$$T_{j_f} < m^2 \ell^{j_f + 1}. \qquad [10]$$

This bound is not tight, and a tighter bound can be obtained. It is also important to mention that in a standard gate-array model (and even in a "qubits in a cavity" model), in which SWAPs are given almost for free, an order of m instead of m^2 is obtained.

Let the relaxation time $\mathcal{T}_1$ of the computation bits be called $\mathcal{T}_{\text{comput-bits}}$ and the relaxation time $\mathcal{T}_1$ for the RRTR bits be called $\mathcal{T}_{\text{RRTR}}$. Note that the dephasing time, $\mathcal{T}_2$, of the computation bits is irrelevant for our algorithm and plays a role only after the cooling is done.

With a short-term goal in mind (see Table 1 in *Appendix C* in the supporting information), we obtain that $m = 20$ can be achieved (for $\ell = 5$) with $\varepsilon_0 = 0.01, j_f = 6, T_{j_f} < 3.1 \times 10^7$ steps, and $n = 260$ bits, or with $\varepsilon_0 = 0.1, j_f = 3, T_{j_f} < 250{,}000$ steps, and $n = 140$ bits. Increasing m to 50 multiplies the initial length by only 2.5 and multiplies the time steps by 6.25. Thus, this more interesting goal can be achieved with $\varepsilon_0 = 0.01, j_f = 6, T_{j_f} < 1.9 \times 10^8$ steps, and $n = 650$ bits, or with $\varepsilon_0 = 0.1, j_f = 3, T_{j_f} < 1.56 \times 10^6$ steps, and $n = 350$ bits (compared with $n \approx 2{,}500$ bits for the reversible repeated-BCS algorithm).

Concentrating on the case of $j_f = 3$ and $\varepsilon_0 = 0.1$, let us calculate explicitly the timing demands. For $m = 20$ bits, we see that the switching time $\mathcal{T}_{\text{switch}}$ must satisfy 250,000 $\mathcal{T}_{\text{switch}} \ll \mathcal{T}_{\text{comput-bits}}$ to allow completion of the purification before the system spontaneously relaxes. Then, with $m^2 = 400$ time steps for each BCS operation, the relaxation time for the RRTR bits must satisfy $\mathcal{T}_{\text{RRTR}} \ll 400\ \mathcal{T}_{\text{switch}}$, if we want the RRTR bits to be ready when we need them the next time. As a result, a ratio of $\mathcal{T}_{\text{comput-bits}} \gg 625\ \mathcal{T}_{\text{RRTR}}$ is required in that case. The more interesting case of $m = 50$ demands $1.56 \times 10^6\ \mathcal{T}_{\text{switch}} \ll \mathcal{T}_{\text{comput-bits}}$, $\mathcal{T}_{\text{RRTR}} \ll 2500\ \mathcal{T}_{\text{switch}}$, and $\mathcal{T}_{\text{comput-bits}} \gg 625\ \mathcal{T}_{\text{RRTR}}$. Note that choosing $\ell = 6$ increases the size by a factor of 5/4 and the time by a factor of $6^4/5^4 \approx 2$. We discuss the possibility of obtaining these numbers in an actual experiment in *Appendix B* in the supporting information.

3.3.2. Estimation of error. Because the cooling algorithm is probabilistic, and so far we have considered only the expected number of purified bits, we need to make sure that in practice the

Downloaded by guest on December 2, 2020

actual number of bits obtained is larger than m with a high probability. This is especially important when one wants to deal with relatively small numbers of bits. We recall that the random variable $\mathcal{L}_j^k$ is the number of bits purified to ε_j, after the kth round of purification step M_{j-1}, each followed by $\mathcal{B}_{\{(j-1)\rightarrow j\}}$. Hence, prior to the CUT $\mathcal{C}_j$ we have $\mathcal{L}_j^\ell$ bits with bias ε_j, where the expected value $\langle \mathcal{L}_j^\ell \rangle = \ell[(1 + \varepsilon_{j-1}^2)/4]\, m > \ell m/4$, and we use $\ell \geq 5$. Of these bits, we keep only the first m qubits, i.e., we keep at most a fraction $4/\ell$ of the average length of the string of desired qubits. Recall also that $\mathcal{L}_j^\ell$ is a sum of independent Bernoulli random variables, and hence one can apply a suitable form of the strong law of large numbers to determine the probability of success, i.e., the probability that $\mathcal{L}_j^\ell \geq m$.

The details of applying a law of large numbers are given in *Appendix E*, which is published as supporting information on the PNAS web site. Here we state only the result. Chernoff's bound implies that the probability of failing to get at least m bits with bias ε_j is

$$\Pr[\mathcal{L}_j^\ell < m] \leq \exp\left(-\frac{1}{2}\left(1-\frac{4}{\ell}\right)^2 \frac{\ell}{4} m\right),$$

which, because of

$$\frac{1}{2}\left(1-\frac{4}{\ell}\right)^2 \frac{\ell}{4} m = \frac{(\ell-4)^2}{8\ell} m,$$

gives

$$\Pr[\mathcal{L}_j^\ell < m] \leq \exp\left(-\frac{(\ell-4)^2}{8\ell} m\right).$$

For the probability of success of the entire algorithm, we have the following conservative lower bound on $\Pr[\mathrm{s}] \equiv Pr$ [success of the algorithm]:

$$\Pr[\mathrm{s}] \geq \left[1-\exp\left(-\frac{(\ell-4)^2}{8\ell} m\right)\right]^{(\ell^{j_f}-1)/(\ell-1)}. \qquad [11]$$

The probability of success is given here for several interesting cases with $j_f = 3$ (and remember that the probability of success increases when m is increased): For $m = 50$ and $\ell = 6$, we get Pr[success of the algorithm] > 0.51. For $m = 50$ and $\ell = 5$, we get Pr[success of the algorithm] $> 2.85 \times 10^{-5}$. This case is of most interest because of the reasonable time scales. Therefore, it is important to mention here that our bound is very conservative because we demanded success in all truncations (see details in *Appendix E*), and this is not really required in practice. For instance, if only $m - 1$ bits are purified to ε_1 in one round of purification, but m bits are purified in the other $\ell - 1$ rounds, then the probability of having m bits at the resulting M_2 process is not zero but is actually very high. Thus, our lower bound presented above should not discourage the belief in the success of this algorithm, because a much higher probability of success is actually expected.

4. Discussion

In this paper, we suggested "algorithmic cooling via polarization heat bath," which removes entropy into the environment and allows compression beyond the Shannon's bound. The algorithmic cooling can solve the scaling problem of NMR quantum computers and can also be used to refrigerate spins to very low temperatures. We explicitly showed how, by using SWAP operations between electron spins and nuclear spins, one can obtain a 50-qubit NMR quantum computer, starting with 350 qubits, and using feasible time scales. Interestingly, the interaction with the environment, usually a most undesired interaction, is used here to our benefit.

Some open questions that are left for further research: (*i*) Are there better and simpler cooling algorithms? (*ii*) Can the above process be performed in a (classical) fault-tolerant way? (*iii*) Can the process be much improved by using more sophisticated compression algorithms? (*iv*) Can the process be combined with a process that resolves the addressing problem? (*v*) Can one achieve sufficiently different thermal relaxation times for the two different spin systems? (*vi*) Can the electron-nuclear spin SWAPs be implemented on the same systems that are used for quantum computing? Finally, the summarizing question is: (*vii*) How far are we from demonstrating experimental algorithmic cooling, and how far are we from using it to yield 20-, 30-, or even 50-qubit quantum computing devices?

The work of P.O.B., T.M., V.R., and F.V. was sponsored in part by Defense Advanced Research Projects Agency (DARPA) Project MDA972-99-1-0017, by the U.S. Army Research Office/DARPA under contract number DAAD19-00-1-0172, and in part by the National Science Foundation under contract number EIA-0113440. The research of F.V. at the Jet Propulsion Laboratory (JPL) was supported in part by a contract with the National Aeronautics and Space Administration. The Revolutionary Computing Technologies Program of JPL's Center for Integrated Space Microsystems supported F.V.'s work. The work of T.M. was supported in part by the Israel Ministry of Defense Research and Technology Unit.

1. Cory, D. G., Fahmy, A. F. & Havel, T. F. (1997) *Proc. Natl. Acad. Sci. USA* **94,** 1634–1639.
2. Gershenfeld, N. A. & Ghuang, I. L. (1997) *Science* **275,** 350–356.
3. Jones, J. A., Mosca, M. & Hansen, R. H. (1998) *Nature (London)* **393,** 344.
4. Knill, E., Laflamme, R., Martinez, R. & Tseng, C.-H. (2000) *Nature (London)* **404,** 368–370.
5. Boykin, P. O., Mor, T., Roychowdhury, V. & Vatan, F. (1999) *Algorithms on Ensemble Quantum Computers* (Los Alamos National Laboratory, Los Alamos, NM), e-print: xxx.lanl.gov/abs/quant-ph/9907067.
6. Shor, P. W. (1997) *SIAM J. Comput.* **26,** 1484–1509.
7. Grover, L. (1996) *Proc. 28th ACM Symp. Theor. Comput.* (ACM Press, New York), 212–219.
8. Warren, W. S. (1997) *Science* **277,** 1688–1689.
9. DiVincenzo, D. P. (1998) *Nature (London)* **393,** 113–114.
10. Braunstein, S. L., Caves, C. M., Jozsa, R., Linden, N., Popescu, S. & Schack, R. (1999) *Phys. Rev. Lett.* **83,** 1054–1057.
11. Yannoni, C. S., Sherwood, M. H., Miller, D. C., Chuang, I. L., Vandersypen, L. M. K. & Kubinec, M. G. (1999) *Appl. Phys. Lett.* **75,** 3563–3565.
12. Kurreck, H., Kirste, B. & Lubitz, W. (1998) *Methods in Stereochemical Analysis,* ed. Marchand, A. P. (VCH, New York).
13. Schulman, L. J. & Vazirani, U. (1999) *Proc. 31st ACM Symp. Theor. Comput.* (ACM Press, New York), pp. 322–329.
14. Ash, R. B. (1990) *Information Theory* (Dover, New York).
15. Cover, T. M. & Thomas, J. A. (1991) *Elements of Information Theory* (Wiley, New York).

Downloaded by guest on December 2, 2020