Bounds on the entanglability of thermal states in liquid-state nuclear magnetic resonance

by

Terri M. Yu

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2003

Author ....
Department of Electrical Engineering and Computer Science
August 22, 2003

Certified by........ ....
Isaac L. Chuang
Associate Professor, Media Laboratory and Department of Physics
Thesis Supervisor

Accepted by......
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Bounds on the entanglability of thermal states in liquid-state nuclear magnetic resonance

by

Terri M. Yu

Submitted to the Department of Electrical Engineering and Computer Science
on August 22, 2003, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Theorists have recently shown that the states used in current nuclear magnetic resonance (NMR) quantum computing experiments are not entangled. Yet it is widely believed that entanglement is a necessary resource in the implementation of quantum algorithms. The apparent contradiction might be resolved by the experimental realization of an entangled NMR state. Designing such an experiment requires us to know whether or not the initial NMR state is *entanglable* – that is, does there exist a unitary transform that entangles the state? This computational and theoretical thesis explores the entanglability of *thermal states* in $N - \alpha$ space where N specifies the number of qubits and α characterizes the polarization of the thermal state. The thermal state is transformed by the Bell unitary $U_{\mathrm{b,s}}$ and the entanglement of the transformed state is measured by negativity. Here we present numerically generated negativity maps of N-α space ($N \leq 12$) and explicit negativity formulas for $U_{\mathrm{b,s}}$-transformed thermal states. We also give a general method that uses the symmetry of a special mixed Bell state family to derive bounds on the entanglement of generic Bell-transformed thermal states. This approach yields analytical bounds on the entanglability of thermal states and gives an upper limit of $N \leq 20,054$ required to entangle a thermal state under ideal experimental conditions.

Thesis Supervisor: Isaac L. Chuang
Title: Associate Professor, Media Laboratory and Department of Physics

Acknowledgments

I wish to thank my advisor, Isaac Chuang, for his patience and thoughtful guidance throughout this project. Much appreciation goes to my fellow collaborators; thanks to Joshua Powell for his dedication and valuable numerical contributions, Andrew Cross for his computational expertise, and Ken Brown for his theoretical insight. I also wish to acknowledge Julia Kempe for pointing out a crucial reference, Matthias Steffen for useful conversations about NMR quantum computing, and Aram Harrow for help on understanding entanglement.

Finally, I would like to express my gratitude to the Quanta Lab members: Jeff Brock, Andrew Cross, Josh Folk, Andrew Houck, Steve Huang, Murali Kota, and Matthias Steffen. Their enthusiasm and sense of humor made it fun to come into work everyday. A special thanks to Josh and Andy who introduced me to their excellent taste in country music. Those upbeat tunes helped to keep my spirits up during the writing of this thesis.

I am indebted to my friends, family, and teachers for their encouragement and aid during my time at MIT. Thank you all, for everything.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Historical background

Entanglement is hidden information that exists as nonlocal correlations between two or more quantum degrees of freedom. The phenomenon was first mentioned in a 1935 article authored by Einstein, Podolsky, and Rosen [EPR35]. They claimed that the existence of entanglement was "unphysical," and therefore quantum mechanics was an incomplete theory. Einstein suggested that quantum mechanics might be superseded by a deterministic model based on hidden local variables. In 1964, Bell presented a rigorous inequality that quantum measurements must satisfy in order for such a hidden local variable theory to hold [Bel65]. He also proved that the predictions of quantum measurement theory did not always satisfy this inequality. An experimental test of Bell's inequality was realized thirty years later. From 1981 to 1982, Aspect and his collaborators observed violations of Bell's inequality in photon pairs [AGR81, AGR82]. Their measurements agreed to within 1% of quantum mechanical predictions [AGR82]. However, Aspect's experiments and similar ones by other groups were criticized for having experimental loopholes [Pea70, Fra85], in particular detector inefficiency[1] and locality difficulties[2]. Experimentalists have ad-

[1]Detector inefficiency is problematic because if the photodetector only records a subset of the actual detection events, it might be the case that the subset happens to give a result that favors quantum mechanics.

[2]The locality difficulty occurs because the experimental results rely on observing each particle in the photon pair in separate detectors at different times. If the two detectors are separated by a

dressed the detection [RKM$^+$01] or locality [WJS$^+$98] loophole separately, but no group has eliminated both simultaneously. Despite these problems, many experiments have confirmed Aspect's work and the general consensus is that entanglement is a real-world phenomenon.

Entanglement remained an intriguing curiosity until the 1990s, which saw the emergence of quantum computation – the study of information processing that can be performed in physical systems which are quantum mechanical in nature. Bennett and his co-workers showed that entangled pairs of particles are useful for information transfer. For instance, quantum teleportation uses an entangled particle pair and two classical bits to move a quantum state between two spatially separated locations [BBC$^+$93]. Moreover, theorists found that a quantum computer is not able only do arbitrary reversible classical operations [Ben80], but it can also execute some operations faster. Shor broke ground in 1994 with a quantum algorithm that could find the prime factors of a number with exponentially fewer steps than the best known classical algorithm [Sho94, Sho97]. Two years later, Grover gave a quantum algorithm capable of searching an unsorted database with square root smaller steps than the most efficient classical algorithm [Gro96].

The theoretical promise of quantum computation was quickly realized in experiment. In 1998, the first quantum algorithms were demonstrated in nuclear magnetic resonance (NMR) experiments, which were conducted by the laboratory groups of Chuang [CVZ$^+$98] and Jones [JM98]. Many more NMR quantum computing experiments followed, including successful implementations of Grover's algorithm [CGK98], quantum teleportation [NKL98], and Shor's algorithm [VSB$^+$01].

1.2 Motivation

What makes quantum computers more powerful than classical computers? The leading candidate is entanglement because it is a feature unique to quantum systems and

space-like distance, then it is possible for the first detector to make a measurement and communicate the result of that measurement (at speed $v \leq c$) to the second detector before it makes its own measurement.

because many quantum algorithms, including Shor's algorithm, seem to require the creation of entangled states.

This conventional view has now been challenged. In 1999, Braunstein et. al. showed that the states used in current NMR quantum computing experiments are never entangled at any point in time [BCJ$^+$99]. This startling conclusion raises many doubts about the validity of NMR quantum computation [Fit00]. How can NMR techniques demonstrate quantum algorithms without entanglement? Are entangled states necessary resources for quantum computation? These questions are yet to be conclusively answered, although two authors of the Braunstein et. al. paper later pointed out that the NMR machines could not be shown to be merely performing classical computation either [SC99]. This vague state of affairs has left researchers with the uneasy thought: how can one build a quantum computer without knowing what crucial resources make it work?

We seek to better understand these issues through a numerical and theoretical investigation of entangled states in NMR quantum computing. Specifically, this thesis addresses the problem of designing an NMR experiment to realize an entangled state. In NMR, we do not have a pure quantum state, but an ensemble of pure states probabilistically distributed as a function of temperature. We have chosen to study this naturally mixed state – the thermal state. Can it be entangled in the laboratory? We will address this central question in what follows.

1.3 Outline

This first chapter introduces the history and motivation for studying entanglement and summarizes my specific contributions to the field in this thesis work. Chapter 2 describes the theory needed to understand the work described in this thesis. It covers entanglement, quantum computing abstractions, and liquid-state NMR quantum computation. We also review the current understanding of entanglement in liquid-state NMR quantum computation. Chapter 3 discusses the approach we have chosen to investigate our problem and explains our implementation decisions.

In Chapters 4, 5, and 6, we describe the results of this thesis. Chapter 4 presents numerical entanglement calculations for the thermal state. Chapter 5 derives formulas for the entanglement of specific Bell-transformed thermal states using direct and empirically motivated analyses. Chapter 6 draws upon Dür and Cirac's study of entanglement in mixed Bell states [DC00] and uses their formalism to derive bounds on the entanglement of general Bell-transformed thermal states.

Chapter 7 concludes this thesis with a summary of our results and a discussion of directions for further research.

1.4 Contributions to this work

I began this work in August 2002. Given my experience in programming, I asked my advisor, Professor Isaac Chuang, for a thesis topic that would involve our Beowulf cluster. Chuang introduced me to the controversy surrounding NMR state entanglement (Section 1.2) and taught me the basics of entanglement and density matrix formalism (Section 2.1). He suggested that I first try entangling thermal states with a Bell state transformation (Section 3.3.1) and contributed some MATLAB code for performing partial transposes (Section 4.3.4). In early September, undergraduate Joshua Powell joined me on my project. I met with Powell once a week and taught him enough quantum mechanics and quantum information theory to do research with me. Meanwhile, I programmed the Bell transformation entanglement calculations in MATLAB for the $\{N/2, N/2\}$ bipartite split (Section 4.3) and produced entanglement maps of NMR parameter space for up to twelve qubits (Section 4.5). Powell and I worked together to optimize the MATLAB code for short run times and low memory usage (Section 4.3.6). We thought about ways to calculate the minimum eigenvalue of a matrix without having to find all the eigenvalues. Graduate student Aram Harrow suggested a variation on the Jacobi method, and Powell wrote a MATLAB-based minimum eigenvalue function, which incorporated Harrow's idea.

It became clear that any computations on matrices larger than ten qubits had to be run in parallel on the Beowulf cluster. Graduate student Geva Patz had been responsible for the initial construction and setup of our Beowulf cluster. In January 2003, graduate student Andrew Cross joined our laboratory group and took over administration of the cluster from Patz. Cross, Powell, and I teamed up to implement entanglement calculations on the Beowulf cluster (Section 4.4). Cross, with some consultation from Patz, concentrated on making cluster operation reliable (Section 4.4.1). Powell and I wrote C functions for parallel calculation, which could be overloaded on top of our old MATLAB entanglement code (Section 4.4.3). Cross also revised the core parallel linear algebra code. In early February, I gave a poster presentation of the NMR entanglement work at the SQUINT (Southwest Quantum Information Technology) conference and learned of some interesting numerical entanglement research by Stockton, Geremia, Doherty, and Mabuchi at Caltech [SGDM03]. Their group focused exclusively on fully symmetric states and significantly reduced the dimension of the matrices needed to compute entanglement. Perhaps we could also exploit symmetry in our calculations. In addition, the Caltech team had experimented with many bipartite splits, while we had only calculated a specific split.

After I returned from SQUINT, I worked on developing a function that could calculate the minimum eigenvalue of a matrix stored on the Beowulf cluster. This function was the last piece of code we needed to implement the entanglement calculation on the cluster. While exploring efficient methods for calculating minimum eigenvalues, I discovered that the minimum eigenvalue I was trying to find always corresponded to one of two eigenvectors (Section 5.2). Not only that, the eigenvectors were a simple function of qubits in the system. This numerical evidence seemed to confirm our intuition that the transformed thermal state had considerable symmetry. At that very moment, Kenneth Brown was visiting our laboratory from K. Birgitta Whaley's group in UC Berkeley. On a hunch, Chuang suggested that Brown and I check to see if the transformed thermal state was block diagonal in the totally symmetric basis. Chuang found a numerical method to construct the irreducible representations of the totally symmetric group, and the three of us wrote MATLAB code to implement it. While thinking about the NMR entanglement problem, Brown found a direct analytical derivation for the $\{1, N-1\}$ bipartite split (Section 5.1). I began working on finding analytical formulas for other bipartite splits.

In late April, Julia Kempe, a visiting researcher from Université de Paris-Sud and UC Berkeley, told me that I could save much work if I used the Dür and Cirac formalism mentioned above (Section 6.1). The Bell transformed thermal states we were studying were not exactly the same as the ones in the formalism, but Chuang constructed a random unitary procedure that connected thermal states to Dür and Cirac's mixed Bell states (Section 6.2.1). I was then able to calculate analytical bounds on entanglement of the Bell transformed thermal states under any bipartite split (Sections 6.2.2 and 6.2.3). After achieving this promising result, Chuang suggested that some key results from majorization theory might allow me to generalize my approach further. In particular, if the Dür-Cirac state and thermal state satisfied the right majorization relation, then the connecting random unitary procedure was proven to exist, eliminating the need for me to search for a construction (Section 6.3.2). Now the problem was shifted to ensuring that the majorization relation was in fact satisfied. I tried several schemes, but only made partial progress (Section 6.3.3). However, the majorization-based approach looks promising for future work.

Chapter 2

Theory and prior work

This chapter introduces the theory of entanglement (Sec. 2.1), quantum computation (Sec. 2.2), and liquid-state nuclear magnetic resonance (NMR) quantum computation (Sec. 2.3) and reviews research on entanglement in NMR quantum computation previous to our work (Sec. 2.4).

The theory sections provide the minimal background for the reader to interpret our approach and results. More comprehensive references are mentioned in the text. The review section focuses on the work of Braunstein et. al. [BCJ$^+$99], explaining how the authors derive bounds on the entanglement of near maximally mixed states and the implications of their results on liquid-state NMR quantum computation.

In what follows, we assume knowledge of undergraduate quantum mechanics at the level of Griffith's text *Introduction to Quantum Mechanics* [Gri95] and a basic understanding of tensor products, density matrices, and angular momentum at the level of Sakurai's text *Modern Quantum Mechanics* [Sak94].

2.1 Entanglement

This section introduces the notion of entanglement. It begins with a simple physical example to illustrate the concept, followed by formal discussions of entanglement classification and terminology and definitions for pure and mixed state entanglement. For further reading, we recommend the text *Quantum Computation and Quantum*

Information by Nielsen and Chuang [NC00] and the quantum computation and information lecture notes by Preskill [Pre98].

2.1.1 Gedankenexperiment

At the beginning of the previous chapter, we described entanglement as nonlocal correlations between two or more quantum degrees of freedom. To illustrate the meaning of this statement, Einstein, Podolsky, and Rosen originally constructed a thought experiment or *gedankenexperiment* [EPR35]. Here we discuss a more modern version due to Bohm [Boh51] and Basdevant, Dalibard, and Grangier [BDG02].

A pi meson can decay into an electron and positron:

$$\pi^0 \rightarrow e^+ + e^- . \tag{2.1}$$

Since the π^0 has zero spin, the electron and positron must be in the spin singlet state [Gri95] given by

$$|\psi\rangle_{ep} = \frac{1}{\sqrt{2}} \left(|\uparrow\rangle \, |\downarrow\rangle - |\downarrow\rangle \, |\uparrow\rangle \right) \tag{2.2}$$

where first ket in each pair corresponds to the electron state and the second ket in each pair corresponds to the positron state and the kets $|\uparrow\rangle$ and $|\downarrow\rangle$ are the spin up and spin down eigenstates of S_z. The joint state in Eq. 2.2 is sometimes called an EPR (Einstein-Rosen-Podolsky) pair and is an example of an *entangled* state.

Two people, Alice and Bob, decide to perform a experiment. They generate a large number of EPR pairs. Alice keeps the electrons and stays in Boston while Bob takes the positrons to Paris. Assume the states of the EPR pairs are perfectly preserved throughout this process – clearly an idealization. Both of them make spin measurements with their detectors aligned on the z-axis and write down their results in an ordered list.

Consider a single EPR pair. If Alice measures the state of the electron and gets $|\uparrow\rangle$, then the total state of the system collapses to $|\uparrow\rangle \, |\downarrow\rangle$. If Bob now measures the state of the positron,[1] he obtains spin down with absolute certainty. We have only described

[1]We assume that the time between Alice and Bob's measurements is less than distance between

the measurements from Alice's point of view, but the analysis would identical for Bob making the first measurement. Alice and Bob's measurement results are always anti-correlated regardless of the order in which the measurements are made. The anti-correlation is also nonlocal.[2] By making a measurement on the spin of her electron, Alice can perfectly predict Bob's measurement result on the corresponding positron, no matter how far Bob has traveled.

Returning to the gedankenexperiment, Alice records her list: $\{\uparrow\uparrow\downarrow\downarrow\uparrow\downarrow\uparrow \ldots\}$. Then Bob returns to Boston and shows Alice his list: $\{\downarrow\downarrow\uparrow\uparrow\downarrow\uparrow\downarrow \ldots\}$. Their measurements results are exactly as we have explained.

Yet Alice's and Bob's measurements look random on a local level. We now show this result mathematically by analyzing the case of a single EPR pair. The quantum state of one EPR pair can be described by the density matrix

$$\begin{aligned} \rho_{AB} &= |\psi\rangle_{ep}\, {}_{ep}\langle\psi| \\ &= \frac{1}{2}\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{aligned} \tag{2.3}$$

in the basis $\{|\uparrow\rangle|\uparrow\rangle, |\uparrow\rangle|\downarrow\rangle, |\downarrow\rangle|\uparrow\rangle, |\downarrow\rangle|\downarrow\rangle\}$. We label Alice with A and Bob with B and let AB label quantum information shared by both Alice and Bob.

Alice only possesses the electron and has no knowledge of Bob's positron. As far as she is concerned, the density matrix relevant to measurements in her Hilbert space is

$$\rho_A = \operatorname{tr}_B(\rho_{AB}) \tag{2.4}$$

where tr_B denotes the partial trace over Bob's Hilbert space.[3] We define the action

them divided by the speed of light, preventing the possibility of classical communication from affecting the outcome.

[2]When we use the word "local", we mean pertaining to either Alice's or Bob's portion of the system only. When we use the word "nonlocal", we mean pertaining to the entire composite system.

[3]This definition gives the right expectation values for a measurement operator that only acts in the Hilbert space of A. For more details, see page 105 in Ref. [NC00].

of tr_B on an arbitrary ket in the Hilbert space of AB as

$$\mathrm{tr}_B(|a\rangle\,|b\rangle\,\langle a'|\,\langle b'|) = |a\rangle\,\langle a'|\,\mathrm{tr}\,(|b\rangle\,\langle b'|) \tag{2.5}$$

where $|a\rangle$ and $|a'\rangle$ are in the Hilbert space of A and $|b\rangle$ and $|b'\rangle$ are in the Hilbert space of B. Applying this definition, we have

$$\begin{aligned}\rho_A &= \mathrm{tr}_B(\rho_{AB}) \\ &= \frac{1}{2}\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\end{aligned} \tag{2.6}$$

in the basis $\{|\uparrow\rangle\,, |\downarrow\rangle\}$. In other words, she measures spin up or spin down with equal probability.[4]

Analogously, Bob's reduced density matrix is

$$\rho_B = \mathrm{tr}_A(\rho_{AB}) \tag{2.7}$$

where tr_A denotes the partial trace over the Hilbert space of A. A similar calculation shows that Bob has the same reduced density matrix as Alice. Thus, Alice and Bob's measurement results are random when viewed separately. This behavior is a hallmark of entanglement: the global (two-particle) behavior of the system may be radically different from the local (one-particle) behavior of the system. In this sense, entanglement is "hidden" information.

We predicted the correlations in the EPR pair from our knowledge of quantum mechanics, but it can also be explained classically. Imagine a candy factory that always puts one red gum ball and one green gum ball into each box. The experiment we described is equivalent to opening each box and distributing one gum ball to Alice and one gum ball to Bob. If Alice gets a red gum ball, then Bob must have the green one and vice versa. Moreover, if the distribution of gum balls is unbiased, both Alice's and Bob's set of gum balls will be approximately evenly divided between red

[4]We could have also obtained this result by inspection of $|\psi\rangle_{ep}$.

and green.

So far, Alice and Bob's EPR pairs appears to be classical. But what if Alice and Bob make their measurements along the x-axis? To predict the measurement results, we must rewrite Eq. 2.2 in terms of the S_x eigenstates $|\pm\rangle$:

$$|\psi\rangle_{ep} = \frac{1}{\sqrt{2}}\left(|+\rangle\,|-\rangle - |-\rangle\,|+\rangle\right) \tag{2.8}$$

where $|\pm\rangle = (|\uparrow\rangle \pm |\downarrow\rangle)/\sqrt{2}$. We have a spin singlet again, but this time in the S_x basis.[5]

Thus if Alice and Bob measure the spin states of their EPR pairs along the x-axis, their measurement results are still anti-correlated. We could explain this result by the candy factory analogy too, except for one important difference: S_z and S_x are not commuting observables. If Alice first measured her electron along the z-axis and Bob then measured his electron along the x-axis, Bob would find $|+\rangle$ and $|-\rangle$ with equal probability; the perfect anti-correlation would be destroyed by the noncommuting measurements. In general, Alice's measurement axis may differ from Bob's measurement axis by some angle θ. It can be shown that classical local variable models cannot explain all of these measurement scenarios. Indeed, the main idea behind Bell's inequality (see Section 1.1) is that classical local variable models and quantum theory predict contradictory measurement results for specific values of θ. All experimental results to date agree with quantum theory.

To summarize, entanglement is a uniquely quantum mechanical phenomenon that possesses nonlocal correlations. Conversely, non-entangled systems are characterized by two features: 1) local measurements are independent of one another and 2) correlations between local measurements can be modeled classically. We shall return to these ideas when we define entanglement formally.

[5]In fact, $|\psi\rangle_{ep}$ will have this antisymmetric form for measurements along any axis because a spin singlet has zero angular momentum and is therefore rotation invariant.

2.1.2 Classification and terminology

The gedankenexperiment we just saw is one example of entanglement, but there are a wide variety of situations where we may call the system entangled. Here we explain how each of these situations are categorized in the literature.

Suppose we are given an entangled physical system. To classify the type of entanglement, we need to ask three questions:

1. What quantum degrees of freedom do each part of the system possess?

2. Which parts of the system are entangled?

3. Is the quantum state of the system pure or mixed?

The first two questions are straightforward, so we do not elaborate on them. As for the third question, we have pure states when the quantum state of a system can be described by a ket whose evolution is described by Schrödinger's equation. Otherwise, we have mixed states, which can only be described by probabilistic ensembles of kets, i.e. density matrices. The Alice and Bob gedankenexperiment deals with pure state entanglement.

In the context of quantum information theory where entanglement is viewed as a communication resource, we often speak of entanglement between "parties," rather than "parts of the system." The simplest case is two-party or *bipartite entanglement.* In the gedankenexperiment we just examined, Alice and Bob were each a party of a bipartite system.

We emphasize that each party may possess one or more quantum degree of freedom and that these degrees of freedom need not be associated with different particles. In the Alice and Bob gedankenexperiment, we had two entangled quantum degrees of freedom: the spin of an electron and the spin of a positron. However, we could just as easily have entanglement between Party A, B, and C in a six-spin system, where Party A hold the first and second spins, Party B hold the third and fourth spins, and Party C hold the fifth and sixth spins. Or, we could have entanglement between the vibrational mode and spin state of a single particle, a situation which has been experimentally realized in trapped ions [SKK$^+$00, GRL$^+$03].

The reader should note that in the literature, *entangled* is synonymous with *non-separable* and *nonentangled* is synonymous with *separable*. This thesis will use both terminologies equally. In fact, most theorists prefer the expressions "nonseparable" and "separable." The reason is that we only understand what it means for a state to be nonentangled (separable) and identify all other states as being entangled (nonseparable).

If a state is separable with respect to every quantum degree of freedom in the system (i.e. the number of parties is equal to the number of quantum degrees of freedom in the above definition), the state is called *fully separable*. Such a state must always be separable, no matter how the quantum degrees of freedom are distributed among parties. This statement will become clear after we define pure and mixed state entanglement.

In this thesis, we also introduce a new word: *entanglable*. We call a state "entanglable" if there exists a unitary, which may be nonlocal, that transforms the state into an entangled one. A state is *nonentanglable* if such a unitary does not exist.

2.1.3 Pure state separability

We now define pure state separability, first considering the bipartite case and then generalizing to an arbitrary number of parties.

Suppose we have a bipartite system with its quantum degrees of freedom distributed among two parties A and B. We denote Party A's Hilbert space as $\mathcal{H}_A$ and Party B's Hilbert space as $\mathcal{H}_B$. The state of such a system can always be written as

$$|\psi\rangle_{AB} = \sum_{i=1}^{d_A}\sum_{j=1}^{d_B} c_{ij}\,|i\rangle \otimes |j\rangle \tag{2.9}$$

where $|i\rangle \in \mathcal{H}_A$, $|j\rangle \in \mathcal{H}_B$, d_A and d_B are the dimensions of $\mathcal{H}_A$ and $\mathcal{H}_B$ respectively, and the weights c_{ij} are complex coefficients such that $\sum_{i,j}|c_{ij}|^2 = 1$.

The bipartite pure state $|\psi\rangle_{AB}$ is separable (nonentangled) if and only if it can be expressed as

$$|\psi\rangle_{AB} = |a\rangle \otimes |b\rangle \tag{2.10}$$

where $|a\rangle \in \mathcal{H}_A$, $|b\rangle \in \mathcal{H}_B$.

Conversely, a nonseparable (entangled) bipartite pure state cannot be expressed in this form.

A separable bipartite pure state can be decomposed into a tensor product of a state for Party A and a state for Party B. Thus Party A's measurements are independent of Party B's measurements, and we can classically model correlations between the measurement results of the two parties.

The expression in Eq. 2.10 is easily generalized to N parties. A state $|\psi\rangle$ is N-separable if and only if it can be expressed as

$$|\psi\rangle = \bigotimes_{k=1}^{N} |\phi\rangle_k \tag{2.11}$$

where $|\phi\rangle_k$ is in the Hilbert space of the kth party.

Let us illustrate the concept of bipartite separability with a few examples. The state $|\uparrow\uparrow\rangle$ is separable because it is a product state: $|\uparrow\rangle \otimes |\uparrow\rangle$. In contrast, the electron-positron pair described in Eq. 2.2 is nonseparable. It is easily shown that this state cannot be resolved into a tensor product of separate spin states for the electron and positron, that is, $|\psi\rangle_{ep} \neq (c_\uparrow |\uparrow\rangle + c_\downarrow |\downarrow\rangle) \otimes (d_\uparrow |\uparrow\rangle + d_\downarrow |\downarrow\rangle)$ where $c_\uparrow$, $c_\downarrow$, $d_\uparrow$, and $d_\downarrow$ are complex coefficients constrained by the usual normalization relations.

In fact, the electron-positron pair possesses a special kind of entanglement; it is called a *maximally entangled* state. When a bipartite state $|\psi\rangle_{AB}$ is maximally entangled, it means that

$$\mathrm{tr}_A(|\psi\rangle_{AB}\ {}_{AB}\langle\psi|) = \mathrm{tr}_B(|\psi\rangle_{AB}\ {}_{AB}\langle\psi|) = I\,, \tag{2.12}$$

exactly what we found earlier. The identity matrix or *maximally mixed state* represents a uniform distribution in any basis. Therefore, measurements on either half of a bipartite, maximally entangled system look perfectly random.

The spin basis $\{\uparrow,\downarrow\}\otimes\{\uparrow,\downarrow\}$ has a complementary basis composed of four maximally entangled states:

$$\left|\phi^{\pm}\right\rangle \equiv \frac{1}{\sqrt{2}}(\left|\uparrow\right\rangle\left|\uparrow\right\rangle \pm \left|\downarrow\right\rangle\left|\downarrow\right\rangle) \tag{2.13}$$

$$\left|\psi^{\pm}\right\rangle \equiv \frac{1}{\sqrt{2}}(\left|\uparrow\right\rangle\left|\downarrow\right\rangle \pm \left|\downarrow\right\rangle\left|\uparrow\right\rangle). \tag{2.14}$$

These states are frequently called *Bell states* and will play an important role in this thesis.

2.1.4 Mixed state separability

Mixed state separability is defined similarly to pure state separability, with the caveat that the probabilistic nature of the system must be taken into account. We first give the condition for bipartite separability, then generalize to an arbitrary number of parties.

Suppose we have a bipartite system with its quantum degrees of freedom distributed among two parties A and B. The joint state can always be expressed as a density matrix

$$\rho_{AB} = \sum_k p_k \left|\psi_k\right\rangle\left\langle\psi_k\right| \tag{2.15}$$

where $\left|\psi_k\right\rangle$ are in the Hilbert space of AB and the weights p_k are probabilities. A density matrix can be interpreted as a probabilistic ensemble of pure states. Note that the states $\left|\psi_k\right\rangle$ need not be orthogonal.

The bipartite mixed state ρ_{AB} is separable (nonentangled) if and only if it can be expressed as

$$\rho_{AB} = \sum_i p_i\, \rho_i^A \otimes \rho_i^B \tag{2.16}$$

where ρ_i^A and ρ_i^B are density matrices such that $\rho_i^A \in \mathcal{H}_A$ and $\rho_i^B \in \mathcal{H}_B$, and the weights p_i are probabilities.

Conversely, a nonseparable (entangled) bipartite mixed state cannot be expressed in this form.

In view of Eq. 2.16, a separable bipartite mixed state may be interpreted as a probabilistic ensemble containing tensor products of density matrices in $\mathcal{H}_A$ with density matrices in $\mathcal{H}_B$. Measurement on ρ_{AB} may be thought of in the following way. We randomly select one of the tensor products in the sum according to $\{p_i\}$, then measure it. Thus Party A's measurement can be thought of as acting on the reduced density matrix $\mathrm{tr}_B(\rho_i^A \otimes \rho_i^B) = \rho_i^A \mathrm{tr}\,(\rho_i^B)$ [NC00]. But since the trace of any density matrix is always one, we immediately see that $\rho_i^A \mathrm{tr}\,(\rho_i^B)$ is simply ρ_i^A, a density matrix in the Hilbert space of A. Similar reasoning shows that B's measurement acts only on ρ_i^B. As a result, Party A's and Party B's measurements are independent of one another.

Furthermore, a separable bipartite mixed state can always be modeled by classical correlations between local hidden variables [Wer89]. We might imagine a classical machine that outputs pure states to Parties A and B according to some probability distribution. A note of caution must be emphasized. If a density matrix ρ can be written in separable form, that does not mean that the system has been physically prepared as a pure state ensemble. The existence of a separable form merely implies that the correlations between the measurement made by Party A and Party B on ρ_{AB} can be modeled classically.

In general, an N-party mixed state ρ is N-separable if and only it can be expressed as

$$\rho = \sum_i p_i \bigotimes_{k=1}^{N} \rho_i^k \,. \tag{2.17}$$

where p_i are probabilities and ρ_i^k are in the Hilbert space of the kth party.

Unfortunately, the definitions of mixed state separability in Eqs. 2.16 and 2.17 are impractical to use because von Neumann proved that there are an infinite number of decompositions for a given mixed state [Per95]. Consider the following example. The maximally mixed state of two spins can be decomposed in at least two ways:

$$\begin{aligned} \rho &= \frac{1}{4}\left(\left|\uparrow\right\rangle\left|\uparrow\right\rangle\left\langle\uparrow\right|\left\langle\uparrow\right| + \left|\uparrow\right\rangle\left|\downarrow\right\rangle\left\langle\uparrow\right|\left\langle\downarrow\right| + \left|\downarrow\right\rangle\left|\uparrow\right\rangle\left\langle\downarrow\right|\left\langle\uparrow\right| + \left|\downarrow\right\rangle\left|\downarrow\right\rangle\left\langle\downarrow\right|\left\langle\downarrow\right|\right) & (2.18) \\ &= \frac{1}{4}\left(\left|\phi^+\right\rangle\left\langle\phi^+\right| + \left|\phi^-\right\rangle\left\langle\phi^-\right| + \left|\psi^+\right\rangle\left\langle\psi^+\right| + \left|\psi^-\right\rangle\left\langle\psi^-\right|\right) . & (2.19) \end{aligned}$$

where $|\phi^{\pm}\rangle$ and $|\psi^{\pm}\rangle$ are the Bell states from Eqs. 2.13 and 2.14. Eq. 2.19 suggests that ρ is an ensemble of maximally entangled states, tempting us to claim that the density matrix is nonseparable. Yet Eq. 2.18 plainly shows that ρ is separable.

In general, we may be given a density matrix decomposed into pure states, some of which are nonseparable. To determine if the density matrix is entangled, we need to check all possible decompositions into separable states. However, the number of decompositions is infinite. Consequently, it is extremely difficult to determine whether a given mixed state is separable. We will touch on this problem when we consider measures of mixed state entanglement.

2.1.5 Measures of entanglement

Having defined entanglement rigorously, we desire a method to quantify the entanglement of an arbitrary pure or mixed state. While N-party entanglement measures exist (for instance, N-tangle [WC01]), we focus on bipartite measures in this thesis. Here we describe the definitive entanglement measure for bipartite pure states, von Neumann entropy [Per95], and the most commonly used entanglement measure for bipartite mixed states, negativity [dHSL98, VW02].

General considerations for pure states

Consider an arbitrary superposition of $|\uparrow\rangle|\uparrow\rangle$ and $|\downarrow\rangle|\downarrow\rangle$:

$$|\gamma\rangle = a|\uparrow\rangle|\uparrow\rangle + b|\downarrow\rangle|\downarrow\rangle \tag{2.20}$$

where a and b are complex coefficients such that $|a|^2 + |b|^2 = 1$. Let us assume that Party A possesses the first spin and Party B possesses the second spin.

When $a = 1$ and $b = 0$, we have the separable state $|\uparrow\rangle|\uparrow\rangle$. When $a = b = 1/\sqrt{2}$, we have the maximally entangled state $|\phi^+\rangle$ from Eq. 2.13. Now let $a = 1/2$ and $b = \sqrt{3}/2$. Intuitively, we expect that the entanglement of this state is somewhere between that of $|\uparrow\rangle|\uparrow\rangle$ and $|\phi^+\rangle$. But how entangled is it?

One way to measure the entanglement of a bipartite state $|\psi\rangle_{AB}$ is to compare

$|\psi\rangle_{AB}$ against a standard state, which we choose to be $|\phi^+\rangle$. To do the comparison, we must either convert $|\phi^+\rangle \mapsto |\psi\rangle_{AB}$ or $|\psi\rangle_{AB} \mapsto |\phi^+\rangle$. In the first process, we use k maximally entangled pairs to form n approximate copies of $|\psi\rangle_{AB}$ and in the second process, we start with n copies of $|\psi\rangle_{AB}$ and distill from them k' maximally entangled pairs. It is reasonable to use the efficiency of these conversion processes as a way to quantify entanglement. Consequently, we define the *entanglement of formation* E_f and the *entanglement of distillation* E_d for bipartite pure states $|\psi\rangle_{AB}$:

$$\begin{aligned} E_f(|\psi\rangle_{AB}) &\equiv \lim_{n\to\infty} \frac{k_{\min}}{n} && (2.21) \\ E_d(|\psi\rangle_{AB}) &\equiv \lim_{n\to\infty} \frac{k'_{\max}}{n}. && (2.22) \end{aligned}$$

For pure states, it has been shown that E_f and E_d are not only equivalent; they are exactly the von Neumann entropies [BBP+96] of the reduced density matrices ρ_A and ρ_B corresponding to $|\psi\rangle_{AB}\,{}_{AB}\langle\psi|$. Therefore, we define the entanglement E of a bipartite pure state $|\psi\rangle_{AB}$ to be

$$\begin{aligned} E(|\psi\rangle_{AB}) &\equiv E_f(|\psi\rangle_{AB}) = E_d(|\psi\rangle_{AB}) && (2.23) \\ &= S(\rho_A) \\ &= S(\rho_B) \end{aligned}$$

where S denotes the von Neumann entropy function and the reduced matrices are given by $\rho_A = \text{tr}_B(|\psi\rangle_{AB}\,{}_{AB}\langle\psi|)$ and $\rho_B = \text{tr}_A(|\psi\rangle_{AB}\,{}_{AB}\langle\psi|)$.

von Neumann entropy

The von Neumann entropy [Per95] is a measure of disorder for pure quantum states and is defined by

$$S(\rho) \equiv -\text{tr}\left[\rho \log_2(\rho)\right]. \qquad (2.24)$$

When an analytic function, like a logarithm, is applied to a matrix, the function is defined in the diagonal basis of the matrix. Since ρ is Hermitian, it may always be diagonalized as $\rho = UDU^\dagger$ with U being a unitary matrix and D being a diagonal matrix. Consequently, we have

$$\log_2(\rho) = UD'U^\dagger \tag{2.25}$$

with $D'_{ij} = D_{ij} = 0$ for $i \neq j$ and $D'_{ii} = \log_2(D_{ii})$.

If this expression is substituted into Eq. 2.24, the entropy becomes

$$\begin{aligned} S(\rho) &= -\mathrm{tr}\,(UDU^\dagger UD'U^\dagger) \\ &= -\sum_i D_{ii} \log_2 D_{ii}\,, \end{aligned} \tag{2.26}$$

using the cyclic property of the trace in the first line. Relabeling the elements D_{ii} as λ_i (the eigenvalues of ρ), we obtain the usual formula for computing von Neumann entropy:

$$S(\rho) = -\sum_i \lambda_i \log_2(\lambda_i)\,. \tag{2.27}$$

Some of the eigenvalues may be zero in which case we define $0 \log_2(0) \equiv 0$.[6]

We note several properties of von Neumann entropy that will be useful in this thesis:

1. Entropy of pure states: $S(|\psi\rangle\langle\psi|) = 0$. Easily proven by writing $|\psi\rangle\langle\psi|$ in an orthogonal basis where ψ is one of the basis states.

2. Upper bound on entropy: $S(\rho) \leq \log_2 d$ where d is the dimension of ρ. The equality holds when ρ is the maximally mixed state $M_d = I_d/d$ with I_d being the d-dimensional identity matrix.

3. Entropy of separable bipartite mixed state: $S(\rho \otimes \sigma) = S(\rho) + S(\sigma)$. Easily proven by noting that the eigenvalues of $\rho \otimes \sigma$ are the products of the separate eigenvalues of ρ and σ.

When ρ may be decomposed as $\rho = \bigotimes_{i=1}^{N} \sigma$, the third property yields a simple

[6]This definition may be argued from $\lim_{x\to 0^+} x \log_2(x) = 0$.

formula for the von Neumann entropy:

$$S(\rho) = NS(\sigma)\,. \tag{2.28}$$

Now let us return to the example of Eq. 2.20 and compute the entanglement of $|\gamma\rangle$. First, we compute the reduced density matrices of Party A and B (which are equal since $|\gamma\rangle$ is invariant under exchange of the two spins):

$$\begin{aligned} \rho_A &= \rho_B \\ &= \begin{bmatrix} |a|^2 & 0 \\ 0 & |b|^2 \end{bmatrix} \end{aligned} \tag{2.29}$$

where ρ_A and ρ_B are the reduced density matrices as defined in Eqs. 2.4 and 2.7. Application of Eqs. 2.23 and 2.27 yields the entanglement

$$\begin{aligned} E(|\gamma\rangle) &= S(\rho_A) \\ &= -|a|^2 \log_2 |a|^2 - |b|^2 \log_2 |b|^2\,. \end{aligned} \tag{2.30}$$

This formula gives $E(|\uparrow\uparrow\rangle) = 0$ and $E(|\phi^+\rangle) = 1$ for the extreme cases of the separable and maximally entangled states. What happens when $a = 1/2$ and $b = \sqrt{3}/2$? We calculate $E(|\gamma\rangle) \approx 0.811$. Indeed, $|\gamma\rangle$ possesses an amount of entanglement between that of $|\uparrow\rangle\,|\uparrow\rangle$ and $|\phi^+\rangle$.

General considerations for mixed states

While the von Neumann entropy is an excellent measure of pure state entanglement, it does not work well for mixed states. The von Neumann entropy captures all of the disorder in a system whether it is due to classical or quantum correlations. Pure states may only possess the quantum type, but mixed states can have both. For example, consider the maximally mixed state (identity matrix). This state has the maximum possible von Neumann entropy, but it is clearly separable according to Eq. 2.17.

Unlike pure states, there is no widely accepted measure of mixed state entanglement. Some well-known bipartite measures of entanglement in the literature include entanglement of formation [BDSW96], relative entropy of entanglement [VP98], and negativity[dHSL98, VW02]. The first two measures are all variationally-based and extremely difficult to compute for arbitrary density matrices with greater than four dimensions. For instance, the entanglement of formation for a bipartite mixed state ρ_{AB} is defined as

$$E_f(\rho_{AB}) = \min_{\{|\psi_k\rangle\}} \left[\sum_k p_k E(|\psi_k\rangle) \right] \tag{2.31}$$

where the probabilities p_k and the pure states $\{|\psi_k\rangle\}$ correspond to a decomposition (see Eq. 2.15). The variational definition forces us to search over all valid decompositions of ρ to find the minimum.

Negativity

In contrast, negativity is simple to compute for density matrices of any dimension and therefore it is the most widely used measure in the literature. We now describe how to compute negativity. The calculation relies on a linear algebra operation called the *partial transpose.*

We denote the partial transpose of ρ with respect to Party A as ρ^{T_A}. Now any bipartite density matrix can be expressed as

$$\rho = \sum_{a,a',b,b'} C_{a,a',b,b'} \left|a\right\rangle \left\langle a'\right| \otimes \left|b\right\rangle \left\langle b'\right| . \tag{2.32}$$

where the eigenstates $|a\rangle$, $|a'\rangle$ and $|b\rangle$, $|b'\rangle$ live in the Hilbert spaces of Parties A and B respectively. The constants $C_{a,a',b,b'}$ are complex and constrained to give ρ unit trace. Using this notation, the partial transpose is defined as

$$\rho^{T_A} = \sum_{a,a',b,b'} C_{a,a',b,b'} \left|a'\right\rangle \left\langle a\right| \otimes \left|b\right\rangle \left\langle b'\right| . \tag{2.33}$$

This linear algebra operation simply swaps the bra and ket (row and column index) in Party A's Hilbert space. The partial transpose depends on Party A's basis, but the choice of basis does not affect the eigenvalues of ρ^{T_A}.

If ρ^{T_A} has all nonnegative eigenvalues, we say that ρ has *positive partial transpose* (PPT). If ρ^{T_A} has at least one negative eigenvalue, we say that ρ has *negative partial transpose* (NPT). Throughout this thesis, we will assume that the phrase "partial transpose" is equivalent to taking the partial transpose with respect to Party A.

Negativity originates from the following criterion for mixed state entanglement due to Peres [Per96] and Horodecki [HHH96]:

> If the partial transpose of a bipartite system ρ has a negative eigenvalue, ρ must be entangled.

It is easy to see that separable states must have a partial transposition that possesses nonnegative eigenvalues. Any separable state can be expressed as

$$\rho_s = \sum_i p_i \; \rho_A^i \otimes \rho_B^i \tag{2.34}$$

where ρ_A^i and ρ_B^i are density matrices in the Hilbert space of Parties A and B respectively and p_i are probabilities such that $\sum_i p_i = 1$. By the definition in Eq. 2.33, the partial transpose of ρ_s is

$$\rho_s^{T_A} = \sum_i p_i \; (\rho_A^i)^T \otimes \rho_B^i \, . \tag{2.35}$$

Since ρ_A^i is Hermitian, $(\rho_A^i)^T = ((\rho_A^i)^\dagger)^* = (\rho_A^i)^*$. Consequently, $(\rho_A^i)^T$ has nonnegative eigenvalues and unit trace, making it a valid density matrix. It follows that $\rho_s^{T_A}$ is also a density matrix with nonnegative eigenvalues and unit trace.

Observe that the Peres-Horodecki criterion merely gives a *sufficient* condition for entanglement. If it fails, the statement is not known to give any information about the separability of the state, except in the two qubit case when it becomes both *necessary and sufficient.* There are, in fact, so-called "bound" entangled states [Hor97] whose partial transpose have positive eigenvalues even though the state is nonseparable. Maximally entangled pure states cannot be distilled from bound entangled states.

After Życkowski, et.al. [dHSL98] and Vidal and Werner [VW02], we define negativity of state ρ as

$$E_n(\rho) = \max\{0, -\lambda_{\min}(\rho^{T_A})\} \tag{2.36}$$

where $\lambda_{\min}(\sigma)$ is the most negative eigenvalue of a density matrix σ. Notice that separable states automatically have zero negativity. The major difficulty with negativity is that it provides no physical intuition since the partial transpose does not correspond to any physical operation (e.g. measurements or unitaries).

2.2 Quantum computation

A computation requires two ingredients: a state where information is stored and operations that alter the state. This section describes these two abstractions in the context of quantum computation. For further reading, we recommend the text *Quantum Computation and Quantum Information* by Nielsen and Chuang [NC00] and the quantum computation and information lecture notes by Preskill [Pre98].

2.2.1 State

Quantum bit

In a classical computer, the bit is the smallest unit of information that can store the state of the system. Likewise, we can define a quantum bit or *qubit* as a superposition of two orthogonal quantum states, which we name $|0\rangle$ and $|1\rangle$:

$$|\psi\rangle = c_0 |0\rangle + c_1 |1\rangle \tag{2.37}$$

with complex coefficients c_0 and c_1 such that $|c_0|^2 + |c_1|^2 = 1$. The labels $|0\rangle$ and $|1\rangle$ are intended to evoke the analogy of logical "0" and "1" in classical computation. It will sometimes be useful to think of a qubit in this manner, although the coherent superposition inherent to the qubit clearly has no classical analogue.

The qubit in Eq. 2.37 may also be expressed as a vector

$$|\psi\rangle = \begin{bmatrix} c_0 \\ c_1 \end{bmatrix}. \tag{2.38}$$

This notation is useful to understand how operations (usually written as matrices) act on qubits.

From Eq. 2.37, we see that three continuous degrees of freedom are needed to describe a qubit (one removed by the normalization constraint). However, an overall phase cannot be detected in a quantum measurement, leaving us with two continuous degrees of freedom. Therefore, we can also specify the qubit by a polar angle θ and an azimuthal angle ϕ:

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle \ . \tag{2.39}$$

This expression is called the Bloch sphere representation. If $|0\rangle$ and $|1\rangle$ are mapped to $\hat{z}$ and $-\hat{z}$ (the "north" and "south" poles of the Bloch sphere), then $|\psi\rangle$ can be thought of as the normalized vector[7] $\vec{n} = (\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\theta)$ on the Bloch sphere, as depicted in Fig. 2-1.

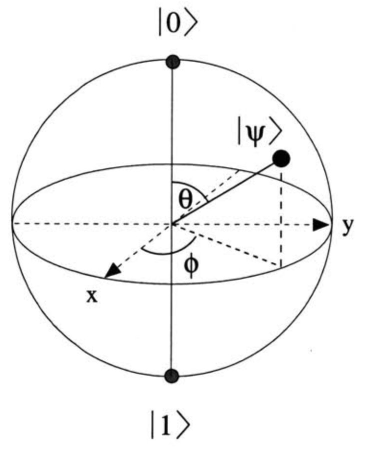

Figure 2-1: Bloch sphere representation of a qubit $|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle$.

Any two level quantum system can be considered a qubit. The simplest example is a spin-1/2 particle. The convention is to map the spins states to computational states such that

$$\begin{aligned} |\uparrow\rangle &\equiv |0\rangle \\ |\downarrow\rangle &\equiv |1\rangle \ . \end{aligned} \tag{2.40}$$

[7]Notation: $\vec{n}$ will always be a normalized vector throughout this thesis.

Multiple qubits

The state of multiple qubits can be expressed as a superposition of tensor products of $|0\rangle$ and $|1\rangle$. A possible two-qubit state is thus $|0\rangle \otimes |0\rangle$. Tensor products becomes cumbersome for many qubits, so we adopt a shorthand where the Kronecker product symbols are dropped: $|\psi\rangle |\phi\rangle \equiv |\psi\rangle \otimes |\phi\rangle$. For example, $|0\rangle |0\rangle \equiv |0\rangle \otimes |0\rangle$. This notation is often simplified further and the digits 0 and 1 are written as a string inside a single ket. For instance, $|00\rangle \equiv |0\rangle \otimes |0\rangle$. The state of two qubits can subsequently be expressed as

$$|\psi\rangle = c_{00}|00\rangle + c_{01}|01\rangle + c_{10}|10\rangle + c_{11}|11\rangle \tag{2.41}$$

where the c_{ij} are complex coefficients such that $|\psi\rangle$ is properly normalized.

In an alternative notation, the binary number labels in Eq. 2.41 may be rewritten in decimal base. For example, the state of any N-qubits can be expressed as

$$|\psi\rangle = \sum_{i=0}^{2^N-1} c_i |i\rangle \ , \tag{2.42}$$

and the N-qubit generalized Bell states can be defined as

$$\left|\Psi_j^\pm\right\rangle = \frac{1}{\sqrt{2}}(|j\rangle \pm \left|2^N - j - 1\right\rangle) \tag{2.43}$$

where $0 \leq j \leq 2^{N-1} - 1$.

Regardless of what notation is used, $|0\rangle$, $|1\rangle$, and tensor product combinations of them are called *computational basis states.* For N qubits, the states $\{|i\rangle\}$ in Eq. 2.42 form the *computational basis.*

2.2.2 Operations

From elementary quantum mechanics, we know that a quantum state evolves according to a unitary matrix U:

$$|\psi(t)\rangle = U(t)\,|\psi(0)\rangle \tag{2.44}$$

where $U(t) = e^{-i\mathcal{H}t/\hbar}$ and $\mathcal{H}$ is the Hamiltonian that models the dynamics of the underlying quantum system.

For the purposes of computation, we only need to know how U maps initial states to final states; the details of the Hamiltonian or how long it is applied are irrelevant. Consequently, we can represent operations on qubits by *quantum gates.*

We draw a one qubit gate as a square box that has an input "wire" on the left and an output "wire" on the right. Thus, time implicitly flows from left to right. A single wire "carries" one qubit.

Some common one-qubit operations are the Pauli operators X, Y, and Z (traditionally also written as σ_x, σ_y, and σ_z) and the Hadamard operator H. Their matrix representations in the computational basis are

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{2.45}$$

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \tag{2.46}$$

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{2.47}$$

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{2.48}$$

with corresponding gate representations shown in Fig. 2-2(a). Notice that X applies the transformation $|0\rangle \mapsto |1\rangle$ and $|1\rangle \mapsto |0\rangle$, which looks like a classical NOT operation. Therefore, an alternative gate representation for X is the NOT symbol depicted in Fig. 2-2(b).

In general, an arbitrary single-qubit unitary can be interpreted as rotating the qubit's Bloch vector by θ around the unit vector $\vec{n}$ in the Bloch sphere:

$$R(\vec{n}, \theta) = e^{-i\frac{\theta}{2}\vec{n}\cdot\vec{\sigma}} \tag{2.49}$$

where $\vec{\sigma}$ is a vector of Pauli matrices (X, Y, Z).

An important two-qubit gate is the controlled NOT or CNOT gate. Its matrix

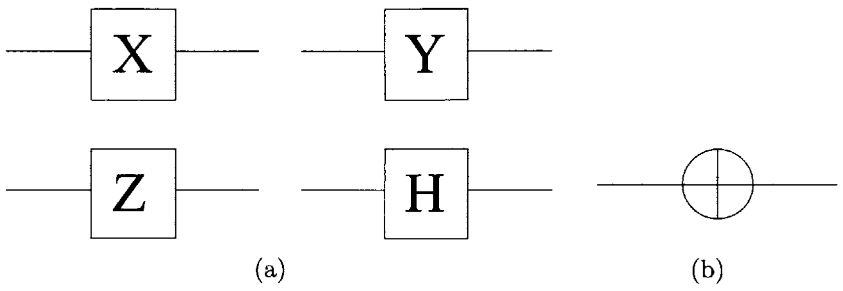

Figure 2-2: (a) Quantum gate representations of Pauli operators X, Y, and Z and Hadamard gate H. (b) Alternative representation of X, the NOT gate.

representation in the computational basis $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$ is

$$U_{\text{cnot}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{2.50}$$

On computational basis states, U_{cnot} behaves exactly like the classical controlled NOT operation. If the first qubit is $|0\rangle$, U_{cnot} does nothing; if the first qubit is $|1\rangle$, U_{cnot} applies a NOT (X) gate to the second qubit. This interpretation leads to the gate representation shown in Fig. 2-3(a). It has two horizontal wires, one for the controlling qubit and one for the target qubit. The dark circle specifies the controlling qubit while the NOT gate acts on the target qubit.

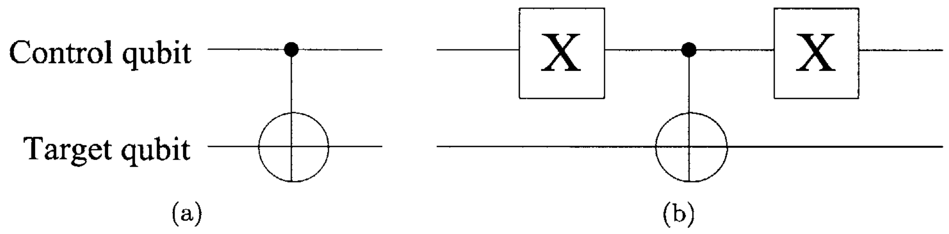

Figure 2-3: (a) Quantum gate representation of the CNOT operation. (b) An example of a quantum circuit.

We can assemble quantum gates into a *quantum circuit* by hooking input wires of quantum gates to the output wires of other quantum gates [Deu89] as shown in Fig. 2-3(b). The steps of a composite quantum operation may be illustrated as a quantum circuit. As an abstraction, quantum circuits look like classical circuits, except the underlying states and operations differ fundamentally in nature.

Finally, we mention an important result in quantum computation. The two-qubit CNOT operation and the continuous set of one-qubit rotations $R(\vec{n}, \theta)$ together can be used to build any unitary operation for an arbitrary number of qubits [NC00].

2.3 Liquid-state NMR quantum computation

This section reviews some basic theoretical facts concerning liquid-state nuclear magnetic resonance (NMR) quantum computation. We begin by discussing the natural quantum state in liquid-state NMR – the thermal state. Then we give a procedure for creating initial states that can be used in quantum computation. Finally, we briefly mention the principles behind unitary operations and readout in liquid-state NMR. For further reading, we recommend introductory articles by Vandersypen, et. al. [VYC02] and Jones [Jon01] and Steffen's PhD thesis [Ste03].

2.3.1 Thermal state

Liquid-state NMR quantum computing experiments are performed on samples containing order $10^{18} - 10^{19}$ identical molecules under a strong, static magnetic field (typically around 12 T). Each molecule possesses a number of spin-1/2 nuclei, which represent the qubits. NMR experiments thus work with an ensemble of quantum computers, one per molecule. The large number of molecules is needed because the signal from a single nuclear spin is exceedingly weak.

The dynamics of this system are dominated by the Zeeman interaction between the nuclear spins and the magnetic field [Gri95], which splits the energy of each spin into two levels. The generic Zeeman interaction for a single spin is

$$\mathcal{H} = -\vec{\mu} \cdot \vec{B} \tag{2.51}$$

where $\vec{\mu}$ is the magnetic dipole moment of the spin and $\vec{B}$ is the magnetic field. It can be shown that

$$\vec{\mu} = \frac{\hbar}{2}\gamma\vec{\sigma} \tag{2.52}$$

where $\vec{\sigma} = (X, Y, Z)$ as before and γ is the gyromagnetic ratio unique to the chemistry of the spin and its environment.

If we choose the magnetic field to be along the z-axis such that $\vec{B} = B_0\hat{z}$ and insert our expression for $\vec{\mu}$, the Hamiltonian can be rewritten as

$$\mathcal{H} = \frac{h\nu}{2}Z \tag{2.53}$$

with the resonance frequency $\nu = \gamma B$ corresponding to the Zeeman energy splitting. Because the resonance frequency depends on B, the strength of the field must be extremely homogeneous over the volume of the sample. Otherwise, each molecule will have a different resonance frequency and the overall measurable signal from the ensemble will be washed out.

In general, we may have many spin-1/2 nuclei per molecule, all with different resonance frequencies. For convenience in the analysis, throughout this thesis, we assume that all the spins have the same Zeeman energy splitting and consider a Hamiltonian of N isotropic nuclear spins:

$$\mathcal{H} = \frac{h\nu}{2}\sum_{i=1}^{N} Z_i\,, \tag{2.54}$$

which has been expressed in the computational/spin bases.[8] Here ν is the characteristic frequency corresponding to the Zeeman energy splitting of one spin, Z_i is taken to be the Pauli Z operator acting on the ith spin, and N is the number of spin-1/2 particles in each molecule. In reality, neighboring spins also interact through an electron mediated force, giving rise to a J-coupling term hJZ_iZ_j between the ith and jth spins. However, the coupling constant J is typically six orders of magnitude smaller than the strength of the Zeeman interaction ν, so we neglect it in this thesis.

[8]In NMR quantum computation, generally the qubits are chosen to be spin-1/2 particles, and the computational basis is taken to be equivalent to the spin basis.

Due to the ensemble nature of liquid-state NMR, the natural quantum state of the system is the *thermal state* given by the Boltzmann distribution:

$$\rho_{\text{th}} = \frac{e^{-\mathcal{H}/kT}}{\mathcal{Z}} \tag{2.55}$$

where $\mathcal{H}$ is the Hamiltonian from Eq. 2.54, k is the Boltzmann constant, T is the absolute temperature, and $\mathcal{Z}$ is a constant chosen such that $\operatorname{tr} \rho_{\text{th}} = 1$.

Notice ρ_{th} is diagonal in the computational basis because the Hamiltonian possesses the same property. The Hamiltonian also has extensive symmetry. It is invariant under the exchange of any two spins and its energy eigenvalues are simply the total azimuthal spin in the computational basis states. The absence of coupling terms in the Hamiltonian also makes the N-qubit thermal state fully separable:

$$\rho_{\text{th}} = \bigotimes_{i=1}^{N} \rho_{\text{th},1} \tag{2.56}$$

where $\rho_{\text{th},1}$ is the thermal state corresponding to any single qubit.

In analytical calculations, we prefer to work with dimensionless quantities if possible and thus define

$$\alpha \equiv h\nu/kT\,. \tag{2.57}$$

This dimensionless variable characterizes the polarization of the thermal state because in the liquid-state NMR regime ($\alpha \ll 1$), the one-qubit thermal state is approximately

$$\rho_{\text{th},1} \approx \frac{1}{2}\begin{bmatrix} 1+\frac{\alpha}{2} & 0 \\ 0 & 1-\frac{\alpha}{2} \end{bmatrix}. \tag{2.58}$$

We now rewrite the thermal state in terms of α and a dimensionless Hamiltonian $\mathcal{H}_d$:

$$\rho_{\text{th}} = \frac{e^{-\mathcal{H}_d \alpha}}{\mathcal{Z}} \tag{2.59}$$

where $\mathcal{H}_d = \frac{1}{2}\sum_{i=1}^{N} Z_i$ and the normalization constant is

$$\mathcal{Z} = (e^{\alpha/2} + e^{-\alpha/2})^N\,. \tag{2.60}$$

The diagonal entries of the thermal state in the computational basis are easily found to be

$$\langle i| \rho_{\text{th}} |i\rangle = \frac{1}{\mathcal{Z}} e^{[N-2w(i)]\alpha/2},\ 0 \leq i \leq 2^N - 1 \tag{2.61}$$

where the Hamming weight $w(i)$ is the number of 1s in the binary expression for i. The number $\langle i| \rho_{\text{th}} |i\rangle$ is the probability for finding the overall spin state in $|i\rangle$. We will frequently refer to this formula.

We also use Eqs. 2.27 and 2.28 to calculate the von Neumann entropy of an N-qubit thermal state:

$$\begin{aligned} S(\rho_{\text{th}}) &= NS(\rho_{\text{th},1}) \qquad (2.62) \\ &= -N\left[p\log_2(p) + (1-p)\log_2(1-p)\right] \end{aligned}$$

where $\rho_{\text{th},1}$ is again the thermal state for a single qubit and $p = \frac{1}{2}\left[1 + \tanh\alpha\right]$.

2.3.2 Initial state preparation

Most quantum algorithms begin with an initial pure state, but liquid-state NMR experiments must be performed at room temperature where $\alpha \ll 1$.[9] In this limit, thermal states are extremely close to the maximally mixed state (see Eq.2.61), far from being pure states. Even for spin species with large Zeeman splittings, the population difference between $|\uparrow\rangle$ and $|\downarrow\rangle$ for a single spin is no more than 10^{-4} in conventional NMR magnetic fields.

To solve this problem, liquid-state NMR quantum computing experiments use initial states of form

$$\rho_{\text{eff}} = (1-\epsilon)M_d + \epsilon\,|0\rangle\,\langle 0| \tag{2.63}$$

where the dimension of ρ_{eff} is $d = 2^N$, $M_d = I_d/d$ is the maximally mixed state (I_d is the d-dimensional identity matrix), and ϵ characterizes the excess population in the ground state $|0\rangle$. In typical NMR quantum computing experiments, ϵ ranges between 10^{-7} and 10^{-4}.

[9]See Section 3.2.1 for further discussion.

Due to the large contribution from the identity matrix, ρ_{eff} is exceedingly mixed. However, if a unitary operation is applied to ρ_{eff}, the identity will remain untouched as $UIU^\dagger = I$. Computations may therefore be performed as if they were only acting on the ground state. For this reason, ρ_{eff} is called an *effective pure state.*[10]

One method of creating an effective pure state is temporal averaging [GC97, KC98]. We conduct multiple experiments in which the thermal state is prepared in a different way each time and sum the results afterwards. Unitary operations are performed on ρ_{th} to cyclically permute the diagonal entries that correspond to the populations of the excited computational basis states. The transformed states are then summed over all possible cyclic permutations. This procedure is summarized by the mathematical statement

$$\rho_{\text{eff}} = \sum_{j=1}^{d-1} p_j U_j \rho_{\text{th}} U_j^\dagger \, . \tag{2.64}$$

where d is the dimension of ρ_{eff}, $p_j = 1/(d-1)$ and U_j is a unitary operation that cyclically permutes the populations of the excited computational basis states $|i\rangle \neq |0\rangle$ $j-1$ times.

To see how the method works, we demonstrate the procedure for two qubits. In this case, ρ_{th} is a 4×4 density matrix whose diagonal entries may be labeled by alphabetical letters:

$$\rho_{\text{th}} = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & d \end{bmatrix} \tag{2.65}$$

where the representation is given in the computational basis.

If the excited state populations b, c and d are cyclically permuted once and twice,

[10]Sometimes an effective pure state is also called a pseudo-pure state in the literature.

we obtain the transformed thermal states

$$\rho_1 = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & b & 0 \\ 0 & 0 & 0 & c \end{bmatrix} \tag{2.66}$$

$$\rho_2 = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & d & 0 \\ 0 & 0 & 0 & b \end{bmatrix}. \tag{2.67}$$

Now we mix the states together uniformly to form the effective pure state

$$\begin{aligned} \rho_{\text{eff}} &= \frac{1}{3}(\rho_{\text{th}} + \rho_1 + \rho_2) \qquad (2.68) \\ &= \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & \frac{b+c+d}{3} & 0 & 0 \\ 0 & 0 & \frac{b+c+d}{3} & 0 \\ 0 & 0 & 0 & \frac{b+c+d}{3} \end{bmatrix} \\ &= \frac{1-\epsilon}{4} I_4 + \epsilon \left|0\right\rangle\left\langle 0\right| \end{aligned}$$

where $\epsilon = a - (b+c+d)/3$.

It is straightforward to generalize this analysis to N qubits and obtain an expression for the excess population in the ground state:

$$\epsilon = \frac{e^{N\alpha/2}}{\mathcal{Z}} - \frac{1 - e^{N\alpha/2}/\mathcal{Z}}{2^N - 1}, \text{ exact} \tag{2.69}$$

$$\epsilon \approx \frac{N\alpha/2}{2^N - 1}, \text{ room temperature} \tag{2.70}$$

where we have used the fact that the ground state population in a thermal state is $e^{-N\alpha/2}/\mathcal{Z}$ (see Eq. 2.61) in the first line and taken the limit $\alpha \ll 1$ in the second line.

2.3.3 Unitary operations and readout signal

Here we mention a few significant facts about NMR unitary operations and readout that are relevant to this thesis.

A liquid-state NMR system qualifies as a universal quantum computer since single-qubit rotations and the CNOT operation can be implemented. Single-qubit rotations are achieved by applying a radio frequency pulse to a spin at its resonant frequency. This signal perturbs the large, static magnetic field $B\hat{z}$ with a small sinusoidal field along an axis orthogonal to $\hat{z}$. The rotation axis is specified by the phase of the sinusoid while the rotation angle is specified by the amplitude of the sinusoid. A CNOT operation is executed by combining the single-qubit radio frequency pulses with time delays where the J-coupling is allowed to evolve freely.

In liquid-state NMR, the measurable signal S of one spin is proportional to the expectation value of X or Y with respect to the spin state ρ:

$$S \propto \operatorname{tr}(\rho X) \text{ or} \tag{2.71}$$

$$S \propto \operatorname{tr}(\rho Y). \tag{2.72}$$

This fact justifies use of the effective pure state. When $\rho = I$, $S = 0$ since X and Y are traceless operators. The measurable signal for $U\rho_{\text{eff}}U^\dagger$ thus corresponds to the excess pure state $U\left|0\right\rangle\left\langle 0\right|U^\dagger$, attenuated by factor ϵ. For more spins, the observable measured is a tensor product of X and Y operators, so the argument holds generally.

2.4 Bounds on the entanglement of near maximally mixed states

This section reviews research concerning the separability and nonseparability of near maximally mixed states. We focus on the bounds due to Braunstein, et. al. [BCJ$^+$99], describing their results and explaining how they were derived. Then we discuss the implications of the separable bound on liquid-state NMR quantum computation.

2.4.1 Braunstein et. al. separable and nonseparable bounds

Formalism

The effective pure state (for small ϵ) is an example of a state with a density matrix very close to the maximally mixed state. These so-called *near maximally states* have the form

$$\rho_\epsilon = (1-\epsilon)M_d + \epsilon\rho' \tag{2.73}$$

where d is the dimension of ρ_ϵ, $M_d = I_d/d$ is the d-dimensional maximally mixed state as before, ρ' is an arbitrary density matrix, and $\epsilon \ll 1$.

The parameter ϵ may be interpreted as the size of a neighborhood around the maximally mixed state. It is known that all of the states in this neighborhood are separable if the size of the neighborhood is sufficiently small [dHSL98]. This suggests that for ϵ below some threshold, ρ_ϵ is always separable.

Braunstein et. al. [BCJ+99] recently established upper and lower bounds on ϵ for a system of N spin-1/2 particles:

$$\text{for } \epsilon \leq \frac{1}{1+2^{2N-1}} \Rightarrow \rho_\epsilon \text{ is always separable} \tag{2.74}$$

$$\text{for } \epsilon > \frac{1}{1+2^{N/2}} \Rightarrow \rho_\epsilon \text{ can be nonseparable}\,. \tag{2.75}$$

The upper bound (Eq. 2.74) states that when ϵ is sufficiently small, ρ_ϵ is separable. The lower bound (Eq. 2.75) states that when ϵ is sufficiently large, there exists some ρ' such that ρ_ϵ is nonseparable.

Derivation of separable bound

Proof of the separable bound in Eq. 2.74 relies on the decomposition of the near maximally mixed state into a particular continuous and separable basis. The reasoning behind the proof may be summarized as follows. If a state has nonnegative coefficients for a separable, continuous decomposition, then the coefficients can be interpreted as probability densities and the state itself must be separable. To determine when $\rho_\epsilon = (1-\epsilon)M_d + \epsilon\rho'$ has nonnegative coefficients, we individually calculate the coefficients for M_d and ρ'. The maximally mixed state M_d is clearly separable, so all its coefficients are nonnegative. The state ρ' may be entangled. If so, it must have at least one negative coefficient. Yet we can always lower ϵ until all coefficients of the overall state ρ_ϵ are nonnegative. The goal of the proof is to determine the value of ϵ where this transition happens.

Now we describe the details of the derivation. A separable basis for N spins is the continuous set of fully separable spin states,[11] parameterized by N Bloch spheres. The decomposition of a density matrix in this basis is

$$\rho = \int d\Omega_1 ... d\Omega_N \; w(\vec{n}_1, ..., \vec{n}_N) \bigotimes_{i=1}^{N} P_{\vec{n}_i} \tag{2.76}$$

where Ω_i is the solid angle of the ith Bloch sphere, $w(\vec{n}_1, ..., \vec{n}_N)$ are the coefficients of the decomposition, and $P_{\vec{n}_i} = \frac{1}{2}(I_2 + \vec{n}_i \cdot \vec{\sigma})$ are the projectors corresponding to the spin state pointing in the $\vec{n}_i$ direction on the Bloch sphere.

Because the basis used in Eq. 2.76 is overcomplete, there are no unique coefficients. We now produce a specific decomposition by introducing a discrete basis formed from tensor products of Pauli matrices. The decomposition of an arbitrary density matrix in this discrete basis is

$$\rho = \frac{1}{2^N} c_{\alpha_1 ... \alpha_N} \bigotimes_{i=1}^{N} \sigma_{\alpha_i} \tag{2.77}$$

where the coefficients $c_{\alpha_1 ... \alpha_N}$ specify the decomposition, the indices α_i take on the values $\{0, 1, 2, 3\}$, and the σ operators are the Pauli matrices defined as $\sigma_0 \equiv I_2$, $\sigma_1 \equiv X$, $\sigma_2 \equiv Y$, and $\sigma_3 \equiv Z$. To avoid unwieldy notation, here we have adopted the Einstein convention and implicitly sum over repeated instances of α_i. The coefficients of the decomposition are given by

$$c_{\alpha_1 ... \alpha_N} = \mathrm{tr}\left[\rho \left(\bigotimes_{i=1}^{N} \sigma_{\alpha_i}\right)\right] . \tag{2.78}$$

[11] The fully separable spin state has form $\bigotimes_{i=1}^{N} |\psi\rangle_i$ where $|\psi\rangle_i$ is an arbitrary state of the ith spin.

The continuous coefficients $w(\vec{n}_1, ..., \vec{n}_N)$ are generated via the relationship

$$\frac{1}{2}\sigma_{\alpha_i} = \frac{3}{4\pi}\int d\Omega_i (\vec{n}_i)_{\alpha_i} P_{\vec{n}_i} \tag{2.79}$$

where $(\vec{n}_i)_{\alpha_i}$ is the α_i-th component of the Bloch vector $\vec{n}_i$ and we set $(\vec{n}_i)_0 \equiv 1/3$.

Inserting this expression into the discrete Pauli decomposition of Eq. 2.77, we get

$$\rho = \int d\Omega_1 ... d\Omega_N \left(\frac{3}{4\pi}\right)^N c_{\alpha_1 ... \alpha_N} \prod_{i=1}^{N} (\vec{n}_i)_{\alpha_i} \left(\bigotimes_{j=1}^{N} P_{\vec{n}_j}\right) . \tag{2.80}$$

Comparing this relation with Eq. 2.76, we find the continuous coefficients

$$w(\vec{n}_1, ..., \vec{n}_N) = \left(\frac{3}{4\pi}\right)^N c_{\alpha_1 ... \alpha_N} \prod_{i=1}^{N} (\vec{n}_i)_{\alpha_i} . \tag{2.81}$$

Substituting Eq. 2.78 for $c_{\alpha_1, ..., \alpha_N}$, we obtain the promised decomposition

$$w(\vec{n}_1, ..., \vec{n}_N) = \left(\frac{1}{4\pi}\right)^N \operatorname{tr}\left[\rho \bigotimes_{i=1}^{N} (I_2 + 3\vec{n}_i \cdot \vec{\sigma})\right] . \tag{2.82}$$

Now let us calculate the coefficients for M_d and ρ' under the representation in Eq. 2.82. The maximally mixed state M_d has uniform coefficients

$$w_{M_d}(\vec{n}_1, ..., \vec{n}_N) = (1/4\pi)^N \tag{2.83}$$

as the total solid angle of any Bloch sphere is 4π.

The density matrix ρ' is unspecified, but we can derive a lower bound on its coefficients. Each matrix in the tensor product $I_2 + 3\vec{n}_i \cdot \vec{\sigma}$ has eigenvalues -2 and 4 because $\vec{n}_i \cdot \vec{\sigma}$ has eigenvalues -1 and 1.[12] The largest eigenvalue of a density matrix is 1. Since the eigenvalues of a product matrix are the products of the factor matrix eigenvalues, the most negative possible value of the trace in Eq. 2.82 is $4^{N-1}(-2) =$

[12]An easy way to understand this result is to recall the Hamiltonian of a spin-1/2 particle in a magnetic field oriented in the $\vec{n}_i$ direction.

-2^{2N-1}, yielding a bound on the coefficients of ρ':

$$w'(\vec{n}_1, ..., \vec{n}_N) \geq -\frac{2^{2N-1}}{(4\pi)^N} . \tag{2.84}$$

Putting together these results, the coefficients of ρ_ϵ, as defined in Eq. 2.82, are constrained by

$$\begin{aligned} w_\epsilon(\vec{n}_1, ..., \vec{n}_N) &= (1-\epsilon) w_{M_d}(\vec{n}_1, ..., \vec{n}_N) + \epsilon w'(\vec{n}_1, ..., \vec{n}_N) && (2.85) \\ &\geq \frac{1-\epsilon}{(4\pi)^N} - \frac{\epsilon\, 2^{2N-1}}{(4\pi)^N} . && (2.86) \end{aligned}$$

These coefficients must be nonnegative when

$$\epsilon \leq \frac{1}{1+2^{2N-1}} , \tag{2.87}$$

which is exactly the separable bound of Eq. 2.74.

Derivation of nonseparable bound

The nonseparable bound of Eq. 2.75 derives from a specific construction for ρ_ϵ. The main steps are:

1. Divide the N-spin system into two parties, each containing $N/2$ spins.

2. Construct a state of form ρ_ϵ such that ρ' corresponds to a specific maximally entangled state shared between the two parties.

3. Project the constructed state into a Werner state [Wer89]. Application of a condition on entangled Werner states yields a bound on nonseparable ρ_ϵ .

Now we go over the details. In the first step, the system is condensed into two spin-$(r-1)/2$ particles where $r = 2^{N/2}$ and each aggregate spin particle is composed by adding together $N/2$ spin-1/2 particles. Moreover, each aggregate spin particle has r angular momentum eigenstates [Sak94]. This construction is simply a physically motivated method to divide the system into two parties.

For the second step, we consider the near-maximally mixed state

$$\rho'_\epsilon = (1-\epsilon)M_{r^2} + \epsilon \left|\psi\right\rangle \left\langle\psi\right| \; . \tag{2.88}$$

with maximally entangled state

$$\left|\psi\right\rangle = \frac{1}{\sqrt{r}} \sum_{n=1}^{r} \left|\ell\right\rangle \left|\ell\right\rangle \; . \tag{2.89}$$

The kets $\left|\ell\right\rangle$ are angular momentum eigenstates. The first ket in each ket pair corresponds to the first aggregate spin and the second ket in each ket pair corresponds on the second aggregate spin.

In the final step, we project ρ'_ϵ into the subspace spanned by $\left|1\right\rangle$ and $\left|2\right\rangle$, yielding

$$\tilde{\rho} = \frac{1}{C}\left[\frac{1-\epsilon}{r^2} I_4 + \frac{\epsilon}{r}\left(\left|1\right\rangle\left|1\right\rangle + \left|2\right\rangle\left|2\right\rangle\right)\left(\left\langle 1\right|\left\langle 1\right| + \left\langle 2\right|\left\langle 2\right|\right)\right] \tag{2.90}$$

where the constant $C = (4/r^2)[1+\epsilon(d/2-1)]$ normalizes $\tilde{\rho}$ properly.

Some rearranging gives

$$\tilde{\rho} = (1-\epsilon')M_4 + \epsilon' \left|\phi\right\rangle\left\langle\phi\right| \tag{2.91}$$

with a new parameter

$$\epsilon' = \frac{2\epsilon/r}{C} = \frac{\epsilon r/2}{1+\epsilon(r/2-1)} \tag{2.92}$$

and a new maximally entangled state

$$\left|\phi\right\rangle = \frac{1}{\sqrt{2}}\left(\left|1\right\rangle\left|1\right\rangle + \left|2\right\rangle\left|2\right\rangle\right). \tag{2.93}$$

Now $\tilde{\rho}$ of Eq. 2.91 is an instance of a Werner state. Because Werner states are always entangled [Wer89, Pop94, BBP$^+$96] for $\epsilon' > 1/3$,[13] we find

$$\epsilon > \frac{1}{1+r} = \frac{1}{1+2^{N/2}}, \tag{2.94}$$

[13]In Chapter 6, we derive a more general version of this condition.

precisely the Braunstein et. al. bound on nonseparable near maximally mixed states.

Improved separable bound

In the past year, Gurvits and Barnum [GB03] tightened the bound on the separability of near maximally-mixed states to

$$\epsilon \leq \frac{1}{2^{N/2-1}\left(2^N-1\right)}. \tag{2.95}$$

This bound scales as $2^{-3N/2}$, an improvement over the corresponding Braunstein et. al. bound, which scales as 2^{-2N}. The derivation of the Gurvits-Barnum result lies beyond the scope of this thesis.

2.4.2 Implications on liquid-state NMR quantum computation

When liquid-state NMR quantum computation was first proposed in 1997, many scientists were skeptical. Most quantum algorithms require entangled states, but the effective pure state contains a minute fraction of pure state. It seemed implausible that such a state could be entangled since its density matrix lies so close to the maximally mixed state, which is clearly separable. Despite these criticisms, numerous quantum algorithms were successfully demonstrated in liquid-state NMR [CVZ+98, CGK98, CPM+98, JM98, JMH98, LBF98, NKL98].

But a year after the initial experiments, Braunstein and his collaborators applied their bounds on near maximally mixed states to NMR and proved that the states used in current NMR quantum computations were never entangled at any moment in time, since ρ_ϵ is separable for $N \leq 12$ under ideal experimental conditions, as seen in Fig. 2-4. The bound they derived in Eq. 2.74 is clearly applicable because NMR machines can only transform a state of form ρ_ϵ to another state of form ρ_ϵ. In fact, the tighter Gurvits-Barnum bound shows that ϵ remains in the separable regime for any computation using less than 23 qubits (also plotted in the same figure). This situation still holds today because the largest NMR quantum computer on record

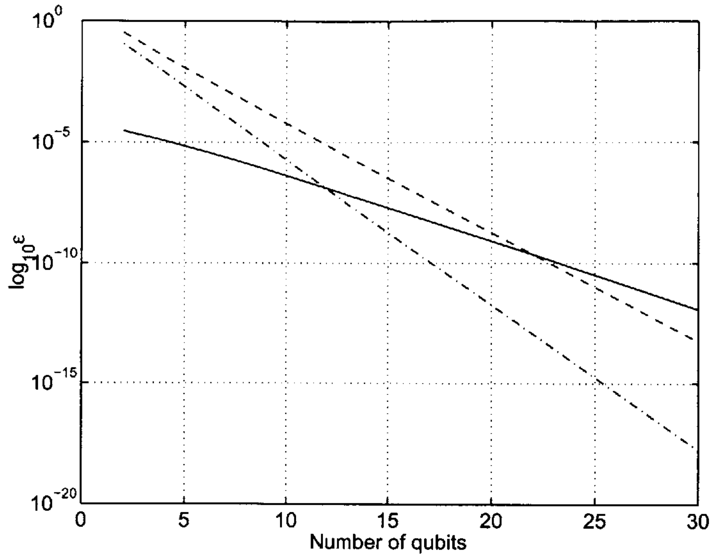

Figure 2-4: Comparison of ϵ in NMR effective pure state (solid line) and constraints on ϵ from the Braunstein et. al. bound on separable ρ_ϵ (dash dotted line) and from the Gurvits and Barnum bound on separable ρ_ϵ (dashed line). Here the effective pure state corresponds to an isotropic proton spin system in a 11.74 T magnetic field at room temperature.

uses 7 qubits [VSB$^+$01].

How should we interpret the Braunstein et. al result? There are two possibilities.

1. NMR machines are merely simulating quantum computation.

2. Entangled states are not a necessary resource for quantum computation.

Neither possibility has been ruled out, but all attempts to classically model separable NMR quantum computation have been unsuccessful thus far. Shortly after the Braunstein et. al. result was published, Schack and Caves constructed a model [SC99] where the NMR quantum state is expressed as a probability distribution over the orientation of Bloch vectors and the effect of a quantum gate is described by a continuous set of transition probabilities that map the initial probability distribution to the final one. The model accurately simulates separable NMR quantum computation. However, compared to the true physical, measurable signal, the signal in the model decreases exponentially as a function of the number of entangling gates (nonseparable unitary operations). In 2002, Caves and Menicucci [MC02] constructed another classical model that also fails because the number of hidden classical variables required increases exponentially as a function of the number of qubits.

Though far from conclusive, these preliminary results admit several interpretations. First, even if NMR machines are not performing true quantum computation, they might still be more efficient than classical computers. Second, the fact that the signal in the first model decreases after the application of each entangling gate suggests that *entangling operations*, rather than entangled states, may be the crucial resource for quantum computation. This conjecture is supported by the recent result that universal quantum computation can be performed with any entangling interaction and local unitary operations [DNBT02].

2.5 Summary

In this chapter, we covered basic definitions and concepts in entanglement, quantum computation, and liquid-state NMR quantum computation and ended with a review of bounds on entanglement in NMR quantum computation.

We defined the notion of separability for quantum states. A bipartite pure state $|\psi\rangle_{AB}$ is separable if and only if it can be expressed as

$$|\psi\rangle_{AB} = |a\rangle \otimes |b\rangle \tag{2.96}$$

where $|a\rangle \in \mathcal{H}_A$ and $|b\rangle \in \mathcal{H}_B$. A bipartite mixed state ρ_{AB} is separable if and only if it can be expressed as

$$\rho_{AB} = \sum_i p_i \; \rho_i^A \otimes \rho_i^B \tag{2.97}$$

where $\rho_i^A \in \mathcal{H}_A$, $\rho_i^B \in \mathcal{H}_B$, and p_i are probabilities.

We also described two measures of bipartite entanglement – von Neumann entropy for pure states and negativity for mixed states. The entanglement of a bipartite pure state ψ_{AB} is given by

$$E(|\psi\rangle_{AB}) = S(\rho_A) = S(\rho_B) \tag{2.98}$$

where S denotes the von Neumann entropy function and the reduced matrices are given by $\rho_A = \mathrm{tr}_B(|\psi\rangle_{AB}\,{}_{AB}\langle\psi|)$ and $\rho_B = \mathrm{tr}_A(|\psi\rangle_{AB}\,{}_{AB}\langle\psi|)$. The von Neumann entropy of a state ρ can be calculated in terms of its eigenvalues λ_i with the formula

$$S(\rho) = -\sum_i \lambda_i \log_2(\lambda_i)\,. \tag{2.99}$$

The negativity of a bipartite mixed state ρ_{AB} is given by

$$E_n(\rho) = \max\{0, -\lambda_{\min}(\rho^{T_A})\} \tag{2.100}$$

where $\lambda_{\min}(\sigma)$ is the most negative eigenvalue of a density matrix σ.

We learned that in quantum computation, the state of the computer can be stored in information units called qubits and the operations on the state are performed by unitary transforms. The qubit is a superposition of two orthogonal states $|0\rangle$ and $|1\rangle$:

$$|\psi\rangle = c_0\,|0\rangle + c_1\,|1\rangle \tag{2.101}$$

with c_0 and c_1 being complex coefficients such that $|c_0|^2 + |c_1|^2 = 1$. The states $|0\rangle$, $|1\rangle$, and tensor product combinations of $|0\rangle$ and $|1\rangle$ are called computational basis states. We can represent a unitary operation by a quantum gate and a series of unitary operations by a quantum circuit.

Quantum computations have been realized on nuclear spins in liquid-state NMR. The natural NMR quantum state is the thermal state

$$\rho_{\mathrm{th}} = \frac{e^{-\mathcal{H}_d \alpha}}{\mathcal{Z}} \tag{2.102}$$

where the dimensionless parameter $\alpha \equiv h\nu/kT$ characterizes the polarization of ρ_{th} (ν is the resonance frequency and T is absolute temperature), $\mathcal{Z}$ is a normalization constant, and $\mathcal{H}_d$ is the dimensionless Hamiltonian

$$\mathcal{H}_d = \sum_{i=1}^{N} Z_i \tag{2.103}$$

where Z_i is the Pauli Z operation acting on the ith spin. The thermal state is inherently mixed while most quantum algorithms require an initial pure state. To solve this problem, transformed thermal states can be mixed together to form an effective pure state

$$\rho_{\text{eff}} = (1-\epsilon)M_d + \epsilon \left|0\right\rangle \left\langle 0\right| \tag{2.104}$$

where the maximally mixed state M_d is I_d/d (I_d being the d-dimensional identity matrix), $|0\rangle$ is the ground state, and ϵ is a parameter much smaller than one. Since M_d is not affected by unitary operations and is not measurable in NMR, computations on ρ_{eff} can be implemented solely on the pure state $|0\rangle$. The effective pure state is an instance of a near maximally mixed state

$$\rho_\epsilon = (1-\epsilon)M_d + \epsilon\rho' \tag{2.105}$$

where ρ' is an arbitrary density matrix.

Although many quantum algorithms have been successfully demonstrated using NMR machines, Braunstein et. al. showed that the states used in current liquid-state NMR quantum computations are separable. Specifically, they found bounds on the separability and nonseparability of near-maximally mixed states in terms of ϵ and the number of qubits N:

$$\text{for } \epsilon \leq \frac{1}{1+2^{2N-1}} \Rightarrow \rho_\epsilon \text{ is always separable} \tag{2.106}$$

$$\text{for } \epsilon > \frac{1}{1+2^{N/2}} \Rightarrow \rho_\epsilon \text{ can be nonseparable}\,. \tag{2.107}$$

As illustrated in Fig. 2-4, the separable bound shows that if effective pure states are used for initialization and $N < 12$, an NMR quantum computer with spins at room temperature under standard laboratory magnetic fields can never realize an entangled state in the laboratory.

Chapter 3

Approach

This chapter outlines the approach we use to investigate entanglement in liquid-state NMR. From this point on, we present original work that builds upon the theoretical background in Chapter 2.

The connection between mixed state entanglement and quantum algorithms is poorly understood. To shed light on this puzzle, we wish to investigate the *theoretical* issues behind mixed state entanglement in an NMR experiment.

Here we discuss the considerations involved in realizing an entangled NMR state. First, the difficulty in experimentally entangling effective pure states leads us to study thermal states (Section 3.1). Then we evaluate the different experimental approaches to entangling a thermal state (Section 3.2). Finally, we explain the central problem we have chosen to pursue in this thesis (Sections 3.3) and describe the methods we will use to solve that problem (Section 3.4).

3.1 Initial NMR states for entanglement

3.1.1 Entangling effective pure states

Current NMR experiments rely on thermal equilibrium at room temperature to set the value of ϵ – the excess population in the ground state in ρ_{eff}. The resulting ϵ is small and gives an effective pure state that is nonentanglable as a short calculation now shows.

We have already found this result in Chapter 2, but it will be useful to re-derive the bounds on ρ_ϵ in terms of α. Recall from Sections 2.3.2 and 2.4 that the fraction of ground state in the effective pure state is given by

$$\begin{aligned}\epsilon &= \frac{e^{N\alpha/2}}{\mathcal{Z}} - \frac{1 - e^{N\alpha/2}/\mathcal{Z}}{2^N - 1}, \text{ exact} && (3.1)\\ \epsilon &\approx \frac{N\alpha/2}{2^N - 1}, \text{ room temperature} && (3.2)\end{aligned}$$

and the bounds on the entanglability of effective pure states are

$$\begin{aligned}\epsilon &\leq \frac{1}{2^{N/2-1}(2^N - 1)} \Rightarrow \rho_{\text{eff}} \text{ nonentanglable} && (3.3)\\ \epsilon &> \frac{1}{1 + 2^{N/2}} \Rightarrow \rho_{\text{eff}} \text{ entanglable}\,. && (3.4)\end{aligned}$$

The reader should understand that "bounds on near-maximally mixed states" are equivalent to "bounds on entanglability of effective pure states" since ρ_{eff} has form ρ_ϵ and unitary operations preserve the form of ρ_ϵ. The term "entanglability" removes reference to the NMR unitary operation that performs $\rho_{\text{eff}} \mapsto \rho_\epsilon$. For Eq. 3.3, we have used the bound due to Gurvits and Barnum (Eq. 2.95) since it is tighter than the Braunstein et. al. bound (Eq. 2.74).

Using the above two sets of equations, we derive conditions that α must satisfy for ρ_{eff} to be nonentanglable and entanglable. Notice that in the nonentanglable case, $\alpha \ll 1$, so the room temperature approximation for ϵ holds. The approximation does not apply to the entanglable case since α is not necessarily small. Setting Eq. 3.3 equal to Eq. 3.2 and Eq. 3.4 equal to Eq. 3.1, we obtain the bounds

$$\begin{aligned}\alpha &\leq \frac{2}{N2^{N/2-1}} \Rightarrow \rho_{\text{eff}} \text{ nonentanglable} && (3.5)\\ \alpha &> -\ln(\sqrt{2} - 1) \Rightarrow \rho_{\text{eff}} \text{ entanglable}\,, && (3.6)\end{aligned}$$

which are plotted in Fig. 3-1. The nonentanglable bound goes as $\alpha \sim N^{-1}2^{-N/2}$, and the entanglable bound is constant with N. Notice that these bounds are far from

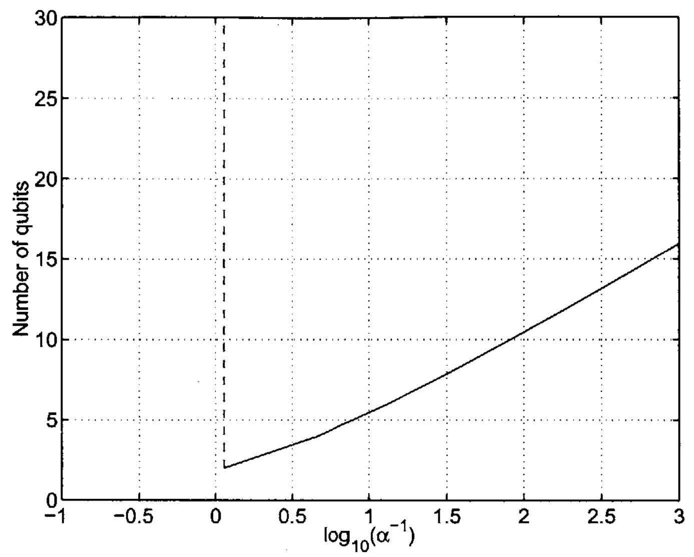

Figure 3-1: Bounds on the nonentanglability (solid line) and entanglability (dashed line) of the effective pure state ρ_{eff} in $N-\alpha$ parameter space where N specifies the number of qubits and α characterizes the polarization of the thermal state. If α is right of the solid line, ρ_{eff} is nonentanglable; if α is left of the dotted line, ρ_{eff} is entanglable.

tight. There is a large area of parameter space between the two bounds for which we have no information; this region might very well contain additional thermal states that can be entangled.[1]

In a typical NMR spectrometer operating with a 11.74 T magnetic field at room temperature, the largest attainable polarization is that of the proton nuclear spin: $\alpha = 8.79 \times 10^{-5}$. Fig. 3-1 shows that an effective pure state with this value of α is separable for $N \leq 22$, agreeing with our calculation from Section 2.4.2. Current NMR quantum computing experiments lie in the separable regime, but if α and/or N were sufficiently increased, an effective pure state could be entangled in the laboratory.

3.1.2 Entangling thermal states

Most NMR quantum computing experiments begin with effective pure states because quantum algorithms use a known initial pure state. However, we are simply interested

[1]In Chapter 6, we give a better bound for the entanglability of effective pure states due to Dür and Cirac [DC00].

in obtaining an entangled state by any means. The thermal state is a better initial state for reasons we now explain.

Recalling the discussion from Section 2.3.2, an effective pure state can be expressed as a mixture of transformed thermal states:

$$\rho_{\text{eff}} = \sum_{j=1}^{d-1} p_j U_j \rho_{\text{th}} U_j^\dagger \tag{3.7}$$

where d is the dimension of ρ_{eff}, $p_j = 1/(d-1)$ and U_j is a unitary operation that cyclically permutes the populations of the excited computational basis states $|i\rangle \neq |0\rangle$ $j-1$ times.

Since the numbers p_j are probabilities, ρ_{eff} satisfies the mathematical definition of a convex combination [Wei02]. Now entanglement is generally a convex function[2] and therefore is reduced under convex combination. Hence, an individual transformed thermal state $U_j \rho_{\text{th}} U_j^\dagger$ may possess more entanglement than an effective pure state of the same dimension. Eq. 3.7 shows that ρ_{eff} may be interpreted as a probabilistic distribution of transformed thermal states. We roll a $d-1$ sided die and obtain the state $U_j \rho_{\text{th}} U_j^\dagger$ with probability p_j. The introduction of this randomness dilutes the entanglement of the system, an undesirable procedure for our objective.

These considerations motivate us to focus on entangling NMR *thermal states* in this thesis. In using thermal states over effective pure states, we hope to uncover entanglable NMR parameter space that was previously ruled out by the bound of Eq. 3.5.

[2]Most theorists choose entanglement measures to be convex. Convexity is desirable in entanglement because if two parties randomly select a state from an ensemble (the situation of a mixed state), the average entanglement the parties possess when they have knowledge of the state should be equal or greater than the entanglement when they have no knowledge of the state.

3.2 Experimental approaches to entangling NMR thermal states

We wish to entangle an NMR thermal state in the laboratory. There are two parts to such an experiment.

1. Prepare the initial thermal state ρ_{th} with number of qubits N and polarization α.

2. Apply a unitary operation to entangle the thermal state: $\rho_{\text{th}} \mapsto U\rho_{\text{th}}U^{\dagger}$.

Let us first focus on Step 1. There are several ways we can alter the parameters to yield an entanglable thermal state:

1. Enhance the initial polarization: $\alpha \uparrow$

   - Lower the temperature of the experiment: $T \downarrow$
   - Increase the strength of the magnetic field: $B \uparrow$
   - Induce polarization above thermal equilibrium by non-magnetic means.

2. Increase the number of qubits (spins) per molecule: $N \uparrow$

Recall that $\alpha \equiv h\nu/kT$ where the resonance frequency $\nu = \gamma B$ and γ is the gyromagnetic ratio (see Section 2.3.1).

We expect that increased polarization will yield a thermal state that is more easily entangled. Doing so moves the thermal state away from the maximally mixed state. Raising the number of qubits should have the same effect, but the intuition behind this conjecture is weaker. As we have already seen, the bound on nonentanglable effective pure states in Eq. 3.5 relaxes as N increases. Moreover, current research seems to indicate that the volume of separable states decreases as Hilbert space dimension grows. Vidal and Tarrach [VT99] find a lower bound for the volume of separable states which decreases exponentially with N. Numerical evidence (see for instance Życzkowski, et. al. [dHSL98]) supports the same trend.

We now examine the viability of each approach. Some of the discussion is drawn from a review article by Jones [Jon00]. As a starting point, we will assume that the ideal state-of-the-art liquid-state NMR quantum computing experiment uses a seven proton nuclear spin molecule at room temperature. The corresponding parameters are $N = 7$, $\alpha = 8.79 \times 10^{-5}$, $B = 11.74$ T, and $T = 298$ K.

3.2.1 Enhancing initial polarization

Lowering the temperature of the experiment is impractical because any appreciable improvement in polarization demands a dramatic drop in temperature, to the point where the sample becomes solid. For instance, boosting α by an order of magnitude corresponds to $T \approx 27$ K. The crystalline structure of a solid allows spins to interact with each other through strong magnetic dipole couplings. Unless the spin system is dilute, these interactions break the massive degeneracy in the previously weakly coupled liquid-state system and smear out the spectrum. A dilute spin system could be used, but we ideally desire large N. There are several proposals for solid-state NMR quantum computers that address these problems [YY99, CLK$^+$00]. However, since these proposals have not yet been realized, we concentrate on liquid-state NMR in this thesis.

Using a larger magnetic field as a way to increase initial polarization is also limited. In NMR spectrometers, the magnetic field is generated from a superconducting coil. However, the coil can only sustain a limited amount of current and thus the largest available NMR magnet provides a 21.1 T field, which would double α. Technologies for delivering higher magnetic fields exist [BLM01]. Unfortunately, they are impractical and ill-suited for liquid-state NMR. DC Bitter resistive magnets can generate fields of 33-45 T, but an enormous apparatus is required to cool the setup. Capacitively-driven magnets can give pulsed fields of 50-60 T, but the pulses only last for a split second, much too short of a time scale for typical NMR experiments. Moreover, these magnets explode after 500-800 pulses. For both the resistive and capacitively-driven magnets, the resulting fields are too spatially inhomogeneous to be used in liquid-state NMR. In addition, these type of magnets require so much energy to run that they are only operated at special institutions like the National High Magnetic Field Laboratory in Florida [Nhm03].

Inducing higher polarization by other means, in particular chemical and optical, is a more promising approach. Two experiments have succeeded in attaining significantly higher polarization. By optically pumping xenon, Verhulst et. al. [VLS+01] have experimentally demonstrated a tenfold polarization increase in a two qubit molecule. Another factor of three or four may be possible if the xenon is isotropically pure. Thus, optical pumping can potentially give an overall polarization increase of 40. Even higher polarizations are possible if hyperpolarized xenon is used. Verhulst et. al. used 1% polarized xenon, but polarizations in excess of 65% have recently been achieved [ZAB02]. Hübler, et. al. [HBG00] have shown that *para* hydrogen (two protons in the singlet state) can be used to increase the polarization of the thermal state. By reacting *para* hydrogen with another molecule, they obtained four orders of magnitude improvement in α compared to thermal equilibrium.

3.2.2 Increasing number of qubits

Raising the number of qubits is also a possibility, but this approach is severely limited by substantial fundamental problems. Since individual spins are addressed by selective excitation (see Section 2.3.3), the resonance frequencies corresponding to the individual spins must be be well-separated. This requirement can be satisfied by using nuclear spins of different chemical species. Unfortunately, only five distinct spin-1/2 nuclei are available (^1H, ^{13}C, ^{15}N, ^{19}F, and ^{31}P). Spins of the same type may also be used, but the largest number of homogeneous spins demonstrated in an experiment thus far is 6 [LKF99]. To compound the problem, when N increases, the molecule size also increases, reducing the period over which the quantum evolution of the spins is coherent.[3] The diminished coherence time, in turn, decreases the resolution of the signal and the number of quantum operations that can be performed during a single

[3]The NMR Hamiltonian also contains strong couplings between spins which are mediated by the magnetic dipole-dipole interaction. These interactions are usually spatially averaged out by the tumbling motion of the molecules in liquid. However, when the molecule size increases, the tumbling rate decreases and the dipolar coupling is no longer completely averaged away.

experiment. Taking these considerations together, the largest practical liquid-state NMR quantum computer might contain 30 qubits.

3.2.3 Algorithmic cooling

Another method, the Schulman-Vazirani procedure [SV99], combines both of the previous approaches – enhancing the polarization and increasing the number of qubits. Schulman and Vazirani propose to start with N_0 qubits in thermal equilibrium at polarization α_0 and extract from these initial qubits a string of order $(\alpha_0)^2 N_0$ qubits each with $p_0 > 1 - \frac{1}{2} N_0^{-10}$ where p_0 is the probability of measuring $|0\rangle$. This method gives excellent polarization but at a high cost for the polarizations used in current NMR experiments. When the initial polarization is $\alpha_0 = 8.79 \times 10^{-5}$, we need approximately 1.29×10^8 initial qubits to obtain just one qubit in (nearly) pure state form. The enormous number of initial qubits are needed because the Schulman-Vazirani procedure preserves the entropy of the system. If entropy conservation is disregarded, better results are possible. Boykin et. al. [BMR$^+$02] propose a scheme where there is a set of spins, in addition to the qubits, that act as a heat bath. This method can be used to create an effective pure state with a initial to final qubit ratio of 50.

If α can be boosted with some of the techniques we previously mentioned, then the Schulman-Vazirani procedure becomes practical. For example, beginning with $N_0 = 30$ and $\alpha_0 = 0.183$, we obtain one highly polarized qubit.

3.2.4 Entangling unitary operations

We now turn to the second step of our entangling NMR experiment. Even though ρ_{th} is clearly separable, we can apply a unitary transform to move ρ_{th} from the separable part of Hilbert space into an entangled region. Currently, it is not known what unitaries might optimally entangle mixed states except for the special case of two qubits [VABM01].

However, progress has been made in related-research in entangling operations, in particular, the ability for a quantum operation or Hamiltonian to create entanglement, the communication resources needed to implement a bipartite unitary operation, and the quantity of classical information that can be communicated using quantum operations.[4] One group has even proposed general quantitative measures to characterize the strength of entangling operators [NDD$^+$02].

3.3 Problem

The last section discussed three approaches to creating an entangled thermal state in NMR. In this thesis, we follow the last approach and concentrate on the problem:

$$\boxed{\textit{Where are the entanglable thermal states in } N-\alpha \textit{ parameter space?}}$$

If we find entanglable parameter space that is experimentally accessible (where N and α are small), we can design an experiment to entangle a thermal state. If we do not, we gain insight into how much the experimental techniques must improve.

Before we can begin to solve the problem, it must be refined into a concrete, answerable question. Specifically, two issues must be addressed:

1. How do we know if a given thermal state is entanglable? In principle, for every point in parameter space, we must apply every possible unitary transform until we find one that entangles the thermal state. We need to choose a limited set of unitaries $\{U(N,\alpha)\}$ to try.

2. How do we know that a transformed thermal state is entangled? We need a measure of mixed state entanglement.

We now describe the set of unitaries and the measure of mixed state entanglement used in this thesis as well as the rationale behind selecting them.

[4]See the first section of Nielsen, et. al.. [NDD$^+$02] for references.

3.3.1 Unitary operations

The set of unitaries $\{U(N, \alpha)\}$ should ideally possess the following qualities:

- Optimally entangling - $\{U(N, \alpha)\}$ should entangle thermal states in the maximum fraction of parameter space that is most experimentally accessible, essentially the region where N and α are small.

- Scalable - $\{U(N, \alpha)\}$ should be at least polynomially describable and realizable.

- Tractable to theoretical analysis - Analytical information about the transformed thermal state provides physical intuition and verifies numerical entanglement calculations or eliminates the need for them altogether.

There are clearly infinite choices for sets of unitaries, but we select the family of transformations U_{b} that maps the computation basis to the Bell state basis. For example, a Bell state unitary might transform a computational basis state $|000\rangle$ to the Bell state $(|000\rangle + |111\rangle)/\sqrt{2}$. We have chosen U_{b} as a starting point because it generates maximal entanglement when applied to the ground state. The thermal state has the greatest population in the ground state, so U_{b} may also be effective in entangling ρ_{th}.

In this thesis, we define a standard Bell state unitary $U_{\mathrm{b,s}}$. The other unitaries in the Bell transformation family are permuted versions of $U_{\mathrm{b,s}}$:

$$U_{\mathrm{b}}(P) = P U_{\mathrm{b,s}} P^{\dagger} \tag{3.8}$$

where P is a permutation matrix.

The operation $U_{\mathrm{b,s}}$ can be represented by a quantum circuit that first applies a Hadamard gate to the first qubit and then uses the first qubit to control NOT operations on the other $N-1$ qubits [NC00]. This circuit is depicted in Fig. 3-2. The symmetry in the circuit reveals that $U_{\mathrm{b,s}}$ is invariant under exchange among the latter $N-1$ qubits.

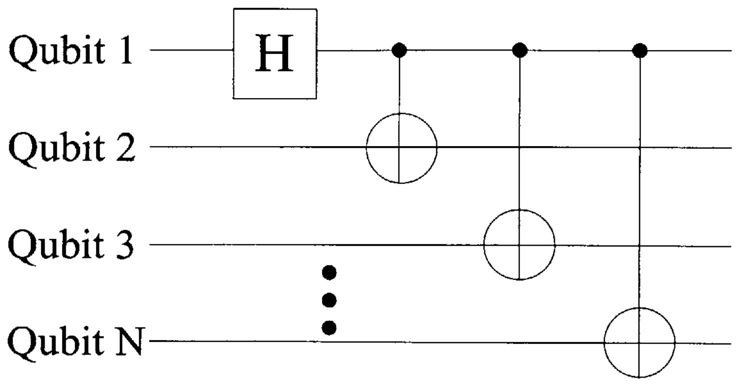

Figure 3-2: Quantum circuit for $U_{\mathrm{b,s}}$ unitary operation.

The unitary $U_{\mathrm{b,s}}$ also has a simple matrix form. For example, $U_{\mathrm{b,s}}$ for $N = 3$ is

$$U_{\mathrm{b,s}} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \end{bmatrix}, \tag{3.9}$$

expressed in the computational basis.

In Chapter 6, we will encounter another Bell unitary, which can be generated by permuting $U_{\mathrm{b,s}}$ with U_{fan}. The unitary U_{fan} is also known as the fanout gate because it bit flips the latter $N-1$ qubits if the first qubit is in state $|1\rangle$. The fanout gate can be viewed as a massive NOT gate on the latter $N-1$ qubits controlled by the first qubit, exactly like $U_{\mathrm{b,s}}$ but without the initial Hadamard gate on the first qubit. The overall unitary $U_{\mathrm{b,s}}U_{\mathrm{fan}}$ is depicted in Fig. 3-3.

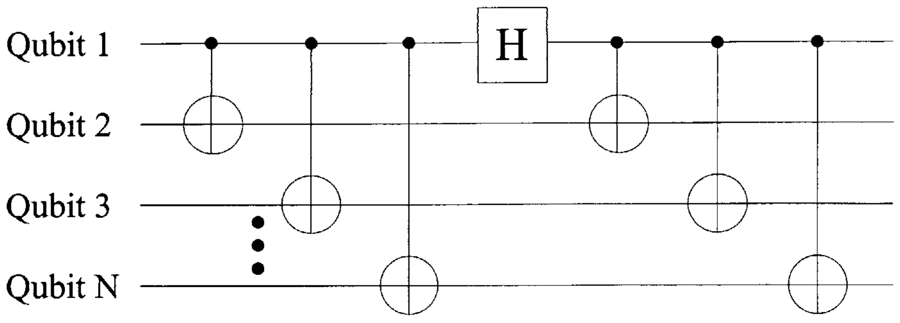

Figure 3-3: Quantum circuit for $U_{\mathrm{b,s}}U_{\mathrm{fan}}$ unitary operation.

The unitary $U_{\mathrm{b,s}}U_{\mathrm{fan}}$ has a matrix form similar to $U_{\mathrm{b,s}}$. For example, $U_{\mathrm{b,s}}U_{\mathrm{fan}}$ for $N = 3$ is

$$U_{\mathrm{b,s}}U_{\mathrm{fan}} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}, \tag{3.10}$$

expressed in the computational basis.

It is easy to see that in the computational basis, $U_{\mathrm{b,s}}$ and $U_{\mathrm{b,s}}U_{\mathrm{fan}}$ always have two diagonals and two anti-diagonals. The straightforward pattern allows a computer program to quickly generate either Bell unitary for any N. Hence, they fulfill the scalability requirement. As we shall see in Chapter 6, any generic Bell unitary U_{b} also satisfies the criterion for analytical tractability.

3.3.2 Measure of mixed state entanglement

We need a way to quantify how well a given unitary entangles thermal states. Recall from the discussion in Section 2.1.5 that negativity is the only computable measure of mixed state entanglement for density matrices with dimension greater than four. Since N may be arbitrarily large, we choose negativity to be our measure of entanglement.

If the negativity varies monotonically in parameter space, then the value where $\lambda_{\text{min}} = 0$ bears special significance. It marks the boundary in parameter space between states that are known to be entangled and states whose separability properties are unknown.

Finally, we explain our notation in specifying bipartite splits. The thermal state is totally symmetric, and the unitary transforms we consider in this thesis are symmetric among the latter $N-1$ spins. Therefore, it is sufficient to assign the first q spins to one party and the remaining $N-q$ spins to another party. We write such a bipartite split as $\{q, N-q\}$. In general, we would also have to consider all permutations of the spins.

3.4 Methods

We use both numerical and analytical methods to study thermal state entanglability in NMR parameter space. The numerical part of this work is the subject of Chapter 4 while the analytical portion is the subject of Chapters 5 and 6.

3.4.1 Numerical methods

Our numerical approach is to generate *negativity maps* of Bell-transformed thermal states in N-α parameter space, using the following procedure:

1. Compute the thermal state ρ_{th} for given N and α.

2. Apply the generalized Bell-state transformation: $\rho_{\text{th}} \mapsto U_{\text{b}}\rho_{\text{th}}U_{\text{b}}^{\dagger}$. We define the Bell transformed state as $\rho_{\text{tf}} \equiv U_{\text{b}}\rho_{\text{th}}U_{\text{b}}^{\dagger}$.

3. Calculate the negativity of the transformed state $E_n(\rho_{\text{tf}})$.

4. Repeat the above for all points in N-α space.

Numerical analysis is an excellent means to explore arbitrary unitary transforms, but the calculations quickly become memory-intensive and time-consuming as the number of qubits grows. When N increases by one, the size of a $2^N \times 2^N$ density matrix increases by four. The exponential scaling in required memory resources makes large qubit computations cost-prohibitive. The computation time of the negativity calculation also grows exponentially. Matrix multiplication must be performed for the unitary transformation and the minimum eigenvalue calculations. The fastest known algorithm for $d \times d$ matrix multiplication is $\mathcal{O}(d^c)$ where $c = 2.3755$ [CW90]. The number of steps needed to implement the partial transpose operation also increases exponentially with the size of Party A's Hilbert space.

For these reasons, we search for ways to reduce the computational resources needed. In some cases, the matrices involved in the computation are sparse. Then we need only store the nonzero elements and their coordinates in the matrix. For our particular problem, the number of nonzero elements in ρ_{th} and U_{b} scales linearly, a vast improvement over the previously mentioned exponential growth in required memory resources. It may also be possible to exploit symmetry if parts of the full density matrix are invariant under particular operations. In these cases, we only have to work with the active part of the matrix.

The matrices in our problem, the thermal state ρ_{th} and the family of Bell unitaries U_{b}, are always sparse in the computational basis. In addition, both ρ_{th} and U_{b} possess substantial symmetry. As we have already seen, ρ_{th} is diagonal in the computational basis and totally symmetric under spin exchange while $U_{\text{b,s}}$ and $U_{\text{b,s}}U_{\text{fan}}$ are invariant under exchange among the latter $N-1$ of the qubits. However, the partial transpose operation poses a problem because it can break the symmetry of the transformed thermal state, depending on the bipartite split.

3.4.2 Analytical methods

Numerical methods are constrained by finite memory resources. However, analytical methods can be used if the negativity calculation possess a high degree of symmetry or if entanglement of the state of interest can be bounded by the entanglement of another state.

As we shall see in Chapter 5, the symmetries of the standard Bell-transformed thermal state $U_{\text{b,s}}\rho_{\text{th}}U_{\text{b,s}}^{\dagger}$ can be exploited to find explicit negativity formulas. Sometimes, the symmetry is obvious (Section 5.1) while in other cases, patterns in numerical data can reveal a buried symmetry of the system (Section 5.2).

Chapter 6 uses a more general method to find bounds on the separability and nonseparability of transformed thermal states. Suppose we have a state ρ' whose entanglement is easy to analytically classify. However, it is difficult to analytically determine whether a transformed thermal state is entangled. If we can find a quantum operation that converts the transformed thermal state into ρ' without generating any new entanglement, then the entanglement of ρ' bounds the entanglement of the transformed thermal state.

3.5 Summary

This chapter described the problem we will pursue in this thesis and the approach that will be used to solve this problem. Before defining our problem, we discussed two important considerations for realizing an entangled NMR state: 1) the initial NMR state and 2) the experimental approaches to entangling an NMR state.

We showed that the thermal state may be easier to entangle than the effective pure state, the conventional initial state used in liquid-state NMR quantum computation. An effective pure state can be expressed as a convex combination of transformed thermal states. Since entanglement is reduced under convex combination, the maximally entangled thermal state must possess at least as much entanglement as the corresponding maximally entangled effective pure state. Therefore, we decided to focus on thermal states in this thesis.

We also discussed three approaches to experimentally entangle a thermal state: 1) enhancing the initial polarization α, 2) increasing the number of qubits N, and 3) applying a unitary operation U to entangle the state. A promising method to increase α was to induce higher polarization via optical pumping or *para* hydrogen. Combining these techniques with algorithmic cooling could yield nearly pure qubits. In contrast, the possibilities for raising the number of qubits were limited and impractical.

In this thesis, we decided to study the third approach, specifically exploring the problem:

Where are the entanglable thermal states in N-α parameter space?

To investigate this question, we chose to study the entanglement of Bell-transformed thermal states. The Bell transforms were given by the set of permuted unitaries

$$U_{\mathrm{b}}(P) = PU_{\mathrm{b,s}}P^{\dagger} \tag{3.11}$$

where P is a permutation matrix and $U_{\mathrm{b,s}}$ is the standard Bell unitary whose quantum circuit diagram is shown in Fig. 3-2. Our measure of entanglement was negativity, as defined in Eq. 2.36.

The entanglement of Bell-transformed thermal states can be classified with either numerical or analytical methods. Here we list the methods used in this thesis:

1. Compute negativity maps of N-α space (numerical).

2. Exploit the symmetry of Bell-transformed thermal states to derive explicit negativity formulas (analytical).

3. Bound the entanglement of a given transformed thermal state with the entanglement of another state (analytical).

Chapter 4

Numerical negativity maps of Bell-transformed thermal states

This chapter focuses on numerical negativity calculations in N-α space performed on either a single processor or a parallel machine. First, we outline the general algorithm that is used to compute a negativity map (Section 4.1). Next, we describe the hardware specifications of our single processor and our parallel machine – a Beowulf cluster (Section 4.2). We then discuss the single processor and Beowulf cluster software packages, how the different modules and/or architectural layers interact, and how the algorithm is implemented in both packages (Sections 4.3 and 4.4). Finally, we calculate negativity and minimum eigenvalue maps on a single processor for up to 12 qubits and evaluate the results (Section 4.5).

4.1 Algorithm

The calculation of a negativity map follows the algorithm:

1. Generate a Bell unitary U_{b} for given N.

2. Compute the thermal state ρ_{th} for given N and α.

3. Transform the thermal state: $\rho_{\mathrm{th}} \mapsto \rho_{\mathrm{tf}} = U_{\mathrm{b}} \rho_{\mathrm{th}} U_{\mathrm{b}}^{\dagger}$.

4. Perform the partial transpose on ρ_{tf} to yield $\rho_{\text{tf}}^{T_A}$.

5. Find the minimum eigenvalue λ_{min} of $\rho_{\text{tf}}^{T_A}$. Store this value, so the information can be used later to calculate the negativity E_n.

6. Repeat the above for all points in N-α space.

The algorithm saves the intermediate minimum eigenvalue because λ_{min} by itself contains useful information and because E_n is trivially derived from λ_{min} via Eq. 2.36. Note that since fractional qubits have no physical meaning, N must be computed at positive integer values. In this thesis, the even split $\{N/2, N/2\}$ bears special interest, so we only compute negativity for even positive integer values of N.

4.2 Hardware

Negativity calculations are implemented on either a single processor or a massive parallel machine. The single processor is a 997 MHz Pentium III PC that has 879 MB RAM and runs LINUX as its operating system. The massive parallel machine consists of 32 dual-processor 1.2 GHz Pentium III PCs also running LINUX. Each PC stores and manipulates a portion of the overall matrix. Of the 32 PCs, 24 computers each have 1 GB RAM and 32K cache while the other 8 computers each have 768 MB RAM. The individual PCs communicate with one another across a 1000BaseT Ethernet fabric using TCP/IP protocol. This type of setup – off-the-shelf CPUs linked together for parallel operation – is generically called a *Beowulf cluster*.

As we discussed in Section 3.4.1, the negativity calculations are constrained by memory resources – the amount of RAM available. The memory bottleneck is matrix multiplication, an operation that requires the simultaneous storage of three matrices: the two factors and the product. Floating point arithmetic typically uses 8 byte doubles. A N-qubit matrix contains $2^N \times 2^N$ doubles for a total memory usage of 2^{2N-17} MB. Therefore, with our setup, a single processor can compute negativity up to $N = 12$ while the Beowulf cluster can calculate negativity up to $N = 15$.[1]

[1]For a $2^{15} \times 2^{15}$ matrix, the memory required per processor is 2^{13} MB / matrix $\times$ 3 matrices /

These numbers assume that the calculations do not optimize for sparse matrices or symmetry.

4.3 Single-processor software

This section describes the numerical software package for calculating negativity maps on a single processor. First, we explain the structure of the package and give an overview of how the negativity map calculation is implemented. Next, we describe the data structure that stores the results of the negativity calculation and the implementation details of each module. Finally, we evaluate the memory usage and run time performance of the package. For the source code, see Appendix A.

4.3.1 Package structure and implementation

The single-processor software package consists of three major module types: execution, calculation, and post-processing. The package structure is depicted in the module dependency diagram of Fig. 4-1. The arrows in the module dependency diagram indicate the flow of execution. The modules are written in LINUX-based MATLAB v6.1 [Mat01] and executed on the same platform. We chose MATLAB because it is easy to program and because it contains many built-in functions for plotting, efficient linear algebra operations, and numerical optimization. The M-file extension `.m` specifies a file as a MATLAB function.

We now explain how the modules interact to implement the negativity map calculation:

1. The execution module `runexpt.m` runs a "numerical experiment" to calculate λ_{min} in user-specified N-α space. It calls the master calculation module `calcmineig.m` for every value of N.

2. The module `calcmineig.m` computes λ_{min} for all values of α while holding N fixed. Since U_{b} only depends on N, `calcmineig.m` generates U_{b} once for

multiplication $\times$ 1 / 32 processors = 768 MB RAM required per processor.

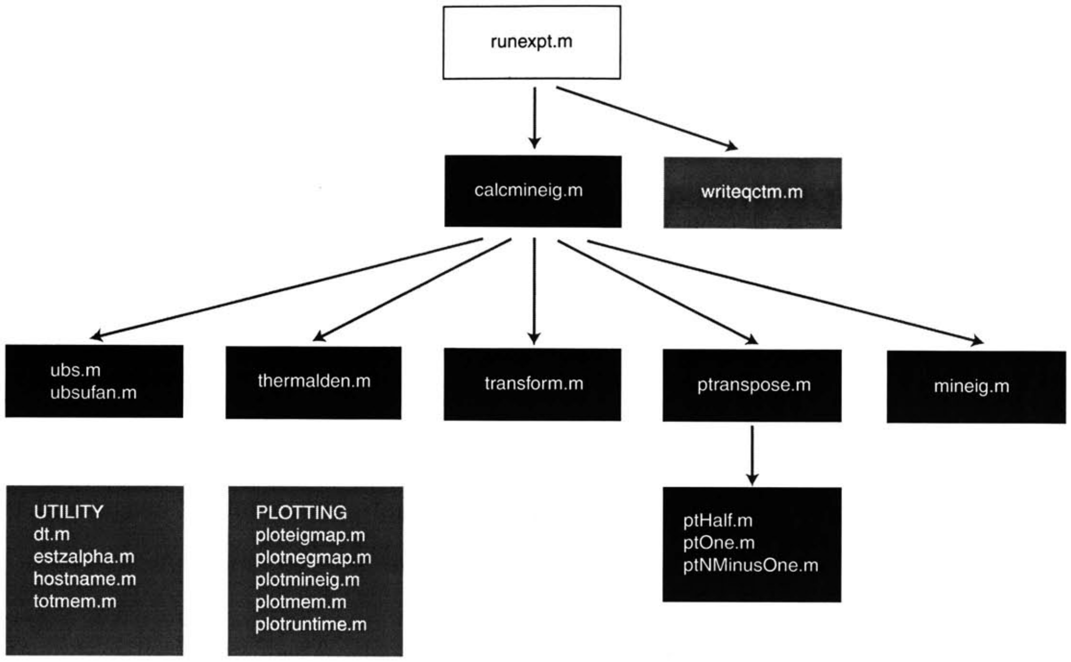

Figure 4-1: Module dependency diagram for single-processor software package. Module types are execution (white), calculation (black), and post-processing (medium gray).

the entire computation over α. At present, the package can either create $U_{\mathrm{b,s}}$ (`ubs.m`) or $U_{\mathrm{b}}U_{\mathrm{fan}}$ (`ubsufan.m`). Then the algorithm of Section 4.1 is performed for each value of α:

(a) The submodule `thermalden.m` generates the thermal density matrix ρ_{th} for given N and α.

(b) The submodule `transform.m` executes the matrix multiplication $U_{\mathrm{b}}\rho_{\mathrm{th}}U_{\mathrm{b}}^{\dagger}$ to give the transformed thermal density matrix ρ_{tf}.

(c) The submodule `ptranspose.m` carries out the partial transpose on the transformed density matrix for a specified bipartite split $\{q, N-q\}$ to give $\rho_{\mathrm{tf}}^{T_A}$. The splits currently implemented are $\{N/2, N/2\}$ (`ptHalf.m`), $\{1, N-1\}$ (`ptOne.m`), and $\{N-1, 1\}$ (`ptNMinusOne.m`).

(d) The submodule `mineig.m` finds the minimum eigenvalue λ_{min} of $\rho_{\mathrm{tf}}^{T_A}$.

After `calcmineig.m` finishes computing $\lambda_{\min}$ for all values of α, it records $\lambda_{\min}$ and other relevant information in a temporary structure.

3. Now the negativity calculation is complete for one value of N and `runexpt.m` calls the post-processing module `writeqctm.m`, which writes the temporary structure plus some additional information to a QCTM data structure (explained in the next section) on disk.

4. Repeat Steps 2-3 for each value of N.

5. After $\lambda_{\min}$ has been calculated for all points in N-α space, `runexpt.m` combines all the QCTM structures into a structure array, which is also written to disk.

The intermediate writing of QCTM structures is done at individual values of N so that if the numerical experiment is interrupted, at least part of the calculation will be saved. For low-level utility tasks and plotting, the package includes other post-processing functions that will be described in Section 4.3.5.

To evaluate the performance of our software, we also collect memory usage and run time data from the calculation submodules. The memory usage is obtained by running the MATLAB function `whos.m` inside a submodule, and the run time is clocked by the command sequence: '`tic; submodule_operations; t=toc;`'. The MATLAB command `tic` begins the timer; `toc` ends the timer and returns the elapsed time to the variable `t`.

4.3.2 QCTM data structure

The Quantum Computing Treasure Map (QCTM) structure contains all the information needed to reconstruct a $\lambda_{\min}$ calculation at fixed N.

Fields

- Filename: string, which always has the format '<user-specified string><date>q <number of qubits>.mat', e.g. '`ubshalf28Jun03-134322q4.mat`'. The user-specified part of the filename is optional.

- Date of file: string, e.g. '`28Jun03-134322`', which should be read as time 13:43:22 on 28 June 2003.
- Hostname of processor that ran the calculation: string, e.g. '`turing.media.mit.edu`'.
- Name of unitary: string, e.g. '`ubs`'.
- Number of qubits: positive even integer.
- Number of qubits in Party A's partition: positive integer between 1 and $N-1$.
- Range in α: array of positive doubles, e.g. $(10^{-3.0}, 10^{-2.9}, 10^{-2.8}, ..., 10^{0.9}, 10^{1.0})$.
- Minimum eigenvalue λ_{min}: array of doubles corresponding to range in α.
- Run time of λ_{min} calculation in seconds: positive double.
- Submodule data: structure storing information about each calculation submodule with fields listed below.
  - Name of submodule: string, e.g. '`thermalden`'.
  - Memory usage in bytes: array of positive integers corresponding to range in α.
  - Run time in seconds: array of positive doubles corresponding to range in α.

4.3.3 Execution module

`runexpt.m`

Function

This module computes λ_{min} in N-α space, calling `calcmineig.m` to perform the calculation for sections of parameter space at each value of N. After calculating λ_{min} for each N, it calls `writeqctm.m` to save λ_{min} and other relevant data inside a QCTM structure, which is written to disk. At the end of the entire calculation, it merges all the QCTM structures into an array, which is also written to disk.

Inputs

- Bell unitary: handle pointing to a submodule that generates the Bell unitary as a function of N.
- Range in N: even positive integer array, e.g. (2, 4, 6, 8, 10, 12).
- Range in α: array of positive doubles.
- Number of qubits in Party A's partition: positive integer array, e.g. for the example range in N given above, we could specify the $\{N-1,1\}$ split with the array (11, 9, 7, 5, 3, 1). This format requires that the size of the array specifying the partition must be the same as the size of the array specifying the range in the number of qubits. Each element in the partition array must be between 1 and $N-1$.

Output

Data from numerical experiment: array of QCTM structures.

4.3.4 Calculation modules

`calcmineig.m`

Function

The master calculation module is `calcmineig.m`, which calls various calculation submodules to find λ_{min} over a range of α values, at fixed N. First, it directs `ubs.m/ubsufan.m` to create the Bell unitary matrix and stores the matrix in a global variable. Recall that this unitary depends solely on N, so it only needs to be calculated once. Then for each value of α, it calls `thermalden.m`, `transform.m`, `ptranspose.m`, and `mineig.m` in sequential order. The output matrix of each submodule is passed onto the next submodule as input.

Inputs

- Bell unitary: handle pointing to a submodule that generates the Bell unitary as a function of N.

- Number of qubits (N): even positive integer.

- Range in α: array of positive doubles.

- Number of qubits in Party A's partition: positive integer between 1 and $N-1$.

Output

A temporary structure whose fields contain information for `writeqctm.m` to create a QCTM structure. The fields of the temporary structure are exactly the same as in the QCTM structure except there are no fields for filename, file date, or hostname.

`ubs.m/ubsufan.m`

Function

Both submodules construct a Bell unitary matrix in the computational basis for given N; `ubs.m` creates $U_{\mathrm{b,s}}$ and `ubsufan.m` creates $U_{\mathrm{b,s}}U_{\mathrm{fan}}$. In either case, the submodule first initializes the elements of a $2^N \times 2^N$ matrix to zero. Then it sets the diagonals and anti-diagonals to $\pm 1/\sqrt{2}$ (see Section 3.3.1).

Input

Number of qubits (N): positive even integer.

Outputs

- Unitary matrix: $2^N \times 2^N$ matrix of doubles.

- Name of unitary: '`ubs`'/'`ubsufan`'.

- Memory usage in bytes: positive integer.

- Run time in seconds: positive double.

`thermalden.m`

Function

This submodule constructs the thermal density matrix in the computational basis for given N and α. It calculates the diagonal entries of ρ_{th} according to Eq. 2.61 and places the values into an array. Then the MATLAB function **`diag.m`** creates a diagonal matrix with the diagonal entries derived from the array. The Hamming weight of the $\langle i| \rho_{\text{th}} |i\rangle$ matrix element is found by using the MATLAB function **`bitget.m`** to count the number of binary digits in i that are 0.

Inputs

- Number of qubits (N): positive even integer.
- Polarization (α): positive double.

Outputs

- Thermal density matrix ρ_{th}: $2^N \times 2^N$ matrix of doubles.
- Memory usage in bytes: positive integer.
- Run time in seconds: positive double.

`transform.m`

Function

This submodule transforms the input thermal density matrix by performing the matrix multiplication $U_{\text{b}}\rho_{\text{th}}U_{\text{b}}^{\dagger}$. Notice that U_{b} is not one of the inputs because it is stored as a global variable in the scope of **`calcmineig.m`**.

Inputs

Thermal density matrix ρ_{th}: $2^N \times 2^N$ matrix of doubles.

Outputs

- Transformed thermal density matrix ρ_{tf}: $2^N \times 2^N$ matrix of doubles.
- Memory usage in bytes: positive integer.
- Run time in seconds: positive double.

`ptranspose.m`

Function

This submodule calculates the partial transpose of a square matrix in the computational basis for either the $\{N/2, N/2\}$, $\{1, N-1\}$, or $\{N-1, 1\}$ splits by calling helper submodules `ptHalf.m`, `ptOne.m`, or `ptNMinusOne.m` respectively. At present, other splits have not been implemented. The submodule returns an error if the square matrix does not have a dimension that is a power of 2 or if the partition is invalid.

Inputs

- Transformed thermal density matrix ρ_{tf}: square matrix of doubles with dimension 2^N.
- Number of qubits in Party A's partition: positive integer between 1 and $N-1$.

Outputs

- Partial transposed matrix $\rho_{\mathrm{tf}}^{T_A}$: square matrix of doubles with dimension 2^N.
- Memory usage in bytes: positive integer.
- Run time in seconds: positive double.

`ptHalf.m/ptOne.m/ptNMinusOne.m`

Function

These helper submodules calculate the partial transpose of $2^N \times 2^N$ matrix for the $\{N/2, N/2\}$, $\{1, N-1\}$, or $\{N-1, 1\}$ splits respectively. As seen in Section 2.1.5, the partial transpose is simply a set of swap operations between matrix elements. For a split $\{q, N-q\}$, a swap only occurs when the outer product $|i\rangle\langle j|$ corresponding to Party A's subspace (the first q qubits) is off-diagonal, that is $i \neq j$. We now describe how each of the submodules computes the partial transpose.

The code in `ptHalf.m` actually implements the partial transpose with respect to Party B, but the change in convention does not affect the calculated $\lambda_{\min}$ since the $\{N/2, N/2\}$ split is symmetric in our problem. The submodule `ptHalf.m` examines the matrix elements where the outer product $|i\rangle\langle j|$ corresponding to the latter $N/2$ qubits is off-diagonal and has $i < j$ (so the same swap is not performed twice). For each off-diagonal outer product, the submodule cycles through the matrix elements corresponding to all possible indices for the first $N/2$ qubits and performs the appropriate swap operation for each matrix element.

For the $\{1, N-1\}$ split, the first qubit only has off-diagonal outer products in the upper righthand corner and the lower lefthand corner of the matrix. In fact, an element in the upper righthand corner always swaps with an element in the lower lefthand corner. Therefore `ptOne.m` swaps all the elements in the upper righthand corner of the matrix to calculate the partial transpose.

For the $\{N-1, 1\}$ split, the partial transpose exchanges elements between the lower triangle (under the diagonal) and the upper triangle (above the the diagonal) of the matrix. Thus **`ptNMinusOne.m`** examines just the elements of the lower triangle. Not all of these matrix elements have an off-diagonal outer product in Party A's subspace (the first $N-1$ qubits). Thus, in addition, the submodule must check to see if a swap is necessary.

Inputs

Same as **`ptranspose.m`**.

Outputs

Same as **`ptranspose.m`**.

Additional considerations

As noted in Section 3.4.1, the number of swap operations in the partial transpose grows exponentially with N. Consequently, it is essential to carry out each swap with as few steps as possible. Consider the MATLAB code for **`ptHalf.m`**:

```
function mpt = ptHalf(matin)

n = length(matin); % n = size of matrix
nhalf = sqrt(n);    % nhalf = size of one party's subspace
mpt = matin;        % mpt = partial transposed matrix
```

```
for a = 0:nhalf:n-1          % Row index of Party B's subspace
  for b = 0:nhalf:n-1        % Column index of Party B's subspace
    for j = 0:nhalf-1        % Row index of Party A's subspace
      for k = j+1:nhalf-1  % Column index of Party A's subspace
        mpt(a+j+1,b+k+1) = matin(a+k+1,b+j+1);  % Swap: line 1
        mpt(a+k+1,b+j+1) = matin(a+j+1,b+k+1);  % Swap: line 2
      end
    end
  end
end
```

Here `matin` is the input matrix and `mpt` is the partial transpose of `matin`. The indices of the swapped elements are sums, and it is evident from the code that each sum appears twice. For instance, `a+j+1` occurs in both lines of the swap. An efficient compiler usually recognizes this kind of redundancy and only computes each sum once. However, the MATLAB compiler apparently does not. If `ptHalf.m` is rewritten in the C programming language, we get a dramatic improvement in run time as seen in Fig. 4-2. When N=4, the C version `ptHalfC.c` runs 3.84 times faster than its MATLAB counterpart. The speedup increases to a factor of 142 at N=10.

`mineig.m`

Function

This submodule calculates the minimum eigenvalue of a square matrix. It calls the MATLAB function `eig.m` to compute all the eigenvalues of the matrix and then applies MATLAB function `min.m` to find the minimum of these numbers.

Input

Partial transposed matrix $\rho_{\text{tf}}^{T_A}$: square matrix of doubles.

Output

- Minimum eigenvalue λ_{min} of $\rho_{\text{tf}}^{T_A}$: double.

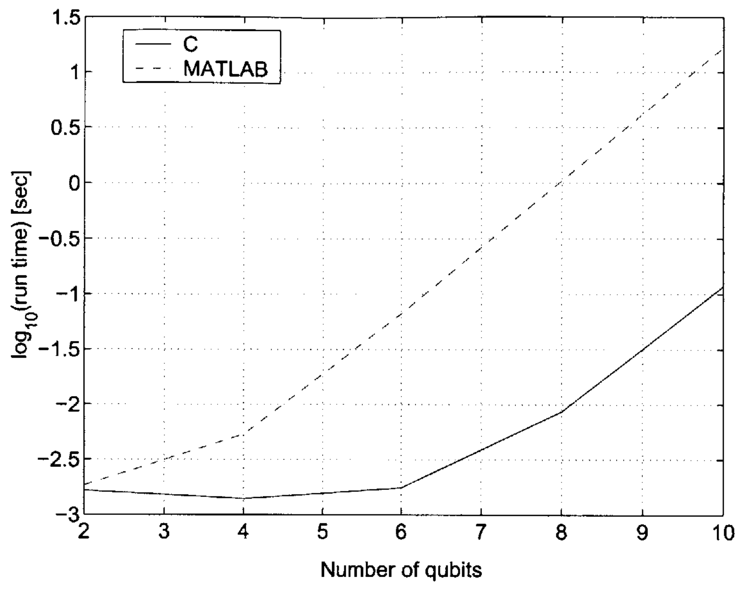

Figure 4-2: Comparison of mean single-processor run times for C and MATLAB version of partial transpose operation ($\{N, 2, N/2\}$ split). The run time data is acquired by executing the modules 100 times at each N.

- Memory usage in bytes: positive integer.
- Run time in seconds: positive double.

4.3.5 Post-processing modules

`writeqctm.m`

Function

This module takes the temporary structure produced by `calcmineig.m`, assigns it a filename, and writes a QCTM structure containing the temporary structure, filename, file date, and hostname of the processor where `writeqctm.m` was executed. This seems trivial, but we keep calculational tasks separate from post-processing tasks to allow easier maintenance of the package.

Inputs

- Information from `calcmineig.m`: structure (see Section 4.3.4).
- Filename (optional): string.

Output

Results of λ_{min} calculation at fixed N: QCTM structure.

Utility submodules

Here we list the utility submodules and their functions:

- `dt.m`: Runs UNIX `date` command to obtain current time.

- `estzalpha.m`: Estimates the α that corresponds to $\lambda_{\text{min}} = 0$, given a QCTM structure. Currently, this submodule is very primitive; it simply returns the α that has λ_{min} closest to 0.

- `hostname.m`: Runs UNIX `hostname` command to obtain the name of the processor where `hostname.m` is called.

- `totmem.m`: Adds up the memory used by variables within the scope of a function given a structure array that is created by calling the MATLAB function `whos.m` inside the function.

Plotting modules

Here we list the plotting modules and their functions:

- `plotnegmap.m`: Plots E_n as a two dimensional colormap with $\log_{10}(\alpha^{-1})$ on the x-axis and N on the y-axis,[2] given an array of QCTM structures. The negativity is calculated from λ_{min} by applying Eq. 2.36.

- `ploteigmap.m`: Plots λ_{min} as a two-dimensional colormap with $\log_{10}(\alpha^{-1})$ on the x-axis and N on the y-axis, given an array of QCTM structures. This module also gives the option to plot contours of λ_{min}.

- `plotmineig.m`: Plots λ_{min} in a line graph as a function of α for each value of N, given an array of QCTM structures.

[2]Throughout this thesis, we will graph NMR parameter space with N on the vertical axis and $\log_{10}(\alpha^{-1})$ on the horizontal axis. We use α^{-1} rather than α so that higher temperature lies towards the right side of the plot.

- `plotmem.m`: Plots the logarithm of the memory used by `ubs.m/ubsfan.m`, `thermalden.m`, `transform.m`, `ptranspose.m`, and `mineig.m` as a function of N.

- `plotruntime.m`: Plots the logarithm of the run times for `ubs.m/ubsfan.m`, `thermalden.m`, `transform.m`, `ptranspose.m`, `mineig.m`, and `calcmineig.m` as a function of N.

4.3.6 Package performance

Here we assess the memory usage and run time performance of the single-processor software package. The mean memory usage and run times of the calculation submodules are plotted in Figs. 4-3 and 4-4 respectively at $N = \{2, 4, 6, 8, 10, 12\}$. Here `ptranspose.m` uses `ptHalfC.c` to calculate the partial transpose. The data is collected by running each module 100 times and averaging over the iterations at each value of N. We also clock the run time of the master calculation module `calcmineig.m` for $U_{\mathrm{b,s}}$-transformed thermal states at $N = \{2, 4, 6, 8, 10, 12\}$ and 100 logarithmically spaced points in the range $\alpha = [10^{-3}, 10^{1}]$.

Our assessment ignores several submodules. First, `ubsufan.m` is not plotted since its performance is well-modeled by `ubs.m`. The helper submodules `ptOne.m` and `ptNMinusOne.m` are also left out of our study since we have not yet written efficient C versions of them.

The memory usage for all submodules should scale as 4^N since memory is primarily consumed by the storage of N-qubit matrices. Fig. 4-3 confirms this expectation. The submodule `transform.m` uses the most memory, followed by `ptranspose.m`. The submodules `ubs.m`, `thermalden.m`, and `mineig.m` use roughly the same amount of memory and consume the least memory among the submodules. This result can be explained by the fact that `transform.m` must store three N-qubit matrices simultaneously (U_{b}, ρ_{th}, and ρ_{tf}) while `ptranspose.m` holds two such matrices (ρ_{tf} and $\rho_{\mathrm{tf}}^{T_A}$) in memory. The submodules `ubs.m`, `thermalden.m`, and `mineig.m` each hold one matrix ($U_{\mathrm{b,s}}/\rho_{\mathrm{th}}/\rho_{\mathrm{tf}}^{T_A}$) in memory. Indeed, the memory usage of `transform.m`

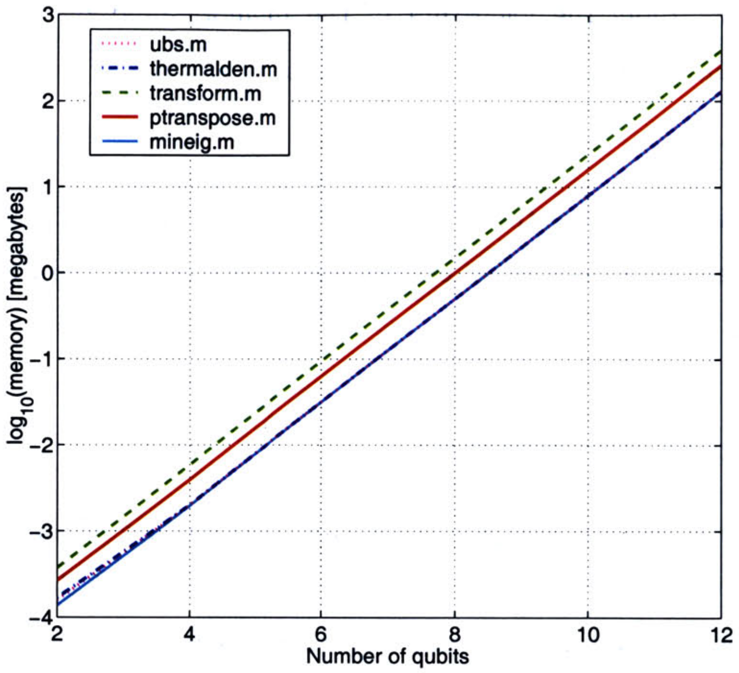

Figure 4-3: Mean single-processor memory usage as a function of N (ordered from top to bottom): **transform.m** (dashed green line), **ptranspose.m** (solid red line), **ubs.m** (dotted magenta line), **thermalden.m** (dash dotted blue line), and **mineig.m** (solid cyan line). The data is acquired by executing the modules 100 times at each N. The partial transpose is computed for the $\{N/2, N/2\}$ split, using **ptHalfC.c**.

is vertically offset from that of **thermalden.m/mineig.m** by $\log_{10} 3$, corresponding to a memory usage ratio of 3:1. Similarly, the separation between **ptranspose.m** and **thermalden.m/mineig.m** is $\log_{10} 2$. As a final verification, the memory used by **transform.m** for $N = 12$ is 384 MB, exactly the size of three $2^{12} \times 2^{12}$ matrices.

Now let us examine the run time performance of the package. In view of absolute run times at $N = 12$, the fastest submodule is **ubs.m**, followed by **thermalden.m**, **ptranspose.m**, **transform.m**, and **mineig.m**. These results are expected because **ubs.m** and **thermalden.m** create matrices whose nonzero entries scale linearly, whereas the other submodules perform matrix operations.

Based on the discussion in Section 3.4.1, we expect the run times to also scale exponentially. Yet Fig. 4-4 shows that this is not the case. All of the submodules except for **thermalden.m** exhibit run times with much shallower slope at $N = \{2, 4, 6\}$

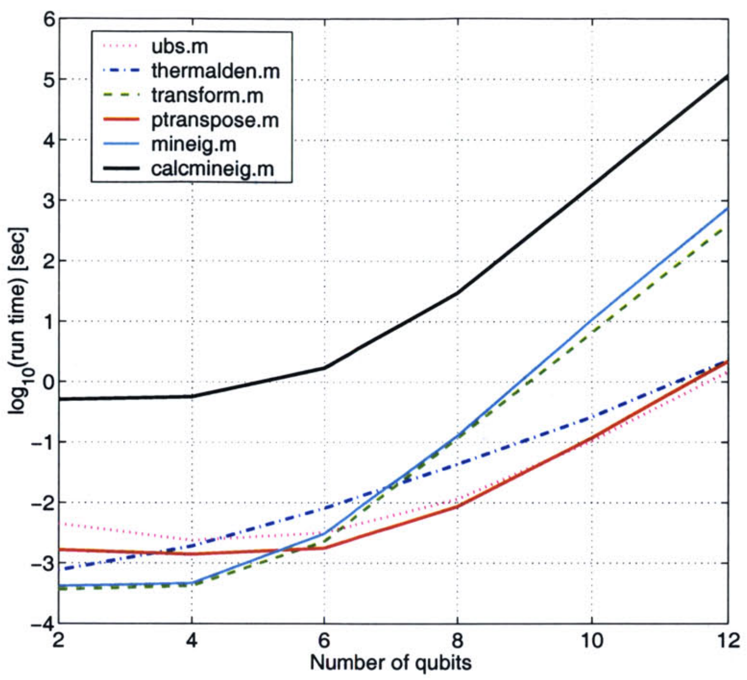

Figure 4-4: Mean single-processor run time as a function of N (ordered from top to bottom at the far left of the graph): `calcmineig.m` (thick solid black line), `ubs.m` (dotted magenta line), `ptranspose.m` (solid red line), `thermalden.m` (dash dotted blue line), `mineig.m` (solid cyan line), and `transform.m` (dashed green line). The data is acquired by executing the modules 100 times at each N. The partial transpose is computed for the $\{N/2, N/2\}$ split, using `ptHalfC.c`.

than $N = \{8, 10, 12\}$. In fact, the run time of `ptranspose.m` decreases from $N = 2$ to $N = 4$. We know that the number of steps required to calculate the transformed thermal density matrix and the partial transpose scale exponentially, so the anomalous behavior must be caused by unknown system overhead like cache size, as the run time data does not show any fluctuations. That `thermalden.m` is not affected may be due to the fact that it only creates a matrix where as the other submodules manipulate matrices through element accesses and matrix multiplication.

We assume that the system overhead does not dominate for $N = \{8, 10, 12\}$ since the run times here are much longer. The logarithmic run times for these qubit values are close to straight lines, implying an exponential scaling. The results of a linear fit analysis for the logarithmic data points is given in Table 4.1 with the prediction error defined as

$$e \equiv \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{\hat{t}_i - t_i}{t_i}\right)^2} \tag{4.1}$$

where $\hat{t}_i$ and t_i denote the predicted and actual run time data points and n is the total number of data points. Note that the fit is done for $\log_2$(run time) and not $\log_{10}$(run time).

Submodule	**Linear fit to $\log_2$(run time)**	**Prediction error**
`ubs.m`	1.75 N - 37.0	0.0077
`thermalden.m`	1.43 N - 16.0	0.0662
`transform.m`	2.94 N - 26.7	0.0100
`ptranspose.m`	2.00 N - 22.9	0.0479
`mineig.m`	3.13 N - 28.0	0.0160
`calcmineig.m`	2.98 N - 19.0	0.0043

Table 4.1: Linear fits to logarithmic mean single-processor run times in Fig. 4-4 for $N = \{8, 10, 12\}$. The prediction error is defined in Eq. 4.1.

The results in the table show that **`thermalden.m`** has the best run time scaling, followed by `ubs.m`, `ptranspose.m`, `transform.m`, and `mineig.m`. The low error suggests that the fits are reasonably accurate. The fits gives reasonable scalings for `transform.m` and `ptranspose.m`. The $d^{2.94}$ scaling (where the dimension $d = 2^N$) of `transform.m` run times is close to the d^3 scaling of naive matrix multiplication. Similarly, the number of swaps in `ptHalfC.c` is $2^{3N/2-1}(1 + 2^{N/2})$, which scales as approximately 2^{2N}. The run times of `ubs.m` and `thermalden.m` should scale linearly, but the actual scaling is a little worse for inexplicable reasons. As `mineig.m` is a built-in MATLAB function, we do not comment on its run time here.

More importantly, the fit for the run time of `calcmineig.m` demonstrates that even with unlimited memory resources, run times are prohibitively long on a single processor for $N > 12$. The same calculation run at $N = 15$ would take 630 days. In the next section, we turn to parallel computing to perform calculations where $N > 12$.

Overall, the submodules `transform.m` and `mineig.m` perform most poorly and dominate the run time of the λ_{min} calculation. The situation could be vastly improved if `transform.m` used sparse matrix multiplication and if `mineig.m` was optimized to look for the smallest eigenvalue as opposed to first calculating all the eigenvalues and then finding the minimum.

4.4 Beowulf cluster software

This section describes our work to adapt the single-processor software package for operation on a Beowulf cluster. First, we explain how the cluster architecture is configured to perform parallel linear algebra. Then we discuss the implementation of the Beowulf cluster package and calculation modules. We end with some remarks on the general performance of the cluster and calculation modules. More information on our Beowulf cluster architecture and software is available in Patz's masters thesis [Pat03] and at our cluster webpage [CC03].

4.4.1 Cluster architecture

Here we explain the layers of the Beowulf cluster architecture: physical, operating system, messaging, and applications. A diagram of the cluster architecture is shown in Fig. 4-5. Each layer depends on the lower layers to run properly. Failure of any layer may cause malfunctions in the layers above it.

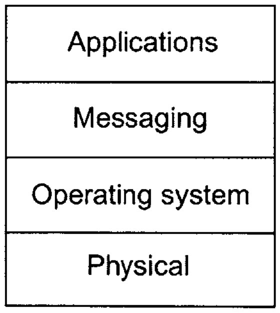

Figure 4-5: Hardware and software layers of Beowulf cluster architecture.

We now explain how each architectural layer is implemented.

Physical

The physical layer consists of 32 Pentium III machines that are linked together for parallel operation. Their hardware specifications were given in Section 4.2. The user controls the cluster machines via the client machine named Qubit, which is a quad processor Pentium III with 3 GB RAM. The cluster machines also have access to files on Alexandria, a server for user home directories. Alexandria, Qubit, and the cluster machines are connected to one another via a gigabit Ethernet network. Fig. 4-6 depicts the physical configuration we have just described.

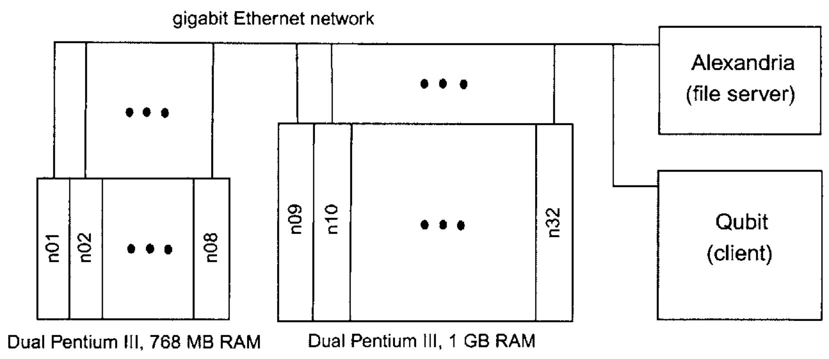

Figure 4-6: Physical configuration of Beowulf cluster.

Any subset of the 32 cluster machines may be selected to perform parallel computations, although some configurations may be more optimal than others depending on the application. Each active machine in the parallel environment is called a *node*.

Operating system

All of the nodes run Red Hat LINUX v7.3 as their operating system. We use the standard operating system configuration with three major modifications:

- Nodes mount the entire filesystem over the network via NFS.
- Users can start remote jobs on any node.
- The cluster allows remote access to user home directories on Alexandria and Qubit.

The ability to mount an entire filesystem over the network allow the nodes to be diskless. When the cluster starts up, each node boots its operating system kernel off a floppy disk and mounts its root file system from Qubit. We selected this setup so that only the user node hard disk needs to maintained.

The ability to start remote jobs on each node is essential to the operation of the message passing software as we explain shortly. Remote access to home directories lets users access the same copy of a file on every node such that only one set of files needs to be managed.

Messaging

PVM (Parallel Virtual Machine) [GBD+94] is the software package that handles primitive message passing between nodes. In typical cluster operation, PVM starts up on Qubit, then spawns copies of itself on user-requested nodes via remote job startup. Once PVM is established across the cluster, nodes can send point-to-point messages, broadcast messages to multiple nodes, or gather information from other nodes. An important function of PVM is synchronization of the nodes. Suppose the cluster performs one step of a calculation. The next step of the calculation may require information to be gathered from all the nodes. PVM can block nodes from continuing to the next step until every node is finished with the current step.

Applications

The cluster runs three major applications: BLACS, ScaLAPACK, and qpMATLAB. The first two packages are industry standard applications for high performance linear algebra on parallel machines while the last package contains our specific routines for implementing quantum simulations on a Beowulf cluster.

The BLACS (Basic Linear Algebra Communication Subprograms) package [DW97] performs inter-process communication that is optimized for high performance linear algebra. It calls primitive PVM routines to implement high level message passing, e.g. sending an array to all the processes holding a particular row of a matrix. In addition, BLACS sets up the cluster for parallel computation. First, it starts processes on nodes where PVM has been established. The number of processes is specified by the user, and more than one process may run on a machine. Once all the processes are running, it creates the *process grid*, which assigns unique grid coordinates to every process. This procedure allows BLACS to easily identify the processes, so it can pass messages between them. It also facilitates the distribution of parallelized matrices and implementation of parallel calculations by ScaLPAPACK. For example, BLACS might start one process on each of the machines n09, n10, n11 and n12 and assign the processes coordinates (0,0), (0,1), (1,0), and (1,1) in a 2×2 process grid.[3]

The ScaLAPACK (Scalable Linear Algebra PACKage) application [BCC+97] implements sophisticated parallelized linear algebra routines. The ScaLAPACK package calls BLACS routines for communication between processes and PBLAS (Parallel Basic Linear Algebra Subprograms) [CDO+95] routines for simple linear algebra operations like addition of vectors and matrix multiplication. By combining the functionality of the BLACS and PBLAS packages, ScaLAPACK implements complex operations such as eigenvalue finding and singular value decomposition. The ScaLAPACK software also handles the distribution of a parallel matrix to all processes in the BLACS-created processor grid such that each process on an active node locally stores and manipulates a section of the overall matrix. When we call linear algebra routines, we think of the matrices as whole objects whose elements are indexed by coordinates (i, j). However, each element (i, j) actually corresponds to a matrix element (k, ℓ) in the local submatrix of a process located at coordinates (p, q) of the processor grid. ScaLAPACK and any parallel computation program must keep track of this mapping. We call the coordinates in the mathematical abstraction *global coordinates* and the coordinates on the physical cluster *local coordinates.*

The qpMATLAB (Quantum Parallel MATLAB) package [Pat03] implements the specialized routines needed to calculate negativity. However, more generally, it is an integrated environment for simulating quantum computation on classical computers. The qpMATLAB package consists of three components: 1) a C-based library of routines to run quantum simulations on a Beowulf cluster, 2) a MATLAB-based user

[3]BLACS uses zero based indexing.

interface to direct the simulations, and 3) management of communication between the user and the cluster.

The qpMATLAB library is written in C for simple integration of the BLACS and ScaLAPACK packages, which use C and Fortran languages. The library contains four types of routines:

1. Matrix operations: matrix-matrix multiplication, matrix-vector multiplication, matrix-scalar multiplication, matrix-scalar division, and matrix-matrix addition. These operations are implemented by pre-existing BLACS and ScaLAPACK routines.

2. Matrix creation: zero matrix, identity matrix, random matrix, thermal density matrix, standard Bell unitary matrix.

3. Matrix manipulation: transfer a matrix from cluster to user, transfer a matrix from user to cluster, return one element of a parallel matrix, partial transpose of a matrix.

4. Utility operations: free the memory of a parallel matrix, free the memory of all parallel matrices on the cluster, shutdown qpMATLAB.

Internal to the library are functions that map global coordinates to local coordinates.

We use a MATLAB-based interface to enhance high-level portability of code between the single processor and the Beowulf cluster. User-cluster communication is implemented by using TCP/IP protocol to transmit serial strings between the two parties.

When a user issues a command in the MATLAB interface on the *client* machine (typically Qubit), that command is communicated to a special process on the cluster named the *master process*, which acts as the *server*. This process then implements the user's command by directing the other processes in the grid to call routines from the qpMATLAB library. The machine running the master process is the *master node*. The non-master processes are *child processes*. We emphasize that the master process can run on any machine on the cluster and that the identity of the master process can be easily changed from session to session.

The client-server communication occurs through two major programs, `qpclient` and `qpserver`. The `qpclient` program runs on the client and transmits user-specified commands to the `qpserver` program running on the master process, which interprets the command and orchestrates the implementation of the appropriate operation on the cluster. When the operation is complete, `qpserver` on the master process returns a message to `qpclient`. If the command was unsuccessful, an error message is returned; otherwise, the message returns a result (e.g. a number or a reference to a parallel matrix).

The application architecture we have described is shown in Fig. 4-7.

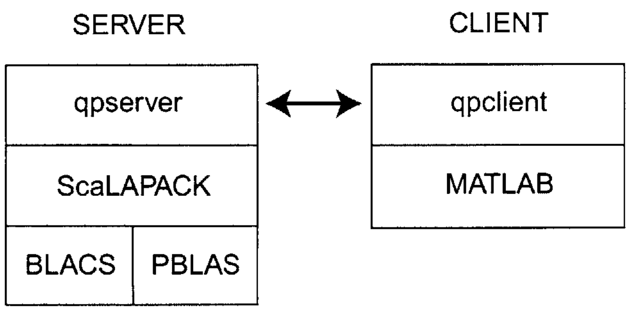

Figure 4-7: Application architecture of Beowulf cluster. The double arrow represents communication between `qpserver` and `qpclient`.

4.4.2 qpMATLAB usage examples

Rather than explaining the implementation details of qpMATLAB, we give examples to show how the package is used. The first example demonstrates the typical startup procedure for a qpMATLAB session, and the second example illustrates the communication processes that occur during the execution of a user-specified command.

Example startup procedure for qpMATLAB session

(done on Qubit)

1. Start PVM on Qubit by typing '`cd /cluster/pvm/lib3; ./pvm`'.

2. Sixteen cluster nodes are added to PVM's host list with the command '`add n09 n10 n11 n12 n13 n14 n15 n16 n17 n18 n19 n20 n21 n22 n23 n24`'.

3. Machine n09 is designated to be the master node by editing the file `qs_ip` to contain the number '`09`'.

4. Remotely log onto the master node (n09) with the command '`rsh n09`'. Then startup the server by running the command '`cd /cluster/matlab-6.1/qpmatlab/Server; ./qpserver m n`' on n09. This action causes BLACS to spawn one process on every machine n09-n24, create a $m \times n$ processor grid and spawns copies of `qpserver` on every process in the grid.

5. On Qubit, change to the qpMATLAB client directory and start the desktop version of MATLAB: '`cd /cluster/matlab-6.1/qpmatlab/Client; matlab -nojvm -nosplash`'.

6. Now the user can type commands at the MATLAB prompt to perform parallel operations on the cluster.

Example qpMATLAB session for creating a 14-qubit $U_{\mathrm{b,s}}$ matrix

This example will describe a qpMATLAB session first from the client's point of view, then from the server's point of view.

Client-side session

1. The user enters the MATLAB command '`U=pubs(14)`' at the console prompt. The argument to `pubs` specifies the number of qubits.

2. The MATLAB function `pubs.m` calls `qpclient` with the command '`val=qpclient(18,16384,16384)`' where the first value in the parentheses is the command number for `pubs.m` and the latter two values are the number of

rows and columns in the desired matrix. The program `qpclient` is actually a MEX file, a specialized C program that can be called inside MATLAB.

3. Now the `qpclient` program performs the following sequence:
   (a) Sends the string '`UUBS 16384 16384`' to the master process on the cluster over a TCP/IP socket.
   (b) Listens on the TCP/IP socket, waiting for the cluster to finish executing the user-specified command.
   (c) Receives the string '`OKSR 1`' from `qpserver` running in the master process. The string '`OKSR`' signals that the command was successful and the string '`1`' is a reference to the $2^{14} \times 2^{14}$ $U_{\rm b,s}$ matrix residing on the cluster.
   (d) Calls a built-in MATLAB function to convert '`1`' from a string to an integer, which is stored in a MATLAB integer array.
   (e) Returns the array pointer to the MATLAB variable `val`.
4. An object of the pmatrix class is created via the call '`pmatout = pmatrix(val,16384,16384)`'. Here `pmatout` is a reference to a structure object containing information about the parallel matrix, including the original array pointer in `val`.
5. The `pubs.m` function returns the value of `pmatout` to the external MATLAB variable `U` and prints the console message '`Parallel distributed matrix: 16384-by-16384 [server ref = 1]`' to let the user know that command was performed successfully.

Server-side session

1. The master process listens on the TCP/IP socket for commands from the client.
2. The master process receives the string '`UUBS 16384 16384`' from `qpclient`.
3. The `qpserver` program running in the master process interprets the string as a command to generate a 16384×16384 $U_{\rm b,s}$ matrix and instructs all the child processes to create a local submatrix containing a portion of $U_{\rm b,s}$.

4. The `qpserver` program running on each child process receives the instruction and calls the specific qpMATLAB routine that creates the local submatrix containing part of $U_{\mathrm{b,s}}$. Meanwhile, the master process creates its own portion of the $U_{\mathrm{b,s}}$ matrix and blocks any further action until the child processes finish.

5. When the matrix generation is complete, the master process sends the string '`OKSR 1`' to `qpclient`. The string '`1`' is a reference to the $U_{\mathrm{b,s}}$ matrix.

6. The master process awaits next command from the client.

Notice that the server only passes references to the client, not the actual parallel matrices. We keep all the calculations on the server side because user-client communication is slow and prone to protocol error for large amounts of transmitted information.

4.4.3 Calculation modules

Here we describe the negativity calculation modules in the qpMATLAB package: `pubs()`, `pthermalden()`, `ptransform()`, and `pptranspose()`. At present, the minimum eigenvalue module `pmineig()` is still incomplete. The character '`p`' prefixes the parallel version of a calculation module, and the notation `()` denotes a qpMATLAB module.

The Beowulf numerical package is designed to overload the single-processor package, so the user may run calculations with either the single processor or Beowulf software transparently. Therefore, the input and output variables are the same as described in Section 4.3.4. The methods to calculate memory usage and run time for each module are also the same as those used for the single-processor modules (see Section 4.3.1).

We now sketch the implementation details of each module. In the description that follows, we have left implicit the calls to determine the mapping between global and local coordinates as these are common to any parallel computation. For the module source code, see Appendix B.

`pubs()`

This module generates the standard Bell unitary $U_{\mathrm{b,s}}$ for given N. The implementation is quite trivial. Each process sets nondiagonal entries to zero and sets diagonal entries to $\pm 1/\sqrt{2}$ according to the pattern in Section 3.3.1.

`pthermalden()`

This module creates the thermal density matrix ρ_{th} for given N and α, following the procedure:

1. Every process sets its nondiagonal entries to zero and sets diagonal entries to values corresponding to the unnormalized thermal density matrix. Thus, the value of the (i, i) entry would be $D_i = e^{[N-2w(i-1)]\alpha/2}$ where $w(i-1)$ is the number of 1s in the binary expression for $i-1$. Here we assume one-based indexing of the matrix elements.

2. Each child process computes the sum of all its diagonal entry values and sends this number to the master process.

3. The master process blocks the child processes until it receives sums from all of the children. It adds the sums together to obtain the normalization constant $\mathcal{Z} = \sum_{i=1}^{2^N} D_i$. Then the master process sends $\mathcal{Z}$ to all the children.

4. All processes divide the values of their diagonal entries by $\mathcal{Z}$ to properly normalize the thermal density matrix.

`ptransform()`

This module computes the Bell transformed thermal state, given parallel matrix references to ρ_{th} and U_{b}. This function is accomplished by matrix multiplication, exactly like the single-processor version `transform.m`. The multiplication is performed in two explicit steps. First, the PBLAS routine for matrix multiplication calculates $U_{\mathrm{b}} * \rho_{\mathrm{th}}$. This product is passed to another PBLAS routine for computing transposed matrix multiplication, which calculates $U_{\mathrm{b}}\rho_{\mathrm{th}} * U_{\mathrm{b}}^T$, yielding the desired transformed thermal state. Here we can use the transpose instead of the Hermitian conjugate since all our matrices are real.

`pptranspose()`

This module calculates the partial transpose of a matrix. Currently, only the $\{N/2, N/2\}$ bipartite split can be computed, although our approach is easily extended to splits with pre-existing single-processor implementations. Unlike the single-processor version, `pptranspose()` performs the partial transpose on the matrix in place to economize memory usage.

Our implementation of `pptranspose()` for the $\{N/2, N/2\}$ split uses the same algorithm as the single-processor version to decide which elements must be swapped. However, in parallel machine, the swap operation is more complicated because the swapped elements are usually located in different processes. We now describe how a generic swap is executed on the cluster.

Following the single-processor algorithm, the master process determines which elements must swap. Suppose first swapped matrix element has complex value c_1 and local coordinates (k_1, ℓ_1) on the process located at (p_1, q_1) in the process grid, and the second swapped matrix element has complex value c_2 and local coordinates (k_2, ℓ_2) on the process located at (p_2, q_2) in the process grid. We also note that `qpserver` automatically defines the process grid coordinates of the master process to be (0,0). There are four cases that we must consider:

1. Swap elements inside the master process, i.e. $p_1 = p_2 = q_1 = q_2 = 0$.

2. Swap elements inside a single remote process, i.e. $p_1 = p_2 \neq 0$ and $q_1 = q_2 \neq 0$.

3. Swap elements between the master process and a remote process, i.e. $p_1 = q_1 = 0$, $p_2 \neq 0$, and $q_2 \neq 0$ or $p_2 = q_2 = 0$, $p_1 \neq 0$, and $q_1 \neq 0$.

4. Swap elements between two different remote processes, i.e. $p_1 \neq p_2$, $q_1 \neq q_2$, and $\{p_1, q_1, p_2, q_2\} \neq 0$.

The swap procedure is facilitated by sending 5-element message arrays between the nodes. The first element of any message array is a single character representing a specific instruction. The codes are:

- '*' = internal swap inside node that receives the message
- '%' = send data to remote node
- '>' = process incoming data
- '$' = wait for other nodes to finish their tasks
- '.' = partial transpose finished

Now we describe the implementations details of each case for a given swap.

1. The first case is trivial.

2. In the second case, the master process sends a message array ['*' k_1 ℓ_1 k_2 ℓ_2] to the process at (p_1, q_1). The process reads the prefix '*' as a directive to swap elements (k_1, ℓ_1) and (k_2, ℓ_2) of its local submatrix.

3. For the third case, we will consider the situation where the first matrix element belongs to the master process and the second matrix element belongs to the remote process since the protocol for the reverse situation follows immediately. First, the master process sends the remote process the message array ['%' k_2 ℓ_2 0 0]. The message prefix '%' tells the receiving process to send data to another process. Therefore, the remote process interprets the message as a directive to transmit the element in its local submatrix at (k_2, ℓ_2) to the master process at (0,0). It sends the master process the message array ['>' Re(c_2) Im(c_2) X X] where the Xs represent garbage information. Similarly, the master array sends the remote node the message ['>' Re(c_1) Im(c_1) X X]. The remote process receives the master process' message and sets value of the matrix element at (p_2, q_2) to c_1. The master process performs the analogous procedure.

4. In the fourth case, the master node transmits the message ['%' k_1 ℓ_1 p_2 q_2] to first remote node and the message ['%' k_2 ℓ_2 p_1 q_1] to the second remote node. These messages direct each remote node to send the appropriate local submatrix element to the other remote node. The first and second remote nodes send each other the requested data using the messages of form ['>' Re(c_i) Im(c_i) X X] and set the value of their local submatrix element to the received data value. This action is exactly the same as in the third case except both the swapping nodes are remote.

While our swap procedure works in principle, the communication traffic grows heavy for large matrices. Since it takes much longer to process the data than to send a message, the nodes are overwhelmed with more swap requests than they can handle. The simplest solution is to continuously record the cumulative number of swap calls in a variable we will call n. If n reaches a fixed limit n_{max}, the master node sends a '\$' message to all the nodes, which instructs them to wait until the remaining pending swap requests are fulfilled. Then the block is released, n is set back to 0, and the master process is free to send more swap instructions.

In Table 4.2, we have listed the mean run times of `pptranspose()` for various values of n_{max}. In all cases, the partial transpose is applied to an 8-qubit matrix, and the parallel environment consists of 4 Pentium III dual-processor machines, each machine having 1 GB RAM and running four processes on a 4×4 process grid.

The data in the table indicates that $n_{\text{max}} = 200$ is optimal. The mean run times do not change significantly when n_{max} is varied between 200 and 800, implying that 4 machines can handle a load of up to 800 cumulative swap requests. However, if we decrease n_{max} below 200, then the mean run time grows quickly as the frequent blocking begins to slow down the partial transpose operation. The particularly large increase in mean run time when n_{max} drops from 25 to 0 demonstrates the importance of limiting communication traffic.

$n_{\max}$	**Mean run time (min)**
0	0.6150
25	0.0881
50	0.0792
100	0.0767
200	0.0758
400	0.0758
800	0.0755

Table 4.2: Mean run time for `pptranspose()` as a function of $n_{\max}$ where $n_{\max}$ is the maximum number of cumulative swap calls allowed before the cluster is blocked. The partial transpose is applied to an 8-qubit matrix, and the parallel environment consists of 4 Pentium III dual-processor machines, each machine having 1 GB RAM and running four processes on a 4×4 process grid. The mean run time is obtained by averaging over 10 runs.

4.4.4 Cluster baseline and qpMATLAB application performance

Cluster benchmark

The baseline cluster memory usage and run time performance was assessed with the High Performance Computing Linpack (HPL) benchmark [PWDC00]. To test the maximum performance of the cluster, HPL used machines n09-n32 to solve a 50000×50000 real linear system ($A\vec{x} = \vec{b}$) on a 8x6 processor grid. From the benchmark results, we estimate a memory usage division of 18.6 GB storage, 3.7 GB overhead, and 1.7 GB free out of a total available memory of 24 GB (from the 24 nodes). In the best run, our Beowulf cluster completed the problem in 59 minutes, equivalent to an effective benchmark of 23.5 gigaflops.[4]

Comparison of our benchmark with the benchmarks of commercially available computers suggests that our Beowulf cluster is comparable to a Sun Fire 12K machine with sixteen 900 MHz processors [Don03].

[4]A gigaflop is equal to 10^9 floating point operations per second.

qpMATLAB calculation module run times

We also clock the run times of the calculation submodules and plot the mean run times at $N = \{4, 6, 8, 10, 12, 14\}$[5] in Fig. 4-8. For these calculations, we used 16 Pentium III 1.2 GHz dual-processor machines with each machine having 1 GB RAM and running one process on a 4×4 processor grid.

Comparing absolute mean run times, **pubs()** is the fastest submodule followed by **pthermalden()**, **ptransform()**, and **pptranspose()**. As in the single-processor assessment, the submodules **pubs()** and **pthermalden()** run substantially faster than the other submodules because they create matrices and do not manipulate them.

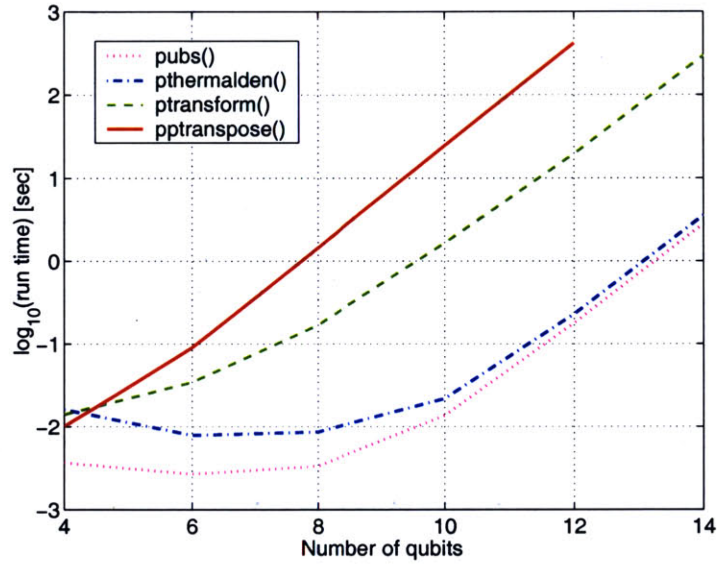

Figure 4-8: Mean cluster run time as a function of N: **pubs()** (dotted magenta line), **pthermalden()** (dash dotted blue line), **ptransform()** (dashed green line), and **pptranspose()** (solid red line). The data is acquired for each module by averaging over 20 runs. The partial transpose is computed for the $\{N/2, N/2\}$ split using the same algorithm as in **ptHalf.m/ptHalfC.c**, and the parallel environment consists of 16 Pentium III 1.2 GHz dual-processor machines with each machine having 1 GB RAM and running one process on a 4×4 processor grid.

To compare the mean run times on the cluster with the mean run times on the single processor, we have plotted their ratios in Fig. 4-9. As the number of qubits increases, all the cluster submodules except **pptranspose()** quickly outpace their

[5]No data point was measured for **pptranspose()** at $N = 14$ because the run time was too long.

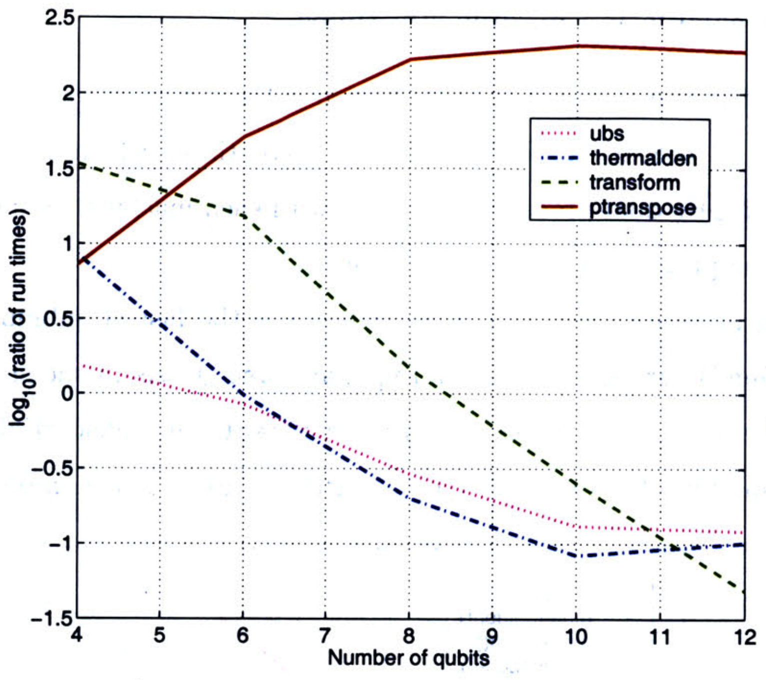

Figure 4-9: Ratio of mean cluster run times from Fig. 4-8 to mean single-processor run times from Fig. 4-4 as a function of N: **ubs** (dotted magenta line), **thermalden** (dash dotted blue line), **transform** (dashed green line), and **ptranspose** (solid red line).

single-processor counterparts with their mean run times growing exponentially faster than the single-processor mean run times for small N. The exponential behavior stabilizes for larger N such that the cluster submodule run times (except for **ptranspose()**) are about a tenth shorter than the corresponding single-processor run times at $N = 12$. The superior cluster performance is due to the large amount of computing power available in 16 dual-processor machines. Even greater speedup might be possible if all 32 cluster machines were included in computation.

The **pptranspose()** submodule is always much slower than **ptranspose.m**. The mean runtime ratio increases exponentially with low N before tapering off at higher N. The worst cluster performance occurs at $N = 10$ where the **pptranspose()** performs the partial transpose 191 times slower than the **ptranspose.m**. As discussed in Section 4.4.3, the sluggishness of **pptranspose()** is due to heavy message traffic over the process grid. We have tried to remedy the situation by blocking communication at fixed intervals, but the messages themselves are uneconomical as they contain a large fraction of header compared to a small portion of actual data. A future solution would be to consolidate the messages by node destination, thereby reducing the traffic problem.

As discussed in Section 4.3.6, we expect the run times to scale exponentially, but clearly this behavior is not exhibited for low N, particularly in the cases of `pubs()` and `pthermalden()`. We again make the assumption that the anomaly is due to unknown system overhead (e.g. cache size) whose impact on run time diminishes at larger N. Consequently, we fit the run times for the last three data points in N and give the results in Table 4.3.

Submodule	**Linear fit to $\log_2$(run time)**	**Prediction error**
`pubs()`	1.92 N - 25.4	0.0077
`pthermalden()`	1.83 N - 23.9	0.0662
`ptransform()`	1.87 N - 18.0	0.0100
`pptranspose()`	2.03 N - 15.7	0.0398

Table 4.3: Linear fits to logarithmic mean cluster run times in Fig. 4-8 for $N = \{8, 10, 12\}$ except in the case of `ptranspose()` where $N = \{6, 8, 10\}$. The prediction error is defined in Eq. 4.1.

Comparing the cluster mean run time scaling in Table 4.3 with that of the single processor in Table 4.1, we see that the scaling of `ptransform()` is in fact better than `transform.m`, the scaling of the partial transpose for both the cluster and single processor is about the same, and the scaling of `pubs()` and `pthermalden()` are significantly worse than their single-processor counterparts. The erratic differences in scaling cannot be explained by the additional complexity of the cluster architecture because it would affect all submodules equally.

4.5 Numerical negativity and minimum eigenvalue maps for standard Bell-transformed thermal states

Here we present negativity and minimum eigenvalue maps for the standard Bell-transformed thermal state $\rho_{\text{tf}} = U_{\text{b,s}}\rho_{\text{th}}U_{\text{b,s}}^{\dagger}$ in N-α space.[6] The calculations are performed on the single-processor software package, according to the steps outlined in Section 4.3.1. All maps were generated by computing λ_{min} at $N = \{2, 4, 6, 8, 10, 12\}$ and 100 logarithmically spaced points in the range $\alpha = [10^{-3}, 10^{1}]$.

The negativity maps for the $\{N/2, N/2\}$, $\{1, N-1\}$, and $\{N-1, 1\}$ splits are plotted in Figs. 4-10, 4-11 and 4-12 respectively. Bounds on the nonseparability of $U_{\text{b,s}}\rho_{\text{th}}U_{\text{b,s}}^{\dagger}$ were determined with `estzalpha.m`, as described in Section 4.3.5. To compare the entanglability of thermal states versus effective pure states, the Braunstein et. al. bound on entanglable ρ_{eff} (Eq. 3.4) and the Gurvits-Barnum bound on nonentangable ρ_{eff} (Eq. 3.3) are also marked in parameter space. We note that the negativity maps look quite uniform for $\log_{10}\alpha^{-1} > 0.5$ because λ_{min} varies very slowly in this region. To illustrate this point, we have plotted λ_{min} for $N = 12$ ($\{N/2, N/2\}$ split) in Fig. 4-13.

Comparing the negativity maps for the three splits, the $\{1, N-1\}$ partition gives the tightest bound on nonseparable ρ_{tf} since its bound lies farthest to the right. We speculate that this result is due to the symmetry of $U_{\text{b,s}}$. The standard Bell unitary appears to give the first qubit a special status because the state of the latter $N-1$ qubits is controlled by the first qubit through a massive CNOT. Moreover, all three splits give bounds on nonseparable ρ_{tf} that capture additional parameter space beyond the bound on entanglable ρ_{eff}. Given that a bound on nonseparable ρ_{tf} is equivalent to a bound on entanglability ρ_{th}, it appears that thermal states are more easily entangled than effective pure states with the same parameters N and α.[7]

[6]We defer discussion of $U_{\text{b,s}}U_{\text{fan}}$-transformed thermal states until Chapter 6 as the motivation for studying these states appears there.

[7]We make this statement in the context of the numerical results. However, in Chapter 6, we derive tighter bounds on both thermal states and effective pure states. The comparison of these

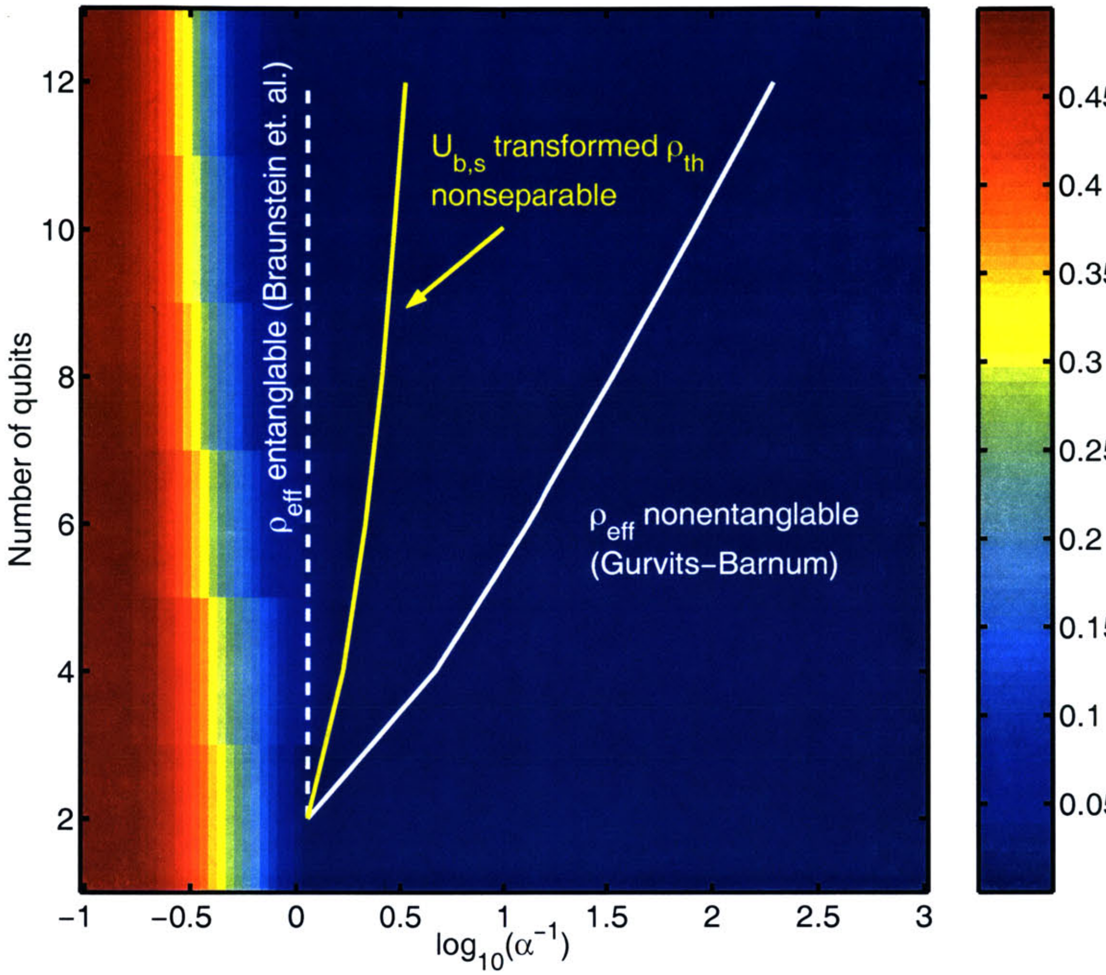

Figure 4-10: Color map of $E_n(\rho_{\text{tf}})$ for $\{N/2, N/2\}$ bipartite split in $N-\alpha$ parameter space, overlaid with Gurvits and Barnum bound on the nonentanglability of ρ_{eff} (solid white line), Braunstein, et. al. bound on the entanglability of ρ_{eff} (dashed white line), and bound on the nonseparability of ρ_{tf} (solid yellow line), which is determined by estimating the α corresponding to $\lambda_{\text{min}} = 0$. The bar on the right ascribes a color to each value of E_n.

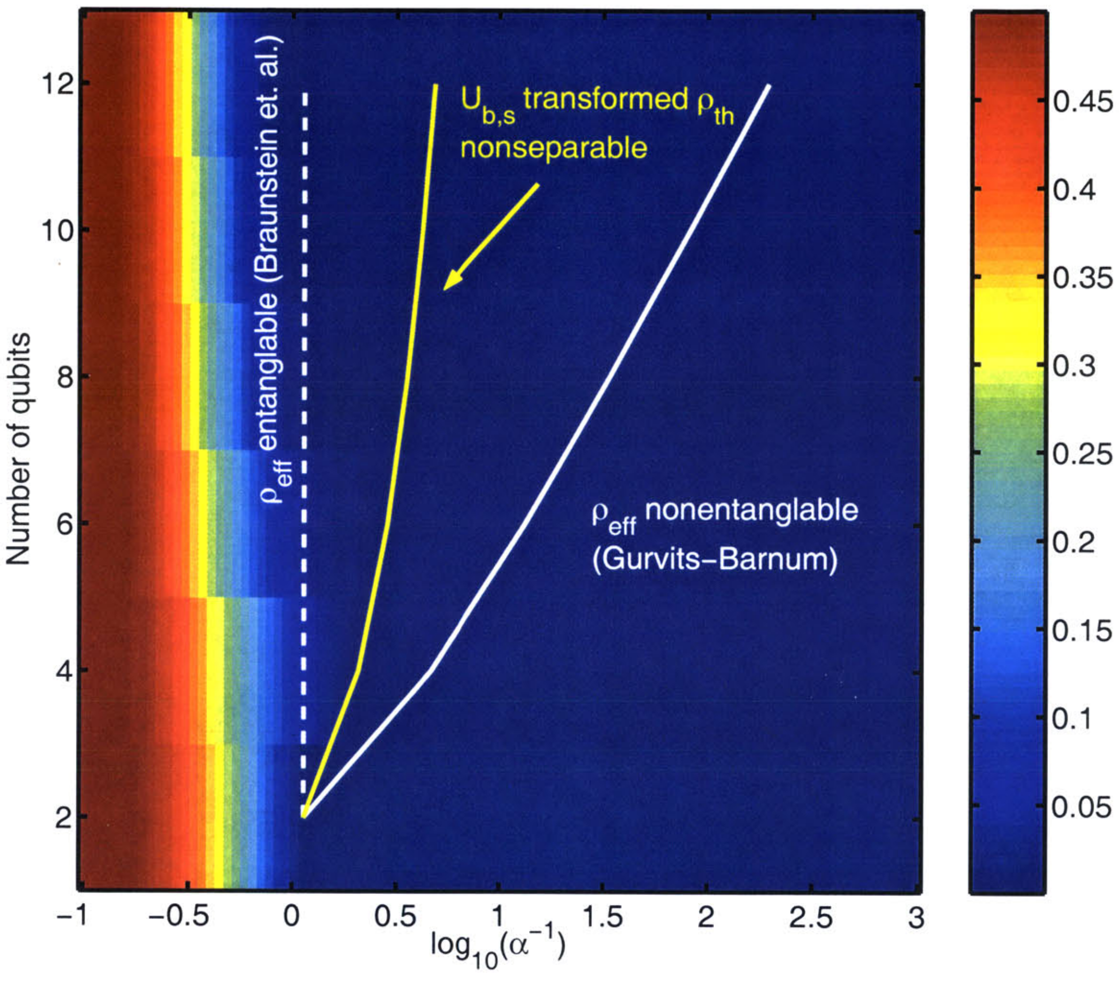

Figure 4-11: Color map of $E_n(\rho_{\mathrm{tf}})$ for $\{1, N-1\}$ bipartite split in $N-\alpha$ parameter space, overlaid with Gurvits and Barnum bound on the nonentanglability of ρ_{eff} (solid white line), Braunstein, et. al. bound on the entanglability of ρ_{eff} (dashed white line), and bound on the nonseparability of ρ_{tf} (solid yellow line), which is determined by estimating the α corresponding to $\lambda_{\mathrm{min}} = 0$. The bar on the right ascribes a color to each value of E_n.

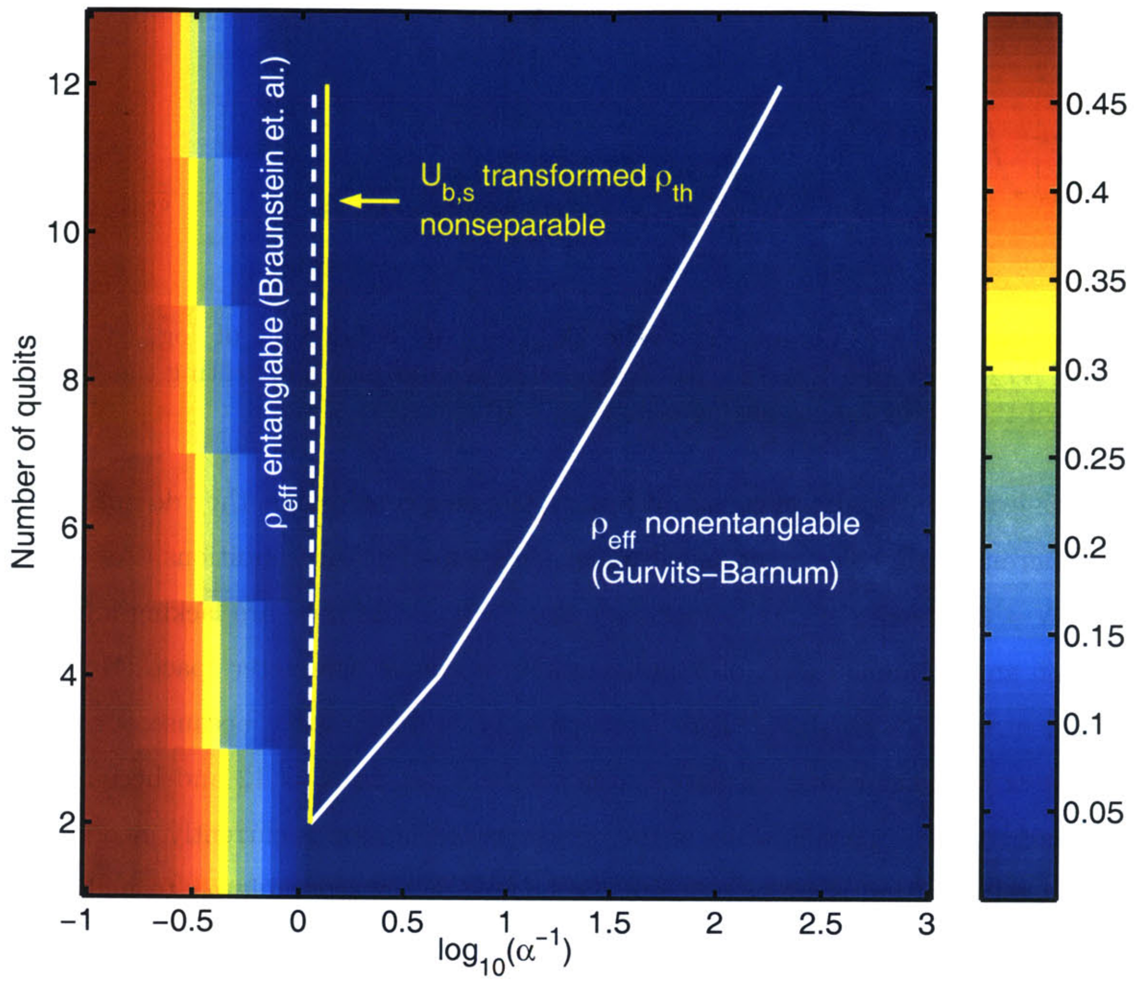

Figure 4-12: Color map of $E_n(\rho_{\text{tf}})$ for $\{N-1, 1\}$ bipartite split in $N - \alpha$ parameter space, overlaid with Gurvits and Barnum bound on the nonentanglability of ρ_{eff} (solid white line), Braunstein, et. al. bound on the entanglability of ρ_{eff} (dashed white line), and bound on the nonseparability of ρ_{tf} (solid yellow line), which is determined by estimating the α corresponding to $\lambda_{\min} = 0$. The bar on the right ascribes a color to each value of E_n.

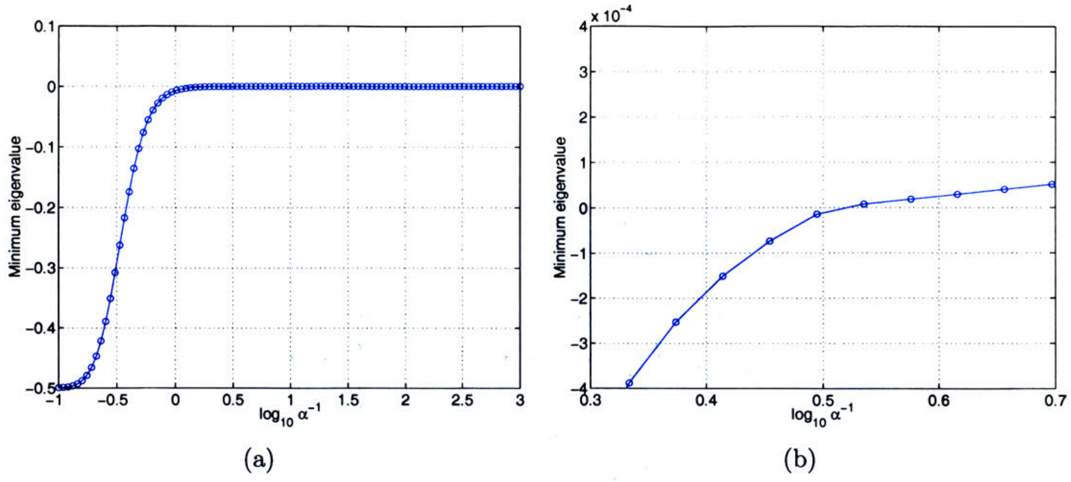

Figure 4-13: (a) Minimum eigenvalue $\lambda_{\text{min}}(\rho_{\text{tf}})$ as a function of $\log_{10}(\alpha^{-1})$ for $\{N/2, N/2\}$ split and $N = 12$. (b) Closeup view of same plot, illustrating the shallow slope of the minimum eigenvalue near $\lambda_{\text{min}} = 0^+$.

While our negativity maps admit some interesting conclusions, they do not tell us if a thermal state can be entangled under ideal experimental conditions. Assuming that $N \leq 30$ is experimentally reachable (see Section 3.2.2), we are lacking information on an enormous region of experimentally accessible parameter space. How can we fill in the missing data? Direct computation is not possible because our single processor only has enough memory to compute E_n up to $N = 12$ and because the Beowulf cluster implementation of the negativity calculation is currently incomplete. One viable solution is to extrapolate the bounds on nonseparable ρ_{tf} to higher N. Unfortunately, this method is unworkable as we now show.

The bound on nonseparable ρ_{tf} is exactly the line of constant $\lambda_{\text{min}} = 0$ as explained in Section 3.3.2. Because we are interested in extrapolating the nonseparable bound, we find the contours for minimum eigenvalues near $\lambda_{\text{min}} = 0$. Here we chose the numbers: $\lambda_{\text{min}} = \{-0.010, -0.005, 0.005, 0.010\}$. In Figs. 4-14, 4-15, and 4-16, we draw the contours over minimum eigenvalue maps for the $\{N/2, N/2\}$, $\{1, N-1\}$, and $\{N-1, 1\}$ splits. For all cases, the contours diverge rapidly from the nonseparable bound. It is impossible to tell whether the extrapolated bound will bend left to follow the negative contours or bend right to follow the positive contours. We conclude that

bounds gives a more ambiguous conclusion.

extrapolation is useless for meaningful analysis of thermal state entanglability. The Beowulf cluster software package for negativity map calculations, if completed in the future, could be helpful in addressing this problem.

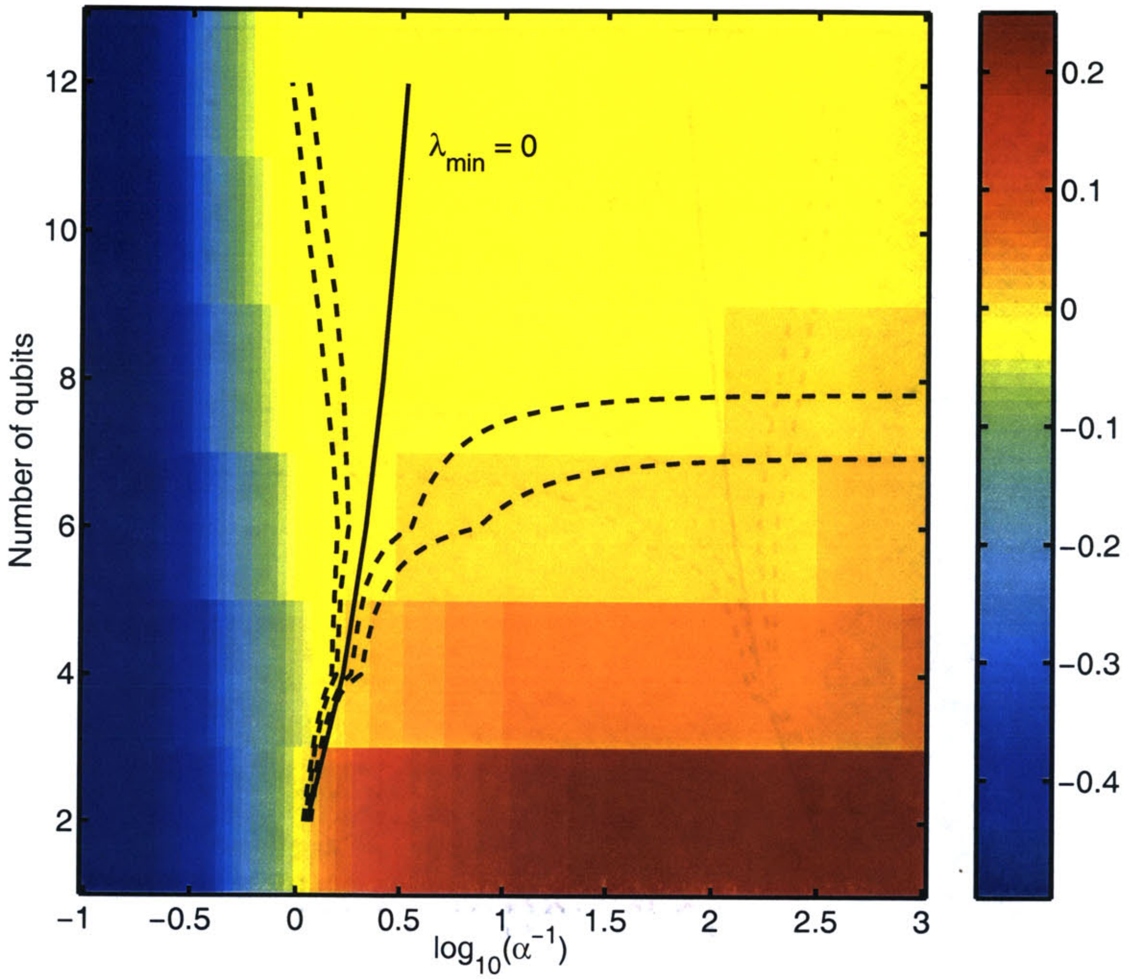

Figure 4-14: Color map of $\lambda_{\min}(\rho_{\text{tf}}^{T_A})$ for $\{N/2, N/2\}$ bipartite split in $N-\alpha$ parameter space, overlaid with bound on nonseparability of ρ_{tf} where $\lambda_{\min} = 0$ (solid black line) and from left to right, contours at $\lambda_{\min} = \{-0.010, -0.005, 0.005, 0.010\}$ (dashed black line). The bar on the right ascribes a color to each value of $\lambda_{\min}$.

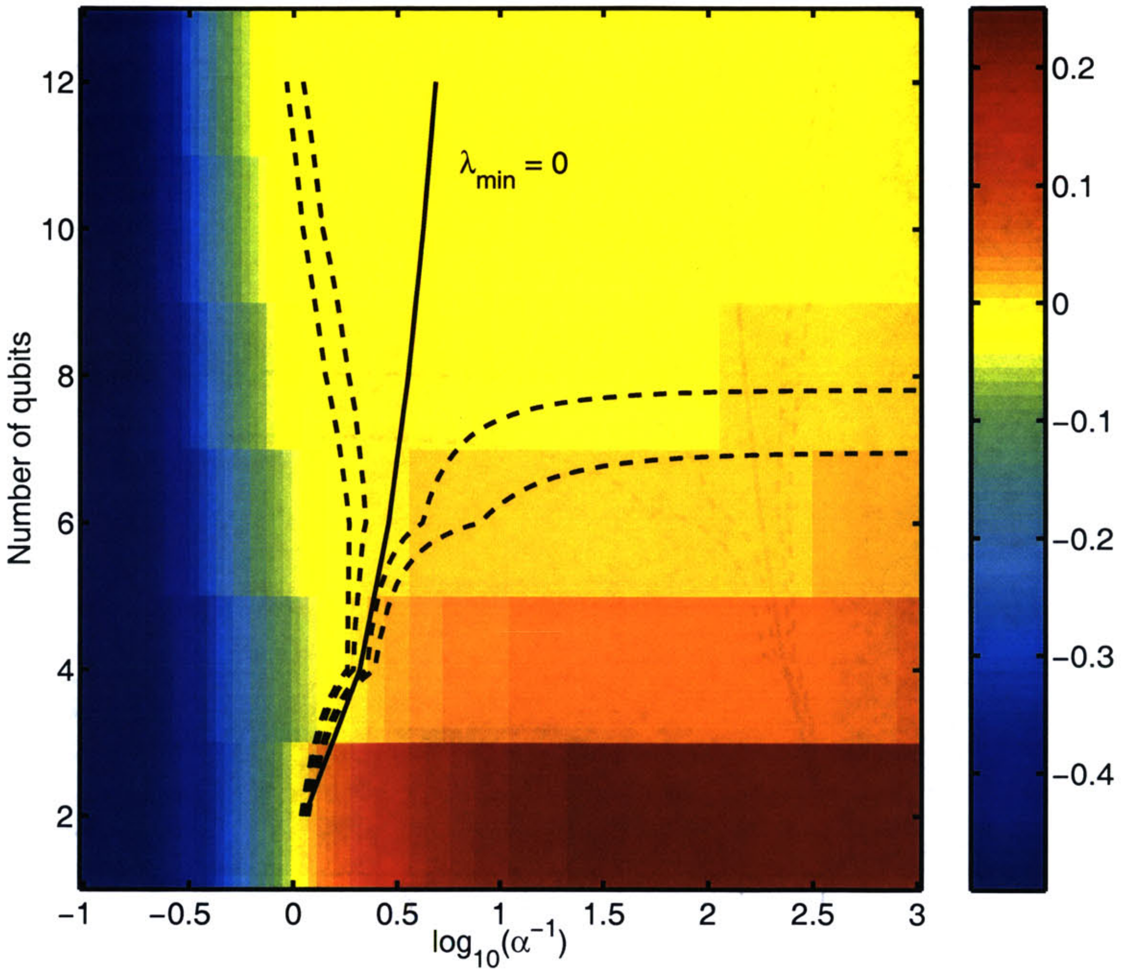

Figure 4-15: Color map of $\lambda_{\min}(\rho_{\mathrm{tf}}^{T_A})$ for $\{1, N-1\}$ bipartite split in $N-\alpha$ parameter space, overlaid with bound on nonseparability of ρ_{tf} where $\lambda_{\min} = 0$ (solid black line) and from left to right, contours at $\lambda_{\min} = \{-0.010, -0.005, 0.005, 0.010\}$ (dashed black line). The bar on the right ascribes a color to each value of $\lambda_{\min}$.

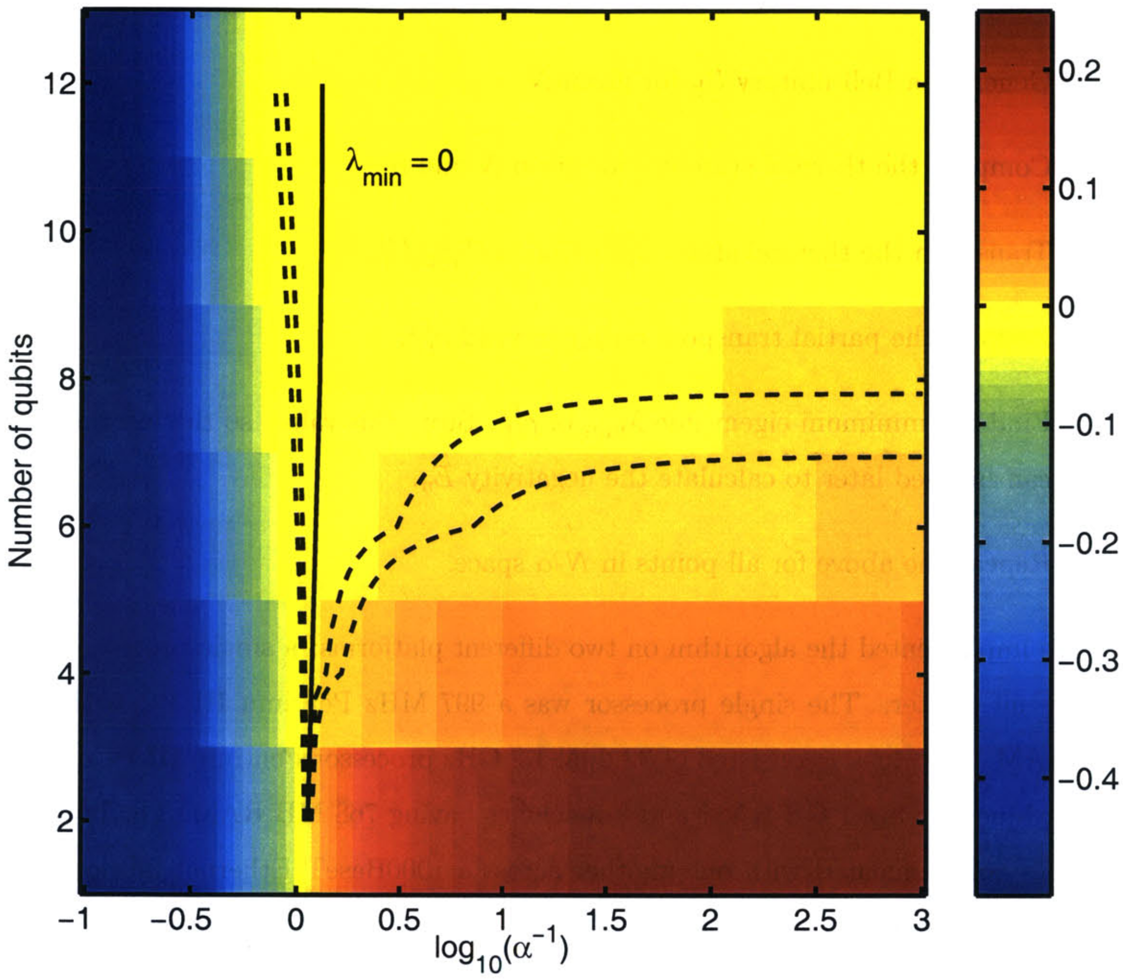

Figure 4-16: Color map of $\lambda_{\min}(\rho_{\text{tf}}^{T_A})$ for $\{N-1, 1\}$ bipartite split in $N-\alpha$ parameter space, overlaid with bound on nonseparability of ρ_{tf} where $\lambda_{\min} = 0$ (solid black line) and from left to right, contours at $\lambda_{\min} = \{-0.010, -0.005, 0.005, 0.010\}$ (dashed black line). The bar on the right ascribes a color to each value of $\lambda_{\min}$.

4.6 Summary

This chapter described the numerical methods that were used to calculate negativity maps in N-α space.

The calculation of a negativity map followed the general algorithm:

1. Generate a Bell unitary U_{b} for given N.

2. Compute the thermal state ρ_{th} for given N and α.

3. Transform the thermal state: $\rho_{\mathrm{th}} \mapsto \rho_{\mathrm{tf}} = U_{\mathrm{b}} \rho_{\mathrm{th}} U_{\mathrm{b}}^{\dagger}$.

4. Perform the partial transpose on ρ_{tf} to yield $\rho_{\mathrm{tf}}^{T_A}$.

5. Find the minimum eigenvalue λ_{min} of $\rho_{\mathrm{tf}}^{T_A}$. Store this value, so the information can be used later to calculate the negativity E_n.

6. Repeat the above for all points in N-α space.

We implemented the algorithm on two different platforms: a single processor and a Beowulf cluster. The single processor was a 997 MHz Pentium III PC with 879 MB RAM. The cluster consisted of 32 dual 1.2 GHz processor Pentium III PCs, with 24 machines having 1 GB RAM and 8 machines having 768 MB RAM. The individual PCs communicated with one another across a 1000BaseT Ethernet fabric using TCP/IP protocol. For both platforms, all machines ran LINUX as their operating system. Based on fundamental memory constraints, the single processor was capable of calculating negativity up to $N = 12$ while the Beowulf cluster could calculate up to $N = 15$.

In the single-processor software package, each step of the above algorithm was implemented as a submodule: `ubs.m` and `ubsufan.m`, `thermalden.m`, `transform.m`, `ptranspose.m`, and `mineig.m`. A negativity map was calculated by executing the module `runexpt.m`, which called a master module named `calcmineig.m` to compute λ_{min} at even positive integer values of N while varying α in parameter space. In turn, `calcmineig.m` called the calculation submodules in sequential order to find λ_{min}. At the end of the calculation at each value of N, `calcmineig.m` gathered the results of the λ_{min} computation, which was written to a QCTM structure by `writeqctm.m`. The package structure and execution flow were depicted in the module dependency diagram of Fig. 4-1.

We evaluated the memory usage and run time performance of the individual calculation submodules in the single-processor package. Mean memory usage was plotted in Fig. 4-3, and mean run time was plotted in Fig. 4-4. For all modules, storage of N-qubit matrices dominated memory usage. The submodule `transform.m` required the most memory because it needed to store three matrices: U_{b}, ρ_{th}, and ρ_{tf}. We did not analyze the run times at lower N because they showed nonexponential scaling. For $N = \{8, 10, 12\}$, `transform.m` and `mineig.m` ran substantially slower than the other modules. This result was expected since the `transform.m` module performs matrix multiplication and `mineig.m` calculates all eigenvalues of a matrix, both time intensive operations when compared with the matrix creation tasks assigned to `ubs.m/ubsufan.m` and `thermalden.m`. Moreover, at these values of N, the run times were well-modeled by exponential scaling. Our fit predicted that a single run of `calcmineig.m` would require 630 days to execute. Thus, aside from memory constraints, any hope of extending the single-processor package to calculating at $N > 12$ would require the run times of `transform.m` and `mineig.m` to be reduced significantly.

Development of the Beowulf cluster software package involved two primary tasks: configuring the cluster architecture to implement parallel linear algebra operations and adapting the single-processor modules to the cluster. The cluster architecture was made of four layers as shown in Fig. 4-5. We briefly describe them here from bottom to top:

1. Physical - The 32 cluster machines, in addition, to being connected to one another, also communicate with a client machine and a file server. The setup is shown in Fig. 4-6.

2. Operating system - RedHat Linux v7.3 fine tuned for parallel operation by

allowing filesystem mounting over the network, capability to start remote jobs on cluster machines, and remote access to files on the client machine and the file server.

3. Messaging - Implemented by PVM, a primitive message passing package.

4. Applications - BLACS, ScaLAPACK, and qpMATLAB. BLACS performed high-level message passing optimized for parallel linear algebra operations, ScaLAPACK implemented complex parallel linear algebra operations, and qpMATLAB was our home-developed software package for simulating quantum computation in parallel environments. A parallel environment was established by calling BLACS routines to spawn processes on available machines and assigned each process coordinates in a processor grid. During a parallel computation, each process stored and manipulated a local portion of the overall matrix.

The qpMATLAB package itself contained three parts: a library of C routines for simulating quantum computation, a MATLAB-based user interface to direct the simulations, and management of communication between the user and the cluster. One designated process, the master process, implemented user commands by directing the other processes to call routines from the qpMATLAB library. All user-cluster communication was passed between the program `qpclient` on the client machine and the program `qpserver` on the master process. The overall application architecture was shown in Fig. 4-7.

The cluster-based negativity calculation modules used the same algorithms as the single-processor versions. Thus the bulk of the development work for the cluster modules involved incorporating the appropriate message passing between processes. Inter-process communication was particularly important for implementing the swap operations needed to perform the partial transpose on a matrix.

We assessed the performance of our cluster in performing simple linear algebra operations and in executing the negativity calculation modules from the qpMATLAB package. Using the HPL benchmark, our cluster solved a 50000×50000 real linear system on 24 dual-processor Pentium III machines and a 8×6 process grid. The problem required 59 minutes to run, giving a benchmark of 23.5 gigaflops. To gauge the efficiency of the qpMATLAB negativity calculation modules, we clocked their run times in a parallel environment consisting of 16 dual-processor Pentium III machines laid out in a 4×4 process grid. In general, the cluster modules performed much faster than their single-processor versions as seen in Figs. 4-8 and 4-9. The mean run time ratio between the cluster and the single processor decreased exponentially with small N, then tapered to approximately 0.1 at $N = 12$. The exception was the cluster-based partial transpose module, which ran more slowly than its single processor counterpart. The slowdown was as much as 191 times at $N = 12$ and was caused by heavy communication traffic derived from swap operations.

Finally, we computed negativity maps of $U_{\mathrm{b,s}}$-transformed thermal states for $N = \{2, 4, 6, 8, 10, 12\}$ and the range $\alpha = [10^{-3}, 10^{1}]$. The maps were plotted in Figs. 4-10, 4-11, and 4-12). The line where $\lambda_{\mathrm{min}} = 0$ gave a bound on the entanglability of thermal states. Experimenting with several bipartite splits, we found that $\{1, N-1\}$ split gave the tightest bound on thermal state entanglability. To see if a thermal state could be entangled under ideal experimental conditions, we required negativity data from parameter space at higher N. Unfortunately, because the cluster software package was unfinished, our negativity calculations were limited to $N \leq 12$. Attempts to extrapolate the entanglability bound also failed since the contours near $\lambda_{\mathrm{min}} = 0$ diverged sharply from the bound (see Figs. 4-14, 4-15, and 4-16).

Chapter 5

Negativity formulas for standard Bell-transformed thermal states

This chapter derives analytical expressions for the negativity of the standard Bell-transformed thermal state $\rho_{\mathrm{tf}} = U_{\mathrm{b,s}}\rho_{\mathrm{th}}U_{\mathrm{b,s}}^{\dagger}$. First, we calculate negativity under the $\{1, N-1\}$ bipartite split using a direct theoretical analysis (Section 5.1). Second, we calculate negativity under the $\{N/2, N/2\}$ bipartite split using an empirically-based analysis (Section 5.2). Finally, we plot entanglement maps for the two splits and discuss the features of these maps as well as the experimental feasibility of realizing ρ_{tf} (Section 5.3).

5.1 Theoretical analysis for $\{1, N-1\}$ bipartite split

In this section, we analytically calculate the negativity of ρ_{tf} under the $\{1, N-1\}$ bipartite split.

The quantum circuit for $U_{\mathrm{b,s}}$ (see Fig. 3-2) suggests that the first qubit is special. The expression for the thermal state from Eq. 2.59 can be rewritten to separate the behavior of the first qubit from the others:

$$\rho_{\mathrm{th}} = \frac{e^{-\mathcal{H}\alpha}}{\mathcal{Z}} \tag{5.1}$$

$$
\begin{aligned}
&= \frac{1}{\mathcal{Z}} \exp\left[-\left(\frac{1}{2}\sum_{i=1}^{N} Z_i\right)\alpha\right] \\
&= \frac{1}{\mathcal{Z}} \exp\left[-\frac{1}{2}Z_1\alpha\right] \exp\left[-\left(\frac{1}{2}\sum_{i=2}^{N} Z_i\right)\alpha\right] \\
&= \frac{1}{\mathcal{Z}}\sum_{j,m} \left[e^{\alpha/2}\left|0\right\rangle\left\langle 0\right| + e^{-\alpha/2}\left|1\right\rangle\left\langle 1\right|\right] e^{-m\alpha}\left|j,m\right\rangle\left\langle j,m\right| .
\end{aligned}
$$

where in the second line, we have substituted Eq. 2.54 and in the third line, we have used the fact that the Z_i commute with one another. The first qubit's state is expressed in the computational basis $\{\left|0\right\rangle, \left|1\right\rangle\}$. The terms $e^{\alpha/2}$ and $e^{-\alpha/2}$ are the energy contributions when the first qubit is spin up $\left|0\right\rangle$ and spin down $\left|1\right\rangle$ respectively. We have inserted a complete set of angular momentum states, $\left|j,m\right\rangle\left\langle j,m\right|$, to represent the collective spin state of the latter $N-1$ qubits [Sak94]. The symbol j denotes the total spin angular momentum, and $m = -j, -j+1, \ldots, j-1, j$ represents the total azimuthal spin angular momentum in the z-direction. Since the Hamiltonian simply counts the total azimuthal spin, the latter $N-1$ qubits contribute an overall factor $e^{-m\alpha}$ to the energy.

Eq. 5.1 can be rewritten more compactly to give

$$
\rho_{\text{th}} = \frac{1}{\mathcal{Z}}\sum_{j,m} e^{\alpha Z_1/2} e^{-m\alpha}\left|j,m\right\rangle\left\langle j,m\right| \tag{5.2}
$$

where Z_1 is the Pauli Z operator acting on the first qubit.

Now we examine the effect of $U_{\text{b,s}}$ on the thermal state. It is easier to analyze the problem if we break the unitary into two parts: the application of the Hadamard gate on the first qubit (H_1) followed by a massive controlled NOT on the other qubits (U_{cnot}) or mathematically, $U_{\text{b,s}} = H_1 U_{\text{cnot}}$.

Applying a Hadamard operation to the first qubit of ρ_{th} (note that $H_1 = H_1^\dagger$) yields the state

$$
\begin{aligned}
\rho'_{\text{th}} &= H_1 \rho_{\text{th}} H_1 \qquad (5.3) \\
&= \frac{1}{\mathcal{Z}}\sum_{j,m} e^{\alpha X_1/2} e^{-m\alpha}\left|j,m\right\rangle\left\langle j,m\right|
\end{aligned}
$$

since $HZH = X$. The operator X_1 represents a Pauli X operation on the first qubit. Expanding $e^{\alpha X_1/2}$, this equation can be rewritten as

$$\begin{aligned}
\rho'_{\text{th}} &= \frac{1}{\mathcal{Z}}\sum_{j,m}\left[\left(\cosh\frac{\alpha}{2}\right)I_1 + \left(\sinh\frac{\alpha}{2}\right)X_1\right]e^{-m\alpha}\left|j,m\right\rangle\left\langle j,m\right| \qquad (5.4)\\
&= \frac{1}{\mathcal{Z}}\sum_{j,m}\left[\cosh\frac{\alpha}{2}(\left|0\right\rangle\left\langle 0\right| + \left|1\right\rangle\left\langle 1\right|) + \sinh\frac{\alpha}{2}(\left|0\right\rangle\left\langle 1\right| + \left|1\right\rangle\left\langle 0\right|)\right]e^{-m\alpha}\left|j,m\right\rangle\left\langle j,m\right| .
\end{aligned}$$

where in the first line, I_1 is the two-dimensional identity matrix acting on the first qubit and in the second line, we have expanded I_1 and X_1 in the outer products of the computational basis states.

To obtain the fully transformed state, we apply the CNOT operation: $U_{\text{cnot}}\rho'_{\text{th}}U^{\dagger}_{\text{cnot}}$. Since $U_{\text{cnot}}\left|0\right\rangle\left|j,m\right\rangle = \left|0\right\rangle\left|j,m\right\rangle$ and $U_{\text{cnot}}\left|1\right\rangle\left|j,m\right\rangle = \left|1\right\rangle\left|j,-m\right\rangle$, we see that ρ_{tf} is composed of two-dimensional subspaces spanned by $\left|0\right\rangle\left|j,m\right\rangle$ and $\left|1\right\rangle\left|j,-m\right\rangle$:

$$\rho_{\text{tf}} = \bigoplus_{j,m}\frac{e^{-m\alpha}}{\mathcal{Z}}\begin{bmatrix}\cosh\frac{\alpha}{2} & \sinh\frac{\alpha}{2}\\ \sinh\frac{\alpha}{2} & \cosh\frac{\alpha}{2}\end{bmatrix} . \qquad (5.5)$$

Next, we calculate the partial transposed state $\rho_{\text{tf}}^{T_A}$ with respect to the $\{1, N-1\}$ split. The partial transpose solely mixes the eigenstates $\left|0\right\rangle\left|j,\pm m\right\rangle$ and $\left|1\right\rangle\left|j,\pm m\right\rangle$ according to Eq. 2.33. Thus we need only look at the subspace

$$\frac{1}{\mathcal{Z}}\begin{bmatrix}e^{m\alpha}\cosh\frac{\alpha}{2} & e^{m\alpha}\sinh\frac{\alpha}{2} & 0 & 0\\ e^{m\alpha}\sinh\frac{\alpha}{2} & e^{m\alpha}\cosh\frac{\alpha}{2} & 0 & 0\\ 0 & 0 & e^{-m\alpha}\cosh\frac{\alpha}{2} & e^{-m\alpha}\sinh\frac{\alpha}{2}\\ 0 & 0 & e^{-m\alpha}\sinh\frac{\alpha}{2} & e^{-m\alpha}\cosh\frac{\alpha}{2}\end{bmatrix} \qquad (5.6)$$

in the basis $\left|0\right\rangle\left|j,-m\right\rangle$, $\left|1\right\rangle\left|j,m\right\rangle$, $\left|0\right\rangle\left|j,m\right\rangle$, and $\left|1\right\rangle\left|j,-m\right\rangle$.

Performing the partial transpose with respect to the $\{1, N-1\}$ split on the above subspace gives

$$\frac{1}{\mathcal{Z}}\begin{bmatrix} e^{m\alpha}\cosh\frac{\alpha}{2} & e^{-m\alpha}\sinh\frac{\alpha}{2} & 0 & 0 \\ e^{-m\alpha}\sinh\frac{\alpha}{2} & e^{m\alpha}\cosh\frac{\alpha}{2} & 0 & 0 \\ 0 & 0 & e^{-m\alpha}\cosh\frac{\alpha}{2} & e^{m\alpha}\sinh\frac{\alpha}{2} \\ 0 & 0 & e^{m\alpha}\sinh\frac{\alpha}{2} & e^{-m\alpha}\cosh\frac{\alpha}{2} \end{bmatrix}. \tag{5.7}$$

This matrix is in block diagonal form, and we can easily solve for its eigenvalues.

Using the fact that

$$\lambda_{\pm} = \frac{\operatorname{tr} M \pm \sqrt{(\operatorname{tr} M)^2 - 4\det(M)}}{2}, \tag{5.8}$$

for a 2×2 matrix M, the eigenvalues of the matrix in Eq. 5.7 are

$$\lambda_{\pm} = \frac{e^{m\alpha}\cosh\frac{\alpha}{2} \pm e^{-m\alpha}\sinh\frac{\alpha}{2}}{\mathcal{Z}}. \tag{5.9}$$

Because the largest possible m-value is $(N-1)/2$ [Sak94], the minimum eigenvalue of $\rho_{\mathrm{tf}}^{T_A}$ for a $\{1, N-1\}$ bipartite split is

$$\boxed{\lambda_{\min} = \frac{e^{-(N-1)\alpha/2}\cosh\frac{\alpha}{2} - e^{(N-1)\alpha/2}\sinh\frac{\alpha}{2}}{\mathcal{Z}}, \quad \{1, N-1\} \text{ split} \qquad (5.10)}$$

which can be inserted into Eq. 2.36 to give the negativity.

When $\lambda_{\min}$ is negative, ρ_{tf} is entangled. Since $\lambda_{\min}$ is monotonic in N and α, the values of N and α where $\lambda_{\min} = 0$ give a bound on ρ_{tf} being nonseparable. From the previous equation, we see that the minimum eigenvalue becomes zero when

$$e^{(N-1)\alpha}\tanh\frac{\alpha}{2} = 1. \tag{5.11}$$

Changing the equal sign to a greater than sign, we obtain a bound on nonseparable ρ_{tf}:

$$e^{(N-1)\alpha}\tanh\frac{\alpha}{2} > 1. \tag{5.12}$$

5.2 Empirical analysis for $\{N/2, N/2\}$ bipartite split

We now derive an analytical formula for the negativity of the Bell-transformed thermal state under the $\{N/2, N/2\}$ partition. The analysis that follows is empirically derived, but it will be theoretically confirmed in the next chapter.

If we plot the eigenvalue levels of the partial transposed density matrix $\rho_{\mathrm{tf}}^{T_A}$ as a function of $\log_{10}(\alpha^{-1})$, we discover that $\lambda_{\min}$ always corresponds to one of two eigenvectors at fixed N. Beginning at low values of α^{-1}, the minimum eigenvalue has a constant eigenvector which suddenly changes to a different eigenvector at a transition point in α^{-1}. In fact, the transition point corresponds to the point where the two lowest eigenvalue levels cross, as shown in Fig. 5-1.

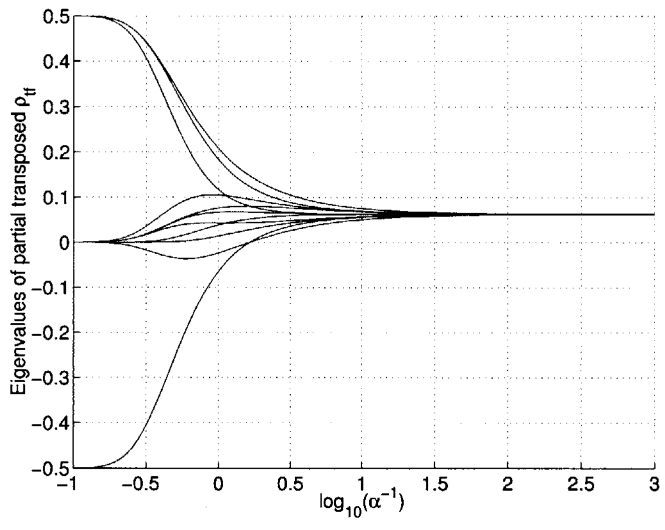

Figure 5-1: Eigenvalue levels of $\rho_{\mathrm{tf}}^{T_A}$ as a function of $\log_{10}(\alpha^{-1})$ for $N = 4$. Note the crossing of the lowest two levels at $\log_{10}(\alpha^{-1}) \approx 0.5$.

Moreover, the two eigenvectors are easily predicted as a function of N and sparse in the computational basis. Based on numerical evidence, we conjecture that the eigenvector corresponding to the minimum eigenvalue is given by

$$\begin{aligned} |v_{\min}\rangle &= |v_-\rangle\,,\ \alpha^{-1} < \alpha_{\mathrm{tr}}^{-1} \\ &= |v_+\rangle\,,\ \alpha^{-1} \geq \alpha_{\mathrm{tr}}^{-1} \end{aligned} \tag{5.13}$$

with α_{tr} being the polarization value where the transition in eigenvectors occurs and

$$|v_-\rangle = \frac{1}{\sqrt{2}}\left(\left|2^{N/2}-1\right\rangle - \left|2^N - 2^{N/2}\right\rangle\right) \tag{5.14}$$

$$|v_+\rangle = \frac{1}{\sqrt{2}}\left(\left|2^{N-1}-1\right\rangle - \left|2^{N-1}\right\rangle\right). \tag{5.15}$$

The decimal labels inside the kets represent the computational basis state when it is expressed in binary notation, i.e. $\left|2^{N-1}-1\right\rangle = |0111\rangle$ when $N=4$. Fig. 5-2 shows the excellent agreement in eigenvalue when these vectors are used.

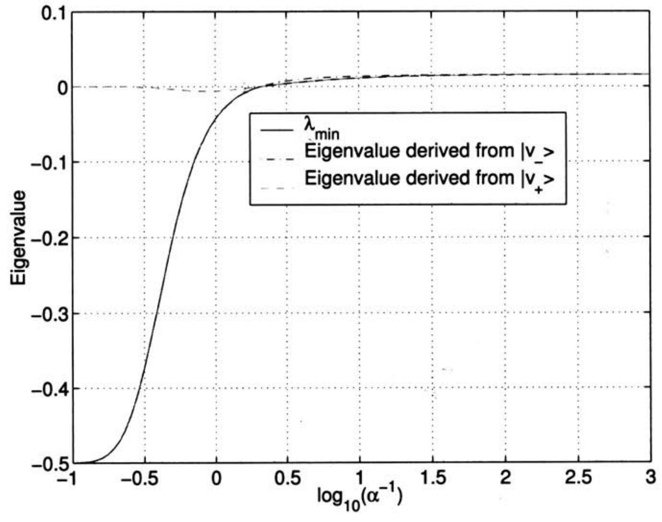

Figure 5-2: Comparison of eigenvalues derived from $|v_-\rangle$ (dash dotted line) and $|v_+\rangle$ (dotted line) to the true minimum eigenvalue λ_{min} (solid line), plotted as a function of $\log_{10}(\alpha^{-1})$ for $N=6$.

We numerically verify the conjecture in Eq. 5.13 for $N = \{2,4,6,8,10\}$ by defining a prediction error $\lambda_{\text{min}} - \hat{\lambda}_{\text{min}}$ where λ_{min} is the minimum eigenvalue of $\rho_{\text{tf}}^{T_A}$ calculated by MATLAB and $\hat{\lambda}_{\text{min}}$ is the minimum eigenvalue derived from the hypothesized eigenvectors above. The prediction error is always smaller than 10^{-15} and thus below MATLAB floating point precision. A sample plot of the prediction error is shown in Fig. 5-3 for the case $N=6$.

How can we interpret these mysterious eigenvectors? One might think that they correspond to the two lowest eigenvalue levels of the partial transposed state. Unfor-

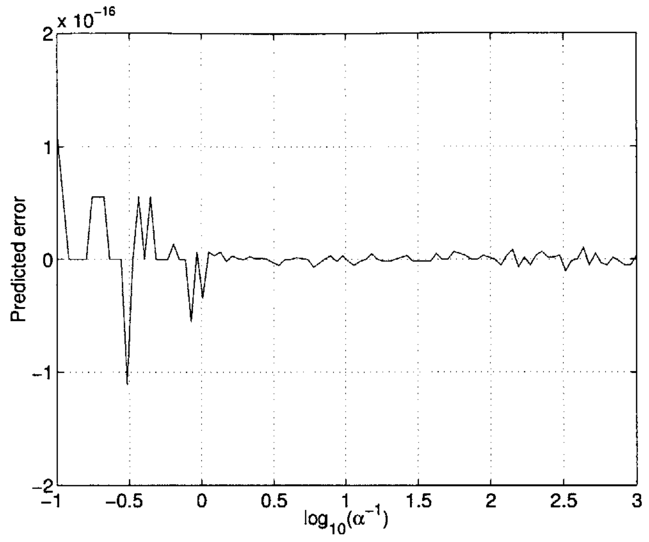

Figure 5-3: Prediction error $\lambda_{\text{min}} - \hat{\lambda}_{\text{min}}$ as a function of $\log_{10}(\alpha^{-1})$ for $N = 6$.

tunately, a more thorough comparison of the actual and predicted eigenvalues proves this hypothesis false. Fig. 5-4 shows that the eigenvector in Eq. 5.15 tracks the second lowest level at very low α^{-1}, but deviates as α^{-1} increases.

Numerical evidence suggests that the transition from one eigenvector to the other is abrupt. However, since we are varying a continuous parameter α, we expect the transition to be smooth; λ_{min} would have the eigenvector $|v_-\rangle$ at low α^{-1}, gradually acquire a component in the direction of $|v_+\rangle$, before switching entirely to eigenvector $|v_+\rangle$. The mixing might occur in such a tiny range of α^{-1} that the behavior escapes numerical examination.

To test this hypothesis, we calculate the mixing terms in the 4×4 computational subspace of $\rho_{\text{tf}}^{T_A}$ spanned by $|v_-\rangle$ and $|v_+\rangle$. The elements of the subspace may be labeled

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix} \tag{5.16}$$

with basis ordering $\left|2^{N/2} - 1\right\rangle$, $\left|2^N - 2^{N/2}\right\rangle$, $\left|2^{N-1} - 1\right\rangle$, and $\left|2^{N-1}\right\rangle$. With this definitions, the mixing terms are A_{13}, A_{14}, A_{23}, A_{24}, A_{31}, A_{32}, A_{41}, and A_{42}.

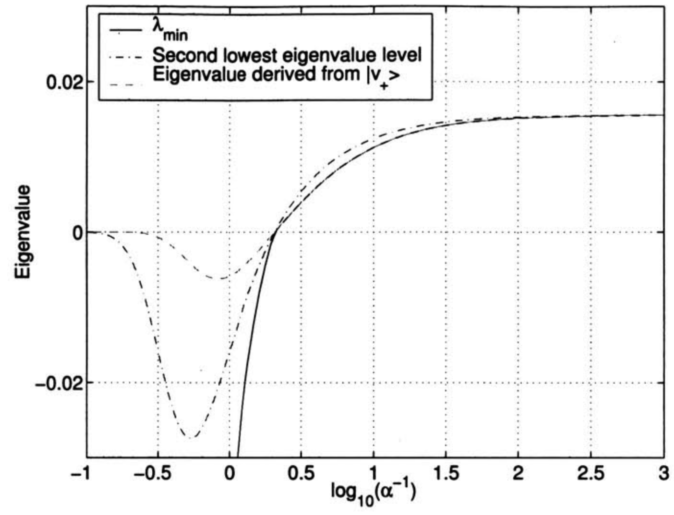

Figure 5-4: Comparison of eigenvalue derived from $|v+\rangle$ (dashed line) to the λ_{min} (solid line) and the second lowest eigenvalue level (dash dotted line), plotted as a function of $\log_{10}(\alpha^{-1})$ for $N = 6$.

Before we calculate them, let us explain our notation. Greek symbols, for instance $A_{\mu\nu}$, are numeric indices. Roman letters, for instance $|i\rangle\langle j|$, represent computational basis states in binary notation. Note that $A_{\mu\nu} = A^{\dagger}_{\mu\nu}$, so we need only calculate half of the mixing terms. This property follows from the facts that ρ_{tf} is real and Hermitian and that the partial transpose preserves Hermiticity: $(|i\rangle\langle j|)^{T_A} = ((|j\rangle\langle i|)^{T_A})^{\dagger}$.

We compute $A_{\mu\nu}$ with the following steps:

1. Write out the outer product $|i\rangle\langle j|$ corresponding to $A_{\mu\nu} = \langle i|\rho_{\text{tf}}^{T_A}|j\rangle$.

2. Perform the $\{N/2, N/2\}$ split partial transpose on the outer product to find $|k\rangle\langle l| = (|i\rangle\langle j|)^{T_A}$. This new outer product $|k\rangle\langle l|$ gives us the matrix coordinates for the entry $\langle k|\rho_{\text{tf}}|l\rangle$ that is equal to $A_{\mu\nu}$.

3. Calculate $\langle k|\rho_{\text{tf}}|l\rangle$. Since $U_{\text{b,s}}$ is so sparse and ρ_{th} is diagonal, $\langle k|\rho_{\text{tf}}|l\rangle$ is equal to a sum of entries in ρ_{th} or zero.

We are essentially backtracking through the original negativity calculation described in Section 3.4.1.

To illustrate the process, we carry out the calculation for A_{41}. The term corresponds to the $\left|2^{N-1}\right\rangle\left\langle 2^{N/2}-1\right|$ outer product and can be rewritten in binary notation as $|10...0:00...0\rangle\langle 00...0:11...1|$.[1] The partial transpose of the matrix element is $|00...0:00...0\rangle\langle 10...0:11...1| = |0\rangle\left\langle 2^{N-1}+2^{N/2}-1\right|$. Therefore we have $\left\langle 2^{N-1}\right|\rho_{\text{tf}}^{T_A}\left|2^{N/2}-1\right\rangle = \langle 0|\,\rho_{\text{tf}}\left|2^{N-1}+2^{N/2}-1\right\rangle$. By looking at the structure of $U_{\text{b,s}}$ and ρ_{th}, we see that $\langle 0|\,\rho_{\text{tf}}\left|2^{N-1}+2^{N/2}-1\right\rangle$ is only nonzero if $2^{N-1}+2^{N/2}=2^N$, which is true for $N=2$.

The calculation results for all the mixing $A_{\mu\nu}$ are summarized in Table 5.1.

Term	$\rho_{\text{tf}}^{T_A}$ **outer product**	ρ_{tf} **outer product**	**Value of term**
$A_{31}=A_{13}$	$\|01...1\|11...1\rangle\langle 00...0\|11...1\|$	$\left\|2^{N/2}-1\right\rangle\left\langle 2^{N-1}-1\right\|$	0 unless $N=2$
$A_{41}=A_{14}$	$\|10...0\|00...0\rangle\langle 00...0\|11...1\|$	$\|0\rangle\left\langle 2^{N-1}+2^{N/2}-1\right\|$	0 unless $N=2$
$A_{32}=A_{23}$	$\|01...1\|00...0\rangle\langle 11...1\|00...0\|$	$\left\|2^{N}-1\right\rangle\left\langle 2^{N-1}-2^{N/2}\right\|$	0 unless $N=2$
$A_{42}=A_{24}$	$\|10...0\|00...0\rangle\langle 11...1\|00...0\|$	$\left\|2^{N}-2^{N/2}\right\rangle\left\langle 2^{N-1}\right\|$	0 unless $N=2$

Table 5.1: Calculation steps to find the values of off-diagonal $A_{\mu\nu}$

The results in Table 5.1 show that there is no mixing unless $N=2$. However, when $N=2$, $|v_-\rangle = |v_+\rangle$. We conclude that vectors $|v_-\rangle$ and $|v_+\rangle$ do not mix under the $\{N/2, N/2\}$ bipartite split.

We can also obtain an analytical formula for the hypothesized minimum eigenvalue. Using the fact that $A\,|v_\pm\rangle = \lambda_\pm\,|v_\pm\rangle$, we find

$$\boxed{\hat{\lambda}_{\text{min}} = \min(\lambda_-, \lambda_+) \qquad (5.17)}$$

where

$$\lambda_- = A_{11} - A_{21} \qquad (5.18)$$

$$\lambda_+ = A_{33} - A_{43}\,. \qquad (5.19)$$

Table 5.2 gives the formulas for A_{11}, A_{21}, A_{33}, and A_{43}, which are calculated in the same way as the mixing terms.

[1]Notational note: The colon in the middle of the ket and bra divide the first $N/2$ qubits from the latter $N/2$ qubits.

Term	Value of term
A_{11}	$\frac{1}{2}\left[\left\langle 2^{N/2}-1\right\vert \rho_{\mathrm{th}} \left\vert 2^{N/2}-1\right\rangle + \left\langle 2^{N/2}+2^{N-1}-1\right\vert \rho_{\mathrm{th}} \left\vert 2^{N/2}+2^{N-1}-1\right\rangle\right]$
A_{21}	$\frac{1}{2}\left[\langle 0\vert \rho_{\mathrm{th}} \vert 0\rangle + \left\langle 2^{N-1}\right\vert \rho_{\mathrm{th}} \left\vert 2^{N-1}\right\rangle\right]$
A_{33}	$\frac{1}{2}\left[\left\langle 2^{N-1}-1\right\vert \rho_{\mathrm{th}} \left\vert 2^{N-1}-1\right\rangle + \left\langle 2^{N}-1\right\vert \rho_{\mathrm{th}} \left\vert 2^{N}-1\right\rangle\right]$
A_{43}	$\frac{1}{2}\left[\left\langle 2^{N-1}-2^{N/2}\right\vert \rho_{\mathrm{th}} \left\vert 2^{N-1}-2^{N/2}\right\rangle - \left\langle 2^{N}-2^{N/2}\right\vert \rho_{\mathrm{th}} \left\vert 2^{N}-2^{N/2}\right\rangle\right]$

Table 5.2: Explicit formulas for $A_{\mu\nu}$ that are needed to derive an analytical formula for λ_{min}.

The formulas in Table 5.2 may be re-expressed in terms of α using the formula

$$\langle i\vert \rho_{\mathrm{th}} \vert i\rangle = e^{[N-2w(i)]\alpha/2} \tag{5.20}$$

from Eq. 2.61. The Hamming weight $w(i)$ is the number of 1s in the binary expression for i.

Now we can calculate the eigenvalues corresponding to $\vert v_{\mathrm{min}}\rangle$ by substituting results from Table 5.2 into Eqs. 5.18-5.19. For $\vert v_-\rangle$, we have

$$\boxed{\begin{aligned} \lambda_- &= \frac{1+e^{-\alpha}-e^{N\alpha/2}+e^{(N-2)\alpha/2}}{2\mathcal{Z}} \\ &= \frac{e^{-\alpha/2}(\cosh\frac{\alpha}{2}-e^{N\alpha/2}\sinh\frac{\alpha}{2})}{\mathcal{Z}}, \quad \{N/2,N/2\} \text{ split} \end{aligned} \quad (5.21)}$$

For $\vert v_+\rangle$, we have

$$\boxed{\begin{aligned} \lambda_+ &= \frac{e^{(-N+2)\alpha/2}+e^{-N\alpha/2}-e^{\alpha}+1}{2\mathcal{Z}} \\ &= \frac{e^{\alpha/2}(e^{-N\alpha/2}\cosh\frac{\alpha}{2}-\sinh\frac{\alpha}{2})}{2\mathcal{Z}}, \quad \{N/2,N/2\} \text{ split} \end{aligned} \quad (5.22)}$$

Eqs. 5.17, 5.21, and 5.22 can be used to calculate the negativity of ρ_{tf} under the $\{N/2,N/2\}$ bipartite split.

An interesting question is where do λ_+ and λ_- cross? Note that $\lambda_+ - \lambda_-$ is a monotonic function in α since

$$\begin{aligned} \lambda_+ - \lambda_- &= \frac{e^{(-N+2)\alpha} + e^{-N\alpha} - e^{2\alpha} - e^{-2\alpha} + e^{N\alpha} - e^{(N-2)\alpha}}{2\mathcal{Z}} \\ &= \frac{\cosh(N\alpha) - \sinh\left[(N-2)\alpha\right] - \cosh(2\alpha)}{\mathcal{Z}}. \end{aligned} \tag{5.23}$$

Therefore, there is only one crossing, which occurs at $\lambda_{\text{min}} = \lambda_+ = \lambda_- = 0$. If we set Eqs. 5.21 and 5.22 to zero, we obtain

$$e^{N\alpha/2} \tanh\frac{\alpha}{2} = 1 \tag{5.24}$$

for both expressions. Changing the equal sign to a greater than sign, we find a bound on nonseparable ρ_{tf}:

$$e^{N\alpha/2} \tanh\frac{\alpha}{2} > 1 \tag{5.25}$$

5.3 Negativity maps for $\{1, N-1\}$ and $\{N/2, N/2\}$ bipartite splits

The results from the previous two sections allow us to plot the negativity E_n in $N - \alpha$ parameter space for the $\{1, N-1\}$ and $\{N/2, N/2\}$ bipartite splits. First, we examine the $\{1, N-1\}$ split. Fig. 5-5 shows the negativity plotted up to 500 qubits. The bound on nonseparable ρ_{tf} bends to the right of the Braunstein et.al. bound on effective pure state nonentanglability, implying that there are entanglable thermal states not captured by the bound.

In addition, λ_{min}, as a function of α, behaves differently at large fixed N than at small fixed N. As the number of qubits increases, the value of α corresponding to $\lambda_{\text{min}} = 0$ decreases, but the derivative of the minimum eigenvalue around this α becomes increasingly shallow. This behavior is seen most clearly in the negativity map plotted in Fig. 5-6 where we have plotted the logarithm of negativity to accentuate small variations from $E_n = 0$. One might speculate that as number of qubits increases, it is easier to entangle a thermal state with $U_{\mathrm{b,s}}$ but the unitary's entanglement power diminishes. Unfortunately, since the quantitative interpretation of negativity is poorly understood, it is hard to justify this statement without additional information.

The negativity maps for the $\{N/2, N/2\}$ case are qualitatively the same as in the $\{1, N-1\}$ split (see Figs. 5-7 and 5-8). The major difference is that the bound on nonseparable ρ_{tf} bends closer to the left; it is less tight for the $\{N/2, N/2\}$ split, as seen in Fig. 5-9. Thus, the $\{1, N-1\}$ split gives better entanglement information.

Finally, can we entangle a thermal state in experimentally accessible parameter space? By numerically solving Eq. 5.12, we find that for the highest experimentally feasible polarization ($\alpha = 8.79 \times 10^{-5}$), we require at least 114,135 qubits to achieve entanglement. This number is clearly impractical, but it dramatically improves upon the Schulman-Vazirani bound of 1.29×10^{8} qubits from Section 3.2.

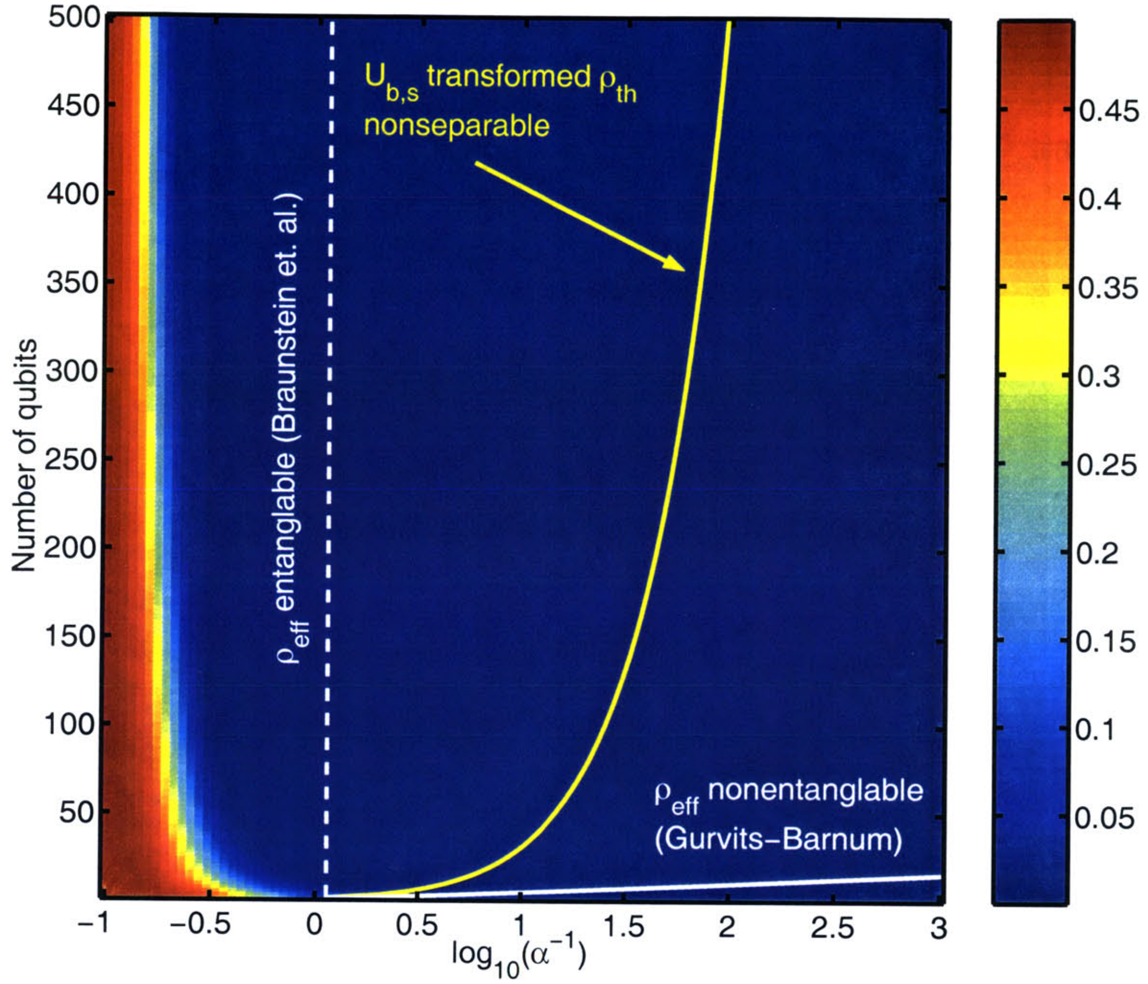

Figure 5-5: Color map of $E_n(\rho_{\text{tf}})$ for $\{1, N-1\}$ bipartite split in $N - \alpha$ parameter space, overlaid with Gurvits and Barnum bound on the nonentanglability of ρ_{eff} (solid white line), Braunstein, et. al. bound on the entanglability of ρ_{eff} (dashed white line), and bound on the nonseparability of ρ_{tf} from Eq. 5.12 (solid yellow line). The bar on the right ascribes a color to each value of E_n.

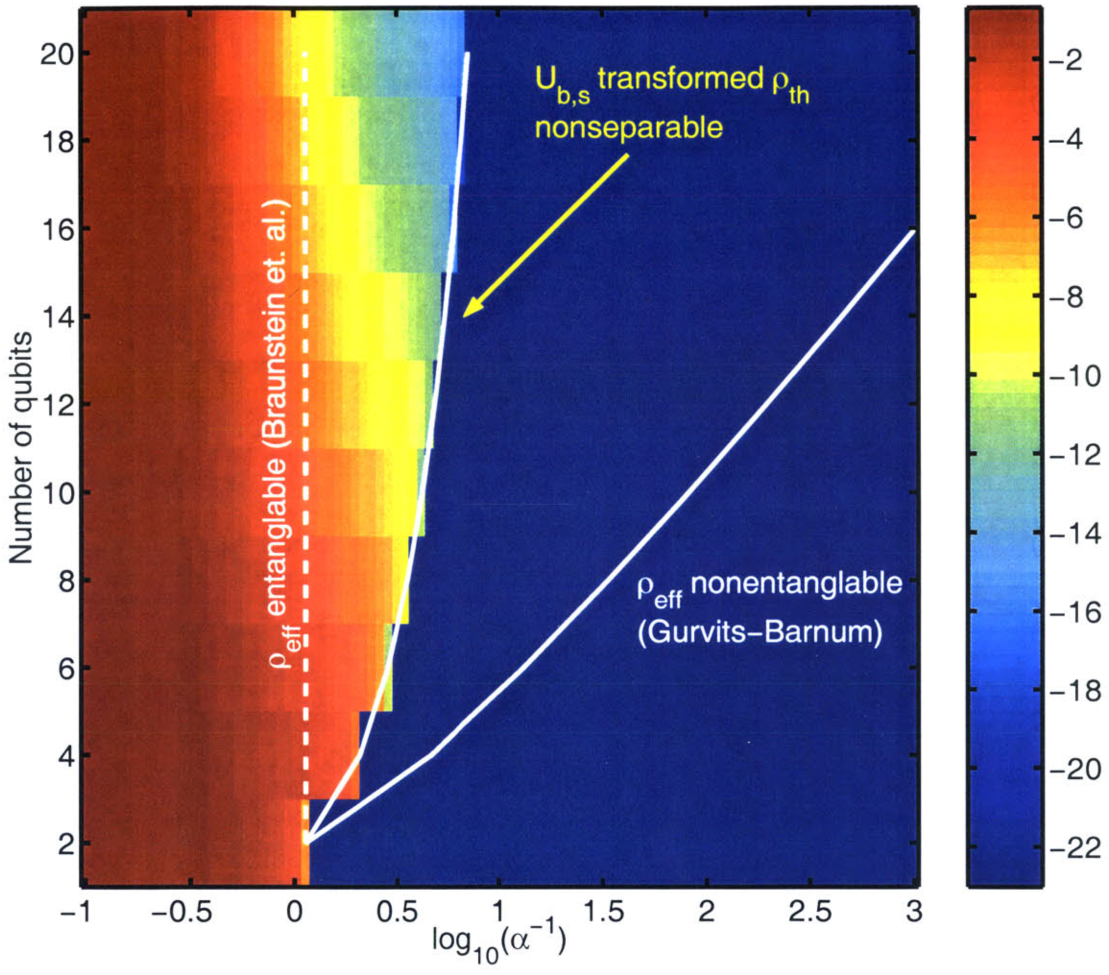

Figure 5-6: Color map of $\log\left[E_n(\rho_{\text{tf}}) + 10^{-10}\right]$ for $\{1, N-1\}$ bipartite split in $N - \alpha$ parameter space, overlaid with Gurvits and Barnum bound on the nonentanglability of ρ_{eff} (solid white line), Braunstein, et. al. bound on the entanglability of ρ_{eff} (dashed white line), and bound on the nonseparability of ρ_{tf} from Eq. 5.12 (solid white line). The bar on the right ascribes a color to each value of $\log\left[E_n(\rho_{\text{tf}}) + 10^{-10}\right]$.

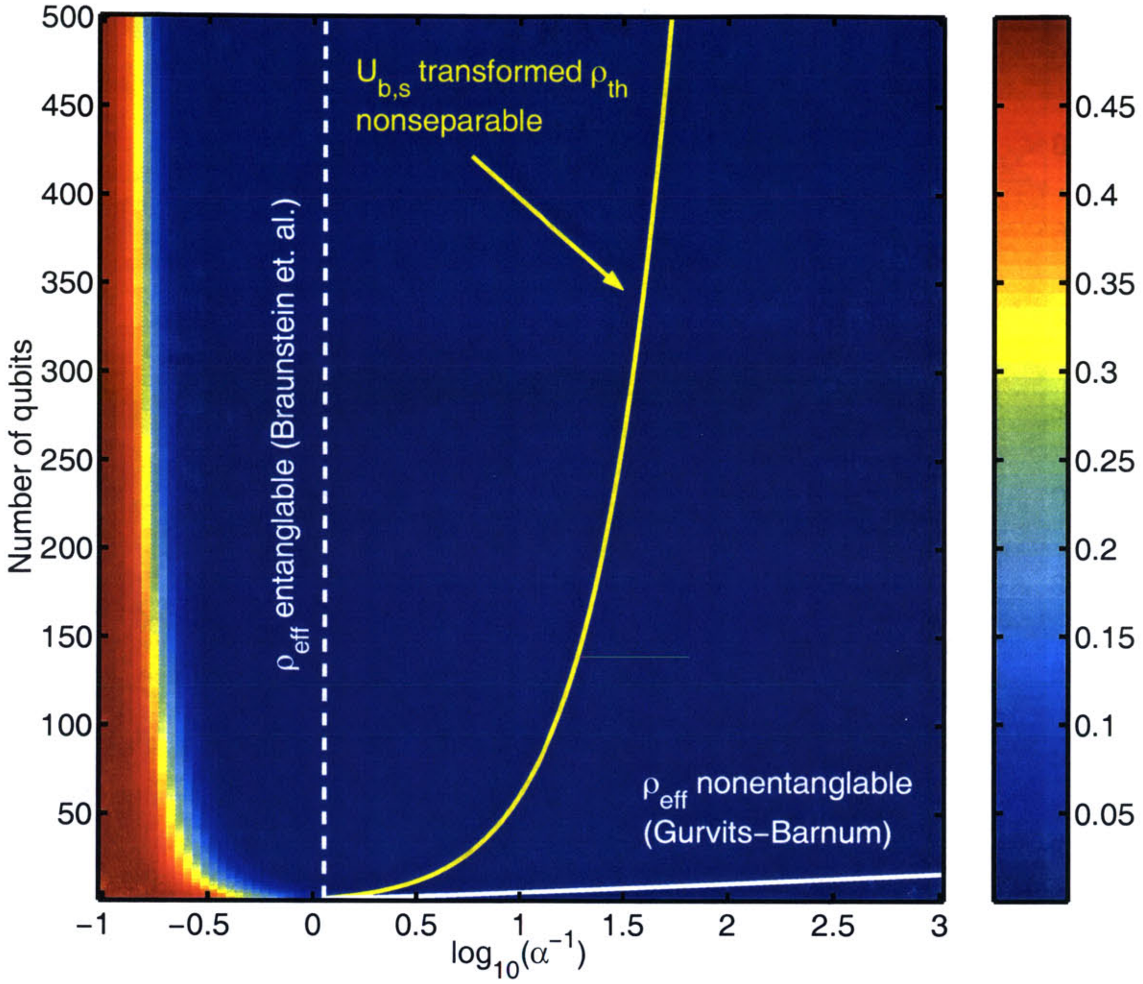

Figure 5-7: Color map of $E_n(\rho_{\text{tf}})$ for $\{N/2.N/2\}$ bipartite split in $N - \alpha$ parameter space, overlaid with Gurvits and Barnum bound on the nonentanglability of ρ_{eff} (solid white line), Braunstein, et. al. bound on the entanglability of ρ_{eff} (dashed white line), and bound on the nonseparability of ρ_{tf} from Eq. 5.25 (solid yellow line). The bar on the right ascribes a color to each value of E_n.

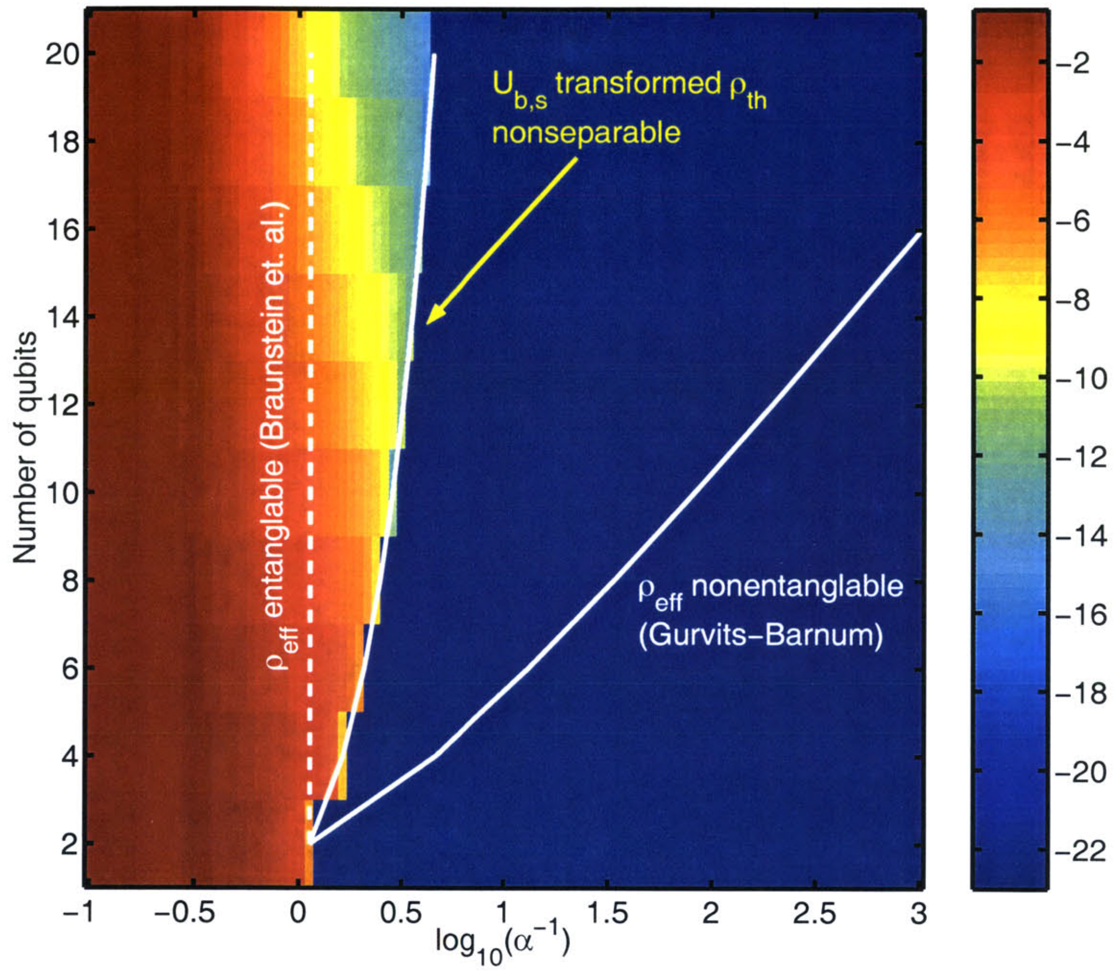

Figure 5-8: Color map of $\log\left[E_n(\rho_{\rm tf}) + 10^{-10}\right]$ for $\{N/2.N/2\}$ bipartite split in $N-\alpha$ parameter space, overlaid with Gurvits and Barnum bound on the nonentanglability of $\rho_{\rm eff}$ (solid white line), Braunstein, et. al. bound on the entanglability of $\rho_{\rm eff}$ (dashed white line), and bound on the nonseparability of $\rho_{\rm tf}$ from Eq. 5.25 (solid white line). The bar on the right ascribes a color to each value of $\log\left[E_n(\rho_{\rm tf}) + 10^{-10}\right]$.

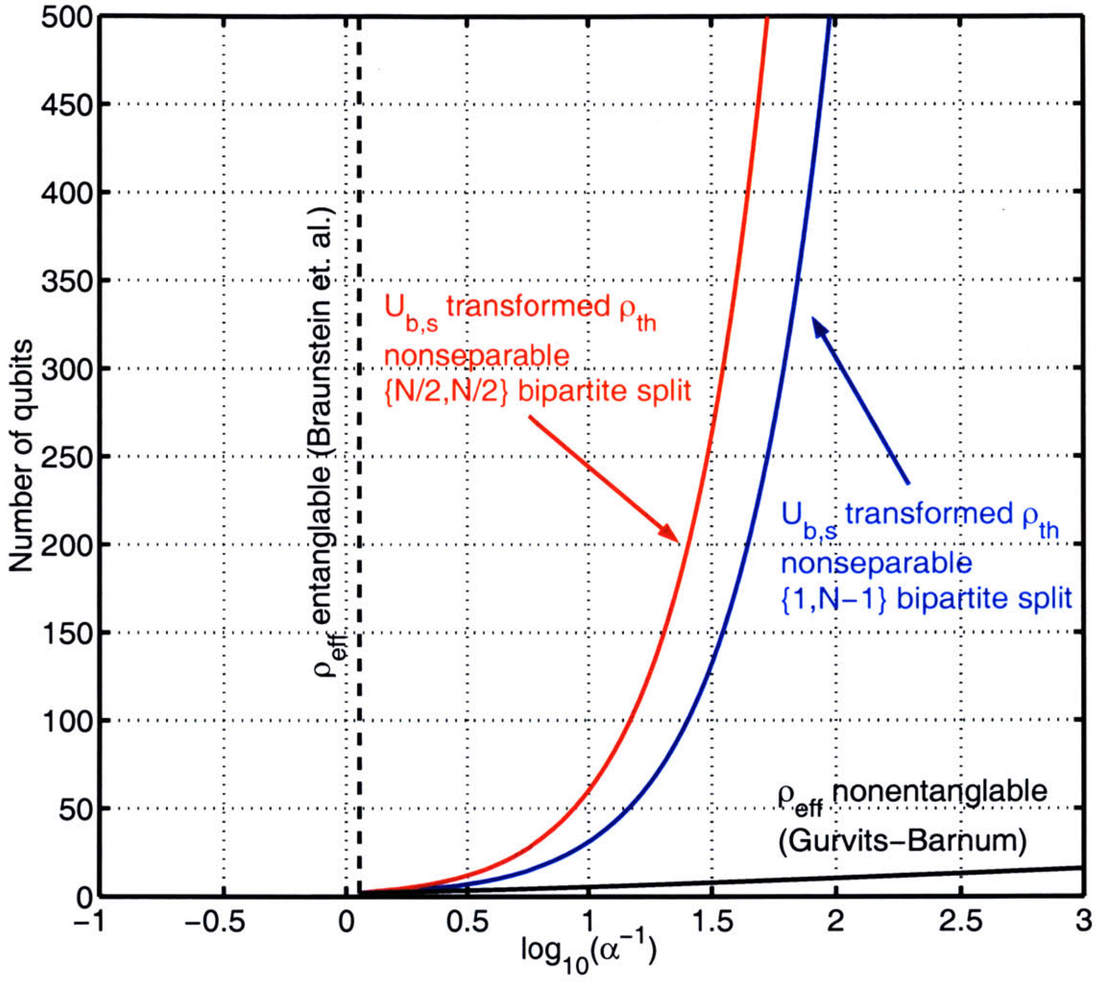

Figure 5-9: Comparison of bounds on nonseparable ρ_{tf} ($\lambda_{\min} = 0$) for $\{1, N-1\}$ (solid blue line) and $\{N/2, N/2\}$ (solid red line) bipartite splits, overlaid with Gurvits and Barnum bound on nonentanglability of ρ_{eff} (solid black line) and Braunstein, et. al. bound on entanglability of ρ_{eff} (dashed black line).

5.4 Summary

In this chapter, we calculated the negativity of the standard Bell-transformed thermal state $\rho_{\mathrm{tf}} = U_{\mathrm{b,s}}\rho_{\mathrm{th}}U_{\mathrm{b,s}}^{\dagger}$ for the $\{1, N-1\}$ and $\{N/2, N/2\}$ bipartite splits.

For the $\{1, N-1\}$ split, we computed the negativity through direct theoretical analysis. By exploiting the symmetry of the $U_{\mathrm{b,s}}$ transform, we found that the minimum eigenvalue of $\rho_{\mathrm{tf}}^{T_A}$ was

$$\lambda_{\min} = \frac{e^{-(N-1)\alpha/2}\cosh\frac{\alpha}{2} - e^{(N-1)\alpha/2}\sinh\frac{\alpha}{2}}{\mathcal{Z}}\,, \tag{5.26}$$

which gave a bound on the nonseparability of ρ_{tf} that was determined by the expression

$$e^{(N-1)\alpha}\tanh\frac{\alpha}{2} > 1\,. \tag{5.27}$$

The negativity E_n could be calculated from $\lambda_{\min}$ with the formula

$$E_n(\rho) = \max\{0, -\lambda_{\min}\}\,. \tag{5.28}$$

For the $\{N/2, N/2\}$ split, we computed the negativity through an empirically motivated analysis. Numerical exploration suggested that the eigenvector corresponding to $\lambda_{\min}$ was

$$\begin{aligned} |v_{\min}\rangle &= |v_-\rangle\,,\ \alpha^{-1} < \alpha_{\mathrm{tr}}^{-1} \\ &= |v_+\rangle\,,\ \alpha^{-1} \geq \alpha_{\mathrm{tr}}^{-1} \end{aligned} \tag{5.29}$$

with α_{tr} being the polarization value where the transition in eigenvectors occurs and

$$|v_-\rangle = \frac{1}{\sqrt{2}}\left(\left|2^{N/2}-1\right\rangle - \left|2^N - 2^{N/2}\right\rangle\right) \tag{5.30}$$

$$|v_+\rangle = \frac{1}{\sqrt{2}}\left(\left|2^{N-1}-1\right\rangle - \left|2^{N-1}\right\rangle\right)\,. \tag{5.31}$$

Knowledge of $|v_{\text{min}}\rangle$ allowed us to predict the minimum eigenvalue

$$\hat{\lambda}_{\text{min}} = \min(\lambda_-, \lambda_+) \tag{5.32}$$

with λ_+ and λ_- given by

$$\begin{aligned} \lambda_- &= \frac{e^{-\alpha/2}(\cosh\frac{\alpha}{2} - e^{N\alpha/2}\sinh\frac{\alpha}{2})}{\mathcal{Z}} && (5.33) \\ \lambda_+ &= \frac{e^{\alpha/2}(e^{-N\alpha/2}\cosh\frac{\alpha}{2} - \sinh\frac{\alpha}{2})}{2\mathcal{Z}}\,. && (5.34) \end{aligned}$$

From these expressions, we derived a bound on the nonseparability of ρ_{tf}:

$$e^{N\alpha/2}\tanh\frac{\alpha}{2} > 1\,. \tag{5.35}$$

These results enabled us to generate entanglement maps for the standard Bell-transformed thermal state. The negativity of $\rho_{\text{tf}}^{T_A}$ and the bounds on nonseparable ρ_{tf} were plotted in Figs. 5-5 and 5-6 for the $\{1, N-1\}$ split and in Figs. 5-7 and 5-8 for the $\{N/2, N/2\}$ splits. Both splits gave entanglable thermal states outside of the Braunstein et. al. bound for entanglable ρ_{eff}. The $\{1, N-1\}$ split, however, yielded more entanglable thermal state parameter space than the $\{N/2, N/2\}$ split.

The overall conclusion from the analyses of this chapter was that at least 114,135 qubits were required to entangle a thermal state at ideal experimental conditions ($\alpha = 8.79 \times 10^{-5}$).

Chapter 6

General separability and distillability bounds for Bell-transformed thermal states

This chapter focuses on the analytical derivation of separability and distillability bounds on Bell-transformed thermal states under any bipartite split. The crux of the derivation is to take a state ρ' with known entanglement properties and find a random unitary operation that performs $U_{\text{b}}\rho_{\text{th}}U_{\text{b}}^{\dagger} \mapsto \rho'$. Since such an operation cannot increase the entanglability of the initial state, the entanglement of ρ' gives a lower bound on the entanglement of the Bell-transformed thermal state $U_{\text{b}}\rho_{\text{th}}U_{\text{b}}^{\dagger}$.

A candidate state for ρ' is provided by a family of mixed Bell states ρ_N whose entanglement has been classified by Dür and Cirac [DC00]. We first review their formalism (Section 6.1) and then apply it to Bell-transformed thermal states (Section 6.2). We derive fully separable and distillable bounds on Bell-transformed thermal states by constructing a random unitary operation that implements $U_{\text{b}}\rho_{\text{th}}U_{\text{b}}^{\dagger} \mapsto \rho_N$. This result gives improved bounds on the entanglability of ρ_{th}. Finally, we sketch a majorization based approach that may yield even tighter bounds without having to specify a particular unitary (Section 6.3).

6.1 Dür-Cirac classification of entanglement in special mixed Bell states

Here we review the Dür and Cirac formalism for classifying entanglement in a specific family of mixed Bell states and an application of this classification to effective pure states. The discussion is drawn from their article in Physical Review A [DC00].

6.1.1 Formalism

Dür and Cirac consider a special family of mixed Bell states parameterized by the number of qubits N. They are defined by

$$\rho_N = \sum_{\sigma=\pm} \lambda_0^\sigma \left|\Psi_0^\sigma\right\rangle \left\langle\Psi_0^\sigma\right| + \sum_{j=1}^{2^{(N-1)}-1} \lambda_j \left(\left|\Psi_j^+\right\rangle \left\langle\Psi_j^+\right| + \left|\Psi_j^-\right\rangle \left\langle\Psi_j^-\right|\right) \tag{6.1}$$

where the generalized Bell states (in the computational basis) are given by

$$\left|\Psi_j^\pm\right\rangle = \frac{1}{\sqrt{2}} \left[|0\rangle\,|j\rangle \pm |1\rangle\,|\bar{j}\rangle\right] , \tag{6.2}$$

$|\bar{j}\rangle \equiv \left|(2^{N-1} - j - 1)\right\rangle$ is the bit-flipped version of $|j\rangle$, and $1 \leq j < 2^{N-1} - 1$.

Notice that $\lambda_0^\pm$ and $\lambda_j^\pm$ are simply the eigenvalues of ρ_N. Moreover, since ρ_N is a density matrix, $\lambda_0^\pm \geq 0$, $\lambda_j^\pm \geq 0$, and $\sum_{i=0}^{2^{(N-1)}-1} \left(\lambda_i^+ + \lambda_i^-\right) = 1$. We also choose $\lambda_0^+ - \lambda_0^- \geq 0$. The state of the first qubit and the latter $N-1$ qubits are given by the first ket/bra and second ket/bra of the pair respectively. This formulation is slightly altered from the notation used by Dür and Cirac for later convenience in matching our thermal state formulas.

We specify a bipartite split with the nonnegative integer k, which when expressed in binary, labels the qubits that are in Party A with 1s and the qubits in Party B with 0s. For example, if $k = 010110$, then the second, fourth, and fifth qubits are in Party A, and the first, third, and sixth qubits are in Party B. With no loss in generality, we require the first qubit to always be in Party B, i.e. the most significant bit is always 0. Therefore, the possible bipartite splits lie in the range $1 \leq k \leq 2^{N-1} - 1$.

This notation allows us to obtain an elegant condition on ρ_N to have positive partial transpose (PPT) or negative partial transpose (NPT). To derive it, we perform the partial transpose on ρ_N in the computational basis. We first note that application of Eq. 6.2 yields

$$\left|\Psi_j^+\right\rangle\left\langle\Psi_j^+\right| + \left|\Psi_j^-\right\rangle\left\langle\Psi_j^-\right| = |0\rangle|j\rangle\langle 0|\langle j| + |1\rangle|\bar{j}\rangle\langle 1|\langle\bar{j}| \,, \tag{6.3}$$

which is diagonal in the computational basis and invariant under partial transposition. In view of Eqs. 6.1 and 6.3, the partial transpose of ρ_N under bipartite split k is

$$\begin{aligned}\rho_N^{T_A} &= \frac{\lambda_0^+ + \lambda_0^-}{2}\left[|0\rangle|00...0\rangle\langle 0|\langle 00...0| + |1\rangle|11...1\rangle\langle 1|\langle 11...1|\right] \\ &+ \frac{\lambda_0^+ + \lambda_0^-}{2}\left[|0\rangle|k\rangle\langle 1|\left\langle\bar{k}\right| + |1\rangle\left|\bar{k}\right\rangle\langle 0|\langle k|\right] \\ &+ \sum_{j=1}^{2^{(N-1)}-1}\left[|0\rangle|j\rangle\langle 0|\langle j| + |1\rangle|\bar{j}\rangle\langle 1|\langle\bar{j}|\right] .\end{aligned} \tag{6.4}$$

The density matrix $\rho_N^{T_A}$ is diagonal in the computational basis except in the subspace spanned by $|0\rangle|k\rangle$ and $|1\rangle\left|\bar{k}\right\rangle$:

$$\begin{bmatrix}\lambda_k & \Delta/2 \\ \Delta/2 & \lambda_k\end{bmatrix} \tag{6.5}$$

where $\Delta = \lambda_0^+ - \lambda_0^-$. The eigenvalues of this subspace are $\lambda_k \pm \Delta/2$. Since $\Delta \geq 0$, we conclude that ρ_N has the following properties:

$$\lambda_k \geq \Delta/2 \Rightarrow \rho_N \text{ has positive partial transpose (PPT)} \tag{6.6}$$

$$\lambda_k < \Delta/2 \Rightarrow \rho_N \text{ has negative partial transpose (NPT)} . \tag{6.7}$$

More importantly, we can characterize the full separability and distillability of ρ_N by taking the partial transposition over all possible bipartite splits. Dür and Cirac proved the following criteria [DC00]:

1. Consider all possible bipartite splits of a N qubit system. If and only if each of these splits has PPT, then ρ_N is fully separable, i.e. separable under any

partition of the system.

2. Consider all possible bipartite splits of a N qubit system. If and only if each of these splits has NPT, then ρ_N is fully distillable, i.e. a maximally entangled pair can be distilled from any two particles in the system.

Thus the PPT/NPT conditions on ρ_N give bounds on the separability and distillability of ρ_N:

$$\Delta \leq 2 \min_{\{k\}}[\lambda_k] \Rightarrow \rho_N \text{ fully separable} \tag{6.8}$$

$$\Delta > 2 \max_{\{k\}}[\lambda_k] \Rightarrow \rho_N \text{ fully distillable} \tag{6.9}$$

where the minimum and maximum are taken over all possible bipartite partitions k.

We note that the bound on full distillability is important because it gives the regions in parameter space where *useful* entanglement (maximally mixed pairs) can be obtained.

6.1.2 Bound on the entanglability of effective pure states

As Dür and Cirac point out, the fully distillable criterion gives a simple bound on the entanglability of effective pure states that is tighter than the Braunstein et. al. expression in Eq. 3.4.

Recall from Section 2.3.2 that the effective pure state is given by

$$\rho_{\text{eff}} = \frac{1-\epsilon}{d} I_d + \epsilon \left|0\right\rangle \left\langle 0\right| \tag{6.10}$$

where ϵ characterizes the fraction of pure state and the dimension $d = 2^N$. We can apply a unitary that transforms $|0\rangle$ to the maximally entangled Bell state $\left|\Psi_0^+\right\rangle$, yielding a new state

$$\rho' = \frac{1-\epsilon}{d} I_d + \epsilon \left|\Psi_0^+\right\rangle \left\langle \Psi_0^+\right| \tag{6.11}$$

that fits the form of ρ_N. It is easy to see that $\lambda_0^+ = (1-\epsilon)/d + \epsilon$, $\lambda_0^- = (1-\epsilon)/d$, and $\lambda_j = (1-\epsilon)/d$. Therefore, $\Delta = \epsilon$.

Using Eq. 6.9, we find that full distillability[1] of ρ_{eff} requires

$$\epsilon > \frac{1}{1+2^{N-1}}. \tag{6.12}$$

Furthermore, recall from Section 2.3.2 that ϵ can be expressed in the parameters N and α:

$$\epsilon = \frac{e^{N\alpha/2}}{\mathcal{Z}} - \frac{1-e^{N\alpha/2}/\mathcal{Z}}{2^N-1}. \tag{6.13}$$

Inserting this expression in Eq. 6.12, we find a bound on the entanglability of effective pure states in NMR parameter space:

$$\alpha > -\ln\left[\left(\frac{2+2^N}{3}\right)^{1/N} - 1\right]. \tag{6.14}$$

Unlike the Braunstein et. al. entangled bound of Eq. 3.4, this new bound depends on the number of qubits.

6.2 Application to Bell-transformed thermal states

Here we apply the Dür-Cirac formalism to obtain bounds on the entanglement of Bell-transformed thermal states. We describe a random unitary operation that maps Bell-transformed thermal states to states of form ρ_N, allowing us to derive analytical bounds on the separability and distillability of thermal states under two specific Bell unitary transforms.

6.2.1 Random unitary operation mapping Bell-transformed thermal states to Dür-Cirac Bell states

Since the thermal state ρ_{th} is diagonal, the transformed state has form

$$\rho_{\text{tf}} = U_{\text{b}}\rho_{\text{th}}U_{\text{b}}^{\dagger} \tag{6.15}$$

[1]According to Eq. 6.8, we also obtain a condition for full separability of the transformed effective pure state in Eq. 6.11. However, this criterion is relevant to a specific group of unitaries and therefore does not give as much information as the Braunstein et. al. and Gurvits-Barnum separable bounds.

$$= \sum_{\sigma=\pm} \lambda_0^\sigma \left|\Psi_j^\sigma\right\rangle \left\langle\Psi_j^\sigma\right| + \sum_{j=1}^{2^{(N-1)}-1} \left(\lambda_j^+ \left|\Psi_j^+\right\rangle \left\langle\Psi_j^+\right| + \lambda_j^- \left|\Psi_j^-\right\rangle \left\langle\Psi_j^-\right|\right) .$$

The state is diagonal in the generalized Bell basis $\left\{\left|\Psi_j^\pm\right\rangle\right\}$, but is not of the form ρ_N because generally $\lambda_j^+ \neq \lambda_j^-$.

Not all is lost. Suppose we can find a *random unitary operation* $\mathcal{E}_\mathrm{U}$ such that $\mathcal{E}_\mathrm{U}(\rho_\mathrm{tf})$ has form ρ_N. The action of any random unitary operation on ρ_tf can always be expressed as a probabilistic mixture of unitaries U_i:

$$\mathcal{E}_\mathrm{U}(\rho) = \sum_i p_i U_i \rho U_i^\dagger \tag{6.16}$$

where p_i is the probability of applying the unitary U_i. We can interpret $\mathcal{E}_\mathrm{U}(\rho)$ as a convex combination of transformed thermal states. In Chapter 3, we showed that the effective pure state had the same form and that the entanglement available from ρ_eff could never be as much the maximum entanglement attainable from ρ_th. Analogous reasoning tells us that a random mixture of unitaries *cannot increase* entanglement; the creation of an effective pure state is merely a specific case of this general statement. Thus, the entanglement of $\mathcal{E}_\mathrm{U}(\rho_\mathrm{tf})$ bounds the entanglement of ρ_tf.

Now we present a random unitary operation $\mathcal{E}_\phi$ that performs the desired transformation. Consider the following procedure:

1. Start with the transformed Bell state ρ_tf, which is diagonal in the generalized Bell basis.

2. Apply mixing operation R_ϕ, where $R = \bigotimes_{i=1}^N R_i$ and

$$R_i = \begin{bmatrix} e^{\phi_i} & 0 \\ 0 & 1 \end{bmatrix} \tag{6.17}$$

in the computational subspace $\{|0\rangle, |1\rangle\}$ of the ith qubit. Essentially, R_i multiplies the ith qubit by a random phase ϕ_i if the qubit is in the state $|0\rangle$. Here we assume that ϕ_i is random and uniformly distributed over $[-\pi, \pi]$ subject to the constraint $\sum_i \phi_i = 2\pi$. This requirement is chosen so that $R_\phi \left|\Psi_0^\pm\right\rangle = \left|\Psi_0^\pm\right\rangle$.

3. Average R_ϕ over all possible values of ϕ_i from 0 to 2π.

Let us verify that $\mathcal{E}_\phi$ produces a state of form ρ_N. First, we calculate the effect of R_ϕ on each generalized Bell state:

$$\begin{aligned} \left|\chi_j^\pm\right\rangle &= R_\phi \left|\Psi_j^\pm\right\rangle \\ &= \bigotimes_{i=1}^{N} R_i \left|\Psi_j^\pm\right\rangle \\ &= \frac{1}{\sqrt{2}}\left[e^{i\phi_j}\left|0\right\rangle\left|j\right\rangle \pm e^{-i\phi_j}\left|1\right\rangle\left|\bar{j}\right\rangle\right] \end{aligned} \tag{6.18}$$

where we define

$$\phi_j \equiv \sum_{\{i|q_i=0\}} \phi_i \,. \tag{6.19}$$

Note that $\left|1\right\rangle\left|\bar{j}\right\rangle$ gains a phase of $-\phi_j$ because it is the bit-flipped version of $\left|0\right\rangle\left|j\right\rangle$.

Next, we find the state that results when R_ϕ acts on the entire Bell-transformed thermal state:

$$R_\phi \rho_{\mathrm{tf}} R_\phi^\dagger = \sum_{j=0}^{2^{(N-1)}-1} \left[\lambda_j^+ \left|\chi_j^+\right\rangle\left\langle\chi_j^+\right| + \lambda_j^- \left|\chi_j^-\right\rangle\left\langle\chi_j^-\right|\right] \tag{6.20}$$

with the outer products being

$$\begin{aligned} \left|\chi_j^\pm\right\rangle\left\langle\chi_j^\pm\right| &= \frac{1}{2}\Big[\left(\left|0\right\rangle\left|j\right\rangle\left\langle 0\right|\left\langle j\right| + \left|1\right\rangle\left|\bar{j}\right\rangle\left\langle 1\right|\left\langle\bar{j}\right|\right) \\ &\quad \pm \left(e^{2i\phi_j}\left|0\right\rangle\left|j\right\rangle\left\langle 1\right|\left\langle\bar{j}\right| + e^{-2i\phi_j}\left|1\right\rangle\left|\bar{j}\right\rangle\left\langle 0\right|\left\langle j\right|\right)\Big]\,. \end{aligned} \tag{6.21}$$

When we average Eq. 6.20 over ϕ_i (recall that ϕ_j is a function of ϕ_i from Eq. 6.19), the last two terms ($j \neq 0$) in Eq. 6.21 vanish because $\prod_{i=1}^N \int_0^{2\pi} d\phi_i e^{\pm 2i\phi_j} = 0$. When $j = 0$, $\mathcal{E}_\phi$ has no effect as $\phi_j = 2\pi$. The final state after application of $\mathcal{E}_\phi$ is consequently

$$\mathcal{E}_\phi(\rho_{\mathrm{tf}}) = \sum_{\sigma=\pm} \lambda_0^\sigma \left|\Psi_0^\sigma\right\rangle\left\langle\Psi_0^\sigma\right| + \sum_{j=1}^{2^{(N-1)}-1} \frac{\lambda_j^+ + \lambda_j^-}{2}\left(\left|\Psi_j^+\right\rangle\left\langle\Psi_j^+\right| + \left|\Psi_j^-\right\rangle\left\langle\Psi_j^-\right|\right), \tag{6.22}$$

which is indeed of form ρ_N. Matching this expression with Eq. 6.1, we find that

$$\lambda_j = \frac{\lambda_j^+ + \lambda_j^-}{2} . \tag{6.23}$$

We emphasize that $\mathcal{E}_\phi$ is just one possible random unitary operation that gives the needed transformation. Moreover, random unitary operations generally *decrease* the entanglement of the system. We ideally desire a procedure that preserves as much thermal state entanglement as possible. In the next subsection, we make a few remarks about the effectiveness of $\mathcal{E}_\phi$ in this regard.

6.2.2 Separability and distillability of thermal states under $U_{\mathrm{b,s}}$

We now calculate separability and distillability bounds on ρ_{th}. First, we examine the case where the transformed thermal state is $\rho_{\mathrm{tf}} = U_{\mathrm{b,s}}\rho_{\mathrm{th}}U_{\mathrm{b,s}}^\dagger$.

Inspection of the matrix in Eq. 3.9 shows that $U_{\mathrm{b,s}}$ maps computational basis states to Bell states in the following manner:

$$\begin{aligned} |j\rangle &\mapsto \left|\Psi_j^+\right\rangle, \ 0 \le j < 2^{N-1} - 1 \qquad (6.24) \\ |j\rangle &\mapsto \left|\Psi_{j-2^{(N-1)}}^-\right\rangle, \ 2^{N-1} \le j < 2^N . \end{aligned}$$

Recall from Section 2.3.1 that the diagonal entries of the thermal state are given by

$$\langle i|\, \rho_{\mathrm{th}}\, |i\rangle = \frac{1}{\mathcal{Z}} e^{[N-2w(i)]\alpha/2}, \ 0 \le i \le 2^N - 1 \tag{6.25}$$

where the Hamming weight $w(i)$ is the number of 1s in the binary expression for $|i\rangle$.

Combining this formula with Eq. 6.24, we find

$$\begin{aligned} \left\langle \Psi_i^+ \middle| \rho_{\mathrm{tf}} \middle| \Psi_i^+ \right\rangle &= \frac{e^{[N-2w(i)]\alpha/2}}{\mathcal{Z}}, \ 0 \le i < 2^{N-1} - 1 \qquad (6.26) \\ \left\langle \Psi_i^- \middle| \rho_{\mathrm{tf}} \middle| \Psi_i^- \right\rangle &= \frac{e^{[N-2w(i)-2]\alpha/2}}{\mathcal{Z}}, \ 2^{N-1} \le i < 2^N . \qquad (6.27) \end{aligned}$$

Now we perform the procedure of the last subsection to obtain a new state $\mathcal{E}_\phi(\rho_{\text{tf}})$. Applying Eq. 6.23, $\mathcal{E}_\phi(\rho_{\text{tf}})$ expressed in form of ρ_N has parameters

$$\Delta = \frac{1}{\mathcal{Z}}\, e^{N\alpha/2}(1 - e^{-\alpha}) \tag{6.28}$$

$$\lambda_k = \frac{1}{2\mathcal{Z}}\, e^{[N-2w(k)]\alpha/2}(1 + e^{-\alpha}) \tag{6.29}$$

where $1 \leq k \leq 2^{(N-1)} - 1$.

Inserting the parameters into Eqs. 6.6 and 6.7, we establish conditions for the thermal state to be PPT/NPT under $U_{\text{b,s}}$ and bipartite split k,

$$\begin{aligned} \tanh\frac{\alpha}{2} &\leq e^{-w(k)\alpha} \Rightarrow \rho_{\text{th}} \text{ PPT under } U_{\text{b,s}} \\ \tanh\frac{\alpha}{2} &> e^{-w(k)\alpha} \Rightarrow \rho_{\text{th}} \text{ NPT under } U_{\text{b,s}}\,. \end{aligned} \tag{6.30}$$

Notice that the $\{1, N-1\}$ and $\{N/2, N/2\}$ bipartite splits correspond to $|k\rangle$ being $|011...1\rangle$ and $|00...0 : 11...1\rangle$,[2] which have Hamming weights $N-1$ and $N/2$ respectively. If we insert these weights into the above equations, then we recover the same separability constraints calculated in Eqs. 5.12 and 5.25. Our procedure $\mathcal{E}_\phi$ seems to be a good one; it does not decrease the entanglement of $U_{\text{b,s}}\rho_{\text{th}}U_{\text{b,s}}^\dagger$ for the $\{1, N-1\}$ and $\{N/2, N/2\}$ splits.

It is straightforward to calculate separability and distillability bounds by minimizing and maximizing Eq. 6.30 over all bipartite splits, and we find

$$\tanh\frac{\alpha}{2} \leq e^{-(N-1)\alpha} \Rightarrow \rho_{\text{th}} \text{ fully separable under } U_{\text{b,s}} \tag{6.31}$$

$$\tanh\frac{\alpha}{2} > e^{-\alpha} \Rightarrow \rho_{\text{th}} \text{ fully distillable under } U_{\text{b,s}}\,. \tag{6.32}$$

The separable bound corresponds to the $\{1, N-1\}$ split while the distillable bound corresponds to the $\{N-1, 1\}$ split.

[2]Following the notation established in Chapter 5, the colon in the middle of the ket separates the first $N/2$ qubits from the latter $N/2$ qubits.

6.2.3 Separability and distillability of thermal states under $U_{\mathrm{b,s}}U_{\mathrm{fan}}$

Here we calculate separability and distillability bounds on thermal states under a Bell unitary that is obtained by acting with a permutation matrix U_{fan} before applying $U_{\mathrm{b,s}}$. This new Bell transform $U_{\mathrm{b,s}}U_{\mathrm{fan}}$ gives much improved results over $U_{\mathrm{b,s}}$, as we now show.

The tightness of any bound derived from the Dür-Cirac formalism depends heavily on Δ, which measures the size of the gap between λ_0^+ and λ_0^-. In the case where we transform ρ_{th} with $U_{\mathrm{b,s}}$, Δ scales as 2^{-N}. We give a short derivation here:

$$\begin{aligned} \Delta &= \frac{1}{\mathcal{Z}}\, e^{N\alpha/2}(1-e^{-\alpha}) \\ &= \frac{e^{N\alpha/2}(1-e^{-\alpha})}{(e^{\alpha/2}+e^{-\alpha/2})^N} \\ &= \frac{1-e^{-\alpha}}{(1+e^{-\alpha})^N} \\ &\sim \frac{\alpha}{2^N}\ (\alpha \ll 1)\,, \end{aligned} \tag{6.33}$$

where we have used Eq. 2.60 for the partition function. We observe that Δ is smaller than it could be. By choosing the Bell transform to be $U_{\mathrm{b,s}}$, we have $\lambda_0^+ = e^{N\alpha/2}/\mathcal{Z}$ and $\lambda_0^- = e^{(N-2)\alpha/2}/\mathcal{Z}$. If we alter the unitary mapping between computational basis states and Bell states, we can shuffle the values of $\lambda_0^\pm$ and λ_j to maximize Δ over all permutations of $U_{\mathrm{b,s}}$. The optimization allows us to achieve tighter bounds on the entanglability of thermal states.

The largest possible gap occurs when we select $\lambda_0^\pm = e^{\pm N\alpha/2}/\mathcal{Z}$, or

$$\Delta = \frac{2\sinh(N\alpha/2)}{\mathcal{Z}}\,. \tag{6.34}$$

Following a similar calculation to the one above, we see this choice scales as

$$\Delta \sim N 2^{-N}\,. \tag{6.35}$$

We can realize this gap with a new mapping:

$$\begin{aligned} |j\rangle &\mapsto \left|\Psi_j^+\right\rangle,\ 0 \le j < 2^{N-1}-1 \\ |j\rangle &\mapsto \left|\Psi_{2^N-j-1}^-\right\rangle,\ 2^{N-1} \le j < 2^N . \end{aligned} \tag{6.36}$$

In fact, this mapping is reproduced by the Bell unitary $U_{\text{b,s}}U_{\text{fan}}$, with the permutation matrix U_{fan} reshuffling the old mapping in Eq. 6.24 to give Eq. 6.36.

Now we calculate the PPT condition for the transformed thermal state $\rho_{\text{tf}} = U_{\text{b,s}}U_{\text{fan}}\rho_{\text{th}}U_{\text{fan}}U_{\text{b,s}}$. Using the mapping in Eq. 6.36 and the formula in Eq. 6.25, we have

$$\begin{aligned} \lambda_k &= \frac{1}{2\mathcal{Z}}\left[e^{-[N-2w(k)]\alpha/2} + e^{[N-2w(k)]\alpha/2}\right] \\ &= \frac{1}{\mathcal{Z}}\cosh\left[(N-2w(k))\frac{\alpha}{2}\right] . \end{aligned} \tag{6.37}$$

Comparing Eqs. 6.34 and 6.37, ρ_{th} under the transform $U_{\text{b,s}}U_{\text{fan}}$ and bipartite split k is PPT/NPT if and only if

$$\tanh(k\alpha/2) \le e^{-[N-w(k)]\alpha} \Rightarrow \rho_{\text{th}} \text{ PPT under } U_{\text{fan}}U_{\text{b,s}} \tag{6.38}$$

$$\tanh(k\alpha/2) > e^{-[N-w(k)]\alpha} \Rightarrow \rho_{\text{th}} \text{ NPT under } U_{\text{fan}}U_{\text{b,s}} . \tag{6.39}$$

Minimizing and maximizing the above conditions over all possible bipartite splits, we find that the fully separable bounds are

ρ_{th} fully separable under $U_{\text{b,s}}U_{\text{fan}}$ if and only if

$$\tanh(N\alpha/4) \le e^{N\alpha/2} \text{ or} \tag{6.40}$$

$$\sinh(N\alpha) \le 1 \tag{6.41}$$

and that the fully distillable bounds are

ρ_{th} is fully distillable under $U_{\text{b,s}}U_{\text{fan}}$ if and only if

$$\tanh\left[(N-1)\alpha/2\right] > e^{-\alpha} \text{ or} \tag{6.42}$$
$$\tanh\frac{\alpha}{2} > e^{-(N-1)\alpha}. \tag{6.43}$$

Eqs. 6.40 and 6.41 are derived from the $\{N/2, N/2\}$ split while Eqs. 6.42 and 6.43 correspond to the $\{1, N-1\}$ and $\{N-1, 1\}$ splits respectively. These two splits give the same bound because λ_k in Eq. 6.37 is symmetric under replacement of $w(k)$ by $N - w(k)$.

6.2.4 Discussion

How do the separable and distillable bounds on thermal states under the two Bell unitaries compare? Are the bounds on thermal state entanglability more favorable than the bounds on effective pure states entanglability? We now analyze these questions.

In Figs. 6-1 and 6-2, we plot the Dür-Cirac derived bounds on full separability and distillability of thermal states under $U_{\text{b,s}}$ (Eqs. 6.31 and 6.32) and $U_{\text{b,s}}U_{\text{fan}}$ (Eqs. 6.41 and 6.43) as well as bounds on the entanglability (Dür and Cirac, Eq. 6.12) and nonentanglability (Gurvits and Barnum, Eq. 2.95) of effective pure states.

Earlier in Section 6.2.2, we remarked that the NPT condition on $U_{\text{b,s}}\rho_{\text{th}}U^\dagger_{\text{b,s}}$ matched the negativity formulas we calculated in Chapter 5 for the $\{1, N-1\}$ and $\{N/2, N/2\}$ splits. Similarly, we can check the fully separable and distillable bounds by numerically finding the α where $\lambda_{\text{min}} = 0$. Figs. 6-3 and 6-4 show that the analytical and numerical values match for $N = \{2, 4, 6, 8, 10\}$. The excellent agreement implies that our random unitary procedure preserves the entanglement of Bell-transformed thermal states.

Comparing $U_{\text{b,s}}$ and $U_{\text{b,s}}U_{\text{fan}}$, the latter unitary clearly gives tighter bounds on full separability and distillability. The Dür-Cirac derived fully separable bounds on ρ_{th} lie far from the Gurvits-Barnum bound on the entanglability of ρ_{eff}, simply because the bounds on ρ_{th} apply to specific unitaries whereas the bounds on ρ_{eff} hold for any quantum operation. Interestingly enough, the fully distillable bound on ρ_{th} under

$U_{\rm b,s}$ exactly coincides with the Braunstein bound on the nonentanglability of $\rho_{\rm eff}$ (not shown in the figures; for comparison, see for instance Figs. 5-5 and 5-6), and the fully separable bound for $U_{\rm b,s}$ exactly matches the fully distillable bound for $U_{\rm b,s}U_{\rm fan}$.

We also evaluate the fully distillable bound for $U_{\rm b,s}U_{\rm fan}$ transformed thermal states against the corresponding bound for effective pure states. The $U_{\rm b,s}U_{\rm fan}$ thermal state bound encompasses more parameter space at low N, but the effective pure state bound does better for $N > 9$. Yet we motivated the study of NMR thermal states in Chapter 3 by observing that thermal states can be more easily entangled than effective pure states. The apparent contradiction suggests that $U_{\rm b,s}U_{\rm fan}$ is not the optimal unitary for entangling thermal states.

Thermal states with parameter values to the left of the $U_{\rm b,s}U_{\rm fan}$ separable bound have at least one NPT bipartite split, meaning that at least one maximally entangled pair can be distilled. Consequently, this bound gives an upper limit on the number of qubits needed to entangle a thermal state in ideal experimental conditions ($\alpha = 8.79\times 10^{-5}$). Solving the separable bound in Eq. 6.41, we compute an upper bound of 20,054 qubits for an entanglable thermal states. This result is a substantial improvement over the lower bound of 114,135 qubits we derived at the end of Chapter 5.

6.3 Majorization approach to obtain bounds on entanglable thermal states

We have seen how the Dür-Cirac formalism is a powerful tool for obtaining bounds on the separability and nonseparability of Bell-transformed thermal states. The key is to find a random unitary operation that maps the Bell-transformed thermal state to a state of form ρ_N. In principle, we can always try to construct the needed quantum operation, but this method is time-consuming and wasteful. Knowing such a procedure exists is enough.

Here we discuss a new approach that uses the concept of majorization to address this problem. Uhlmann's theorem [Uhl70, Uhl71, Uhl72, Uhl73] provides a majoriza-

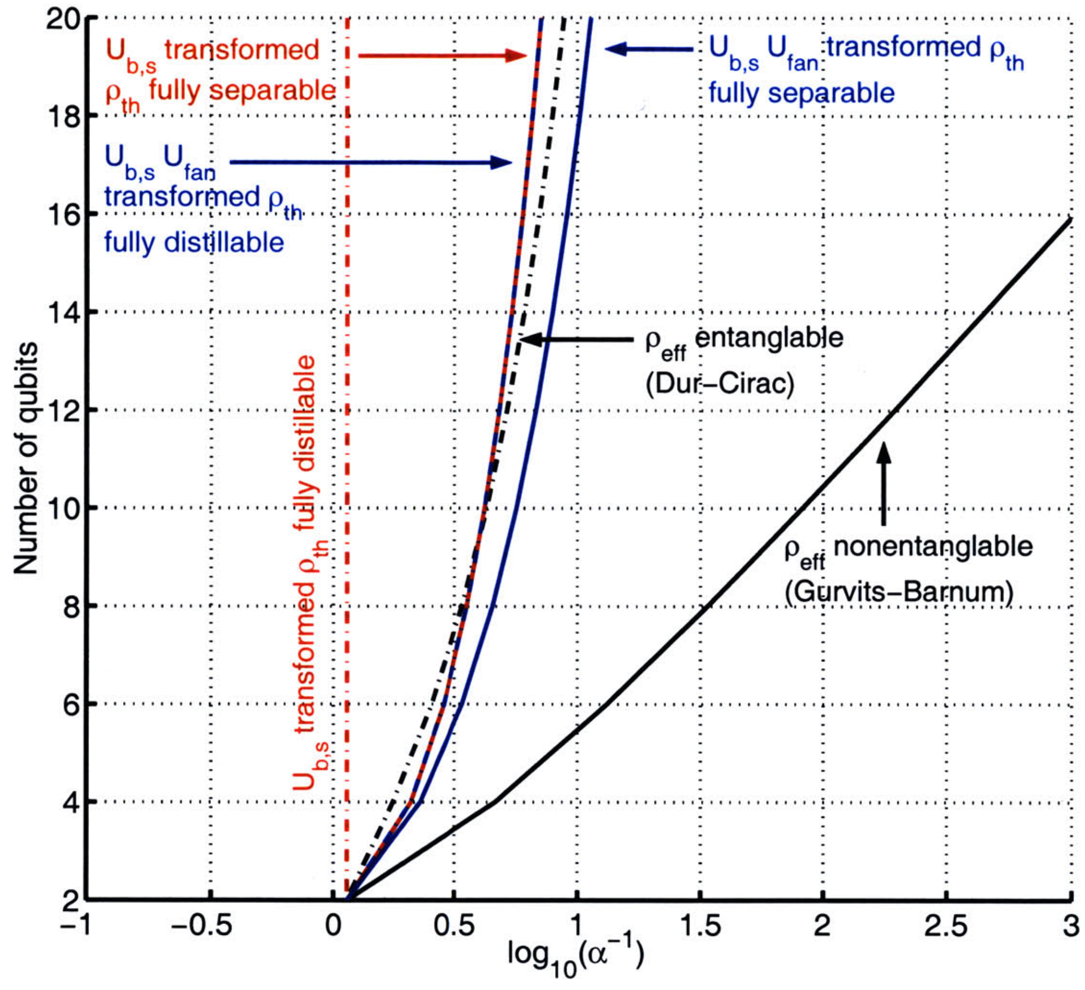

Figure 6-1: Comparison of bounds on the separability and nonseparability of thermal states versus bounds on the entanglability and nonentanglability of effective pure states derived from Dür-Cirac formalism (plotted up to 20 qubits): ρ_{th} fully separable under $U_{\text{b,s}}$ (solid red), ρ_{th} fully separable under $U_{\text{b,s}}U_{\text{fan}}$ (solid blue), ρ_{th} fully distillable under $U_{\text{b,s}}$ (dash dotted red), ρ_{th} fully distillable under $U_{\text{b,s}}U_{\text{fan}}$ (dash dotted blue), ρ_{eff} entanglable (dash dotted black), ρ_{eff} nonentanglable (solid black).

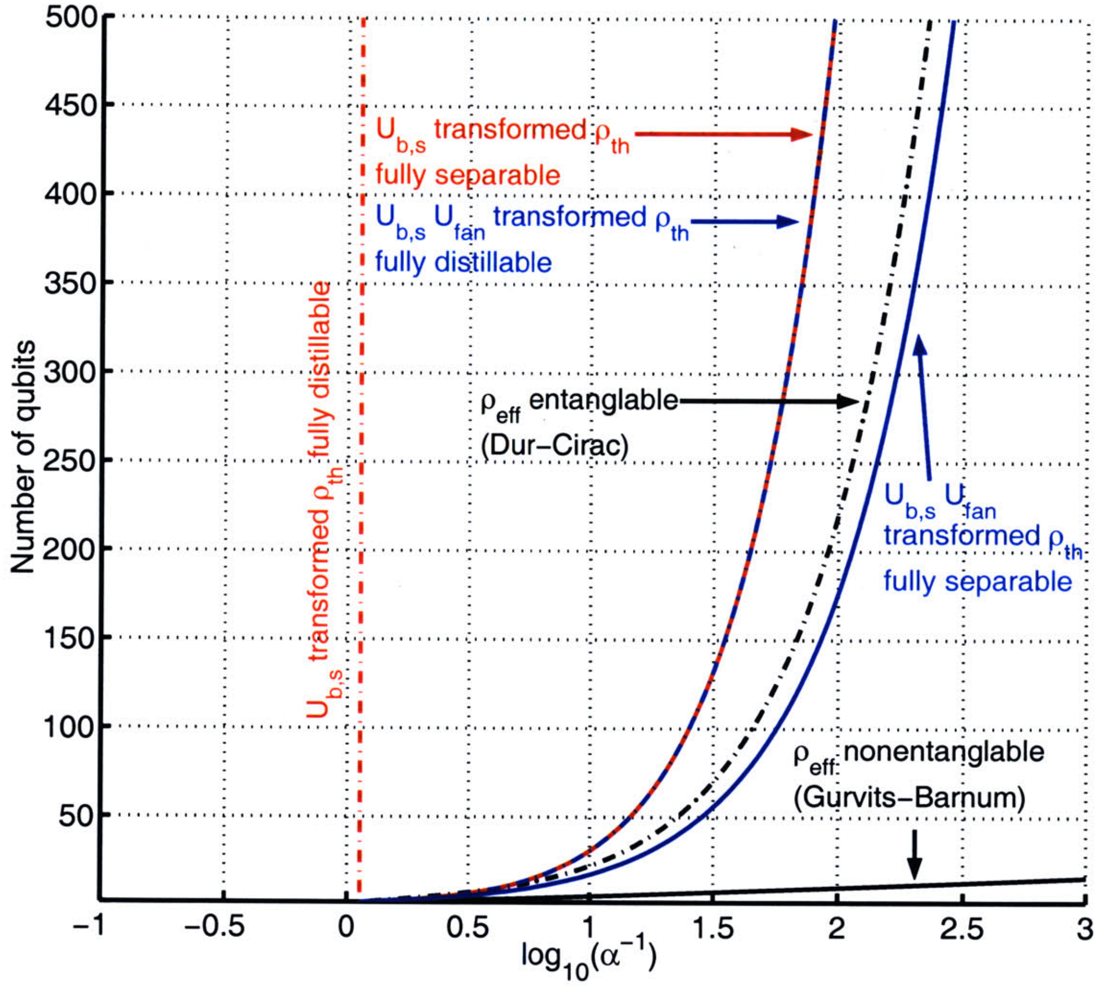

Figure 6-2: Comparison of bounds on the separability and nonseparability of thermal states versus bounds on the entanglability and nonentanglability of effective pure states derived from Dür-Cirac formalism (plotted up to 500 qubits): ρ_{th} fully separable under $U_{\mathrm{b,s}}$ (solid red), ρ_{th} fully separable under $U_{\mathrm{b,s}}U_{\mathrm{fan}}$ (solid blue), ρ_{th} fully distillable under $U_{\mathrm{b,s}}$ (dash dotted red), ρ_{th} fully distillable under $U_{\mathrm{b,s}}U_{\mathrm{fan}}$ (dash dotted blue), ρ_{eff} entanglable (dash dotted black), ρ_{eff} nonentanglable (solid black).

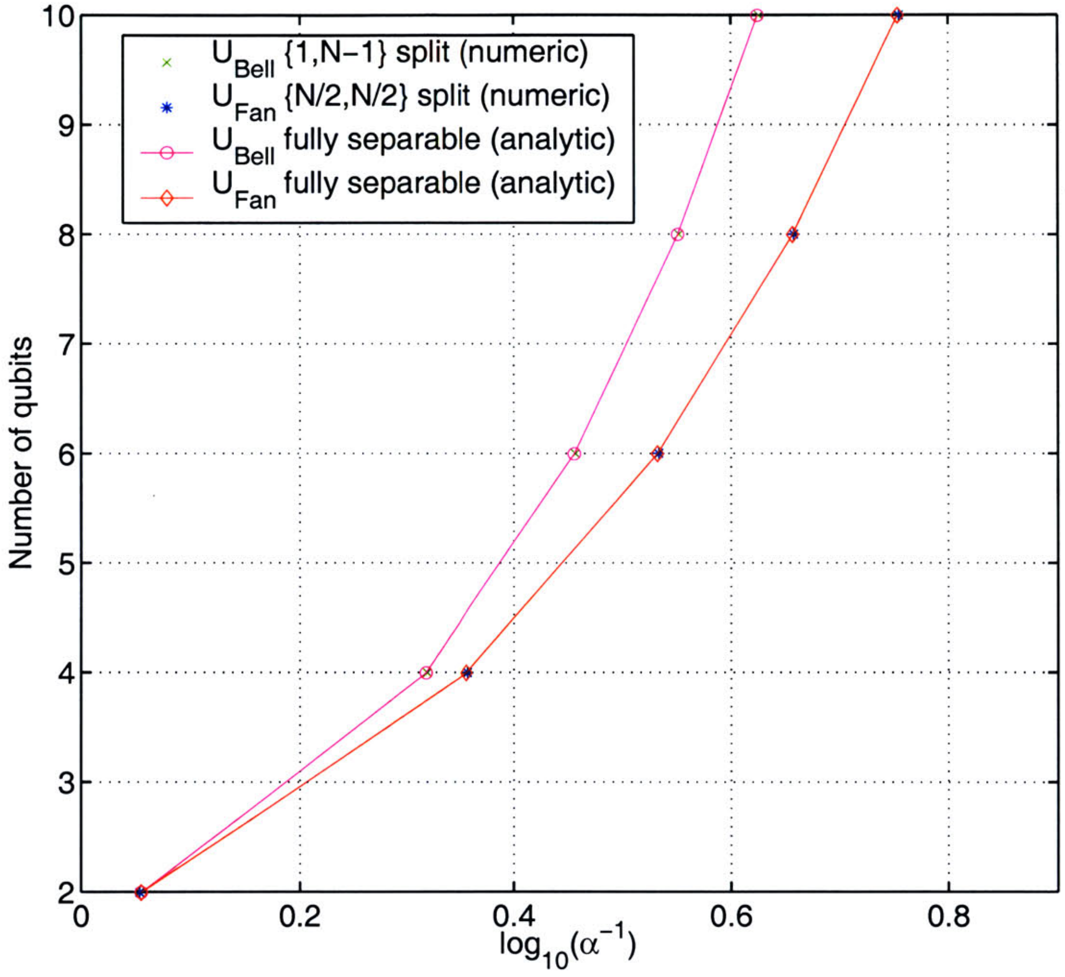

Figure 6-3: Comparison of Dür-Cirac derived fully separable bounds on Bell-transformed thermal states versus direct numerical calculation of negativity. The numeric values are computed by finding α such that $\lambda_{\min} = 0$.

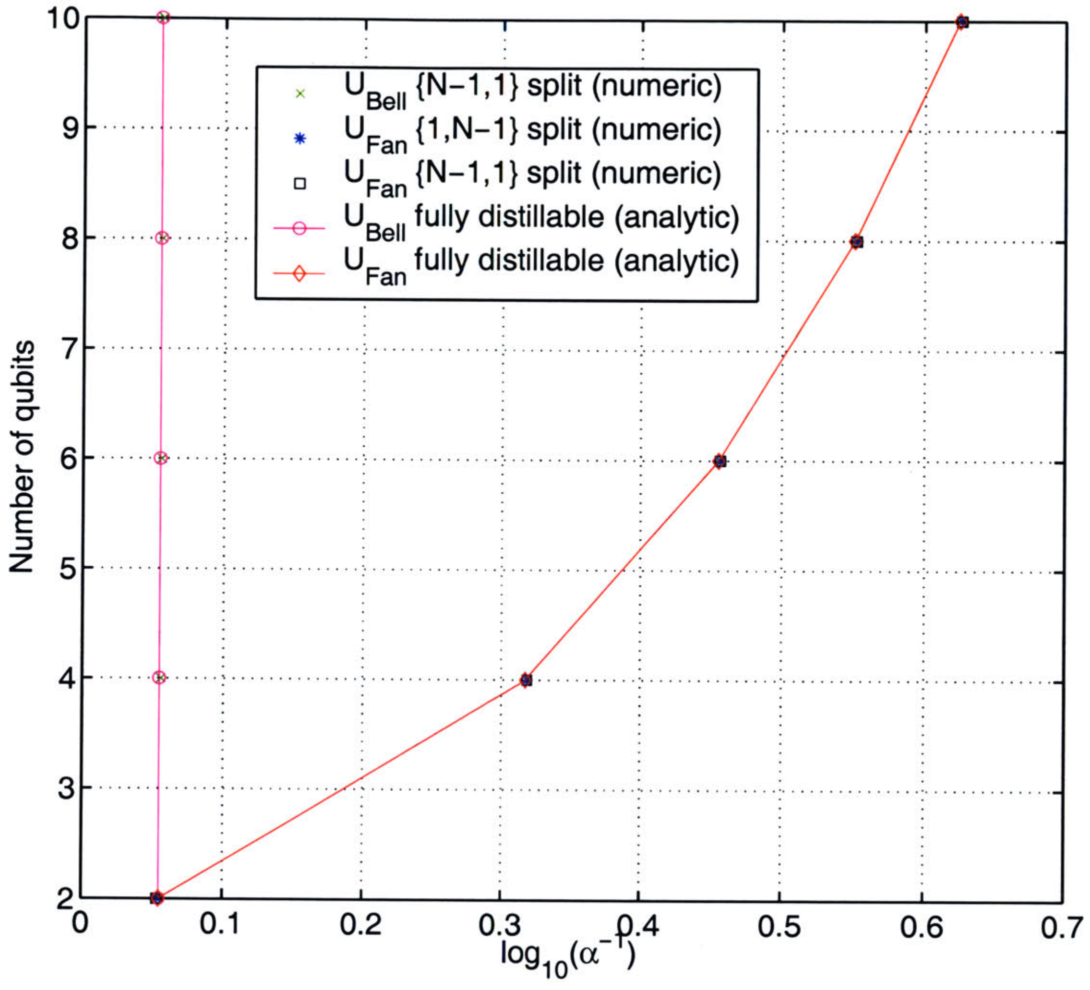

Figure 6-4: Comparison of Dür-Cirac derived fully distillable bounds on Bell-transformed thermal states versus direct numerical calculation of negativity. The numeric values are computed by finding α such that $\lambda_{\min} = 0$.

tion criterion for the existence of a random unitary operation that maps one density matrix to another. The majorization relation between two density matrices also constrains their von Neumann entropies.

In this section, we first define majorization and explain its physical significance. The account closely follows Nielsen's lecture notes [Nie02]. Then we describe Uhlmann's theorem and the majorization condition on von Neumann entropy and illustrate how these tools could *potentially* give tighter bounds on entanglable thermal states. Finally, a numerical approach for deriving the bounds is outlined, although no results are presented.

6.3.1 Majorization

Majorization is a mathematical relation that determines whether one probability distribution is more disordered than another. We first give an intuitive definition of majorization, followed by a definition more amenable to actual calculation.

Suppose we have two real vectors, $\vec{r} = (r_1, r_2, ... r_{d_1})$ and $\vec{s} = (s_1, s_2, ..., s_{d_2})$. If $d_1 \neq d_2$, we pad the shorter vector with extra zeros so that the dimension of both vectors is $d = \max(d_1, d_2)$. We say "*$\vec{s}$ majorizes $\vec{r}$*" if there exists a set of permutation matrices P_j and probability distribution p_j such that

$$\vec{r} = \sum_j p_j P_j \vec{s}. \tag{6.44}$$

The statement "$\vec{s}$ majorizes $\vec{r}$" is equivalent to the mathematical shorthand $\vec{r} \prec \vec{s}$. We may think of $\vec{r}$ as being more "disordered" than $\vec{s}$ because it can be expressed as a random sum of permuted copies of $\vec{s}$. An easy way to understand the concept of majorization is to consider the completely random distribution $(1/d, 1/d, 1/d, ..., 1/d)$ where d is the dimension. Any vector will majorize this distribution. Simply take $p_j = 1/d$ and let P_j be cyclic permutation performed j times. This result agrees with the intuition that a perfectly uniform distribution contains the maximum amount of disorder.

While the definition in Eq. 6.44 furnishes us with a clear physical understanding of majorization, a more practical definition is the following. Let $\vec{r}^{\uparrow}$ and $\vec{s}^{\uparrow}$ be the vectors $\vec{r}$ and $\vec{s}$ with their components (zero-padded if necessary) arranged in non-decreasing order, that is $r_1^{\uparrow} \leq r_2^{\uparrow} \leq ... \leq r_d^{\uparrow}$ and vice versa. We have $\vec{r} \prec \vec{s}$ if and only if

$$\sum_{j=1}^{k} r_j^{\uparrow} \geq \sum_{j=1}^{k} s_j^{\uparrow} \tag{6.45}$$

for $k = 1, 2, ..., d$ and with the equality holding for $k = d$.

When majorization is applied to quantum mechanics, we take $\vec{r}$ and $\vec{s}$ to be vectors containing the eigenvalues of density matrices. Given two density matrices ρ and σ, we have $\rho \prec \sigma$ if and only if $\vec{\lambda}(\rho) \prec \vec{\lambda}(\sigma)$ where $\vec{\lambda}(\rho)$ is a vector containing the eigenvalues of ρ.

This interpretation makes sense because the eigenvalues define a probability distribution on an orthogonal set of states. Thus, majorization amounts to comparing the disorder of two probability distributions. The concept is related to von Neumann entropy in the sense that $\rho \prec \sigma \Rightarrow S(\rho) \leq S(\sigma)$ [Nie02].

6.3.2 Uhlmann's theorem and majorization constraint on von Neumann entropy

Majorization has many beautiful applications to quantum measurement, quantum operations, and density matrix decomposition. We now discuss two fundamental results that will be useful for our purposes: Uhlmann's theorem and the majorization constraint on von Neumann entropy.

Uhlmann's theorem states that given two Hermitian matrices A and B, there exists a random unitary operation $\mathcal{E}_{\mathrm{U}}$ such that $A = \mathcal{E}_{\mathrm{U}}(B)$ if and only if B majorizes A. Mathematically, we write

$$A \prec B \Leftrightarrow \exists\ \mathcal{E}_{\mathrm{U}} \text{ s.t. } A = \mathcal{E}_{\mathrm{U}}(B) \tag{6.46}$$

where the action of $\mathcal{E}_{\mathrm{U}}$ is given in Eq. 6.16.

Suppose ρ_{th} majorizes ρ', a state of form ρ_N. Then Uhlmann's theorem says that

$\rho' = \sum_i U_i \rho_{\text{th}} U_i^\dagger$. The same convexity argument from Section 6.2.1 establishes that the entanglement of ρ' gives a lower bound on the maximum entanglement attainable from thermal states.

We have converted the problem of calculating the entanglement of $U\rho_{\text{th}}(N,\alpha)U^\dagger$ to finding a density matrix ρ' that majorizes $\rho_{\text{th}}(N,\alpha)$. If ρ' has an analytically tractable formulation (in terms of N and α), we can derive analytical bounds on entanglable thermal states.

Although we have changed our approach, the majorization approach is not necessarily simpler. We need to guess a form for ρ' that will majorize ρ_{th} and compare as many as 2^N eigenvalues to see if the majorization actually holds. There is, however, a constraint on the majorization that can be much easier to check. A well-known result[3] says that given two density matrices ρ and σ, if σ majorizes ρ, the von Neumann entropy $S(\sigma)$ must be at least as much as the von Neumann entropy $S(\rho)$:

$$\rho \prec \sigma \Rightarrow S(\rho) \geq S(\sigma) \,. \tag{6.47}$$

Notice the majorization relation is merely necessary and not sufficient.

We can calculate the von Neumann entropy of ρ and σ and check if the entropies satisfy the correct relation for majorization in a particular direction. For example, if $S(\rho) < S(\sigma)$, then it is impossible for σ to majorize ρ. Since there is often an analytical formula for the von Neumann entropy, we can sometimes avoid the time-consuming task of checking that the desired majorization holds.

6.3.3 Application to thermal states

Let us apply these two majorization theorems to the problem of entangling NMR thermal states. First, we use the entropy relation in Eq. 6.47 to establish a condition specifying when ρ' does not majorize ρ_{th}. The condition yields a lower bound on the number of qubits N needed to entangle a thermal state under ideal experimental conditions ($\alpha = 8.79 \times 10^{-5}$). Second, we outline an algorithm that uses Uhlmann's

[3]For a proof, see Ref. [Nie02].

theorem to iteratively search for a better bound by incrementing N until the majorization relation $\rho_{\text{th}} \prec \rho'$ holds. The entropy-based bound sets the initial value of N for the algorithm.

Entropy constraint on thermal state majorization

Here we derive an entropy constraint on $\rho' \prec \rho_{\text{th}}$. The following analysis assumes fixed N and α, although this assumption is not always explicitly stated. Now the majorization always fails when $S(\rho') < S(\rho_{\text{th}})$. Thus we must maximize $S(\rho')$ to obtain a tight constraint on the majorization. Intuitively, $S(\rho')$ is maximized when the eigenvalues of ρ' are distributed as equally as possible. At the same time, ρ' must be nonseparable. Otherwise, its entanglement is zero, and we do not gain any information on thermal state entanglement.

Consider the following construction for ρ', which we call ρ_S. We set $\lambda_0^- = 0$ for convenience, letting $\Delta = \lambda_0^+$. For ρ_S to be nonseparable, at least one bipartite split must be NPT. We can choose any split k such that $\lambda_k = \Delta/2 = \lambda_0^+/2$. To maximize the entropy, we evenly distribute the remainder of the trace over the other $2^N - 4$ eigenvalues. Our construction becomes

$$\begin{aligned}
\lambda_0^- &= 0 && (6.48)\\
\lambda_k &= \lambda_0^+/2,\ k \neq 0 && (6.49)\\
\lambda_j &= \frac{1-2\lambda_0^+}{2^N-4},\ 1 \leq j \leq 2^{(N-1)} - 1,\ j \neq k && (6.50)
\end{aligned}$$

with λ_0^+ yet to be determined. This construction has entropy

$$\begin{aligned}
S(\rho_S) &= -\lambda_0^+ \log_2 \lambda_0^+ - 2\frac{\lambda_0^+}{2}\log_2\frac{\lambda_0^+}{2} - (2^N-4)\frac{1-2\lambda_0^+}{2^N-4}\log_2\frac{1-2\lambda_0^+}{2^N-4} && (6.51)\\
&= -2\lambda_0^+ \log_2 \lambda_0^+ + \lambda_0^+ - (1-2\lambda_0^+)\log_2\frac{1-2\lambda_0^+}{2^N-4}\,.
\end{aligned}$$

To maximize $S(\rho_S)$ with respect to λ_0^+, we compute

$$\frac{\partial S}{\partial \lambda_0^+} = -2\log_2 \lambda_0^+ - \frac{2\lambda_0^+}{\lambda_0^+ \ln 2} + 1 + 2\log_2\frac{1-2\lambda_0^+}{2^N-4} \qquad (6.52)$$

$$\begin{aligned} &= \log_2 2 - 2\log_2 \lambda_0^+ + 2\log_2 \frac{1-2\lambda_0^+}{2^N-4} \\ &= \log_2 \left[\frac{2}{(\lambda_0^+)^2} \left(\frac{1-2\lambda_0^+}{2^N-4} \right)^2 \right]. \end{aligned}$$

Setting the above expression to zero (and checking that the second derivative is negative at this point), we get

$$\lambda_0^+ = \frac{\sqrt{2}}{2^N - 4 + 2\sqrt{2}}. \tag{6.53}$$

Now we can verify our assumption that only split k is NPT:

$$\begin{aligned} \lambda_j &< \lambda_k \\ \frac{1-2\lambda_0^+}{2^N-4} &< \lambda_0^+/2 \\ \lambda_0^+ &> \frac{1}{2^{N-1}} \end{aligned} \tag{6.54}$$

where we have used Eqs. 6.49 and 6.50 in the second line. Substituting Eq. 6.53 into the last expression above, we see that $\lambda_j < \lambda_k$ holds for all positive N.

Examination of Eq. 6.51 shows that for low N, $S(\rho_S) < S(\rho_{\text{th}})$. Furthermore, the deviation of $S(\rho_S)$ from the maximum entropy N decreases exponentially. In contrast, at fixed α, the thermal state entropy deviation from maximum entropy scales linearly with N (see Eq. 2.62). Therefore, at some value of N, the entropy of ρ_S will exceed that of the thermal state. Applying Eq. 6.51 and 6.53, numerical calculation shows that at $\alpha = 8.79 \times 10^{-5}$, $S(\rho_S) \geq S(\rho_{\text{th}})$ when $N > 26$.

The state ρ_S is not the best construction. In fact, if we let λ_0^- be a free parameter and numerically maximize $S(\rho')$ with respect to both λ_0^+ and λ_0^-, we obtain slightly higher entropies that give a lower bound: $N > 25$. However, ρ_S may be useful in the future since it is analytically tractable.

Algorithm for upper bound on N required to entangle a thermal state in ideal experimental conditions

Here we present an algorithm for finding a lower bound on the number of qubits necessary to entangle an NMR thermal state. Assume that the polarization is fixed at $\alpha = 8.79 \times 10^{-5}$ and that we possess an analytical formulation for ρ'.

We begin the algorithm initially at $N = 25$, according to the entropy bound we calculated previously. Then for each value of N, we calculate and compare the entropies of ρ' and ρ_{th} to see if the possibility exists for ρ_{th} to majorize ρ'. If it does, the next step is to check if the majorization relation actually does hold. When the majorization succeeds, Uhlmann's theorem applies and we are finished. Otherwise, we increment N and try again. Here is a detailed description of the algorithm:

1. Set the initial number of qubits to $N = 25$, according to the entropy bound calculated previously.

2. Calculate the thermal state von Neumann entropy $S(\rho_{\text{th}})$.

3. Calculate the von Neumann entropy $S(\rho')$. If ρ' depends on parameters other than N, maximize $S(\rho')$ with respect to the extra parameters.

4. If $S(\rho') \geq S(\rho_{\text{th}})$, check if the desired majorization relation holds: $\rho' \prec \rho_{\text{th}}$.

5. If the majorization relation does not hold, increment the number of qubits by one ($N \mapsto N + 1$) and go back to Step 2. Otherwise, stop.

The missing link is a construction for ρ' that is efficiently majorized by ρ_{th}. It must be different from ρ_S because in our previous formulation, $\lambda_0^- = 0$ and therefore ρ_S can never majorize another density matrix. We leave the construction of ρ' for future work.

6.4 Summary

In this chapter, we used the separability conditions on Dür's and Cirac's ρ_N Bell states to bound the entanglement of thermal states transformed under two Bell unitaries.

For $U = U_{\text{b,s}}$, we found that $U\rho_{\text{th}}U^\dagger$ was fully separable if and only if

$$\tanh\frac{\alpha}{2} \leq e^{-(N-1)\alpha} \tag{6.55}$$

and that $U\rho_{\text{th}}U^\dagger$ was fully distillable if and only if

$$\tanh\frac{\alpha}{2} > e^{-\alpha}. \tag{6.56}$$

For $U = U_{\text{b,s}}U_{\text{fan}}$, we found that $U\rho_{\text{th}}U^\dagger$ was fully separable if and only if

$$\begin{aligned} \tanh(N\alpha/4) &\leq e^{N\alpha/2} \text{ or} \\ \sinh(N\alpha) &\leq 1 \end{aligned} \tag{6.57}$$

and that $U\rho_{\text{th}}U^\dagger$ was fully distillable if and only if

$$\begin{aligned} \tanh\left[(N-1)\alpha/2\right] &> e^{-\alpha} \text{ or} \\ \tanh\frac{\alpha}{2} &> e^{-(N-1)\alpha}. \end{aligned} \tag{6.58}$$

These separable and distillable bounds were plotted in NMR parameter space as depicted in Figs. 6-1 and 6-2. The unitary $U_{\text{b,s}}U_{\text{fan}}$ gave tighter bounds on thermal state entanglability than $U_{\text{b,s}}$. However, the bounds on effective pure state entanglability gave a larger volume of entangled parameter space for $N > 9$, suggesting that Bell unitaries may not be the optimal entangling unitaries for thermal states.

Finally, we discussed an alternative, majorization-based approach to finding bounds on the entanglability of thermal states. It had two attractive features. First, the approach was analytically tractable unlike the numerical computation of negativity. Second, it was more general than our application of the Dür-Cirac formalism, which was limited to Bell transforms. Unfortunately, our approach lacked a construction for a density matrix of form ρ_N that is majorized by a given thermal state. Still, future attempts to bound the entanglability of NMR thermal states may fruitfully follow this approach.

Chapter 7

Conclusion

This thesis was inspired by the controversy surrounding entanglement in liquid-state NMR quantum computation. Braunstein et. al. showed that the states used in current NMR quantum computing experiments are not entangled. Yet it is widely believed that entangled states are needed to perform quantum computations. Based on this evidence, some have claimed that NMR machines do not perform true quantum computation. Others have said that the role of entanglement in quantum computation is poorly understood.

We sought to address the controversy by studying the entanglability of NMR thermal states in a parameter space defined by the number of qubits N and the polarization α. Our goal was to bound the areas in N-α space where thermal states could be entangled. If these regions were also experimentally accessible, it might be possible to create an entangled NMR state in the laboratory, beginning with an initial thermal state. This idea was novel because conventional implementations of quantum algorithms in NMR begin with effective pure states. We became interested in thermal states because more entanglement may be attainable from a thermal state than from a near-maximally mixed state.

To simplify our problem, we investigated the entanglement of thermal states that had been transformed by a family of unitaries U_{b} that mapped computational basis states to Bell states. First, we pursued a numerical approach. A negativity map was computed on a single processor, for the standard Bell-transformed thermal state

$U_{\mathrm{b,s}}\rho_{\mathrm{th}}U_{\mathrm{b,s}}^{\dagger}$. The map gave a bound on the entanglability of thermal states, but memory requirements restricted $N \leq 12$. Due to the strong diverging behavior of the contours near the bound, extrapolation of the bound to higher N was infeasible.

To map larger regions of parameter space, we began adapting the single processor negativity software to a Beowulf cluster. This work, however, was put aside when we discovered alternative analytical approaches that exploited the symmetry of Bell-transformed thermal states. We found explicit negativity formulas for the standard Bell-transformed state ($\rho_{\mathrm{tf}} = U_{\mathrm{b,s}}\rho_{\mathrm{th}}U_{\mathrm{b,s}}^{\dagger}$) under the $\{1, N-1\}$ and $\{N/2, N/2\}$ bipartite splits. Derivation of the $\{1, N-1\}$ split negativity formula relied on the invariance of the latter $N-1$ qubits under the $U_{\mathrm{b,s}}$ and partial transpose operations. A simple pattern in the eigenvectors of the partial transposed state $\rho_{\mathrm{tf}}^{T_A}$ provided the crucial step to deriving the $\{N/2, N/2\}$ bipartite split negativity formula. A calculation based on these negativity formulas showed that at least 114,135 qubits were required to entangle a thermal state in ideal experimental conditions.

After our initial breakthrough, we found a more general method that could be used to analytically bound the nonseparability of Bell-transformed thermal states for any U_{b} and any bipartite split. We constructed a random unitary operation $\mathcal{E}_{\mathrm{U}}$ that could convert a given Bell-transformed thermal state into a Dür-Cirac mixed Bell state ρ_N, whose entanglement could easily be determined. This operation allowed us to bound the entanglement of $U_{\mathrm{b}}\rho_{\mathrm{th}}U_{\mathrm{b}}^{\dagger}$ with a negative partial transpose condition on ρ_N. With these tools, we derived conditions on the full separability and distillability of thermal states transformed by the Bell unitaries $U_{\mathrm{b,s}}$ and $U_{\mathrm{b,s}}U_{\mathrm{fan}}$. The distillability condition provided improved bounds on the entanglability of thermal states, which, for $N \leq 9$, captured more entanglable parameter space than the corresponding bound on near-maximally mixed states. From these results, we found a new upper limit of 20,054 qubits required to entangle a thermal state in ideal experimental conditions.

The success of this method suggested a promising new direction. For a given transformed thermal state and Dür-Cirac state, Uhlmann's theorem gave a majorization criterion that the pair had to satisfy for $\mathcal{E}_{\mathrm{U}}$ to exist. If the correct majorization relation was satisfied, we could apply the same formalism as before to bound the entanglement of the transformed thermal state. Now the burden of the analysis was shifted from the difficult task of constructing the required random unitary operation to the somewhat easier task of constructing a state of form ρ_N that could be majorized by $\rho_{\text{th}}(N, \alpha)$. Unfortunately, we have not yet developed a systematic way to construct the desired ρ_N.

Although our results thus far do not motivate an NMR experiment, we have developed a numerical and analytical toolkit that will facilitate further study of thermal state entanglability. First, we created a software package for calculating the negativity of transformed thermal states. The software may be easily extended to search for unitaries that entangle the maximum fraction of parameter space. On a single processor, the search can be implemented by using MATLAB numerical optimization functions to minimize λ_{min} with respect to a set of parameters that specify a unitary transformation. Second, we began building a software library to implement negativity calculations on a Beowulf cluster. Although that work is unfinished, we have since then acquired a more developed parallel linear algebra package, which we plan to use for future cluster calculations. This software, Matlab *P v2.0, is being developed by the Applied Computing Group at the Massachusetts Institute of Technology [Cho02] and allows easy integration of user-written packages into the main library. In addition, we may explore optimization for sparse matrices, as prototype applications for these special matrices already exist [Li03]. Finally, we found a powerful and generically applicable analytical method for deriving bounds on thermal state entanglability. The combination of majorization techniques with this approach may lead to tighter bounds yet.

More importantly, we have completed the first study of NMR entanglement that goes beyond near-maximally mixed states. Our work, while preliminary, demonstrates that much fertile ground remains to be explored in the field of mixed state entanglement.

Appendix A

Single processor negativity map source code

This appendix gives MATLAB source code for generating entanglement maps as described in Chapter 4.

The execution function is `runexpt.m`. It takes a function handle for the transforming unitary, vectors specifying a square block of parameter space in α and N, a vector specifying the partition (which must match the N data points), and an output file name and then saves a QCTM (Quantum Computing Treasure Map) structure that contains the calculated minimum eigenvalue of the partial transposed state at every point in the specified parameter plus some useful information. The QCTM structure can be passed to a plotting function like `ploteigmap.m` or `plotneg.m` to output a minimum eigenvalue or negativity map. To load a saved QCTM file, use the command '`load filename.mat`'.

Here is an example for generating a negativity map with the unitary $U_{\mathrm{b,s}}$ under the $\{N/2, N/2\}$ bipartite split.

```
alpha = logspace(−3,1,100); % alpha ranges from 0.001 to 10
nq = (2:2:10); % number of qubits = {2,4,6,8,10}
partHalf = nq./2; % Use partition {N/2,N/2}
option = 0; % Set option to just plot negativity and no contours
qctmvec = runexpt(@ubell, partHalf, alpha, nq, 'bellhalf');
```

```
plotneg(qctmvec, option);
```

A.1 Code for calculating negativity data

```
% File:     calcmineig.m
% Usage:    qctminfo = calcmineig(@genunit, part, alpha, nq)
% Date:     30-Sept-02
% Author:   Terri Yu <teryu@mit.edu>
% Function: Calculates minimum eigenvalue of partial transposed
%           thermal state for a specified number of qubits and a
%           range in alpha.
% Notes:    alpha is a dimensionless quantity equal to h\nu/kT.
% Input:    @genunit - function handle pointing to unitary generator
%           part - number of qubits in first half of bipartite split
%           alpha - a vector of points specifying the range in alpha
%           nq - number of qubits (not a vector)
% Returns:  qctminfo - structure of information for writeqctm.m
%           qctminfo.alpha - alpha vector
%           qctminfo.mineig - minimum eigenvalues for each alpha
%           qctminfo.unitary - a string describing the unitary
%                              that was used to transform thermal state
%           qctminfo.nqubits - number of qubits
%           qctm.partition - number of qubits in first half of
%                            bipartite split
%           qctminfo.funcdata - memory usage and runtimes for
%                               individual functions
%           qctminfo.runtime - total runtime
```

```
function qctminfo = calcmineig(genunit, part, alpha, nq)

global U;

alen = length(alpha);
mineig = zeros(alen,1);
% Save names of functions into structure
funcdata(1).name = 'thermalden';
funcdata(2).name = 'transform';
funcdata(3).name = 'ptranspose';
funcdata(4).name = 'mineig';
funcdata(5).name = 'genunit';
% Start timer
tic;
% Generate unitary transform for nqubits
[U, Uname, funcdata(5).mem, funcdata(5).rt] = feval(genunit,nq);
% Calculate minimum eigenvalues
for i=1:alen
  [rho, funcdata(1).mem(i), funcdata(1).rt(i)] = thermalden(alpha(i), nq);
  [rho, funcdata(2).mem(i), funcdata(2).rt(i)] = transform(rho);
  [rho, funcdata(3).mem(i), funcdata(3).rt(i)] = ptranspose(rho, part);
  [mineig(i), funcdata(4).mem(i), funcdata(4).rt(i)] = mineig(rho);
end
for i=1:4
  funcdata(i).mem = funcdata(i).mem';
  funcdata(i).rt = funcdata(i).rt';
end
% Stop timer
runtime = toc;
% Make structure to feed into writeqctm.m
```

```
[rows cols] = size(alpha);
% Make sure alpha is saved as a column vector
if (cols == 1)
  qctminfo.alpha = alpha;
else
  qctminfo.alpha = alpha';
end
qctminfo.mineig = mineig;
qctminfo.unitary = Uname;
qctminfo.nqubits = nq;
qctminfo.partition = part;
qctminfo.funcdata = funcdata;
qctminfo.runtime = runtime;
% Clear memory of large matrices
clear rho U ID X Z H;


% File:      dt.m
% Usage:     dt()
% Date:      ??
% Author:    Isaac Chuang <ichuang@media.mit.edu>
% Function: Returns date using UNIX command
% Input:     Nothing
% Returns:   dt - date in format: <day><month><year>-time

function dt = dt()

[s,x] = unix('date +"%d%b%y-%H%M%S"');
dt = x(1:end-1);


% File:      transform.m
% Usage:    [rhoent, mem, rt] = transform(rho)
% Date:     30-Sept-02
% Author:   Terri Yu <teryu@mit.edu>
% Function: Transforms given density matrix using globally
%           defined unitary U
% Input:    rho - density matrix (assume it's square)
% Returns:  rhotf - transformed state, U*rho*U'
%           mem - memory used in bytes
%           rt - runtime in seconds

function [rhotf, mem, rt] = transform(rho)

global U

% Start timing
tic;
% Perform unitary transformation on rho
rhotf = U * rho * U';
% Count memory usage
s = whos;
mem = totmem(s);
% End timer
rt = toc;
% Clear variables
clear rho starttime s endtime;

% File:     estzalpha.m
% Usage:    zalpha = estzalpha(qctmvec)
% Date:     01-Jun-03
% Author:   Terri Yu <teryu@mit.edu>
% Function: Estimates the alpha where the minimum eigenvalue goes to zero
% Input:    qctmvec - QCTM structure
% Returns:  zalpha - vector of alpha where the minimum eigenvalue goes
%                  to zero, corresponding to each QCTM structure

function zalpha = estzalpha(qctmvec)

qlen = length(qctmvec);
for i=1:qlen
  % Find alpha data point corresponding to minimum eigenvalue
  % closest to zero
  [mineig index] = min(abs(qctmvec(i).mineig));
  % Force alpha and mineig to both be column vectors
  [rows cols] = size(qctmvec(i).alpha);
  if (cols == 1)
    x = qctmvec(i).alpha(index-1:index+1);
  else
    x = qctmvec(i).alpha(index-1:index+1)';
  end
  [rows cols] = size(qctmvec(i).mineig);
  if (cols == 1)
    y = qctmvec(i).mineig(index-1:index+1);
  else
    y = qctmvec(i).mineig(index-1:index+1)';
  end
  % Fit line to alpha data point and its two closest neighbors
  p = polyfit(x,y,1);
  % Find zero of line
  zalpha(i) = roots(p);
end
```

```
% File:     hostname.m
% Usage:     hostname()
% Date:      01-Oct-02
% Author:    Terri Yu <teryu@mit.edu>
% Function: Returns hostname of machine using UNIX command
% Input:     Nothing
% Returns:   Hostname string

function hostname = hostname()

[s, x] = unix('hostname');
hostname = x;

% File:     mineig.m
% Usage:     lambda = mineig(rho)
% Date:      17-Sept-02
% Author:    Terri Yu <teryu@mit.edu>
% Function: Uses MATLAB eig function to find the minimum eigenvalue
%            of a matrix, assuming that the matrix is square and real
% Input:     mtx - a square matrix
% Returns:   mineig - minimum eigenvalue of mtx
%            mem - memory usage of function
%            rt - runtime of function

function [mineig, mem, rt] = mineig(mtx)

% Start timing
tic;
% Returns eigenvalue with smallest real part
```

```
mineig = min(eig(mtx));
% Count memory usage
s = whos;
mem = totmem(s);
% End timer
rt = toc;
% Clear variables
clear starttime endtime s;

% File:      ptHalf.m
% Usage:     [mpt, mem, rt] = ptHalf(mat)
% Date:      22-Oct-02
% Author:    Isaac Chuang <ichuang@media.mit.edu>
%            Terri Yu <teryu@mit.edu>
% Function: Compute the partial transpose of a n-qubit
%            matrix density matrix using the {N/2,N/2} split.
% Notes:     Assumes that the input matrix is square and even
%            (which is the case if we truly have an n-qubit
%            density matrix.
% Input:     mat - input (square) matrix
% Returns:   mpt  - output partial transpose matrix
%            mem - memory used in bytes
%            rt - runtime in seconds

function [mpt, mem, rt] = ptHalf(mat)

% Start time
tic;
mpt = ptHalfC(mat);
% Count memory use
```

```
s = whos;
mem = totmem(s);
% End timer
rt = toc;
% Clear variables
clear starttime endttime s;



/*
  File:   ptHalf.c
  Date:   14-Oct-02
  Author: I. Chuang <ike@media.mit.edu>
  Partial transpose function written in C as a MEX (matlab extension) file.
  Usage:   mout = ptHalf(min)
  Where:
  min   - input matrix (must be square, and dimension a power of two)
  mout  - output matrix
*/

#include <math.h>
#include "mex.h"

/* Input Arguments */

#define M_IN  prhs[0]

/* Output Arguments */

#define M_OUT plhs[0]
```

```
void usage()
{
 printf("Usage:   mout = ptHalfC(min)\n");
 printf("\nWhere:\n\n");
 printf("min - input matrix (must be square and dimension a power of
   two)\n");
 printf("mout - output matrix\n");
}

void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray*prhs[] )

{
    double *ptr_min, *ptr_mout;
    double tmp;
    unsigned int a,b,i,j,m,n,size,msize,nsize;

    /* Check for proper number of arguments */

    if (nrhs != 1) {
      usage();
      mexErrMsgTxt("One input argument required.");
    } else if (nlhs > 1) {
      usage();
      mexErrMsgTxt("Too many output arguments.");
    }

    /* should also really check that matrix is type REAL (doubles) here */

    /* Check the dimensions of M_IN */
```

```
msize = mxGetM(M_IN);
nsize = mxGetN(M_IN);

if(msize!=nsize){
  usage();
  mexErrMsgTxt("Input matrix not square.");
}

m = rint(log((double)msize)/log(2));

if(1<<m != msize){
  usage();
  mexErrMsgTxt("Input matrix dimension not power of two.");
}

/* setup */

size = sqrt(msize);

M_OUT = mxCreateDoubleMatrix(msize, nsize, mxREAL);

ptr_mout = mxGetPr(M_OUT); /* assign memory pointers */
ptr_min = mxGetPr(M_IN);

/* M_OUT = M_IN */
memcpy(ptr_mout,ptr_min,sizeof(double)*msize*nsize);

/* note that matlab stores its matrices columnwise! */
```

```
    /* Do the transposition */

    for (a=0;a<size;a++){
      for (b=0;b<size;b++){
        m = a*size;              /* Calculate the temporary vertical offset */
        n = b*size;              /* Calculate the temporary horizontal offset */
        for (i=0;i<size;i++){
          for (j=0;j<i;j++){
            tmp = *(ptr_mout+(m+i)*nsize+n+j);
            *(ptr_mout+(m+i)*nsize+n+j) = *(ptr_mout+(m+j)*nsize+n+i);
            *(ptr_mout+(m+j)*nsize+n+i) = tmp;
          }
        }
      }
    }
    return;
}

% File:     ptNMinusOne.m
% Usage:    [mpt, mem, rt] = ptNMinusOne(mat)
% Date:     11-Apr-03
% Author:   Terri Yu <teryu@mit.edu>
% Function: Compute the partial transpose of a n-qubit
%           matrix density matrix using the {N-1,1} split.
% Notes:    Assumes that the input matrix is square and even
%           (which is the case if we truly have an n-qubit
%           density matrix.
% Input:    mat - input (square) matrix
% Returns:  mpt  - output partial transpose matrix
%           mem - memory used in bytes
%           rt - runtime in seconds

function [mpt, mem, rt] = ptNMinusOne(mat)

% Start timer
tic;
matsize = length(mat);  % size of matrix, assume it's square!
n = log2(matsize);
mpt = mat;
% Swap in lower triangle of matrix
% a = rows, b = cols
for a=1:matsize
  for b=1:a-1
    % Check if first N-1 qubits are the different
    if (bitshift(a-1,-1) ~= bitshift(b-1,-1))
      % Check if states have the same parity
      if (mod(a+b,2) == 0)
        mpt(a,b) = mat(b,a);
        mpt(b,a) = mat(a,b);
      % Check if a has even binary representation
      elseif (mod(a-1,2) == 0)
        mpt(a,b) = mat(b-1,a+1);
        mpt(b-1,a+1) = mat(a,b);
      % Then a has odd binary representation
      else
        mpt(a,b) = mat(b+1,a-1);
        mpt(b+1,a-1) = mat(a,b);
      end
    end
  end
```

```
end
% Count memory use
s = whos;
mem = totmem(s);
% End timer
rt = toc;
% Clear variables
clear starttime endttime s;

% File:     ptOne.m
% Usage:    [mpt, mem, rt] = ptOne(mat)
% Date:     11-Apr-03
% Author:   Terri Yu <teryu@mit.edu>
% Function: Compute the partial transpose of a n-qubit
%           matrix density matrix using the {1,N-1} split.
% Notes:    Assumes that the input matrix is square and even
%           (which is the case if we truly have an n-qubit
%           density matrix.
% Input:    mat - input (square) matrix
% Returns:  mpt  - output partial transpose matrix
%           mem - memory used in bytes
%           rt - runtime in seconds

function [mpt, mem, rt] = ptOne(mat)

% Start timer
tic;
matsize = length(mat);  % size of matrix, assume it's square!
n = log2(matsize);
mpt = mat;
```

```
msb = 2^(n-1); % numerical value of most significant bit
% Swap in upper righthand corner of matrix
% a = rows, b = cols
for a=1:matsize/2
  for b=matsize/2+1:matsize
    mpt(a,b) = mat(a+msb,b-msb);
    mpt(a+msb,b-msb) = mat(a,b);
  end
end
% Count memory use
s = whos;
mem = totmem(s);
% End timer
rt = toc;
% Clear variables
clear starttime endttime s;


% File:     ptranspose.m
% Usage:     [mpt, mem, rt] = ptranspose(mat, part)
% Date:      22-Oct-02
% Author:    Terri Yu <teryu@mit.edu>
% Function: Performs partial transpose of a n-qubit matrix,
%           given the number of qubits in first half of the
%           bipartite split.
%           Only {1,N-1}, {N/2,N/2}, and {N-1,1} partitions implemented.
% Notes:     Prints error if input matrix size is not a power of 2
%           and if the total number of qubits is less than the
%           number specified in first half of the bipartite split
% Input :   mat - a 2^n x 2^n matrix where n is an even positive integer
%           part - number of qubits in first half of biparite split
% Returns:  mpt - partial transposed verson of mat
%           mem - memory used in bytes
%           rt - runtime in seconds

function [mpt, mem, rt] = ptranspose(mat, part)

% Start timer
tic;
nq = log2(length(mat));
if ((round(nq) == nq) & (nq > part))
  switch part
    % {1,N-1} split
    case 1,
      mpt = ptOne(mat);
    % {N/2,N/2} split
    case nq/2,
      mpt = ptHalf(mat);
    % {N-1,1} split
    case nq-1,
      mpt = ptNMinusOne(mat);
    otherwise,
      mpt = mat;
      disp('Error -- partition not recognized');
  end
else
  disp('Error -- input matrix is invalid');
end
% Count memory usage
s = whos;
mem = totmem(s);
```

```
% End timer
rt = toc;
% Clear variables
clear starttime s endtime;


% File:     runexpt.m
% Usage:    qctmvec = runexpt(genunit, part, alpha, nq, filename)
% Date:     29-Oct-02
% Author:   Terri Yu <teryu@mit.edu>
% Function: Calculates minimum eigenvalue for each point in a vector
%           for number of qubits, given the specified alpha range,
%           the unitary generating function, and the partition.
%           Writes a QCTM structure for each qubit value using the
%           filename '<filename><date>q<nq(i)>.mat'.
%           Writes an overall QCTM vector structure using the
%           filename '<filename>vec<date>.mat'.
% Input:    @genunit - function handle pointing to unitary generator
%           part - vector specifying number of qubits in first half
%                  of bipartite split
%           alpha - vector specifying the range in alpha
%           nq - vector specifying the number of qubits
%           filename - string for beginning of filename
% Returns:  qctmvec - vector of QCTM structures corresponding
%                     to each component of nq

function qctmvec = runexpt(genunit, part, alpha, nq, filename)

nqlen = length(nq);
date = dt;
```

```
for i=1:nqlen
  qctminfo = calcmineig(genunit, part(i), alpha, nq(i))
  qctmvec(i) = writeqctm(qctminfo, strcat(['filename' ' ' strcat(filename,
    date)]));
end
vecfile = strcat('/home/terri/entanglement/data/', filename, date,
  'vec.mat');
save(vecfile,'qctmvec');


% File:     thermalden.m
% Usage:    [rho, mem, rt] = thermalden(alpha, n)
% Date:     30-Sept-02
% Author:   Terri Yu <teryu@mit.edu>
% Function: Calculates the thermal density matrix = exp(-H * alpha)
%           for a specified number of qubits.
%           The Hamiltonian H is the homogenous sum of single Sz
%           operations on each qubit.
%           For example, for 3 qubits, H = 1/2*(ZII + IZI + IIZ)
%           [here we set \hbar = 1].
% Input:    alpha - dimensionless quantity h\nu/kT
%           n - number of qubits
% Returns:  rho - thermal density matrix
%           mem - memory used in bytes
%           rt - runtime in seconds

function [rho, mem, rt] = thermalden(alpha, n)

% Start timer
tic;
% Initialize Hamiltonian
```

```
hvec = zeros(2^n,1);
for i=1:2^n
  weight = 0;
  if (i ~= 1)
    for j=1:ceil(log2(i-1))+1
      weight = weight + bitget(i-1,j);
    end
  end
  hvec(i) = -1 * (n - 2 * weight);
end
hvec = hvec/2;
rho = diag(exp(-hvec*alpha));
clear hvec; % Remove large matrix from memory
rho = rho/trace(rho);  % Make sure rho is normalized
% Count memory usage
s = whos;
mem = totmem(s);
% End timer
rt = toc;
% Clear variables
clear starttime n s endtime alpha c;


% File:      totmem.m
% Usage:     musage = totmem(varstruct)
% Date:      04-Oct-02
% Author:    Terri Yu <teryu@mit.edu>
% Function:  Calculates amount of memory used with a struct array input
%            that is created externally by running the "whos" command.
% Notes:     The struct only has information about the variables in
%            its scope.  Typically, it is created inside a function
%            and therefore totmem() calculates the memory usage for
%            that function.
% Input:     varstruct - structure that contains information on
%                            workspace variables
% Returns:   musage = total memory usage in the scope of varstruct

function musage = totmem(varstruct)
size = length(varstruct);
musage = 0;
for i=1:size
  musage = musage + varstruct(i).bytes;
end


% File:      ubs.m
% Usage:     [U, Uname, mem, rt] = ubs(nq)
% Date:      23-May-03
% Author:    Terri Yu <teryu@mit.edu>
% Function: Creates a standard unitary transform that maps the computational
%            basis to the Bell state basis
%            Example for 2 qubits:
%            U = [1 0 1 0; 0 1 0 1; 0 1 0 -1; 1 0 -1 0]
% Input:     nq - number of qubits
% Returns:   U - unitary transform
%            Uname - string describing unitary U
%            mem - memory usage of function
%            rt - runtime of function in seconds

function [U, Uname, mem, rt] = ubs(nq)

% Start timer
```

```
tic;
size = 2^nq;
U = zeros(size);
for i=1:size/2
  U(i,i) = 1;
  U(size-i+1,i) = 1;
end
for i=size/2+1:size
  U(i-size/2,i) = 1;
  U(3*size/2-i+1,i) = -1;
end
U = U/sqrt(2);
Uname = 'ubs';
% Count memory usage
s = whos;
mem = totmem(s);
% End timer
rt = toc;
% Clear variables
clear starttime endtime s;


% File:     ubsufan.m
% Usage:    [U, Uname, mem, rt] = ubsufan(nq)
% Date:     30-May-03
% Author:   Terri Yu <teryu@mit.edu>
% Function: Creates a unitary transform that maps the computational
%           basis to the Bell state basis
%           Example for 2 qubits:
%           U = [1 0 0 1; 0 1 1 0; 0 1 -1 0; 1 0 0 -1]
% Input:    nq - number of qubits
% Returns:  U - unitary transform
%           Uname - string describing unitary U
%           mem - memory usage of function
%           rt - runtime of function in days

function [U, Uname, mem, rt] = ubsufan(nq)

% Start timer
tic;
size = 2^nq;
U = zeros(size);
for i=1:size/2
  U(i,i) = 1;
  U(size−i+1,i) = 1;
end
for i=size/2+1:size
  U(size−i+1,i) = 1;
  U(i,i) = −1;
end
U = U/sqrt(2);
Uname = 'ubsufan';
% Count memory usage
s = whos;
mem = totmem(s);
% End timer
rt = toc;
% Clear variables
clear starttime endtime s;

% File:     writeqctm.m
% Usage:    qctm = writeqctm(qctminfo, options)
% Date:     18-Nov-02
% Author:   Terri Yu <teryu@mit.edu>
% Function: Saves a structure "qctm" with the following data:
%             qctm.filename - date followed by number of qubits
%             qctm.filedate - date file was created
%             qctm.hostname - computer calculation was run on
%             qctm.unitary - a string describing the unitary
%                            that was used to transform thermal state
%             qctm.nqubits - number of qubits
%             qctm.partition - number of qubits in first half of
%                              bipartite split
%             qctm.alpha - alpha data
%             qctm.mineig - minimum eigenvalue data
%             qctm.runtime - total runtime
%             qctm.funcdata - memory/runtime data for each function
%
%           If options = 'plot', the function will also plot the data.
%           If options = 'filename <filename-string>', the function will save
%               qctm using the filename: '<filename-string>q<#qubits>.mat'
%           If options = 'beowolf <filename-string> <machine-number>',
%               the function will save qctm using filename:
%               '<filename-string>q<#qubits>r<machine-number>.mat'
%           If neither 'filename' or 'beowolf' are used as options,
%               then qctm will be saved using filename: '<data>q<#qubits>.mat'
% Input:    qctminfo - QCTM info structure created by calcmineig.m
%           options  - a string specifying plotting and file name options
%                      (see above)
% Returns: qctm - QCTM structure containing data from numerical experiment
```

```
function qctm = writeqctm(qctminfo, options)

% Construct filename
[arg1, rem] = strtok(options);
date = dt;
if (strcmp(arg1,'beowolf'))
  [arg2, rem] = strtok(rem);
  [arg3, rem] = strtok(rem);
  filename = sprintf('%sq%dr%02d', arg2, qctminfo.nqubits, str2num(arg3));
elseif (strcmp(arg1, 'filename'))
  [arg2, rem] = strtok(rem);
  filename = strcat(arg2, 'q', int2str(qctminfo.nqubits));
else
  filename = strcat(date, 'q', int2str(qctminfo.nqubits));
end
% Calculate total memory used on average for one data point
totusage = 0;
for i=1:4
  totusage = totusage + mean(qctminfo.funcdata(i).mem);
end
% If arg1 = 'plot', make plot of minimum eigenvalue versus temperature
if strcmp(arg1,'plot')
  figure;
  semilogx(1./qctminfo.alpha,qctminfo.mineig);
  xlabel('log_10(\alpha^{-1})');
  ylabel('Minimum eigenvalue');
  tlabel = strcat('[', filename, ']', '-Minimum eigenvalue versus
    temperature');
  title(tlabel);
  grid on;
```

```
end
% Create quantum computing treasure map structure
qctm.filename = filename;
qctm.filedate = date;
qctm.hostname = hostname;
qctm.unitary = qctminfo.unitary;
qctm.nqubits = qctminfo.nqubits;
qctm.partition = qctminfo.partition;
qctm.alpha = qctminfo.alpha;
qctm.mineig = qctminfo.mineig;
qctm.runtime = qctminfo.runtime;
qctm.funcdata = qctminfo.funcdata;
% Save qctm structure as .mat file in data directory
save(strcat('/home/terri/entanglement/data/',filename,'.mat'),'qctm');


% File:     zcutoff.m
% Usage:    vout = zcutoff(vin)
% Date:     29-Sept-02
% Author:   Terri Yu <teryu@mit.edu>
% Function: Every entry in vector "vin" greater than 0 is set to 0.
%           The other entries are left alone.
% Notes:    Assumes input vector is real
% Input:    vin - input vector of numbers
% Returns:  vout - output vector with positive entries forced to zero

function vout = zcutoff(vin)

size = length(vin);
vout = zeros(size, 1);
for i=1:size
```

```
    if (vin(i) > 0)
        vout(i) = 0;
    else
        vout(i) = vin(i);
    end
end
```

A.2 Code for plotting negativity data

```
% File:     ploteigmap.m
% Usage:    [x, y, z] = ploteigmap(qctmvec, titlestr)
% Date:     30-Sept-02
% Author:   Terri Yu <teryu@mit.edu>
% Function: Plots a colormap of minimum eigenvalue in parameter
%           space with log_{10}(\alpha^{-1}) on the x-axis and number of
%           qubits on the y-axis, given a vector of QCTM structures.
% Notes:    alpha is a dimensionless quantity equal to h\nu/kT.
%           Requires that the range of alpha is the same for each
%           QCTM structure; otherwise an error is displayed.
% Input:    qctmvec - vector of QCTM structures
%           titlestr - additional notes in title of plot
% Returns:  x - vector for log_{10}(\alpha^{-1})
%           y - vector for number of qubits
%           z - minimum eigenvalue

function [x, y, z] = ploteigmap(qctmvec, titlestr)

qlen = length(qctmvec);
alpha = qctmvec(1).alpha;
flag = 0;
```

```
% Checks that qctmvec(i).alpha is consistent for each
% QCTM structure in the qctmvec vector
if (qlen > 1)
  i = 2;
  while ((flag == 0) & (i <= qlen))
    if (length(alpha) ~= length(qctmvec(i).alpha))
      flag = 1;
    else
      check = (qctmvec(i).alpha == alpha);
      flag = (sum(check) ~= length(check));
    end
    i = i + 1;
  end
end
% If alpha is consistent for all the structures,
% construct matrix containing minimum eigenvalues
if ~flag
  nq = qctmvec(1).nqubits;
  mineig = qctmvec(1).mineig;
  for i=2:qlen
    nq = [nq qctmvec(i).nqubits];
    mineig = [mineig qctmvec(i).mineig];
  end
  nq = nq';
  mineig = mineig';
  % Gurvits bound for separability
  gsep = -1/2*log(2.*(2.^(nq/2 - 1)./(2.^(nq/2 - 1) + 1)).^(1./nq) - 1);
  % Braunstein bound for nonseparability
  bnonsep = -1/2*log(sqrt(2)-1)*ones(length(nq),1);
  % Construct vector of transition alpha values
  % These are the values of alpha where the minimum
  % eigenvalue is zero
  for i=1:qlen
    trans(i) = estzalpha(qctmvec(i).alpha);
  end
  figure;
  imagesc(log10(alpha.^(-1)), nq, mineig);
  axis xy;
  hold;
  plot(log10(gsep.^(-1)), nq, 'k', log10(bnonsep.^(-1)), nq, 'k--');
  plot(log10(trans.^(-1)), nq, 'k-.');
  xlabel('log10(\alpha^{-1})');
  ylabel('Number of qubits');
  date = dt;
  title(strcat('Map of minimum eigenvalue [', date, ']-', titlestr));
  colormap(jet);
  colorbar;
  % Set return values for future plotting
  x = log10(alpha.^(-1));
  y = nq;
  z = mineig;
else
  disp('Error: alpha of each qctm structure must be the same');
end

% File:     plotmem.m
% Usage:    plotmem(qctmvec)
% Date:     ??-Sept-02
% Author:   Terri Yu <teryu@mit.edu>
% Function: Plots the logarithm of the average overall and individual
%           function memory usage in bytes vs. number of qubits,
%           given a vector of QCTM structures.
%           Each QCTM structure stores the memory usage for
%           calculations at every data point in alpha, which we
%           average over in the plot.
% Notes:     alpha is a dimensionless quantity equal to h\nu/kT.
% Input:     qctmvec - vector of QCTM structures
% Returns:  Nothing

function plotmem(qctmvec)

qctmlen = length(qctmvec);
for i=1:qctmlen
  nq(i) = qctmvec(i).nqubits;
  % Total memory used by calculation for one alpha data point
  totusage(i) = qctmvec(i).meantotusage;
  % Memory used for thermalden()
  td(i) = mean(qctmvec(i).funcdata(1).mem);
  % Memory used for transform()
  tf(i) = mean(qctmvec(i).funcdata(2).mem);
  % Memory used for ptranspose()
  pt(i) = mean(qctmvec(i).funcdata(3).mem);
  % Memory used for mineig()
  me(i) = mean(qctmvec(i).funcdata(4).mem);
  % Memory used for genunit()
  gu(i) = mean(qctmvec(i).funcdata(5).mem);
end
totusage=log10(totusage);
td=log10(td);
tf=log10(ent);
```

```
pt=log10(pt);
me=log10(pt);
figure;
subplot(2,1,1);
plot(nq,totusage);
title('Average total memory usage');
xlabel('Number of qubits');
ylabel('log_{10}(memory) [bytes]');
subplot(2,1,2);
plot(nq,td,nq,tf,nq,pt,nq,me,nq,gu);
title('Average memory usage for individual functions');
xlabel('Number of qubits');
ylabel('log10(memory) [bytes]');
legend('thermalden','transform','ptranspose','mineig','genunit');

% File:     plotmineig.m
% Usage:    plotmineig(qctmvec)
% Date:     30-Sept-02
% Author:   Terri Yu <teryu@mit.edu>
% Function: Plots minimum eigenvalue versus alpha^(-1) with a
%           semi-log x-axis, given a vector of QCTM structures.
%           Each QCTM structure is plotted in a separate figure.
% Notes:    alpha is a dimensionless quantity equal to h\nu/kT.
% Input:    qctmvec - QCTM structure

function plotmineig(qctmvec)

qctmlen = length(qctmvec);
date = dt;
for i=1:qctmlen
```

```
  figure;
  semilogx((qctmvec(i).alpha).^(−1),qctmvec(i).mineig,'-o');
  xlabel('log_{10} \alpha^{-1}');
  ylabel('Minimum eigenvalue');
  str = strcat('Minimum eigenvalue vs. temperature [', date, '], ');
  str = strcat(str,int2str(qctmvec(i).nqubits), ' qubits');
  title(str);
  grid on;
end

% File:     plotnegmap.m
% Usage:     [x, y, z] = plotnegmap(qctmvec, option)
% Date:      31-Mar-03
% Author:    Terri Yu <teryu@mit.edu>
% Function: Plots a colormap of negativity for the partial
%           transposed thermal state in parameter space
%           with log_{10}(\alpha^{-1}) on the x-axis and number of
%           qubits on the y-axis, given a vector of QCTM structures.
%           Converts QCTM minimum eigenvalue to negativity with
%           the formula: negativity = max{0,-mineig}.
% Notes:     alpha is a dimensionless quantity equal to h\nu/kT.
% Input:     alpha - vector specifying range in alpha
%           nq - vector specifying range in number of qubits
%                (components must be even positive integers)
%           option - if a nonzero number is specified, contours are
%                     also plotted for the minimum eigenvalues
%                     {-0.01, -0.005, 0.005, 0.01}.
% Returns:  x - vector for log_{10}(\alpha^{-1})
%           y - vector for number of qubits
%           z - negativity
```

```
function [x, y, z] = plotnegmap(qctmvec, option)

qlen = length(qctmvec);
alpha = qctmvec(1).alpha;
flag = 0;
% Check to see that data structure is consistent
if (qlen > 1)
  i = 2;
  while ((flag == 0) & (i <= qlen))
    if (length(alpha) ~= length(qctmvec(i).alpha))
      flag = 1;
    else
      check = (qctmvec(i).alpha == alpha);
      flag = (sum(check) ~= length(check));
    end
    i = i + 1;
  end
end
% Data structure OK, proceed
if ~flag
  nq = qctmvec(1).nqubits;
  zmineig = zcutoff(qctmvec(1).mineig);
  mineig = qctmvec(1).mineig;
  for i=2:qlen
    nq = [nq qctmvec(i).nqubits];
    zmineig = [zmineig zcutoff(qctmvec(i).mineig)]; % cutoff mineig
    mineig = [mineig qctmvec(i).mineig]; % preserved mineig
  end
  nq = nq';
```

```
zmineig = (-1*zmineig)';
mineig = (-1*mineig)';
% Gurvits bound on separability of effective pure states
gsep = -log(2.*(2.^(nq/2 - 1)./(2.^(nq/2 - 1) + 1)).^(1./nq) - 1);
% Braunstein bound on nonseparability of effective pure states
bnonsep = -log(sqrt(2)-1)*ones(length(nq),1);
figure;
% Plot negativity colormap
imagesc(log10(alpha.^(-1)), nq, zmineig);
axis xy;
colormap(jet);
H = colorbar;
%set(H, 'LineWidth', 3);
hold;
if option
  % Plot negativity contours
  v = [-0.01 -0.005 0.005 0.01];
  xi = log10(alpha.^(-1));
  yi = nq;
  zi = mineig;
  [C, H] = contour(xi,yi,zi,v,'y--');
  %set(H, 'LineWidth', 3);
end
% Plot various bounds on effective pure state
plot(log10(gsep.^(-1)), nq, 'k', log10(bnonsep.^(-1)), nq, 'k--');
xlabel('log10(\alpha^{-1})');
ylabel('Number of qubits');
date = dt;
str = strcat('Map of negativity, N(\rho)=max(min\{-\lambda_i\},0) [');
str = strcat(str, date, ']');
```

```
  title = str;
  % Set return values
  x = log10(alpha.^(-1));
  y = nq;
  z = zmineig;
else
  disp('Error: alpha of each qctm structure must be the same');
end

% File:     plotruntime.m
% Usage:    plotruntime(qctmvec)
% Date:     30-Sept-02
% Author:   T. Yu <teryu@mit.edu>
% Function: Plots runtime versus number of qubits, given a vector of
%           QCTM structure "qctm"

function plotruntime(qctmvec)

qctmlen = length(qctmvec);
for i=1:qctmlen
  nq(i) = qctmvec(i).nqubits;
  % total run time for calcmineig
  totrt(i) = qctmvec(i).runtime;
  % thermalden runtime
  td(i) = mean(qctmvec(i).funcdata(1).rt);
  % transform runtime
  tf(i) = mean(qctmvec(i).funcdata(2).rt);
  % ptranspose runtime
  pt(i) = mean(qctmvec(i).funcdata(3).rt);
  % mineig runtime
  me(i) = mean(qctmvec(i).funcdata(4).rt);
  % genunit runtime
  gu(i) = mean(qctmvec(i).funcdata(5).rt);
end
figure;
clf;
% Plot total run time
subplot(2,1,1);
hold on;
plot(nq, log10(totrt), '*');
xlabel('Number of qubits');
ylabel('log_{10}(run time) [sec]');
date = dt;
title(strcat('Runtime vs. number of qubits [', date, ']-'));
grid on;
% Plot runtimes for individual functions
subplot(2,1,2);
plot(nq,log10(td),nq,log10(ent),nq,log10(pt),nq,log10(me));
xlabel('Number of qubits');
ylabel('log_{10}(run time) [min]');
date = dt;
title(strcat('Runtime vs. number of qubits [', date, ']'));
legend('thermalden','transform','ptranspose','mineig','genunit');
grid on;
```

Appendix B

Beowulf cluster negativity calculation module source code

Here we give the server side code for generating the standard Bell transform and thermal density matrix and for calculating the partial transpose operation. To match the text in Chapter 4 with the source code, the reader should bear in mind that the following module and function names are equivalent: `pubs()` = `qp_cnth()`, `pthermalden()` = `qp_thdm()`, and `pptranspose()` = `qp_ptran()`.

```
/* Name: qp_cnth(matrix *)
   Author: Terri Yu and Joshua Powell
   Generates a submatrix of a CNOT+Hadamard matrix and stores it in 'm'.
   More specifically this generates a square matrix representing the
   operator
         CNTH = CNOT_{1n}...CNOT_{13}CNOT_{12}H_1
   mapping the computational basis to the Bell basis.
*/

void qp_cnth(matrix * m) {
  int ictxt, nprow, npcol, myrow, mycol, icurrow, icurcol, lda;
  long i, j, ii, jj;
```

```
DBG("qp_cnth(): creating CNOT+Hadamard matrix %d\n", m->num);
ictxt = m->desc[1];
Cblacs_gridinfo( ictxt, &nprow, &npcol, &myrow, &mycol );
icurrow = m->desc[6];
icurcol = m->desc[7];
lda = m->desc[8];

DBG("qp_cnth(): nprow=%d myrow=%d npcol=%d mycol=%d m->x=%d\n",
        nprow, myrow, npcol, mycol, m->x);
DBG("qp_cnth(): icurrow=%d icurcol=%d lda=%d\n", icurrow, icurcol, lda);

for(jj=1; jj<=m->x/npcol; jj++) {
  j = (((jj-1)/m->desc[5])*npcol+mycol) * m->desc[5] -
        (jj%m->desc[5]) + 1;
  for(ii=1; ii<=m->y/nprow; ii++) {
    i = (((ii-1)/m->desc[4])*nprow+myrow) * m->desc[4] -
      (ii%m->desc[4]) + 1;
    if(i<=((m->y)/2-1) && j<=((m->x)/2-1)) {
      CSET(m->m, ii+(jj-1)*lda,(i==j)?SQRTHALF:0.0,0.0);
      DBG("qp_cnth(): top left corner\n");
    }
    if(i<=((m->y)/2-1) && j>((m->x)/2-1)) {
      CSET(m->m, ii+(jj-1)*lda,((i+(m->x)/2)==j)?SQRTHALF:0.0,0.0);
      DBG("qp_cnth(): top right corner\n");
    }
    if(i>((m->y)/2-1) && j<=((m->x)/2-1)) {
      CSET(m->m, ii+(jj-1)*lda,((i+j+1)==(m->x))?SQRTHALF:0.0,0.0);
      DBG("qp_cnth(): bottom left corner\n");
    }
```

```
      if(i>((m->y)/2-1) && j>((m->x)/2-1)) {
        CSET(m->m, ii+(jj-1)*lda,((i+j+1)==(3*(m->x)/2))?
          -1*SQRTHALF:0.0,0.0);
        DBG("qp_cnth(): bottom right corner\n");
      }
      DBG("qp_cnth(): ID=(%d,%d) [%d]=(%d,%d) [%f+%fi] on (%d,%d)
        p=%ld\n",
              ii, jj, ii+(jj-1)*lda, i, j, RGET(m->m, ii+(jj-1)*lda),
              IGET(m->m, ii+(jj-1)*lda), myrow, mycol, m->m);
    }
  }
  DBG("qp_cnth(): done\n");
}

/* Function to calculate base 2 logarithm */

float log2(float num) {
  float result;

  result = log(num)/log(2.0);
  return result;
}

/* Calculates Hamming weight, the number of 1's */

int weight(long num) {
  int count;

  count = 0;
  while (num > 0) {
    if ((num & 0x1) == 1)
      count ++;
    num >>= 1;
  }
  return count;
}

/* Generate thermal density matrix */
/* Master process */
int qp_thdm(matrix * m, FLOAT alpha) {
  int ictxt, nprow, npcol, myrow, mycol, lda, flag;
  long i, j, ii, jj, pi, pj;
  FLOAT n, val, trace;
  FLOAT comm[SMALL_MSGSIZE];

#ifdef FDEBUG
  fprintf(stderr, "Generating thermal density matrix for serial = %d\n",
    m->num);
#endif
  ictxt = m->desc[1];
  Cblacs_gridinfo(ictxt, &nprow, &npcol, &myrow, &mycol);
  lda = m->desc[8];
  Cblacs_barrier(ictxt, "All");
  ((char *)comm)[0] = '$';
  ((FLOAT *)comm)[1] = alpha;
  Cigebs2d(ictxt, "All", " ", SMALL_MSGSIZE, 1, comm, SMALL_MSGSIZE);
  n = (FLOAT)log2((float)m->x);
  trace = 0.0;
  flag = 0;
#ifdef FDEBUG
```

```
  fprintf(stderr, "Calculate entries of thermal density matrix\n");
#endif
  for(jj=1; jj<=m->x/npcol; jj++) {
    j = (((jj-1)/m->desc[5])*npcol+mycol) * m->desc[5] - (jj%m->desc[5])
      + 1;
    for(ii=1; ii<=m->y/nprow; ii++) {
      i = (((ii-1)/m->desc[4])*nprow+myrow) * m->desc[4] - (ii%m->desc[4])
        + 1;
      if (i == j) {
        val = exp((n - 2.0 * (FLOAT)weight(i)) * alpha);
        CSET(m->m, ii+(jj-1)*lda, val, 0.0);
        trace += val;
      }
      else
        CSET(m->m, ii+(jj-1)*lda, 0.0, 0.0);
    }
  }
  Cblacs_barrier(ictxt, "All");
#ifdef FDEBUG
  fprintf(stderr, "Normalizing by trace\n");
#endif
  for (pi = 0; pi < nprow; pi++)
    for (pj = 0; pj < npcol; pj++)
      if ((pi != 0) || (pj != 0)) {
        ((char *)comm)[0] = '>';
        ((FLOAT *)comm)[1] = 0.0;
        Csgesd2d(ictxt, SMALL_MSGSIZE, 1, comm, SMALL_MSGSIZE, pi, pj);
        Csgerv2d(ictxt, SMALL_MSGSIZE, 1, comm, SMALL_MSGSIZE, pi, pj);
        if (((char *)comm)[0] == '<')
          trace += ((FLOAT *)comm)[1];
        else
          flag = -1;
      }
    Cblacs_barrier(ictxt, "All");
    ((char *)comm)[0] = '*';
    ((FLOAT *)comm)[1] = trace;
    Cigebs2d(ictxt, "All", " ", SMALL_MSGSIZE, 1, comm, SMALL_MSGSIZE);
#ifdef FDEBUG
  fprintf(stderr, "Divide by trace\n");
#endif
  for(jj=1; jj<=m->x/npcol; jj++) {
    j = (((jj-1)/m->desc[5])*npcol+mycol) * m->desc[5] - (jj%m->desc[5])
      + 1;
    for(ii=1; ii<=m->y/nprow; ii++) {
      i = (((ii-1)/m->desc[4])*nprow+myrow) * m->desc[4] - (ii%m->desc[4])
        + 1;
      if (i == j)
        CSET(m->m, ii+(jj-1)*lda, RGET(m->m, ii+(jj-1)*lda)/trace, 0.0);
    }
  }
  return flag;
}

/* Child process */
int qp_thdm_child(matrix * m) {
  int ictxt, nprow, npcol, myrow, mycol, lda, flag;
  long i, j, ii, jj, pi, pj;
  FLOAT n, val, trace, alpha;
  FLOAT comm[SMALL_MSGSIZE];
```

```
#ifdef FDEBUG
  fprintf(stderr, "[Child] Generating thermal density matrix for serial =
    %d\n", m->num);
#endif
  ictxt = m->desc[1];
  Cblacs_gridinfo(ictxt, &nprow, &npcol, &myrow, &mycol);
  lda = m->desc[8];
  flag = 0;
  Cblacs_barrier(ictxt, "All");
  Cigebr2d(ictxt, "All", " ", SMALL_MSGSIZE, 1, comm,
    SMALL_MSGSIZE, 0, 0);
  if (((char *)comm)[0] == '$')
    alpha = ((FLOAT *)comm)[1];
  else {
    alpha = 0.0;
    flag = -1;
  }
  /* Cblacs_barrier(ictxt, "All"); */
#ifdef FDEBUG
  fprintf(stderr, "[Child] Calculating entries of thermal density
    matrix\n");
#endif
  n = (FLOAT)log2((float)m->x);
  trace = 0.0;
  for(jj=1; jj<=m->x/npcol; jj++) {
    j = (((jj-1)/m->desc[5])*npcol+mycol) * m->desc[5] - (jj%m->desc[5])
      + 1;
    for(ii=1; ii<=m->y/nprow; ii++) {
      i = (((ii-1)/m->desc[4])*nprow+myrow) * m->desc[4] - (ii%m->desc[4])
      + 1;
```

```
        if (i == j) {
          val = exp((n - 2.0 * (FLOAT)weight(i)) * alpha);
          CSET(m->m, ii+(jj-1)*lda, val, 0.0);
          trace += val;
        }
        else
          CSET(m->m, ii+(jj-1)*lda, 0.0, 0.0);
      }
    }
    Cblacs_barrier(ictxt, "All");
#ifdef FDEBUG
    fprintf(stderr, "[Child] Normalize by trace\n");
#endif
    Csgerv2d(ictxt, SMALL_MSGSIZE, 1, comm, SMALL_MSGSIZE, 0, 0);
    if (((char *)comm)[0] == '>') {
      ((char *)comm)[0] = '<';
      ((FLOAT *)comm)[1] = trace;
      Csgesd2d(ictxt, SMALL_MSGSIZE, 1, comm, SMALL_MSGSIZE, 0, 0);
      Cblacs_barrier(ictxt, "All");
      Cigebr2d(ictxt, "All", " ", SMALL_MSGSIZE, 1, comm, SMALL_MSGSIZE,
        0, 0);
      if (((char *)comm)[0] == '*')
        trace = ((FLOAT *)comm)[1];
      else
        flag = -1;
    }
    else
      flag = -1;
#ifdef FDEBUG
    fprintf(stderr, "[Child] Divide by trace\n");
```

```
#endif
  for(jj=1; jj<=m->x/npcol; jj++) {
    j = (((jj-1)/m->desc[5])*npcol+mycol) * m->desc[5] - (jj%m->desc[5])
      + 1;
    for(ii=1; ii<=m->y/nprow; ii++) {
      i = (((ii-1)/m->desc[4])*nprow+myrow) * m->desc[4] - (ii%m->desc[4])
      + 1;
      if (i==j)
        CSET(m->m, ii+(jj-1)*lda, RGET(m->m, ii+(jj-1)*lda)/trace, 0.0);
    }
  }
  return flag;
}

void
qp_ptran(matrix * m) {
  FLOAT comm[5];
  int ictxt, nprow, npcol, myrow, mycol;
  int n, nhalf, ra, ca, rb, cb;
  int i1, j1, i2, j2, pi, pj;
  int count = 0;
  ictxt = m->desc[1];
  Cblacs_gridinfo( ictxt, &nprow, &npcol, &myrow, &mycol );
  n = m->x; /* Size of matrix, assumes matrix is square */
  nhalf = sqrt(m->x); /* Assumes matrix dimension is a square */
#ifdef DEBUG
  fprintf(stderr, "n: %d, nhalf: %d\n", n, nhalf);
#endif
  for(rb = 0; rb <= n-1; rb += nhalf) { /* Row B */
    for(cb = 0; cb <= n-1; cb += nhalf) { /* Column B */
```

```
        for(ra = 0; ra <= nhalf-1; ra++) { /* Row A */
          for(ca = ra+1; ca <= nhalf-1; ca++) { /* Column A */
            i1 = ra + rb + 1;
            j1 = ca + cb + 1;
            i2 = ca + rb + 1;
            j2 = ra + cb + 1;
            qp_swap(m,i1-1,j1-1,i2-1,j2-1);
#ifdef PPT
            fprintf(stderr, "Swapping (%d, %d) and (%d, %d)\n", i1, j1, i2, j2);
#endif
            count++;
            if(count > SWAPNUM) {
              ((char *)comm)[0]='$';
              for(pi=0; pi<nprow; pi++)
                for(pj=0; pj<npcol; pj++)
                  if(pi!=0 || pj!=0)
                    Csgesd2d(ictxt, 5, 1, comm, 5, pi, pj);
              Cblacs_barrier(ictxt, "All");
              count = 0;
            }
          }
        }
      }
    }
#ifdef DEBUG
            fprintf(stderr, "Swapping (%d, %d) and (%d, %d)\n", 0, 3, 3, 0);
#endif
/*  qp_swap(m, 0, 3, 3, 0);*/
  /* Tell children that partial transpose is complete */
#ifdef DEBUG
```

```
  fprintf(stderr, "Sending message to children to quit partial
    transpose\n");
#endif
  ((char *)comm)[0]='.';
  for(pi=0; pi<nprow; pi++)
    for(pj=0; pj<npcol; pj++)
      if(pi!=0 || pj!=0)
        Csgesd2d(ictxt, 5, 1, comm, 5, pi, pj);
}
/* Swap function for master process */
/* TY: I don't do any with the flag, might want error checking in the future */
int
qp_swap(matrix *m, int i1, int j1, int i2, int j2) {
  FLOAT comm[5];
  int pi, pj, pi1, pj1, pi2, pj2, ii1, jj1, ii2, jj2;
  int ictxt, nprow, npcol, myrow, mycol, lda;
  FLOAT tmp_re, tmp_im;
  int flag;
  /* Get BLACS info */
  ictxt = m->desc[1];
  Cblacs_gridinfo(ictxt, &nprow, &npcol, &myrow, &mycol);
  lda = m->desc[8];
  /* Find process coordinates corresponding to global matrix coordinates */
  pi1 = (i1/m->desc[4])%nprow;
  pj1 = (j1/m->desc[5])%npcol;
  pi2 = (i2/m->desc[4])%nprow;
  pj2 = (j2/m->desc[5])%npcol;
  /* Transform global to local coordinates */
  ii1 = (i1/(m->desc[4] * nprow))*m->desc[4] + (i1%m->desc[4]);
  jj1 = (j1/(m->desc[5] * npcol))*m->desc[5] + (j1%m->desc[5]);
  ii2 = (i2/(m->desc[4] * nprow))*m->desc[4] + (i2%m->desc[4]);
  jj2 = (j2/(m->desc[5] * npcol))*m->desc[5] + (j2%m->desc[5]);
#ifdef PPT
  fprintf(stderr, "Initiating swap between process1 (%d, %d) and process2
    (%d, %d) with local coordinates (%d, %d) and (%d, %d)\n", pi1, pj1,
    pi2, pj2, ii1, jj1, ii2, jj2);
#endif
  /* Check if the swap happens inside same process/node */
  if ((pi1 == pi2) && (pj1 == pj2))
    /* Check if master owns the matrix elements we want to swap */
    if ((pi1 == 0) && (pj1 == 0)) {
#ifdef DEBUG
      fprintf(stderr, "Swap inside master\n");
#endif
      tmp_re = RGET(m->m, ii1+jj1*lda+1);
      tmp_im = IGET(m->m, ii1+jj1*lda+1);
      CSET(m->m, ii1+jj1*lda+1, RGET(m->m, ii2+jj2*lda+1), IGET(m->m,
        ii2+jj2*lda+1));
      CSET(m->m, ii2+jj2*lda+1, tmp_re, tmp_im);
      flag = 0;
    }
    /* Otherwise direct remote process to swap its matrix elements*/
    else {
#ifdef DEBUG
      fprintf(stderr, "Swap inside remote process\n");
#endif
      ((char *)comm)[0] = '*';
      ((int *)comm)[1] = ii1;
      ((int *)comm)[2] = jj1;
      ((int *)comm)[3] = ii2;
      ((int *)comm)[4] = jj2;
      Csgesd2d(ictxt, 5, 1, comm, 5, pi1, pj1);
      flag = 0;
    }
  /* Master has part of the matrix and needs to swap with a remote node */
  else if ((pi1 == 0) && (pj1 == 0)) {
#ifdef DEBUG
      fprintf(stderr, "Swap between master (process 1) and remote node\n");
#endif
      /* Master tells remote node which matrix element it needs and */
      /* where to send it */
    ((char *)comm)[0] = '%';
    ((int *)comm)[1] = ii2;
    ((int *)comm)[2] = jj2;
    ((int *)comm)[3] = pi1;
    ((int *)comm)[4] = pj1;
    Csgesd2d(ictxt, 5, 1, comm, 5, pi2, pj2);
    /* Send remote node data for swap */
    ((char *)comm)[0]='>';
    ((FLOAT *)comm)[1] = RGET(m->m, ii1+jj1*lda+1);;
    ((FLOAT *)comm)[2] = IGET(m->m, ii1+jj1*lda+1);;
    ((int *)comm)[3] = ((int *)comm)[4] = 0;
    Csgesd2d(ictxt, 5, 1, comm, 5, pi2, pj2);
    /* Get ready to receive data */
    flag = 1;
    while(flag == 1) {
      Csgerv2d(ictxt, 5, 1, comm, 5, pi2, pj2);
      if(((char *)comm)[0] == '>') {
#ifdef PPT
        fprintf(stderr, "Master received data from process (%d,%d)\n", pi2,
          pj2);
#endif
        tmp_re = ((FLOAT *)comm)[1];
        tmp_im = ((FLOAT *)comm)[2];
#ifdef PPT
        fprintf(stderr, "(Master) tmp_re: %f, tmp_im: %f\n", tmp_re,
          tmp_im);
#endif
        CSET(m->m,ii1+jj1*lda+1, tmp_re, tmp_im);
        flag = 0;
      }
      else
        flag = -1;
    }
      flag = 0;
  }
  else if ((pi2 == 0) && (pj2 == 0)) {
#ifdef DEBUG
      fprintf(stderr, "Swap between master (process 2) and remote node\n");
#endif
    /* Master tells remote node which matrix element it needs and */
    /* where to send it */
    ((char *)comm)[0]='%';
    ((int *)comm)[1]=ii1;
    ((int *)comm)[2]=jj1;
    ((int *)comm)[3]=pi2;
    ((int *)comm)[4]=pj2;
    Csgesd2d(ictxt, 5, 1, comm, 5, pi1, pj1);
    /* Master sends data to remote node */
    ((char *)comm)[0]='>';
    ((FLOAT *)comm)[1] = RGET(m->m, ii2+jj2*lda+1);;
    ((FLOAT *)comm)[2] = IGET(m->m, ii2+jj2*lda+1);;
    ((int *)comm)[3] = ((int *)comm)[4] = 0;
    Csgesd2d(ictxt, 5, 1, comm, 5, pi1, pj1);
    /* Master prepares to receive data from remote node */
    flag = 1;
    while(flag == 1) {
      Csgerv2d(ictxt, 5, 1, comm, 5, pi1, pj1);
      if(((char *)comm)[0] == '>') {
#ifdef PPT
        fprintf(stderr, "Master received data from process (%d,%d)\n", pi2,
          pj2);
#endif
        tmp_re = ((FLOAT *)comm)[1];
        tmp_im = ((FLOAT *)comm)[2];
#ifdef PPT
        fprintf(stderr, "(Master) tmp_re: %f, tmp_im: %f\n", tmp_re,
          tmp_im);
#endif
        CSET(m->m,ii2+jj2*lda+1, tmp_re, tmp_im);
        flag = 0;
      }
      else
        flag = -1;
    }
    flag = 0;
  }
  else {
#ifdef DEBUG
      fprintf(stderr, "Swap between remote nodes\n");
```

```
#endif
    /* Tell first process which matrix element it needs to send where */
    ((char *)comm)[0]='%';
    ((int *)comm)[1]=ii1;
    ((int *)comm)[2]=jj1;
    ((int *)comm)[3]=pi2;
    ((int *)comm)[4]=pj2;
    Csgesd2d(ictxt, 5, 1, comm, 5, pi1, pj1);
    /* Tell second process which matrix element it needs to send where */
    ((char *)comm)[0]='%';
    ((int *)comm)[1]=ii2;
    ((int *)comm)[2]=jj2;
    ((int *)comm)[3]=pi1;
    ((int *)comm)[4]=pj1;
    Csgesd2d(ictxt, 5, 1, comm, 5, pi2, pj2);
    flag = 0;
  }
  return flag;
}
/* Partial transpose function for child process */
void
qp_ptran_child(matrix *m) {
  FLOAT comm[5];
  int pi, pj, ii, jj, ii1, jj1, ii2, jj2;
  int ictxt, nprow, npcol, myrow, mycol, lda;
  FLOAT tmp_re, tmp_im;
  int flag;
  /* Get BLACS info */
  ictxt = m->desc[1];
  Cblacs_gridinfo(ictxt, &nprow, &npcol, &myrow, &mycol);
  lda = m->desc[8];
  flag = 1;
  /* Child waits for broadcast */
  while (flag == 1) {
#ifdef DEBUG
    fprintf(stderr, "child node waiting for broadcast\n");
#endif
    Csgerv2d(ictxt, 5, 1, comm, 5, 0, 0);
#ifdef DEBUG
    fprintf(stderr, "got something: %c\n", ((char *)comm)[0]);
#endif
    switch(((char *)comm)[0]) {
    case '*': /* Swap elements within local matrix */
      ii1 = ((int *)comm)[1];
      jj1 = ((int *)comm)[2];
      ii2 = ((int *)comm)[3];
      jj2 = ((int *)comm)[4];
      tmp_re = RGET(m->m, ii1+jj1*lda+1);
      tmp_im = IGET(m->m, ii1+jj1*lda+1);
      CSET(m->m, ii1+jj1*lda+1, RGET(m->m, ii2+jj2*lda+1), IGET(m->m,
        ii2+jj2*lda+1));
      CSET(m->m, ii2+jj2*lda+1, tmp_re, tmp_im);
#ifdef DEBUG
      fprintf(stderr, "completed swap inside remote node\n");
#endif
      break;
    case '%': /* Swap elements with another node */
      /* Coordinates of matrix element that it needs to send */
      ii = ((int *)comm)[1];
      jj = ((int *)comm)[2];
      /* Coordinates of process data needs to be sent to */
      pi = ((int *)comm)[3];
      pj = ((int *)comm)[4];
#ifdef PPT
      fprintf(stderr, "Process (%d, %d) received command to send data to
        process (%d, %d)\n", myrow, mycol, pi, pj);
#endif
      /* Send data to remote node */
      ((char *)comm)[0] = '>';
      ((FLOAT *)comm)[1] = RGET(m->m, ii+jj*lda+1);
      ((FLOAT *)comm)[2] = IGET(m->m, ii+jj*lda+1);
      ((int *)comm)[3] = ((int *)comm)[4] = 0;
      Csgesd2d(ictxt, 5, 1, comm, 5, pi, pj);
      /* Receive data from remote node */
      Csgerv2d(ictxt, 5, 1, comm, 5, pi, pj);
      switch(((char *)comm)[0]) {
      case '>':
        tmp_re = ((FLOAT *)comm)[1];
        tmp_im = ((FLOAT *)comm)[2];
#ifdef PPT
        fprintf(stderr, "tmp_re: %f, tmp_im: %f\n", tmp_re, tmp_im);
#endif
        CSET(m->m, ii+jj*lda+1, tmp_re, tmp_im);
        break;
      default:
        flag = -1;
      }
      break;
    case '$': /* Stop here */
      Cblacs_barrier(ictxt, "All");
      break;
    case '.': /* Stop listening */
      flag = 0;
      break;
    default:
      flag = -1;
    }
  }
}

int qp_peek(matrix *m, int s, int row, int col){

  FLOAT re, im;
  FLOAT comm[4];
  FLOAT buf[4];
  int ictxt, nprow, npcol, myrow, mycol, icurrow, icurcol, lda;
  long i, j;
  int li, lj;

  ictxt = m->desc[1];
  Cblacs_gridinfo( ictxt, &nprow, &npcol, &myrow, &mycol );
  icurrow = m->desc[6];
  icurcol = m->desc[7];
  lda = m->desc[8];

  i = processIndex(row, nprow, m->desc[4]);
  j = processIndex(col, npcol, m->desc[5]);
  li = localIndex(row, nprow, m->desc[4]);
  lj = localIndex(col, npcol, m->desc[5]);
  if(s!=0) { /* We are the parent */
```

```
DBG("qp_peek(): (%d,%d) maps to process (%d,%d) and local (%d,%d)\n",
    row, col, (int)i, (int)j, li, lj);
if( i==0 && j==0 )
  {
    /* The server process has the matrix element */
    /* The ?GET macros require 1-based indicies */
    re = RGET(m->m, li + lj*lda + 1);
    im = IGET(m->m, li + lj*lda + 1);
  } else {
    /* One of the children has the matrix element */
    Csgerv2d(ictxt, 4, 1, comm, 4, i, j);
    DBG("qp_peek(): Got %f+%f from (%d,%d)\n", (float)comm[0],
        (float)comm[1], (int)i, (int)j);
    re = (float)comm[0];
    im = (float)comm[1];
  }

/* Transmit the element to the client */
DBG("qp_peek(): server sending %f+%fi\n", re, im);
((char *)buf)[0]='#';
((char *)buf)[1]='\n';
buf[1] = re ;
buf[2] = im;
if(write(s, buf, sizeof(buf))==-1)
  perror("Write");
if(read(s, buf, sizeof(buf))==-1)
  perror("Read");

} else { /* We are a child */
```

```
    if(m==NULL){
      fprintf(stderr, "qp_peek(): matrix pointer is null\n");
      return 1;
    }
    if( i==myrow && j==mycol )
      {
        comm[0] = RGET(m->m, li + lj*lda + 1);
        comm[1] = IGET(m->m, li + lj*lda + 1);
        DBG("qp_peek(): send %f+%fi from (%d,%d)\n",
            comm[0], comm[1], myrow, mycol);
        Csgesd2d(ictxt, 4, 1, comm, 4, 0, 0);
      }
  }
  return 0;
}

/* Name: void qp_rand(matrix *)
   Author: Joshua Powell & Terri Yu

   Fills a matrix's entries with random floating point numbers between 0
   and 1.
   JP formerly called this function qp_fvec() because it was used to create
   a fake random vector.
*/

void
qp_rand(matrix * m) {
  int ictxt, nprow, npcol, myrow, mycol, icurrow, icurcol, lda;
  long i, j, ii, jj, iimax, jjmax;
  float new_entry;
  long new_entry1;
  long new_entry2;
  unsigned t;

#ifdef DEBUG
  fprintf(stderr, "Generating random matrix for serial = %d\n", m->num);
#endif
  ictxt = m->desc[1];
  Cblacs_gridinfo(ictxt, &nprow, &npcol, &myrow, &mycol);
  icurrow = m->desc[6];
  icurcol = m->desc[7];
  lda = m->desc[8];
  new_entry = 0.0;                /* Ratio of two randoms */
  new_entry1 = 0;                 /* First random */
  new_entry2 = 0;                 /* Second random */

#ifdef DEBUG
  fprintf(stderr, "nprow: %d myrow: %d npcol: %d mycol: %d\n",
          nprow, myrow, npcol, mycol);
  fprintf(stderr, "icurrow: %d icurcol %d lda; %d\n", icurrow, icurcol, lda);
#endif

  /* jjmax = (m->x == 1)?1:m->x/npcol; */
  /* iimax = (m->y == 1)?1:m->y/nprow; */
  if (m->x == 1) {
    jjmax = m->y/npcol;
    iimax = 1;
  }
  else if (m->y == 1) {
    jjmax = 1;
```

```
      iimax = m->x/nprow;
    }
    else {
      jjmax = m->x/npcol;
      iimax = m->y/nprow;
    }
    t = (unsigned) time(NULL);
    for (jj=1; jj<=jjmax; jj++){
      j = (((jj-1)/m->desc[5])*npcol+mycol) * m->desc[5] - (jj%m->desc[5])
        + 1;
      for(ii=1; ii<=iimax; ii++){
        i = (((ii-1)/m->desc[4])*nprow+myrow) * m->desc[4] - (ii%m->desc[4])
          + 1;
        if ((((m->x != 1) && (m->y != 1)) || ((m->x == 1) &&
            (myrow == 0)))
          || ((m->y == 1) && (mycol == 0))) {
          t++;
          srand(t);
          new_entry1 = rand();          /* Assign first random long integer */
          new_entry2 = rand();          /* Assign second random long integer*/
          /* Divide smaller rand by larger one */
          new_entry = (new_entry1 > new_entry2)?((float)new_entry2)/
            ((float)new_entry1):((float)new_entry1)/((float)new_entry2);
          /* Set entry in left-most column equal to ratio */
          CSET(m->m, ii+(jj-1)*lda, new_entry, 0.0);
#ifdef DEBUG
        fprintf(stderr, "ID: (%d, %d) [%d] = (%d, %d) [%f+ %fi] on (%d, %d)
          p=%ld\n",
               ii, jj, ii+(jj-1)*lda, i, j, RGET(m->m, ii+(jj-1)*lda),
               IGET(m->m, ii+(jj-1)*lda), myrow, mycol, m->m);
#endif
      }
      else
        /* Set entry in left-most column equal to ratio */
        CSET(m->m, ii+(jj-1)*lda, 0.0, 0.0);
    }
  }
}
```

Bibliography

[AGR81] A. Aspect, P. Grangier, and G. Roger. Experimental tests of realistic local theories via Bell's Theorem. *Phys. Rev. Lett.*, **47**, 460–463, 1981.

[AGR82] A. Aspect, P. Grangier, and G. Roger. Experimental realization of Einstein-Podolsky-Rosen-Bohm gedankenexperiment: A new violation of Bell's inequalities. *Phys. Rev. Lett.*, **49**, 91–94, 1982.

[BBC$^+$93] C. H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W. K. Wootters. Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels. *Phys. Rev. Lett.*, **70**, 1895–1899, 1993.

[BBP$^+$96] C. H. Bennett, G. Brassard, S. Popescu, B. Schumacher, J. A. Smolin, and W. K. Wooters. Purification of noisy entanglement and faithful teleportation via noisy channels. *Phys. Rev. Lett.*, **76**, 722–725, 1996.

[BCC$^+$97] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. ScaLAPACK Users Guide, 1997. http://www.netlib.org/scalapack/slug/.

[BCJ$^+$99] S. L. Braunstein, C. M. Caves, R. Jozsa, N. Linden, S. Popescu, and R. Schack. Separability of very noisy mixed states and implications for NMR quantum computing. *Phys. Rev. Lett.*, **83**, 1054–1057, 1999.

[BDG02] J.-L. Basdevant, J. Dalibard, and P. Grangier. Entangled states, EPR paradox, and Bell's inequality. In *Quantum Mechanics*. Springer-Verlag, Berlin, Germany, 2002.

[BDSW96] C. H. Bennett, D. P. DiVincenzo, J. A. Smolin, and W. K. Wootters. Mixed state entanglement and quantum error correction. *Phys. Rev. A*, **54**, 3824, 1996.

[Bel65] J. S. Bell. *Physics*, **1**, 195–200, 1965.

[Ben80] P. Benioff. The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines. *J. Stat. Phys.*, **22**, 563–591, 1980.

[BLM01] C. Berthier, L. P. Levy, and G. Martinez, editors. *High magnetic fields: applications in condensed matter physics and spectroscopy.* Springer Verlag, Berlin, Germany, 2001.

[BMR$^+$02] P. O. Boykin, T. Mor, V. Roychowdhury, F. Vatan, and R. Vrijen. Algorithmic cooling and scalable NMR quantum computers. *Proceedings of the National Academy of Sciences (U.S.A.)*, **99**, 3388–3393, 2002.

[Boh51] D. Bohm. *Quantum Theory.* Prentice Hall, Englewood Cliffs, NJ, 1951.

[CC03] A. C. Cross and I. L. Chuang. Quanta Beowulf Cluster, 2003. http://qubit.media.mit.edu.

[CDO$^+$95] J. Choi, J. Dongarra, S. Ostrouchov, A. Petitet, D. Walker, and R. C. Whaley. A proposal for a set of parallel basic linear algebra subprograms, July 1995. http://www.netlib.org/lapack/lawnspdf/lawn100.pdf.

[CGK98] I. L. Chuang, N. Gershenfeld, and M. Kubinec. Experimental implementation of fast quantum searching. *Phys. Rev. Lett.*, **80**, 3408–3411, 1998.

[Cho02] R. Choy. Matlab*p 2.0: Interactive supercomputing made practical. Master's thesis, Massachusetts Institute of Technology, June 2002.

[CLK$^+$00] D. G. Cory, R. Laflamme, E. Knill, L. Viola, T. F. Havel, N. Boulant, G. Boutis, E. Fortunato, S. Lloyd, R. Martinez, C. Negrevergne, M. Pravia, Y. Sharf, G. Teklemariam, Y. S. Weinstein, and W. H. Zurek. NMR based quantum information processing: achievements and prospects. *Fortschr. Phys.*, **48**, 875–907, 2000.

[CPM$^+$98] D. G. Cory, M. D. Price, W. Maas, E. Knill, R. Laflamme, W. H. Zurek, T. F. Havel, and S. S. Somaroo. Experimental quantum error correction. *Phys. Rev. Lett.*, **81**, 2152–2155, 1998.

[CVZ$^+$98] I. L. Chuang, L. M. K. Vandersypen, X. Zhou, D. W. Leung, and S. Lloyd. Experimental realization of a quantum algorithm. *Nature*, **393**, 143–146, 1998.

[CW90] D. Coppersmith and S. Winograd. Multiplication via arithmetic progressions. *Journal of Symbolic Computation*, **9**, 251–280, 1990.

[DC00] W. Dür and J. I. Cirac. Classification of multiqubit mixed states: Separability and distillability properties. *Phys. Rev. A*, **61**, 042314, 2000.

[Deu89] D. Deutsch. Quantum computational networks. *Proc. R. Soc. Lond. A*, **425**, 73, 1989.

[dHSL98] K. Życzkowski, P. Horodecki, A. Sanpera, and M. Lewenstein. Volume of the set of separable states. *Phys. Rev. A*, **58**, 883–892, 1998.

[DNBT02] J. L. Dodd, M. A. Nielsen, M. J. Bremner, and R. T. Thew. Universal quantum computation and simulation using any entangling hamiltonian and local unitaries. *Phys. Rev. A*, **65**, 040301, 2002.

[Don03] J. J. Dongarra. Performance of various computers using standard linear equations software, 2003. http://www.netlib.org/benchmark/performance.ps.

[DW97] J. J. Dongarra and R. C. Whaley. LAPACK Working Note 94, A User's Guide to the BLACS v1.1, 1997. http://www.netlib.org/blacs/lawn94.ps.

[EPR35] A. Einstein, B. Podolsky, and N. Rosen. Can quantum-mechanical description of physical reality be considered complete? *Phys. Rev.*, **47**, 777–780, 1935.

[Fit00] R. Fitzgerald. What really gives a quantum computer its power? *Physics Today*, pp. 20–22, January 2000.

[Fra85] J. D. Franson. Bell's theorem and delayed determinism. *Phys. Rev. D*, **31**, 2529–2532, 1985.

[GB03] L. Gurvits and H. Barnum. Separable balls around the maximally mixed multipartite quantum states. *arXive eprint quant-ph/0302102*, 2003.

[GBD+94] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine, a users' guide and tutorial for networked parallel computing.* MIT Press, Cambridge, MA, 1994. http://www.netlib.org/pvm3/book/pvm-book.html.

[GC97] N. Gershenfeld and I. L. Chuang. Bulk spin-resonance quantum computation. *Science*, **275**, 350–356, 1997.

[Gri95] D. J. Griffiths. *Introduction to Quantum Mechanics.* Prentice-Hall, Upper Saddle River, NJ, 1995.

[GRL+03] S. Gulde, M. Riebe, G. P. T. Lancaster, C. Becher, J. Eschner, H. Häffner, F. Schmidt-Kaler, I. L. Chuang, and R. Blatt. Implementing the Deutsch-Jozsa algorithm on an ion-trap quantum computer. *Nature*, **421**, 48–50, 2003.

[Gro96] L. K. Grover. Quantum mechanics helps in searching for a needle in a haystack. *Phys. Rev. Lett.*, **79**, 350–356, 1996.

[HBG00] P. Hübler, J. Bargon, and S. J. Glaser. Nuclear magnetic resonance quantum computing exploiting the pure spin state of *para* hydrogen. *Journal of Chemical Physics*, **113**, 2056–2059, 2000.

[HHH96] M. Horodecki, P. Horodecki, and R. Horodecki. Separability of mixed states: Necessary and sufficient conditions. *Phys. Lett. A*, **223**, 1–8, 1996.

[Hor97] P. Horodecki. Separability criterion and inseparable mixed states with positive partial transposition. *Phys. Lett. A*, **232**, 333–349, 1997.

[JM98] J. A. Jones and M. Mosca. Implementation of a quantum algorithm on a nuclear magnetic resonance quantum computer. *J. Chem. Phys.*, **109**, 1648–1653, 1998.

[JMH98] J. A. Jones, M. Mosca, and R. H. Hansen. Implementation of a quantum search algorithm on a quantum computer. *Nature*, **393**, 344–346, 1998.

[Jon00] J. A. Jones. NMR quantum computation: A critical evaluation. *Fort. der. Physik*, **48**, 909–924, 2000.

[Jon01] J. A. Jones. NMR quantum computation. *Progr. NMR Spectr.*, **38**, 325–360, 2001.

[KC98] E. Knill and I. L. Chuang. Effective pure states for bulk quantum computation. *Phys. Rev. A*, **57**, 3348–3363, 1998.

[LBF98] N. Linden, H. Barjat, and R. Freeman. An implementation of the Deutsch-Jozsa algorithm on a three-qubit NMR quantum computer. *Chem. Phys. Lett.*, **296**, 61–67, 1998.

[Li03] X. S. Li. SuperLU, 2003. http://crd.lbl.gov/~xiaoye/SuperLU.

[LKF99] N. Linden, E. Kupče, and R. Freeman. NMR quantum logic gates for homonuclear spin systems. *Chem. Phys. Lett.*, **311**, 321–327, 1999.

[Mat01] MATLAB Version 6.1.0.450 Release 12.1, 2001. http://www.mathworks.com/access/helpdesk/help/techdoc/matlab.shtml.

[MC02] N. C. Meniucci and C. M. Caves. Local realistic model for the dynamics of bulk-ensemble NMR information processing. *Phys. Rev. Lett.*, **88**, 167901, 2002.

[NC00] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press, Cambridge, MA, 2000.

[NDD$^+$02] M. A. Nielsen, C. M. Dawson, J. L. Dodd, A. Gilchirst, D. Mortimer, T. J. Osborne, M. J. Bremner, A. W. Harrow, and A. Hines. Quantum dynamics as a physical resource. *arXive eprint quant-ph/0208077*, 2002.

[Nhm03] National High Magnetic Field Laboratory (NHMFL), 2003. http://www.magnet.fsu.edu.

[Nie02] M. A. Nielsen. An introduction to majorization and its applications to quantum mechanics, 2002. http://www.qinfo.org/talks/2002/maj/book.ps.

[NKL98] M. A. Nielsen, E. Knill, and R. Laflamme. Complete quantum teleportation using nuclear magnetic resonance. *Nature*, **396**, 52–55, 1998.

[Pat03] G. Patz. A parallel environment for simulating quantum computation. Master's thesis, Massachusetts Institute of Technology, June 2003.

[Pea70] P. Pearle. Hidden variable example based upon data rejection. *Phys. Rev. D*, **2**, 1418–1425, 1970.

[Per95] A. Peres. *Quantum Theory: Concepts and Methods.* Kluwer, Dordrecht, 1995.

[Per96] A. Peres. Separability criterion for density matrices. *Phys. Rev. Lett.*, **77**, 1413–1415, 1996.

[Pop94] S. Popescu. Bell's inequalities versus teleportation: What is nonlocality? *Phys. Rev. Lett.*, **72**, 797, 1994.

[Pre98] J. Preskill. Lecture Notes for Physics 229: Quantum Information and Computation, 1998. http://www.theory.caltech.edu/~preskill/ph229.

[PWDC00] A. Petitet, R. C. Whaley, J. Dongarra, and A. Cleary. HPL - A portable implementation of the high-performance linpack benchmark for distributed-memory computers, 2000. http://www.netlib.org/benchmark/hpl.

[RKM$^+$01] M. A. Rowe, D. Kielpinski, V. Meyer, C. A. Sackett, W. M. Itano, C. Monroe, and D. J. Wineland. Experimental violation of a Bell's inequality with efficient detection. *Nature*, **409**, 791–794, 2001.

[Sak94] J. J. Sakurai. *Modern Quantum Mechanics: Revised Edition.* Addison Wesley Longman, Reading, MA, 1994.

[SC99] R. Schack and C. M. Caves. Classical model for bulk-ensemble NMR quantum computation. *Phys. Rev. A*, **60**, 4354–4362, 1999.

[SGDM03] J. K. Stockton, J. M. Geremia, A. C. Doherty, and H. Mabuchi. Characterizing the entanglement of symmetric many-particle spin-$\frac{1}{2}$ systems. *Phys. Rev. A*, **67**, 022112, 2003.

[Sho94] P. W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, 1994.

[Sho97] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Sci. Statist. Comput.*, **26**, 1484, 1997.

[SKK$^+$00] C.A. Sackett, D. Kielpinsky, B.E. King, C. Langer, V. Meyer, C.J. Myatt, M. Rowe, Q.A. Turchette, W.M. Itano, D.J. Wineland, and C. Monroe. Experimental entanglement of four particles. *Nature*, **404**, 256–258, 2000.

[Ste03] M. Steffen. *A Prototype Quantum Computer Using Nuclear Spins in Liquid Solution.* Ph.D. thesis, Stanford University, June 2003.

[SV99] L. J. Schulman and U. Vazirani. Scalable NMR quantum computation. *Proc. 31'st ACM STOC (Symp. Theory of Computing)*, pp. 322–329, 1999.

[Uhl70] A. Uhlmann. On the Shannon entropy and related functionals on convex sets. *Rep. Math. Phys.*, **1**, 147–159, 1970.

[Uhl71] A. Uhlmann. Sätze über dichtematrizen. *Wiss. Z. Karl-Marx-Univ. Leipzig*, **20**, 633–637, 1971.

[Uhl72] A. Uhlmann. Endlich-dimensionale dichtematrizen i. *Wiss. Z. Karl-Marx-Univ. Leipzig*, **21**, 421–452, 1972.

[Uhl73] A. Uhlmann. Endlich-dimensionale dichtematrizen i. *Wiss. Z. Karl-Marx-Univ. Leipzig*, **22**, 139–177, 1973.

[VABM01] F. Verstraete, K. Audenaert, De Bie, and De Moor. Maximally entangled mixed states of two qubits. *Phys. Rev. A*, **64**, 012316, 2001.

[VLS$^+$01] A. S. Verhulst, O. Liivak, M. H. Sherwood, H.-M. Vieth, and I. L. Chuang. Non-thermal nuclear magnetic resonance quantum computing using hyperpolarized xenon. *Appl. Phys. Lett.*, **79**, 2480–2482, 2001.

[VP98] V. Vedral and M. B. Plenio. Entanglement measures and purification procedures. *Phys. Rev. A*, **57**, 1619–1633, 1998.

[VSB$^+$01] L. M. K. Vandersypen, M. Steffen, G. Breyta, C. S. Yannoni, M. Sherwood, and I. L. Chuang. Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance. *Nature*, **414**, 883–887, 2001.

[VT99] G. Vidal and R. Tarrach. Robustness of entanglement. *Phys. Rev. A*, **59**, 141–155, 1999.

[VW02] G. Vidal and R. F. Warner. A computable measure of entanglement. *Phys. Rev. A*, **65**, 032314, 2002.

[VYC02] L. M. K. Vandersypen, C. S. Yannoni, and I. L. Chuang. Liquid state NMR quantum computing. In D. M. Grant and R. K. Harris, editors, *The encyclopedia of nuclear magnetic resonance, Advances in NMR*. John Wiley and Sons, West Sussex, England, 2002.

[WC01] A. Wong and N. Christensen. Creating Bell states and decoherence effects in a quantum-dot system. *Phys. Rev. A*, **63**, 062307, 2001.

[Wei02] E. W. Weisstein. *CRC Concise Encyclopedia of Mathematics, Second Edition.* CRC Press, Boca Raton, Florida, 2002.

[Wer89] R. F. Werner. Quantum states with Einstein-Podolsky-Rosen correlations admitting a hidden-variable model. *Phys. Rev. A*, **40**, 4277–4281, 1989.

[WJS$^+$98] G. Weihs, T. Jennewein, C. Simon, H. Weinfurter, and A. Zeilinger. Violation of Bell's inequality under strict Einstein locality conditions. *Phys. Rev. Lett.*, **81**, 5039–5043, 1998.

[YY99] F. Yamaguchi and Y. Yamamoto. Crystal lattice quantum computer. *App. Phys. A*, **68**, 1–8, 1999.

[ZAB02] A. L. Zook, B. B. Adhyaru, and C. R. Bowers. High capacity production of $> 65\%$ spin polarized xenon-129 for NMR spectroscopy and imaging. *Journal of Magnetic Resonance*, **159**, 172–182, 2002.