Alpen-Adria-Universität Klagenfurt

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND CYBERSECURITY
MSc Artificial Intelligence & Cybersecurity

LARGE PROJECT

Simulation of an Heisenberg model Hamiltonian for a three-particle system on IBM Quantum's 7-qubit Jakarta system

AUTHOR:
Francesco Pinzauti

ADVISOR:
Prof. Elisabeth Oswald

2021-2022 academic year

ABSTRACT

The aim of this work is to attempt a competition by IBM on an open problem related to quantum simulation. This writeup is divided in three chapters. In the first chapter we give a brief and quick overview at the field of quantum computation. This it is not meant to be by any means complete, and we remind to [Mic11] for further deepening. However this first chapter should give to the reader all the tools in order to be able to read this work.

In the second chapter we take a look at quantum simulation, what it is, how it is done, and how it is done in practice. Again, this is not mean to be complete at all, as we describe the only techniques that will be used in the challenge.

We finish with the third chapter, where we introduce the actual competition and the strategy we have adopted to solve it.

CONTENTS

1 FUNDAMENTAL CONCEPTS

Information is physical.

Rolf Landauer

The term *quantum computer* is synonymous with the quantum circuit model of computation. This chapter provides an introduction to the postulates of quantum computation and a general overlook of quantum circuits and their basic elements.

1.1 THE POSTULATES

We shall start with a general overview of the basic postulates of quantum mechanics. These postulates provide a connection between the physical world and the mathematical formalism of quantum mechanics upon quantum computation is built on.

Postulate 1. Associated to any isolated physical system is a Hilbert space $\mathcal{H}$ known as the *state space* of the system. The system is completely described by its *state vector* $|\psi(t)\rangle$, which is a unit vector in the system's state space.

Postulate 2. The time evolution of the state of a closed quantum system is described by a unitary operator. That is, for any evolution of the closed system there exists a unitary operator[1] $\widehat{U}(t_2, t_1)$ such that if the initial state of the system is $|\psi(t_1)\rangle$ then after the evolution the state of the system will be:

$$|\psi(t_2)\rangle = \hat{U}(t_2, t_1)\,|\psi(t_1)\rangle \quad \text{with} \quad \hat{U}^\dagger \hat{U} = \hat{\mathbb{I}}\,.$$

Postulate 3. Quantum measurements are described by a collection $\{\hat{M}_m\}$ of measurement operators. These are operators acting on the state space of the system being measured. The index m refers to the measurement outcomes that may occur in the experiment. If the state of the quantum system is $|\psi(t)\rangle$ immediately before the measurement then the probability that result m occurs is given by

$$p(m) = \langle\psi|\hat{M}_m^\dagger \hat{M}_m|\psi\rangle$$

1 Since $\widehat{U}(t_2, t_1)$ is a unitary operator, which is the generalization of the rotation operator to complex spaces, we may describe the time evolution of state vectors as rotations (not necessarily spatial) in Hilbert space.

and the state of the system after the measurement is

$$\frac{\hat{M}_m \left|\psi\right\rangle}{\left\langle\psi\right|\hat{M}_m^\dagger \hat{M}_m\left|\psi\right\rangle}\,.$$

The measurement operators satisfy the *completeness equation*

$$\sum_m \hat{M}_m^\dagger \hat{M}_m = \hat{\mathbb{1}}$$

which express the fact that, if the states are normalized (and that will be always required), probabilities sum to one:

$$\sum_m p(m) = \sum_m \left\langle\psi\right|\hat{M}_m^\dagger \hat{M}_m\left|\psi\right\rangle = \sum_m \left\langle\psi|\psi\right\rangle = 1\,. \tag{1.1}$$

Postulate 4. The state space of a composite physical system is the tensor product of the state spaces of the component physical systems. If we have systems numbered 1 through N:

$$\mathcal{H} = \bigotimes_{i=1}^{N} \mathcal{H}_i \quad \text{and} \quad \dim \mathcal{H} = \prod_{i=1}^{N} \dim \mathcal{H}_i\,.$$

1.2 QUANTUM BITS

The definition of qubit (quantum bit) immediately follows from postulate 1:

Definition 1.1. A *qubit* is a physical system Q whose Hilbert space $\mathcal{H}_Q$ has dimension $\dim \mathcal{H}_Q = 2$.

Because of postulate 1 and according to the definition of vector space we see that every linear combination of a state vector

$$\left|\psi\right\rangle = a\left|\alpha\right\rangle + b\left|\beta\right\rangle \quad \left|\alpha\right\rangle, \left|\beta\right\rangle \in \mathcal{H}_\psi \quad a, b \in \mathbb{C} \quad |a|^2 + |b|^2 = 1 \tag{1.2}$$

is still part of the state space and it still describes the physics of the system. (There is only a constraint: the state has to be normalized according to (1.1), such a rescaling is possible and will be assumed hereafter.)

Main difference between bits and qubits.

Here lies the main difference between bits and qubits: whereas in classical computation only 0 and 1 states are allowed, in quantum computation also superposition states are perfectly acceptable. What does a superposition state physically mean? If we measure for example (1.2) the probability of being in the state $\left|\alpha\right\rangle$ is $|a|^2$ and the probability of being in the state $\left|\beta\right\rangle$ is $|b|^2$.

According (again) to the first postulate the state of a qubit is a vector in a two-dimensional Hilbert space. Let us define its basis:

Definition 1.2. The orthonormal basis of the two-dimensional Hilbert space describing a qubit is called *computational basis* and it's composed of the states $\left|0\right\rangle$ and $\left|1\right\rangle$ known as *computational basis states*.

How a qubit is physically made? It can be a $1/2$ spin particle, an atomic system whose dynamics is described by two (non-degenerate) energy levels and so on. Whatever we choose to be our physical realization of the qubit we have a Hermitian operator associated with the observable chosen. The computational basis then will be composed by the eigenstates of the Hermitian operator associated with the observable[2], those state (whatever the operator is) will be labelled as $\{|0\rangle, |1\rangle\}$ according to definition 2.

Let us use, for example, a $1/2$ spin particle. We know that the Hermitian operator associated with spin is S_z or $\hat{\sigma}_z = \frac{2}{\hbar} S_z$ that fits our scope because it has two eigenstates and two non-degenerate eigenvalues:

An example of how a qubit can be physically implemented.

$$\begin{aligned} \hat{\sigma}_z |\chi_+\rangle &= |\chi_+\rangle \\ \hat{\sigma}_z |\chi_-\rangle &= -|\chi_-\rangle \, . \end{aligned} \tag{1.3}$$

These eigenstates (i.e. the spinors) span a two-dimensional Hilbert space and can be chosen as the computational basis.

1.2.1 Quantum register

The definition of quantum register, quantum analogue of the classical register, immediately follows from postulate 4:

Definition 1.3. A n size *quantum register* is a system QR with $\dim \mathcal{H}_{QR} = 2^n$.

In other words, a quantum register is a system comprising multiple qubits.

The simplest case is a system $N = 2$ with two qubits Q_1 and Q_2. If we define the basis of $\mathcal{H}_{Q_1}$ and $\mathcal{H}_{Q_2}$ as $\{|0\rangle_1, |1\rangle_1\}$ and $\{|0\rangle_2, |1\rangle_2\}$ one possible basis of H_{QR} is

Quantum register with two qubits.

$$\{|0\rangle_1 \otimes |0\rangle_2, |1\rangle_1 \otimes |0\rangle_2, |0\rangle_1 \otimes |1\rangle_2, |1\rangle_1 \otimes |1\rangle_2\} \tag{1.4}$$

and as expected we have $\dim \mathcal{H}_{QR} = 4$.

1.2.2 Entanglement

Consider two arbitrary quantum systems Q_1 and Q_2, with respective Hilbert spaces $\mathcal{H}_{Q_1}$ and $\mathcal{H}_{Q_2}$. The Hilbert space of the composite system is the tensor product:

$$\mathcal{H}_{Q_1} \otimes \mathcal{H}_{Q_2} \, ,$$

2 To every Hermitian operator Ω defined on a space $\mathcal{H}$ there exist (at least) a basis of the space $\mathcal{H}$ consisting of the orthonormal eigenvectors of the operator [Sha11, p. 36].

if the first system is in state $|\psi\rangle_{Q_1}$ and the second in state $|\psi\rangle_{Q_2}$ the state of the composite system is

$$|\psi\rangle_{Q_1} \otimes |\psi\rangle_{Q_2} .$$

States of the composite system that can be represented in this form are called *separable states* while

Definition 1.4. A composite system such that $|QR\rangle \neq \otimes_i |Q_i\rangle$ is an *entangled state*.

An example of a separable state.

If we consider two qubits:

$$\begin{aligned} |Q_1\rangle &= a\,|0\rangle_1 + b\,|1\rangle_1 \quad a, b \in \mathbb{C}, \quad |a|^2 + |b|^2 = 1 \\ |Q_2\rangle &= c\,|0\rangle_2 + d\,|1\rangle_2 \quad c, d \in \mathbb{C}, \quad |c|^2 + |d|^2 = 1 \end{aligned}$$

the overall state of the system is:

$$\begin{aligned} |QR\rangle = |Q_1\rangle \otimes |Q_2\rangle = ac\,|0\rangle_1 \otimes |0\rangle_2 \\ + bc\,|1\rangle_1 \otimes |0\rangle_2 + ad\,|0\rangle_1 \otimes |1\rangle_2 + bd\,|1\rangle_1 \otimes |1\rangle_2 \end{aligned}$$

that is a separable state.

An example of an entangled state.

If instead we have:

$$|\Phi^+\rangle = \frac{|0\rangle_1 \otimes |0\rangle_2 + |1\rangle_1 \otimes |1\rangle_2}{\sqrt{2}}$$

we immediately see that this is an entangled state[3] as there is no way of writing it as $|Q_1\rangle \otimes |Q_2\rangle$.

1.3 QUANTUM CIRCUITS

A *quantum circuit* is a set of elementary quantum operations, that is a model for quantum computation in which a computation is a sequence of quantum gates[4].

The basic blocks of a quantum circuit are quantum channels, single-qubit gates, two-qubit gates and the measurement operation which allow us to retrieve the result of the algorithm implemented in the circuit. We shall analyze each component in the following sections.

It follows from the second postulate that the dynamical evolution of the qubit due to the various elements of the circuit (gates and channels) is described by a unitary operator. Let us review some of its proprieties:

Theorem 1.1. *There always exist an operator $\hat{H}$ such that $\hat{U}(t) = e^{-i\hat{H}t/\hbar}$ with $\hat{H} = \hat{H}^\dagger$ the Hamiltonian describing the system [Sha11, p. 145].*

3 It is one of the Bell states, four specific maximally entangled quantum states of two qubits.

4 The anaolgy with the classical gates is only conceptual, unlike classical computer data is not sent into the gate but the gate is instead a operation applied to all the machine.

Note that the the Hamiltonian being Hermitian guarantees the unitarity of our operator. After the evolution the norm of the input is conserved (which implies that the state is still valid according to postulate 1) since

Theorem 1.2. *A unitary operator preserves the inner product.*

Proof.

$$\langle \hat{U}(t)\psi|\hat{U}(t)\psi\rangle = \langle\psi|\hat{U}^{\dagger}(t)\hat{U}(t)|\psi\rangle = \langle\psi|\psi\rangle$$

□

Thanks to theorem 1 and postulate 2 we are able to understand the dynamical evolution of the qubit (or the quantum register) due to each gate (or channel)

State evolution in the circuit.

$$|Q\rangle_{\text{out}} \equiv \hat{U}(\tau)\,|Q\rangle_{\text{in}} = e^{-i\tau\hat{H}_Q/\hbar}\,|Q\rangle_{\text{in}} \tag{1.5}$$

where τ is the time which is physically necessary for the element of the circuit to complete its action, H_Q is the qubit (or quantum register) Hamiltonian and $|Q\rangle_{\text{in}}$, $|Q\rangle_{\text{out}}$ are the input and output.

Theorem 1.3. *For every element of the circuit (gate or transmission channel) acting as $|Q\rangle_{\text{out}} = \hat{U}\,|Q\rangle_{\text{in}}$ there exist $\hat{U}^{\dagger}$ such that $|Q\rangle_{\text{in}} = \hat{U}^{\dagger}\,|Q\rangle_{\text{out}}$.*

Proof. It immediately follows from the unitary of the operator $\hat{U}\hat{U}^{\dagger} = \hat{\mathbb{I}} \rightarrow \hat{U}^{-1} = \hat{U}^{\dagger}$ □

In other words, every operation executed by the circuit is *reversible* and operation with a different number of inputs and outputs (perfectly acceptable in classical computation, an example in 1.1) are not possible.

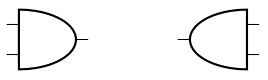


Figure 1.1: Classical inverse gate.

1.3.1 Quantum channels

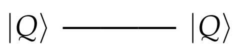


Figure 1.2: Ideal quantum channel

A *quantum channel* represents any process that realizes a point-to-point transfer of the quantum information embodied into the state of one qubit $|Q\rangle$. We can picture a quantum channel as a pipeline intended to carry quantum information[5]. We are going to describe

5 Time proceeds from left to right, wires represent qubits.

memoryless quantum channels (i.e. the output of a channel at a given time depends only upon the corresponding input and not any previous ones). A quantum channel should not alter the information but just transmit it, thus, from (1.5):

Definition 1.5. A *quantum channel* is an evolution operator with $\hat{H} = \hat{\mathbb{I}}$.

The evolution due to the quantum channel can be readily built:

$$|Q\rangle_{\text{out}} = \hat{U}(\tau_{\text{ch}})\,|Q\rangle_{\text{in}} = e^{-i\tau_{\text{ch}}\hat{\mathbb{I}}}\,|Q\rangle_{\text{in}} = e^{-i\tau_{\text{ch}}/\hbar}\,|Q\rangle_{\text{in}} \tag{1.6}$$

as a phase factor does not change the state of the qubit.[6]

Such condition can be obtained in two ways:

FLYING QUBIT When the object embodying the qubit can physically move. Its repositioning should be shielded enough to avoid any interaction that could result in $\hat{H} \neq \hat{\mathbb{I}}$.

STILL QUBIT If the object embodying the qubit cannot move an auxiliary medium is necessary, with the interaction in the medium properly designed so as to guarantee that condition (1.6) be fulfilled.

1.3.2 Single-qubit gates

Quantum gates are not meant to only transport information like quantum channels, their dynamical evolution is supposed to alter the state. Single-qubit gates G_1 perform an operation on one single qubit. G_1 represents the dynamical evolution of the qubit and can therefore be realized by properly designing an Hamiltonian according to (1.5):

$$|Q\rangle_{\text{out}} = G_1\,|Q\rangle_{\text{in}} = e^{-i\hat{H}_Q\tau_{G_1}/\hbar}\,|Q\rangle_{\text{in}}\;.$$

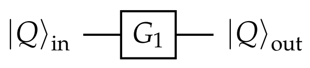


Figure 1.3: One-qubit gate

Once a computational basis has been chosen we can indicate the corresponding matrix representation for the basis vectors as the following

$$|0\rangle \to \begin{pmatrix}1\\0\end{pmatrix} \quad |1\rangle \to \begin{pmatrix}0\\1\end{pmatrix}\,. \tag{1.7}$$

Among the most important gates there are *Pauli X gate* and *Pauli Z gate*, their matrix representation according to 1.7 is[7]

$$X = \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix} \quad Z = \begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}\,.$$

6 $|\psi\rangle = |\psi'\rangle = |\psi\rangle\, e^{i\phi}$ since the probability of measuring a specific eigenvalue ω does not change: $p'(\omega) = \langle\psi e^{-i\phi}|\,\mathbb{P}_\omega\,|\psi e^{i\phi}\rangle = \langle\psi|\mathbb{P}_\omega|\psi\rangle = p(\omega) \quad \forall\omega\,.$

7 Their matrix representation equals Pauli matrices, hence the name.

The Pauli-X gate is the quantum equivalent of the NOT gate for classical computers. The Pauli-Z gate leaves the basis state $|0\rangle$ unchanged and maps $|1\rangle$ to $-|1\rangle$. Due to this nature, it is sometimes called phase-flip.

Hadamard gate

Finally the *Hadamard gate*. The matrix representation is the following:

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix} \tag{1.8}$$

Therefore from (1.7) we have the action of the gate on the basis states (and using a linear combination of those the action on any qubit):

$$\begin{cases} H|0\rangle = \dfrac{|0\rangle + |1\rangle}{\sqrt{2}} \\ H|1\rangle = \dfrac{|0\rangle - |1\rangle}{\sqrt{2}} \end{cases}$$

This gate is fundamental as it creates a superposition state (i.e. a measurement will have equal probabilities to result in $|1\rangle$ or $|0\rangle$) therefore is often the first step of quantum algorithms.

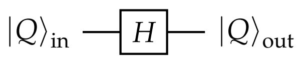


Figure 1.4: Hadamard gate

Let us take an example of a possible physical implementation. Suppose the qubit being implemented by a 1/2 spin particle according to (1.3). If the particle interact with a magnetic field $\mathbf{B} = B\mathbf{n}$ with $\mathbf{n} = (\frac{1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}})$ the Hamiltonian representing the interaction is

Hadamard gate realized trough a spin particle interacting in a magnetic field.

$$\hat{H} = g\mu_B \mathbf{B}\cdot\hat{\mathbf{S}} = \frac{gB\mu_B}{2}\mathbf{n}\cdot\hat{\sigma} = h\mathbf{n}\cdot\hat{\sigma} \quad \text{with} \quad h \equiv \frac{gB\mu_B}{2}$$

where $\hat{\mathbf{S}} = (\hat{S}^x, \hat{S}^y, \hat{S}^z)$ is the spin operator, $\hat{\sigma} = (\hat{\sigma}^x, \hat{\sigma}^y, \hat{\sigma}^z)$ is the Pauli vector, g is the g-factor and μ_B is the Bohr magneton.

We know that

Theorem 1.4.

$$e^{i\hat{\sigma}\cdot\mathbf{n}\theta} = \hat{\mathbb{I}}\cos\theta + i\hat{\sigma}\cdot\mathbf{n}\sin\theta$$

Proof.

$$\begin{aligned} e^{i\hat{\sigma}\cdot\mathbf{n}\theta} &= \sum_{m=0}^{\infty}\frac{(i\theta)^m}{m!}(\hat{\sigma}\cdot\mathbf{n})^m = \\ &= \hat{\mathbb{I}}\sum_{m=0}^{\infty}(-1)^m\frac{\theta^{2m}}{(2m)!} + i\hat{\sigma}\cdot\mathbf{n}\sum_{m=0}^{\infty}(-1)^m\frac{\theta^{2m+1}}{(2m+1)!} = \\ &= \hat{\mathbb{I}}\cos\theta + i\hat{\sigma}\cdot\mathbf{n}\sin\theta \end{aligned}$$

□

thus we can write the evolution operator as

$$\hat{U}(\tau_{G_1}) = e^{-ih\tau_{G_1}\hat{\sigma}\cdot\mathbf{n}} = \hat{\mathbb{I}}\cos(h\tau_{G_1}) - \frac{i}{\sqrt{2}}(\hat{\sigma}^x + \hat{\sigma}^z)\sin(h\tau_{G_1})$$

and its matrix representation is

$$U(\tau_{G_1}) = \begin{pmatrix} \cos(h\tau_{G_1}) - \frac{i}{\sqrt{2}}\sin(h\tau_{G_1}) & -\frac{i}{\sqrt{2}}\sin(h\tau_{G_1}) \\ -\frac{i}{\sqrt{2}}\sin(h\tau_{G_1}) & \cos(h\tau_{G_1}) + \frac{i}{\sqrt{2}}\sin(h\tau_{G_1}) \end{pmatrix}$$

that we can compare with (1.8) (we are trying to build an Hadamard gate) to obtain

$$\tau_{G_1} = \frac{\pi}{2h}.$$

Only with this constraint we have a Hadamard gate, thereafter the time that is physically necessary for the gate to complete its action is not just a label, but must be regarded as a genuine physical time, depending on fundamental constants and tunable Hamiltonian parameters.

1.3.3 Two-qubit gates

Two-qubit gates perform an operation on two qubits simultaneously, they are represented by a unitary operator acting on $\mathcal{H}_{QR} = \mathcal{H}_{Q_1} \otimes \mathcal{H}_{Q_2}$ such that, according to (1.5),

$$|QR\rangle_{\text{out}} = G_2 |QR\rangle_{\text{in}} = e^{-i\hat{H}_{QR}\tau_{G_2}/\hbar} |QR\rangle_{\text{in}}$$

where as always τ_{G_2} is the time that the gate takes to accomplish its task and H_{QR} it is the quantum register Hamiltonian. Note that if the dynamical evolution of the quantum register is of course unitary, the evolution of the two qubits is not. We can readily verify that the matrix representation of those operators is a 4x4 matrix as they act on a four-dimensional Hilbert space.

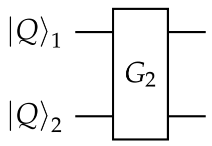


Figure 1.5: Two-qubit gate

Controlled gates

Suppose $\hat{U}$ is an arbitrary single-qubit unitary operation. A *controlled-U* operation is a two-qubit operation with a control and a target qubit. If the control qubit is set then $\hat{U}$ is applied to the target qubit, otherwise the target qubit is left alone.

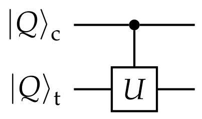


Figure 1.6: Controlled gate

Control-U gates that can convert a separable state to an entangled one are called *entangling gates*. Let us take an example where we consider the following quantum register

$$|QR\rangle_{\text{in}} = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)_c \otimes |\psi\rangle_t$$

Entangling gates

with $|\psi\rangle$ being the target state and the subscript c indicating the controlled one. If we apply the Control-U gate we have

$$|QR\rangle_{\text{out}} = \text{CU}\,|QR\rangle_{\text{in}} = \frac{1}{\sqrt{2}}\left(|0\rangle \otimes |\psi\rangle_t + |1\rangle_c \otimes \hat{U}\,|\psi\rangle_t\right)$$

which is an entangled quantum register if $\langle\psi|\hat{U}|\psi\rangle \neq 0$.

C-NOT gates

Control-U gates with U being the NOT operation are said *C-NOT gates*.

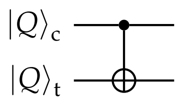


Figure 1.7: C-NOT gate

The action of the CNOT gate can be represented by the matrix

$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

and it follows the action of the gate on the basis states (1.7)

$$\begin{cases} \text{CNOT}\,|0\rangle_c \otimes |0\rangle_t = |0\rangle_c \otimes |0\rangle_t \\ \text{CNOT}\,|1\rangle_c \otimes |0\rangle_t = |1\rangle_c \otimes |1\rangle_t \\ \text{CNOT}\,|0\rangle_c \otimes |1\rangle_t = |0\rangle_c \otimes |1\rangle_t \\ \text{CNOT}\,|1\rangle_c \otimes |1\rangle_t = |1\rangle_c \otimes |0\rangle_t\ . \end{cases}$$

We see that the effect of the CNOT gate is flipping the second qubit (the target qubit) if and only if the first qubit (the control qubit) is $|1\rangle$.

Universal quantum gates

We know that in classical computation a small set of gates (e.g. NOR gates or alternatively NAND gates) can be used to compute an arbitrary classical function, those gates are called universal gates, in quantum computation

Definition 1.6. A set of gates is said to be universal for quantum computation if any unitary operation may be approximated to arbitrary accuracy by a quantum circuit involving only those gates.

Theorem 1.5. *Single-qubit and CNOT gates can be used to implement an arbitrary unitary operation on n qubits, and therefore are universal for quantum computation [Mic11, p. 191].*

1.3.4 Measurement

Theorem 1.6. *Without loss of generality, any unterminated quantum wires (qubits which are not measured) at the end of a quantum circuit may be assumed to be measured [Mic11, p. 187].*

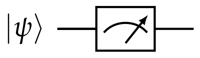


Figure 1.8: Meter

We have discussed all postulates except postulate 3 which is the root of this last element of the circuit. This postulate is intrinsically probabilistic (as quantum mechanics is) and one could argue that the implementation of deterministic algorithms in something probabilistic is a contradiction in terms. That is not the case, let us start defining the features of a quantum circuit in order to be able to execute an algorithm:

- The input and output of the algorithm should be in a separable state (i.e. not entangled).
- The gates of the circuit should not depend on the form of the input, the algorithm has to be a universal process independent of the input.
- The input and output of the algorithm should be written in terms of the computational basis[8] (the results of a measure are the eigenvalues associated to their eigenstates, those eigenstates represent the computational basis of the operator chosen).
- A type of measure of the observable associated with the operator that defines the computational basis should be built.

8 If the input and/or the output is a quantum register the computational basis is a tensor product like (1.4).

Those conditions are enough for a quantum circuit to be *consistent* that is with definite inputs and outputs and with an output that guarantees a certain outcome once measured.

1.4 COMPUTATIONAL COMPLEXITY

Let us define some notation.

Definition 1.7 (Big $\mathcal{O}$ notation)**.** Let f be a real or complex valued function and g a real valued function. Let both functions be defined on some unbounded subset of positive real numbers. One writes

$$f(x) = \mathcal{O}(g(x)) \quad \text{as} \quad x \to a$$

if

$$\limsup_{x \to a} \frac{|f(x)|}{g(x)} < \infty.$$

Definition 1.8 (Big Ω Knuth notation)**.** Let f be a real or complex valued function and g a real valued function. Let both functions be defined on some unbounded subset of positive real numbers. One writes

$$f(x) = \Omega(g(x)) \Longleftrightarrow g(x) = \mathcal{O}(f(x)).$$

We take for granted some basic definitions of classical computational complexity theory and we proceed defying

Definition 1.9. BPP is the class of languages that can be solved in polynomial time by probabilistic Turing machines with error probability bounded by $1/3$. Using standard boosting techniques the error probability can then be made exponentially small in k by iterating the algorithm k times.

Definition 1.10. PSPACE is the class of languages that may be solved on a Turing machine using a polynomial number of working bits, with no limitations on the amount of time that may be used by the machine.

Definition 1.11. BPQ is the class of languages that can be solved in polynomial time by quantum Turing machines with error probability bounded by $1/3$. The error probability can be made exponentially small as in **BPP**.

There are some evidence [Ben+97] that **BQP** $\neq$ **BPP** (i.e. polynomial-time quantum Turing machines are more powerful than polynomial-time probabilistic Turing machines). Since **BPP** is regarded as the class of all efficiently computable languages, this provides evidence that quantum computers could be inherently more powerful than classical computers in a model-independent way.

However exactly where **BQP** fits with respect to **P**, **NP** and **PSPACE** is as yet unknown. What is known is that quantum computers can solve all the problems in **P** efficiently and that there are no problems outside of **PSPACE** which they can solve efficiently. Therefore, **BQP** lies somewhere between **P** and **PSPACE**. It has been demonstrated that

$$\mathbf{P} \subseteq \mathbf{BPP} \subseteq \mathbf{BQP} \subseteq \mathbf{PSPACE}$$

and so proving that **BPP** $\subset$ **BQP** would definitively establish that **P** $\subset$ **PSPACE**, solving a major outstanding problem in computer science.

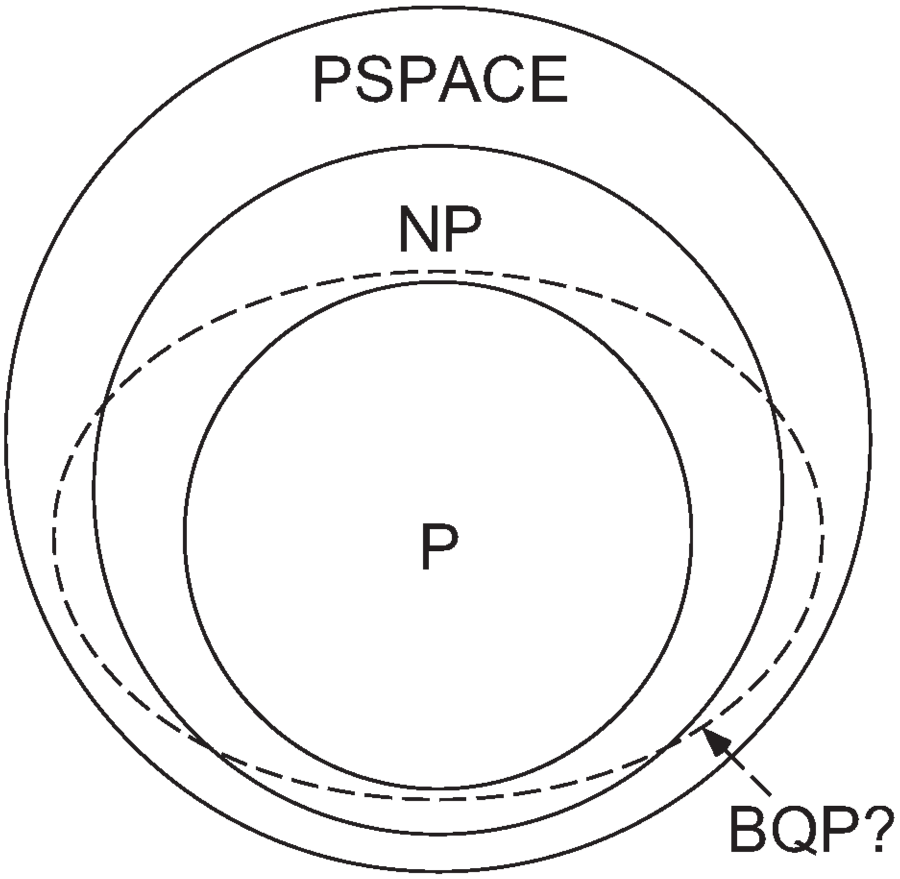


Figure 1.9: The relationship between classical and quantum complexity classes.

Another open computational problem is **NP** $\overset{?}{\subseteq}$ **BQP**. We know that we can adopt a quantum search algorithm to solve any **NP**-complete[9] problem.

We can demonstrate that, because we know that the quantum search algorithm is optimal, this means that it is not possible to search an N item search space in $\mathcal{O}(\log_2 N)$. If such an algorithm had existed, it would have allowed us to solve **NP**-complete problems efficiently

9 We can demonstrate that solving **NP**-complete problems in polynomial times would mean solving every other **NP** problem in polynomial time.

on a quantum computer by transforming problems in **NP** into search problems. This strongly suggests that **NP** is not included in **BQP**.

This does not rule out the possibility that **NP** $\subset$ **BQP**. What this result do establish is that there is no search-based method for attacking **NP**-complete problems. However we note that it is widely believed that the search space of **NP**-complete problems has no structure and that the best possible method for solving such problems is to adopt an unstructured search method. Furthermore for many problems in the **NP**-complete class there is no better method known than exhaustive search of all the possible solution. If one takes this point of view this indicates that **BQP** does not contain **NP**-complete problems.

2 QUANTUM SIMULATION

Can physics be simulated by a universal computer? [...] the physical world is quantum mechanical, and therefore the proper problem is the simulation of quantum physics [...] the full description of quantum mechanics for a large system with R particles [...] has too many variables, it cannot be simulated with a normal computer with a number of elements proportional to R [. . . but it can be simulated with] quantum computer elements. [...] Can a quantum system be probabilistically simulated by a classical (probabilistic, I'd assume) universal computer? [...] If you take the computer to be the classical kind I've described so far [..] the answer is certainly, No!

Richard P. Feynman (1982)

Quantum simulation is the area of quantum computation which seems, at the time of writing, to provide the most useful and efficient applications. In this chapter we are going to present the main and most basic tools of quantum simulation, focusing on a specific techniques: trotterization.

2.1 TROTTERIZATION

Each physical system is described by its Hamiltonian. As discussed in chapter 1 we know that the evolution of a quantum system is governed by the Schrödinger equation [Sha11]:

$$i\frac{d}{dt}\left|\psi(t)\right\rangle = H\left|\psi(t)\right\rangle \tag{2.9}$$

Our goal is to find the solution to this equation in a reasonable time and with reasonable accuracy.

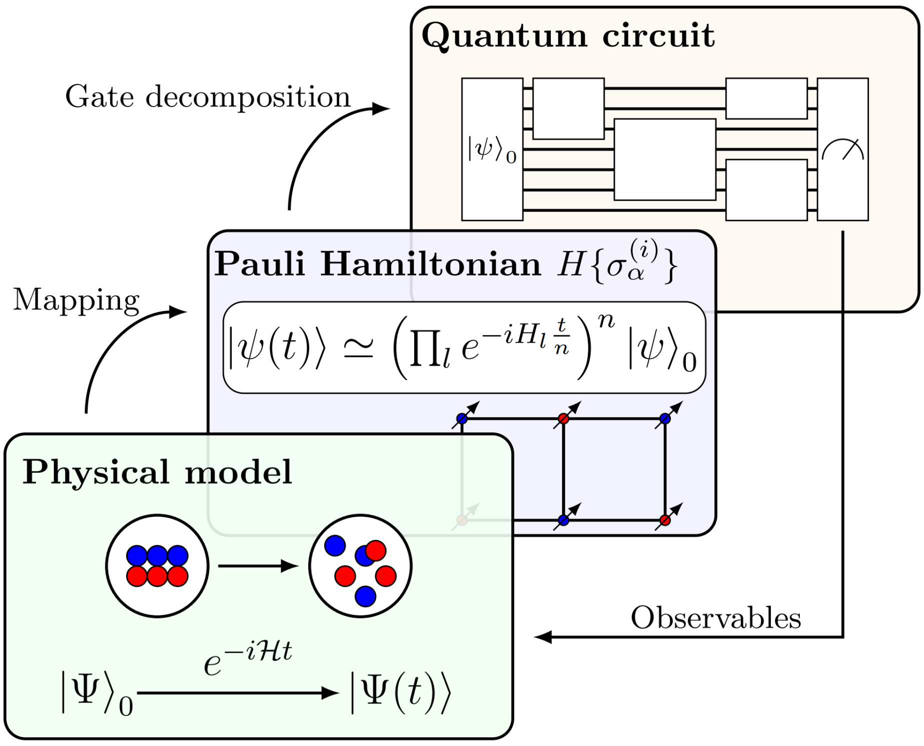


Figure 2.1: Quantum simulation

When dealing with real particles in space this reduces to:[1]

$$i\frac{\partial}{\partial t}\psi(x) = \left[-\frac{1}{2m}\frac{\partial^2}{\partial x^2} + V(x)\right]\psi(x) \tag{2.10}$$

For one qubit evolving according to this equation a system of two differential equations must be solved which results in 2^n equations for n qubits. This is incredibility hard to compute on a classical machine, that is why quantum computers are so crucial in this field.

We know the solution of equation 2.9 for a time independent H:[2]

$$|\psi(t)\rangle = e^{-iHt}|\psi(0)\rangle \tag{2.11}$$

H is usually extremely difficult to exponentiate and in order to be executed on a quantum computer the exponential should be rewritten in terms of its native gates[3].

But, what does it mean to take e to the power of an operator H?

1 What we are doing here is just rewriting the Hamiltonian H as the sum of kinetic and potential energy, i.e. $H = T + V$ and projecting the equation on the x basis, using the conventional representation $\langle x|\psi\rangle = \psi(x)$. This results in $V = V(x)$ and $T = -\frac{1}{2m}\frac{\partial^2}{\partial x^2}$. A complete explanation of the projection is outside the scope of this work.

2 We are going to deal only and exclusively with time-independent Hamiltonians.

3 This is not entirely true, e^{-iHt} should be rewritten in terms of quantum computation gates, but as long as our machine has a set of universal quantum gates implemented any quantum gate can be simulated, even if not native.

Working with the the matrix representation of H, the time evolution $U_H(t) = e^{-iHt}$ can be numerically and algebraically evaluated using a taylor series expansion ($e^x = 1 + x + x^2/2! + x^3/3! + ...$) . This is often where computers step in. Although, single Pauli operators (e.g. $H = \sigma_x$) have a nice algebraic result. On a computer, many different software packages have methods for computing e^A where A is a matrix (e.g. scipy or opflow).

We do need to be careful when the matrix that is being exponentiated is a sum of operators that do not commute[Sha11]:

$$[A, B] = AB - BA \neq 0 \tag{2.12}$$

For example, if $H = \sigma_x + \sigma_z$, we need to take care in working with $e^{it(\sigma_x+\sigma_z)}$ since the Pauli operatos σ_x and σ_z do not commute $[\sigma_x, \sigma_z] = \sigma_x\sigma_z - \sigma_z\sigma_x = -2i\sigma_y \neq 0$.

If we want to apply $e^{it(\sigma_x+\sigma_z)}$ on a quantum computer, however, we will want to decompose the sum in the exponential into a product of exponentials. Each product can then be implemented on the quantum computer as a single or two-qubit gate.

Continuing the example, σ_x and σ_z do not commute ($[\sigma_x, \sigma_z] \neq 0$), so we cannot simply decompose the exponential into a product of exponentials $e^{-it(\sigma_x+\sigma_z)} \neq e^{-it\sigma_x}e^{-it\sigma_z}$. This is where approximation methods come in such as Trotterization (more on that in the next section).

2.1.1 Time evolution

The problem of Hamiltonian simulation is thus stated as follows: Given a Hamiltonian H and an evolution time t, output a sequence of computational gates that implement

$$U = e^{-iHt} \tag{2.13}$$

.

This problem is meaningful because simulating the dynamics of a quantum system is an essential problem in quantum physics and quantum chemistry. As we have seen the Hamiltonian simulation problem can be solved with an exponential number of gates on a classical computer and now wee se that it requires only a polynomial number on a quantum computer, which is a huge improvement.

2.1.2 Lie product formula

In order to decompose our time evolution operator U into a sequence of gates, Lie product formula plays a crucial role.

This is because in most physical systems, the Hamiltonian can be written as a sum over many local interactions. Specifically, for a system of n particles,

$$H = \sum_{k=1}^{L} H_k, \tag{2.14}$$

where each H_k acts on at most a constant c number of systems, and L is a polynomial in n. [Sha11]

We can then rewrite the exponent of 2.13, which we want to decompose as a sequence of one and two-qubit gates, and then apply:

Theorem 2.1. *Let A and B be Hermitian operators. Then for any real t*

$$\lim_{n\to\infty} \left(e^{iAt/n}e^{iBt/n}\right)^n = e^{i(A+B)t} \tag{2.15}$$

[Mic11]

Note that this is true even if A and B do not commute.

Proof. By definition,

$$e^{iAt/n} = I + \frac{1}{n}iAt + O\left(\frac{1}{n^2}\right)$$

and thus

$$e^{iAt/n}e^{iBt/n} = I + \frac{1}{n}i(A+B)t + O\left(\frac{1}{n^2}\right)$$

Taking products of these gives us

$$\left(e^{iAt/n}e^{iBt/n}\right)^n = I + \sum_{k=1}^{n} \left(\begin{array}{c} n \\ k \end{array}\right) \frac{1}{n^k}[i(A+B)t]^k + O\left(\frac{1}{n}\right)$$

and since $\left(\begin{array}{c} n \\ k \end{array}\right) \frac{1}{n^k} = \left(1 + O\left(\frac{1}{n}\right)\right)/k!$, this gives

$$\lim_{n\to\infty} \left(e^{iAt/n}e^{iBt/n}\right)^n = \lim_{n\to\infty} \sum_{k=0}^{n} \frac{(i(A+B)t)^k}{k!}\left(1 + O\left(\frac{1}{n}\right)\right) + O\left(\frac{1}{n}\right) = e^{i(A+B)t}$$

. □

Since the limit of this formula is infinite, we have to truncate the series when implementing this formula on a quantum computer. The truncation introduces error in the simulation that we can bound by a maximum simulation error ϵ. such that

$$||e^{-iHT} - U|| \leq \epsilon$$

.

This truncation is known as *Trotterization* and it's the core of the simulation of non-commuting Hamiltonians on quantum computers.

THE ALGORITHM Let us summarize the complete algorithm for quantum simulation [Mic11]:

INPUTS The inputs of the quantum computer are:

1. A Hamiltonian $H = \sum_k H_k$ acting on an N-dimensional system, where each H_k acts on a small subsystem of size independent of N.
2. An initial state $|\psi_0\rangle$, of the system at $t = 0$.
3. A positive, non-zero accuracy δ.
4. A time t_f at which the evolved state is desired.

OUTPUTS A state $|\tilde{\psi}(t_f)\rangle$ such that $\left|\left\langle \tilde{\psi}(t_f) \left| e^{-iHt_f} \right| \psi_0 \right\rangle\right|^2 \geq 1 - \delta$.

RUNTIME $O(\text{poly}(1/\delta))$ operations.

PROCEDURE Choose a representation such that the state $|\tilde{\psi}\rangle$ of $n = \text{poly}(\log N)$ qubits approximates the system and the operators $e^{-iH_k \Delta t}$ have efficient quantum circuit approximations. Select an approximation method and Δt such that the expected error is acceptable (and $j\Delta t = t_f$ for an integer j), construct the corresponding quantum circuit $U_{\Delta t}$ for the iterative step, and do:

1. $|\tilde{\psi}_0\rangle \leftarrow |\psi_0\rangle \,; j = 0$ initialize state
2. $\rightarrow |\tilde{\psi}_{j+1}\rangle = U_{\Delta t} |\tilde{\psi}_j\rangle$ iterative update
3. $\rightarrow j = j + 1$; goto 2 until $j\Delta t \geq t_f$ loop
4. $\rightarrow |\tilde{\psi}(t_f)\rangle = |\tilde{\psi}_j\rangle$ final result

2.2 STATE TOMOGRAPHY

How we check if the result of our simulation is correct? State tomography is a method for determining the quantum state of a qubit, or qubits, even if the state is in a superposition or entangled. Repeatedly measuring a prepared quantum state may not be enough to determine the full state, in state tomography, a quantum circuit is repeated with measurements done in different bases to exhaustively determine the full quantum state (including any phase information). IBM is using this technique to determine the full quantum state after the quantum simulation. That state is then compared to the exact expected result to compute a fidelity. Although this fidelity only gives information on how well the quantum simulation produces one particular state, it's a more lightweight approach than a full process tomography calculation. In short, a high fidelity measured by state tomography doesn't guarente a high fidelity quantum simulation, but a low fidelity state tomography does imply a low fidelity quantum simulation.

In our problem the tomography is done on qubit 1,3,5 (more on that in chapter 3). The result is a number between 0 and 1 with 1 meaning a perfect result. Error mitigation can be applied before the state tomography according to the rules. [doc22a]

2.3 STATE OF THE ART

Before introducing the state of the art let us discuss a bit the physical realization of qubits.

2.3.1 Different qubit implementations

At the time of writing there are currently two technologies for implementing qubits:

- Trapped ions.
- Superconducting circuit.

Let us focus on the type of qubit used in the machine we are going to use:

SUPERCONDUCTING QUBITS To create a solid-state qubit, like any other kind of qubit, we need to isolate a two-level quantum system. To date, efforts to make solid-state qubits have focused on superconductors and semiconductors. While interesting results have been obtained with two semiconductor approaches, quantum dots and single-donor systems, the superconducting approach is currently the most advanced.

To maintain coherence it is essential to keep electron-electron interactions, and also interactions between electrons and other degrees of freedom (such as phonons in the solid), under control. Superconductors have the advantage in this regard because the electrons condense into Cooper pairs that form a single superfluid. This superfluid is able to move through the metal lattice without any resistance (i.e. without interactions) because it takes a certain amount of energy, known as the energy gap, to break up the Cooper pairs.

In aluminum, a popular material for making superconducting quantum circuits, the energy gap corresponds to a frequency of 90 GHz at a temperature of 20 mK. This gap is an order of magnitude greater than the typical energy difference between the two levels in a superconducting qubit, which means that we can “drive” the qubit without breaking up the Cooper pairs and jeopardizing the quantum coherence of the system.

The behavior of the electron superfluid is completely determined by a single quantum wavefunction. The amplitude of this wavefunction determines the number of Cooper pairs, while the value of the phase is related to the supercurrent and any magnetic field that is present. The amplitude and phase of the wavefunction are conjugate variables, that is, they are related by an uncertainty principle that means we cannot measure both of their values with arbitrary precision at the same time.

The two primary types of superconducting qubit, the charge qubit and the flux qubit, are directly related to these two variables: charge qubits are associated with the amplitude, while flux qubits are related to the phase. [Wor22]

State of the art

In figure 2.2 we report a summary of state-of-art experimental digital quantum simulations from [Tac+19]. Open circles represent results obtained on superconducting circuits quantum processors, while squares correspond to experimental quantum simulations on trapped ions processors. The color code corresponds to different target models being simulated[4]: two-spin Transverse field Ising model (TIM_2), two-spin XY (XY_2) and XYZ (XYZ_2) models, 3- and 6-spin many body interactions, two-spin Heisenberg model (Heis $_2$), and 2to 4 -mode Fermi Hubbard model $(\text{FH}_x$, with $x = 2, 3, 4)$.

As we are going to see in chapter 3 Jakarta (the device used in our experiments) is using superconducting Qubits, specifically the *Transmon*. The description of the Transmon is way beyond the scope of this work but you can find more info on that here [Qis22].

We are interested, as again we are going to see in chapter 3, to two-spin Heisenberg model (Heis $_2$). That means that the state of the art is around 0.7 in our case[5].

It is very interesting to see that accuracy seems to decrease with trotterization steps. Why? According to the formula of chapter 2 it should be exactly the opposite! We are speculating here as there are no known source proving this but our guess is that increasing the number of trotterization steps increase the circuit and thus increase noise in the simulation. This seems a reasonable explanation and it is the reason why we will limit ourself in the number of steps.

4 They are simply different Hamiltonian: their description would be redundant and it is purposely skipped, in chapter 3 we are going to describe our Hamiltonian of interest.
5 Why? Again wait for chapter 3, the problem limit us to four steps of trotterization.

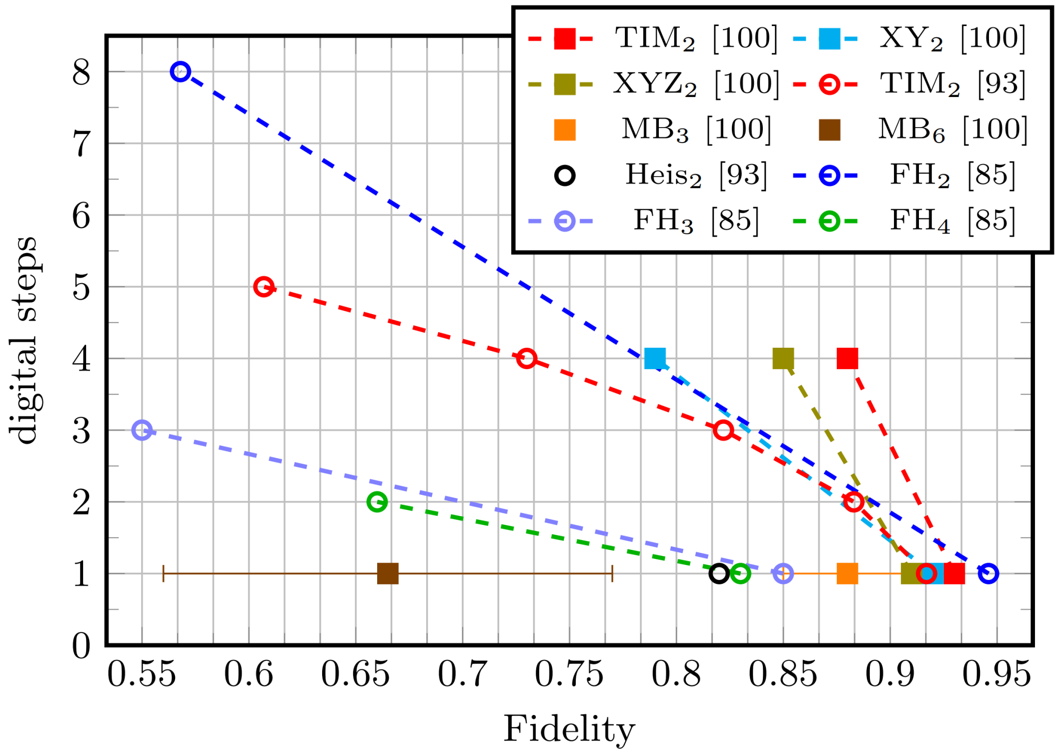


Figure 2.2: State of the art

3 THE CHALLENGE

Provando e riprovando.

Accademia del Cimento, Florence, 1675.

Here we start explaining the actual competition. First of all, what we have to do it is simply simulate an Hamiltonian as explained in chapter 2, nothing more. The Hamiltonian will be presented shortly in 3.1. There are however some restriction:

1. Trotterization must be used in the decomposition and the number of steps should be greater or equal to four.

2. The simulation will be tested and executed on a specific device, Jakarta, that we will describe in 3.3

Note that the access to the real device is tied to a long (very long!) queue and time of execution itself is quite long too. That's why IBM provide backend simulators, i.e. classical computer that simulates specifically quantum backends. Of course we will use sim˙noisy˙jakarta = QasmSimulator.from˙backend(provider.get˙backend('ibmq˙jakarta')) which is the simulator of Jakarta on a classical device. Of course simulations are not as near as precise as executions on the real device, however they can guide you towards the right direction, saving execution on the real device for more fine-grained refinements.

There are even backends simulators without noise, however they were not used as, after some testing, they seemed completely out of touch with the results of the real device.

3.1 THE XXX MODEL

The quantum Heisenberg model, developed by Werner Heisenberg, is a statistical mechanical model used in the study of critical points and phase transitions of magnetic systems, in which the spins of the magnetic systems are treated quantum mechanically. The model is based on the Hamiltonian [Wik22]:

$$H = -1/2 \sum_{j=1}^{N} (J_x \sigma_x^j \sigma_x^{j+1} + J_y \sigma_y^j \sigma_y^{j+1} + J_z \sigma_z^j \sigma_z^{j+1}) \tag{3.16}$$

where J is the coupling constant. We will use a simplified version. This version of the general Heisenberg spin model is called XXX because the same J value multiplies each pair of Pauli operators.

$$H_{\text{Heis}} = \sum_{\langle ij \rangle}^{N} J \left(\sigma_x^{(i)} \sigma_x^{(j)} + \sigma_y^{(i)} \sigma_y^{(j)} + \sigma_z^{(i)} \sigma_z^{(j)} \right). \tag{3.17}$$

N is the number of spin-1/2 particles in model.The i and j superscripts label which qubit they act on. For example, $\sigma_x^{(1)}$ would be the σ_x operator acting on only qubit 1 (which is the 2nd qubit since indexing starts at 0). The sum notation $\langle ij \rangle$ means the sum is over nearest neighbors (only qubits next to each other interact), and J is the interaction strength, which we will set $J = 1$.

Of course $\sigma_x, \sigma_y, \sigma_z$ are the known pauli matrices:

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

$$\sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$

$$\sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

We will work with the explicit case of $N = 3$ with the 3 spins arranged in a line. Written out fully, the Hamiltonian is

$$\begin{aligned} H_{\text{Heis3}} = {} & \sigma_x^{(0)} \sigma_x^{(1)} + \sigma_x^{(1)} \sigma_x^{(2)} + \sigma_y^{(0)} \sigma_y^{(1)} \\ & + \sigma_y^{(1)} \sigma_y^{(2)} + \sigma_z^{(0)} \sigma_z^{(1)} + \sigma_z^{(1)} \sigma_z^{(2)}. \end{aligned} \tag{3.18}$$

Now that we have a Hamiltonian (H_{Heis3}), we can use it to determine how the quantum system of 3 spin-1/2 particles changes in time.

To compute the matrix representation of H_{Heis3}, we are actually missing some pieces namely the identity operator I and the tensor product $\otimes$ symbol. They are both often left out in when writing a Hamiltonian, but they are implied to be there.

Writing out the full H_{Heis3} including the identity operators and tensor product symbols we obtain:

$$\begin{aligned} H_{\text{Heis3}} = {} & \sigma_x^{(0)} \otimes \sigma_x^{(1)} \otimes I^{(2)} + I^{(0)} \otimes \sigma_x^{(1)} \otimes \sigma_x^{(2)} \\ & + \sigma_y^{(0)} \otimes \sigma_y^{(1)} \otimes I^{(2)} + I^{(0)} \otimes \sigma_y^{(1)} \otimes \sigma_y^{(2)} \\ & + I^{(0)} \otimes \sigma_z^{(0)} \otimes \sigma_z^{(1)} + I^{(0)} \otimes \sigma_z^{(1)} \otimes \sigma_z^{(2)}. \end{aligned} \tag{3.19}$$

Knowing the Hamiltonian, we can determine how quantum states of that system evolve in time by solving the Schrödinger equations

$$i\hbar \frac{d}{dt} |\psi(t)\rangle = H |\psi(t)\rangle \tag{3.20}$$

For simplicity, let's set $\hbar = 1$. We know that the Hamiltonian H_{heis3} does not change in time, so the solution to the Schrödinger equation is an exponential of the Hamiltonian operator

$$U_{\text{Heis3}}(t) = e^{-itH_{\text{Heis3}}} = \exp\left(-itH_{\text{Heis3}}\right) \tag{3.21}$$

$$U_{\text{Heis3}}(t) = \exp\left[-it\sum_{\langle ij\rangle}^{N=3}\left(\sigma_x^{(i)}\sigma_x^{(j)} + \sigma_y^{(i)}\sigma_y^{(j)} + \sigma_z^{(i)}\sigma_z^{(j)}\right)\right] \tag{3.22}$$

$$U_{\text{Heis3}}(t) = \exp\left[-it\left(\sigma_x^{(0)}\sigma_x^{(1)} + \sigma_x^{(1)}\sigma_x^{(2)} + \sigma_y^{(0)}\sigma_y^{(1)} + \sigma_y^{(1)}\sigma_y^{(2)} + \sigma_z^{(0)}\sigma_z^{(1)} + \sigma_z^{(1)}\sigma_z^{(2)}\right)\right] \tag{3.23}$$

Now that we have the time evolution operator $U_{\text{Heis3}}(t)$, we can simulate changes in a state of the system ($|\psi(t)\rangle$) over time $|\psi(t)\rangle = U_{\text{Heis3}}(t)|\psi(t=0)\rangle$.

Our goal will be to decompose $U_{\text{Heis3}}(t)$ into one and two-qubit gates executable in the Jakarta device.

3.2 QISKIT PULSE AND NATIVE GATES

Something that we have still not explained is how to implement our decomposition in the machine. First of all, our code is written in Qiskit, a Python SDK, that we will describe better in appendix A. However tha challenge give us two possibilities:

1. Using native gates.
2. Using Qiskit Pulse.

In the first case, we simply call the native gates with the predefined commands[1] and so we are forced to limit our decomposition to the available gates.

Qiskit Pulse on the other ands offers low-level control of a device's qubits. Pulse allows users to program the physical interactions happening on the superconducting chip. This can be a powerful tool for streamlining circuits, crafting new types of gates, getting higher fidelity readout, and more.

We are going to use the first strategy, as it is the most straightforward to implement for a single person working on the project.

3.3 JAKARTA

Jakarta is a 7-qubit machine, with a Falcon r5.11H processor.

1 Of course we can change parameters, phase and so on.

Why 7 qubit is our Hamiltonian has only tree states? The other four states can be used as ancilla qubits in the decomposition and/or to reduce noise in noise reduction techniques. State Tomography at the end will be done only on qubit 1,3,5 (and you can't change which qubits will me measured) while other qubits will not be measured and so they are free to use as one like.

GATES We can use the native gates of the machine, the following set of gates are universal, therefore any gate can be implemented even without Qiskit Pulse, however there could be a significant improvement using only native gates[2]. Here is the list:

ID: Identity

CNOT: CNOT Gate

Z: Z Gate

SX: Pauli-X Gate Squared

X: Pauli-X Gate

In figure 3.1 and figure 3.2 we see the full specifications of Jakarta. Specifically in figure 3.2 we see the various errors in the native gates. Note that those errors fluctuate quite significantly over time.

In figure 3.3, figure 3.4 and figure 3.5 we see the different errors in the native gates divided per qubit.

Initially it was attempted to exploit those different errors in order to create a decomposition that uses as less as possibile gates on qubit with high errors on those gates. However, over time, this strategy has not proved to be successful, especially for the already mentioned high fluctuation of errors during time.

3.4 THE SOLUTION

We want to decompose

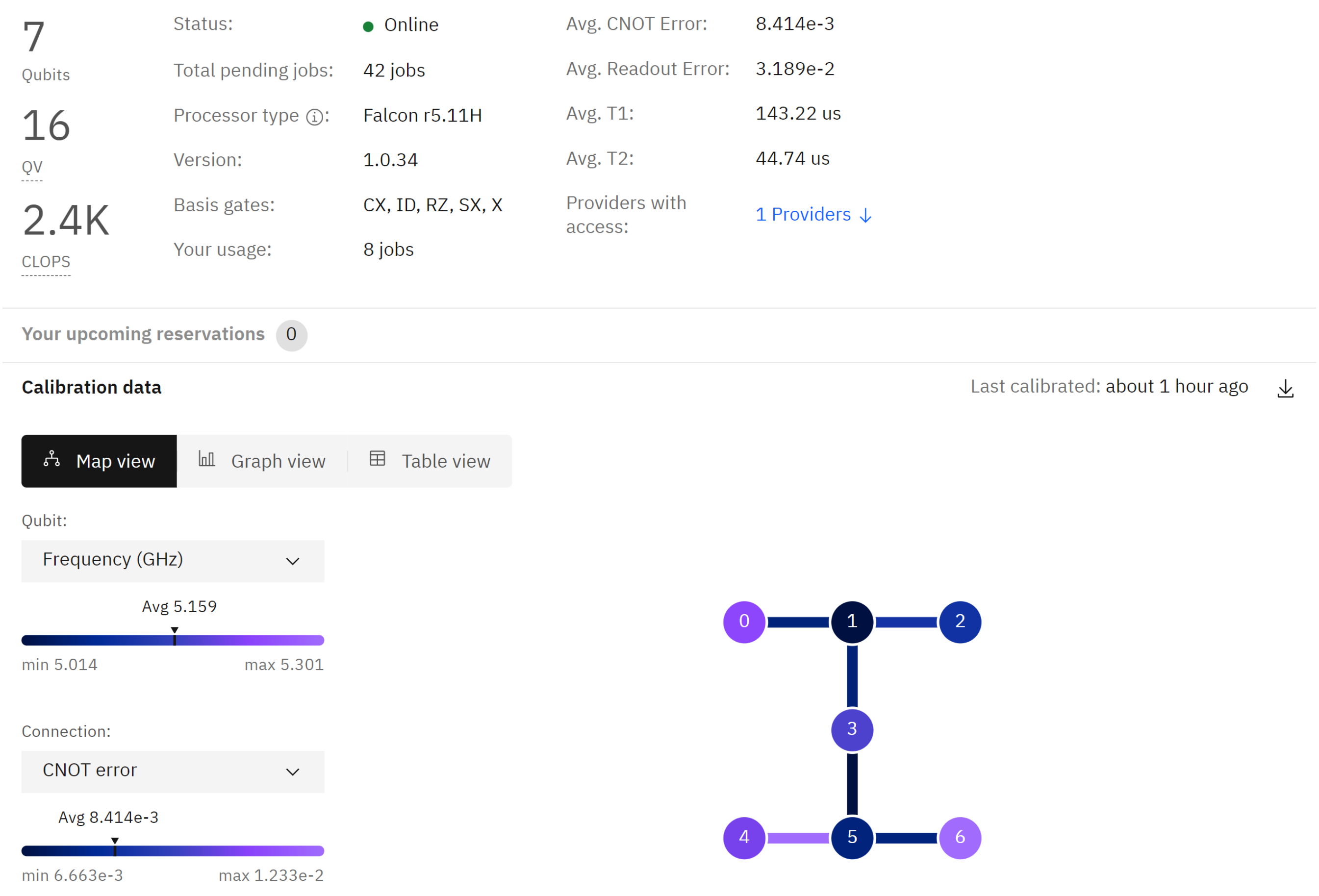
$$U_{\text{Heis3}}(t) = \exp\left[-it \sum_{\langle ij \rangle}^{N=3} \left(\sigma_x^{(i)}\sigma_x^{(j)} + \sigma_y^{(i)}\sigma_y^{(j)} + \sigma_z^{(i)}\sigma_z^{(j)}\right)\right] \tag{3.24}$$

into single and two-qubit gates.

Since the Pauli operators do not commute with each other [Sha11] the exponential $U_{\text{Heis3}}(t)$ cannot be split into a product of simpler exponentials.

However, we can approximate $U_{\text{Heis3}}(t)$ as a product of simpler exponentials through Trotterization. Consider a subsystem of 2 spin-1/2 particles within the larger 3 spin system. The Hamiltonian on spins

2 e.g. avoiding overlapping of errors of the native gates



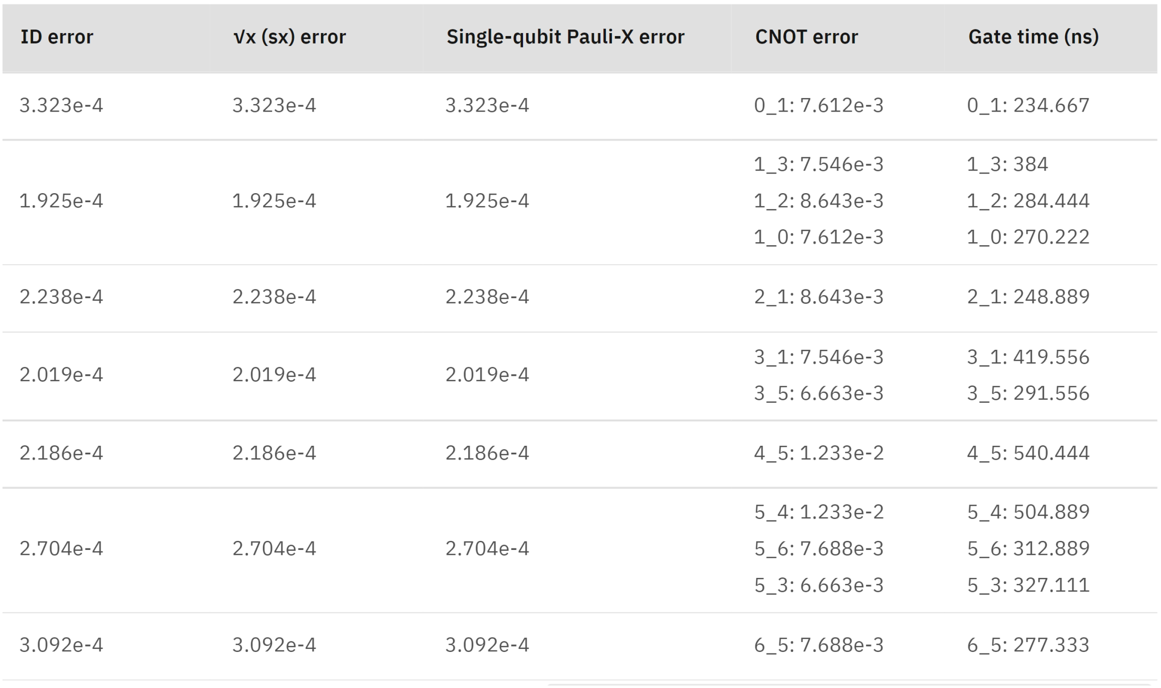
7 Qubits	Status:	● Online	Avg. CNOT Error:	8.414e-3
16 QV	Total pending jobs:	42 jobs	Avg. Readout Error:	3.189e-2
2.4K CLOPS	Processor type ⓘ:	Falcon r5.11H	Avg. T1:	143.22 us
	Version:	1.0.34	Avg. T2:	44.74 us
	Basis gates:	CX, ID, RZ, SX, X	Providers with access:	1 Providers ↓
	Your usage:	8 jobs		

Your upcoming reservations 0

Calibration data

Last calibrated: about 1 hour ago

Figure 3.1: The machine



ID error	√x (sx) error	Single-qubit Pauli-X error	CNOT error	Gate time (ns)
3.323e-4	3.323e-4	3.323e-4	0_1: 7.612e-3	0_1: 234.667
1.925e-4	1.925e-4	1.925e-4	1_3: 7.546e-3 1_2: 8.643e-3 1_0: 7.612e-3	1_3: 384 1_2: 284.444 1_0: 270.222
2.238e-4	2.238e-4	2.238e-4	2_1: 8.643e-3	2_1: 248.889
2.019e-4	2.019e-4	2.019e-4	3_1: 7.546e-3 3_5: 6.663e-3	3_1: 419.556 3_5: 291.556
2.186e-4	2.186e-4	2.186e-4	4_5: 1.233e-2	4_5: 540.444
2.704e-4	2.704e-4	2.704e-4	5_4: 1.233e-2 5_6: 7.688e-3 5_3: 6.663e-3	5_4: 504.889 5_6: 312.889 5_3: 327.111
3.092e-4	3.092e-4	3.092e-4	6_5: 7.688e-3	6_5: 277.333

Figure 3.2: Gates and errors

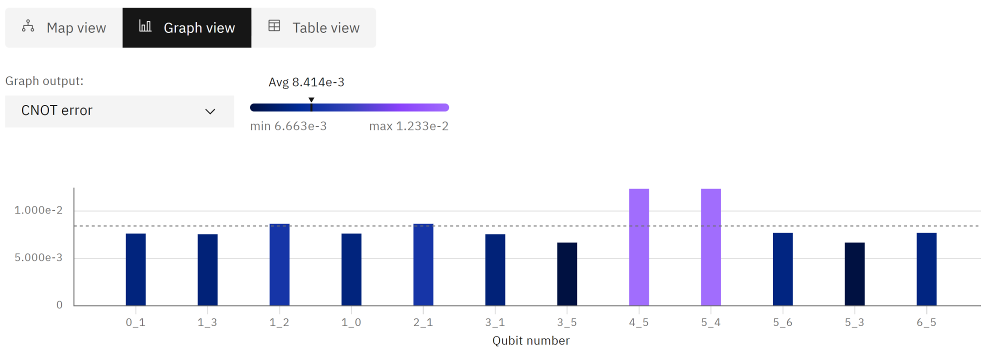


Figure 3.3: CNOT errors

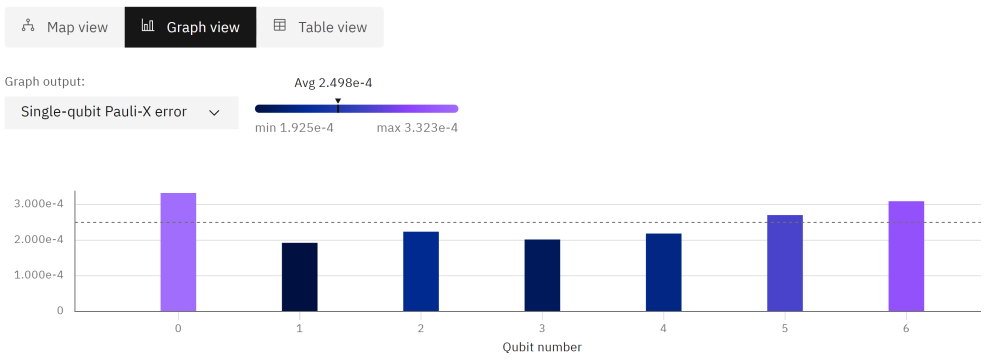


Figure 3.4: Single-qubit Pauli-X errors

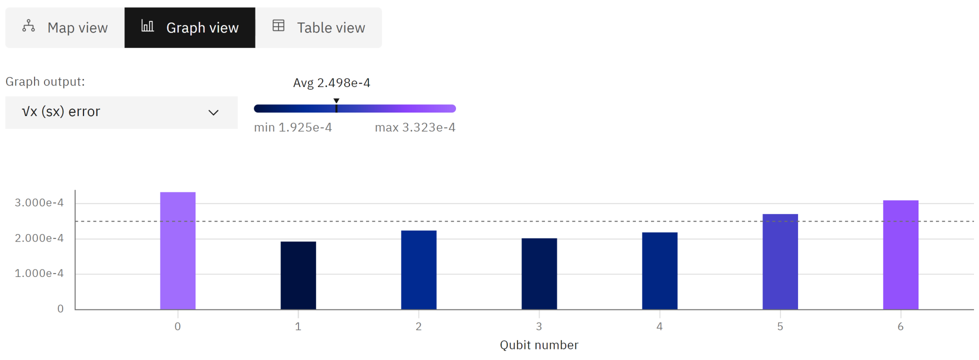


Figure 3.5: Sx errors

i and j ($i, j \in \{0, 1, 2\}$) would be $H^{(i,j)}_{\text{Heis2}} = \sigma^{(i)}_x \sigma^{(j)}_x + \sigma^{(i)}_y \sigma^{(j)}_y + \sigma^{(i)}_z \sigma^{(j)}_z$. Rewriting $U_{\text{Heis3}}(t)$ in terms of the two possible subsystems within the total $N = 3$ system you will simulate,

$$U_{\text{Heis3}}(t) = \exp\left[-it\left(H^{(0,1)}_{\text{Heis2}} + H^{(1,2)}_{\text{Heis2}}\right)\right]. \tag{3.25}$$

$H^{(0,1)}_{\text{Heis2}}$ and $H^{(1,2)}_{\text{Heis2}}$ do not commute, so

$$U_{\text{Heis3}}(t) \neq \exp\left(-itH^{(0,1)}_{\text{Heis2}}\right)\exp\left(-itH^{(1,2)}_{\text{Heis2}}\right) \tag{3.26}$$

.

But, this product decomposition can be approximated with Trotterization which says $U_{\text{Heis3}}(t)$ is approximately a short evolution of $H^{(0,1)}_{\text{Heis2}}$ (time = t/n) and followed by a short evolution of $H^{(1,2)}_{\text{Heis2}}$ (time = t/n) repeated n times

$$U_{\text{Heis3}}(t) = \exp\left[-it\left(H^{(0,1)}_{\text{Heis2}} + H^{(1,2)}_{\text{Heis2}}\right)\right] \tag{3.27}$$

$$U_{\text{Heis3}}(t) \approx \left[\exp\left(\frac{-it}{n}H^{(0,1)}_{\text{Heis2}}\right)\exp\left(\frac{-it}{n}H^{(1,2)}_{\text{Heis2}}\right)\right]^n. \tag{3.28}$$

n is the number of Trotter steps, and as n increases, the approximation should becomes more accurate but as we have already anticipated experimentally this not seems to be the case.

But now we have a state of the art solution for decomposing Heis2 into quantum gates [VD04] [Chi+19] and this is shown in figure 3.6, we can then implement this decomposition of Heis2 into our decomposition of Heis3.

In the decomposition of Heis2 we have:

$$RX(\theta) = \exp\left(-i\frac{\theta}{2}X\right) = \begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix} \tag{3.29}$$

and

$$RZ(\lambda) = \exp\left(-i\frac{\lambda}{2}Z\right) = \begin{pmatrix} e^{-i\frac{\lambda}{2}} & 0 \\ 0 & e^{i\frac{\lambda}{2}} \end{pmatrix}. \tag{3.30}$$

Those are not native gates but as already said they can be implemented as Jakarta has a universal set of quantum gates, and we can change the phase factor directly without the need of using Pulse.

At the end our circuit is represented in figure 3.7 where we can see the 7 qubits, the 4 trotterization steps and the measure at the end. The circuit of 3.6 is contained in each trotterization step.

Finally, those are the results:

NOISY SIMULATION: 0.4405 ± 0.0011 (N=4)

REAL DEVICE: 0.3108 ± 0.0034 (N=4)

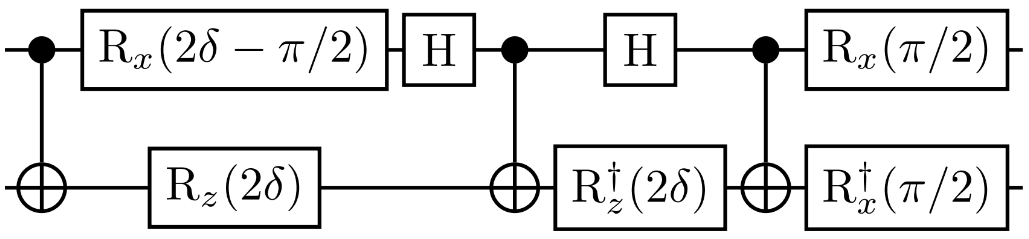


Figure 3.6: Heis2 decomposition

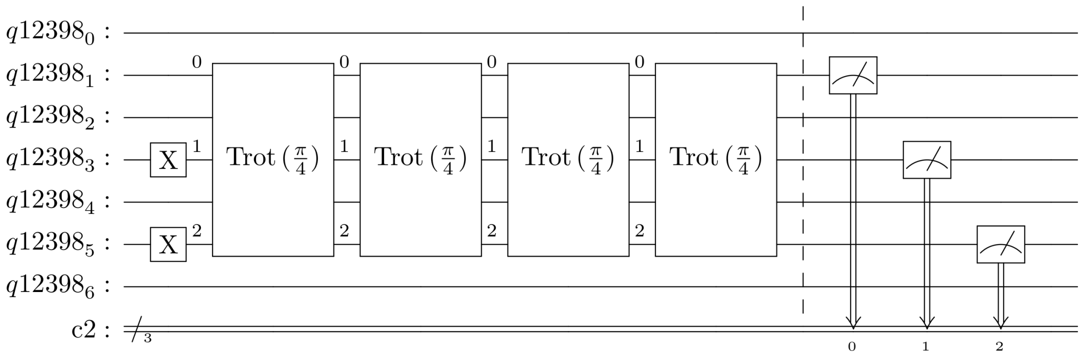


Figure 3.7: Circuit overview

You may ask why there were not tried more complicated decomposition, the reason is that at the end reducing the original Hamiltonian to Hamiltonians with already recognized state of the art decomposition resulted to be the strategy that provided the better results, as completely raw decomposition directly to gates resulted in significantly weaker results.

In addition to that the least number of trotterization steps as the problem required were used. This is because, contrary to what Lie's formula presented in chapter 2 seems to suggests, experimentally it was very clear that augmenting trotterization steps significantly reduced state tomography results, probably because no noise suppression techniques were used in order to deal with the the fact that the circuit increases with more Trotterization steps.

A CODE

Here we report the code, written in Qiskit and executed on the device.

But what is Qiskit? Qiskit is an open-source software development kit (SDK) for working with quantum computers at the level of circuits, pulses, and algorithms. It provides tools for creating and manipulating quantum programs and running them on prototype quantum devices on IBM Quantum Experience or on simulators on a local computer.

It follows the circuit model for universal quantum computation, and can be used for any quantum hardware (currently supports superconducting qubits and trapped ions) that follows this model.

Note that even if this SDK is developed and maintained by IBM it is fully Open Source and can be used to work in systems that are not IBM's (if they implement support to the language).

```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams.update(–'font.size': 16”)  # enlarge matplotlib fonts

# Import qubit states Zero (—0¿) and One (—1¿), and Pauli
↪   operators (X, Y, Z)
from qiskit.opflow import Zero, One, I, X, Y, Z

# Suppress warnings
import warnings
warnings.filterwarnings('ignore')

def H˙heis3():
# Interactions (I is the identity matrix; X, Y, and Z are Pauli
↪   matricies; ˆ is a tensor product)
XXs = (IˆXˆX) + (XˆXˆI)
YYs = (IˆYˆY) + (YˆYˆI)
ZZs = (IˆZˆZ) + (ZˆZˆI)

# Sum interactions
H = XXs + YYs + ZZs

# Return Hamiltonian
return H

# Returns the matrix representation of U˙heis3(t) for a given time
↪   t assuming an XXX Heisenberg Hamiltonian for 3 spins-1/2
↪   particles in a line
```

```
def U˙heis3(t):
    # Compute XXX Hamiltonian for 3 spins in a line
    H = H˙heis3()

    # Return the exponential of -i multipled by time t multipled by
     ↪   the 3 spin XXX Heisenberg Hamilonian
    return (t * H).exp˙i()

    # Define array of time points
ts = np.linspace(0, np.pi, 100)

# Define initial state —110¿
initial˙state = OneˆOneˆZero

# Compute probability of remaining in —110¿ state over the array of
 ↪   time points
 # ˜initial˙state gives the bra of the initial state (¡110—)
 # @ is short hand for matrix multiplication
 # U˙heis3(t) is the unitary time evolution at time t
 # t needs to be wrapped with float(t) to avoid a bug
 # (...).eval() returns the inner product ¡110—U˙heis3(t)—110¿
 #  np.abs(...)**2 is the modulus squared of the innner product which
  ↪   is the expectation value, or probability, of remaining in —110¿
probs˙110 = [np.abs((˜initial˙state @ U˙heis3(float(t)) @
 ↪   initial˙state).eval())**2 for t in ts]

# Plot evolution of —110¿
plt.plot(ts, probs˙110)
plt.xlabel('time')
plt.ylabel(r'probability of state $—110“rangle$')
plt.title(r'Evolution of state $—110“rangle$ under $H˙–Heis3″$')
plt.grid()
plt.show()

# Importing standard Qiskit modules
from qiskit import QuantumCircuit, QuantumRegister, IBMQ,
 ↪   execute, transpile
from qiskit.providers.aer import QasmSimulator
from qiskit.tools.monitor import job˙monitor
from qiskit.circuit import Parameter

# Import state tomography modules
from qiskit.ignis.verification.tomography import
 ↪   state˙tomography˙circuits, StateTomographyFitter
from qiskit.quantum˙info import state˙fidelity

```

```
# suppress warnings
import warnings
warnings.filterwarnings('ignore')

# load IBMQ Account data

IBMQ.save˙account(TOKEN)  # replace TOKEN with your API token string (https://quantum-computing.ibm.com/lab/docs/iql/manage/account/ibmq)
provider = IBMQ.load˙account()

from qiskit.providers.aer import AerSimulator# Get backend for experiment
provider = IBMQ.get˙provider(hub='ibm-q-community', group='ibmquantumawards', project='open-science-22')
jakarta = provider.get˙backend('ibmq˙jakarta')
properties = jakarta.properties()
# Simulated backend based on ibmq˙jakarta's device noise profile
sim˙noisy˙jakarta = QasmSimulator.from˙backend(provider.get˙backend('ibmq˙jakarta'))

# Noiseless simulated backend
sim = QasmSimulator()

t = Parameter('t')

Solution˙qr = QuantumRegister(2)
Solution˙qc = QuantumCircuit(Solution˙qr, name='3-CNOT')

Solution˙qc.cnot(0,1)
Solution˙qc.rx(2*t - np.pi/2,0)
Solution˙qc.rz(2*t,1)
Solution˙qc.h(0)
Solution˙qc.cnot(0,1)
Solution˙qc.h(0)
Solution˙qc.rz(2*t, 1).inverse()
Solution˙qc.cnot(0,1)
Solution˙qc.rx(np.pi/2, 0)
Solution˙qc.rx(np.pi/2, 1).inverse()

Solution = Solution˙qc.to˙instruction()

# Combine subcircuits into a single multiqubit gate representing a single trotter step
num˙qubits = 3

```

```
Trot˙qr = QuantumRegister(num˙qubits)
Trot˙qc = QuantumCircuit(Trot˙qr, name='Trot')

for i in range(0, num˙qubits - 1):
    Trot˙qc.append(Solution, [Trot˙qr[i], Trot˙qr[i+1]])

# Convert custom quantum circuit into a gate
Trot˙gate = Trot˙qc.to˙instruction()

# The final time of the state evolution
target˙time = np.pi

# Number of trotter steps
trotter˙steps = 4  ### CAN BE ¿= 4

# Initialize quantum circuit for 3 qubits
qr = QuantumRegister(7)
qc = QuantumCircuit(qr)

# Prepare initial state (remember we are only evolving 3 of the 7
↪   qubits on jakarta qubits (q˙5, q˙3, q˙1) corresponding to the state
↪   —110¿)
qc.x([3,5])  # DO NOT MODIFY (—q˙5,q˙3,q˙1¿ = —110¿)

# Simulate time evolution under H˙heis3 Hamiltonian
for ˙ in range(trotter˙steps):
    qc.append(Trot˙gate, [qr[1], qr[3], qr[5]])

# Evaluate simulation at target˙time (t=pi) meaning each trotter
↪   step evolves pi/trotter˙steps in time
qc = qc.bind˙parameters(–t: target˙time/trotter˙steps˝)

# Generate state tomography circuits to evaluate fidelity of
↪   simulation
st˙qcs = state˙tomography˙circuits(qc, [qr[1], qr[3], qr[5]])

# Display circuit for confirmation
#st˙qcs[-1].decompose().decompose().draw('latex')  # view
↪   decomposition of trotter gates
st˙qcs[-1].draw('mpl')  # only view trotter gates

shots = 8192
reps = 8
backend = sim˙noisy˙jakarta
config = backend.configuration()
# reps = 8
# backend = jakarta

jobs = []
for ˙ in range(reps):
    # execute
    job = execute(st˙qcs, backend, shots=shots)
    print('Job ID', job.job˙id())
    jobs.append(job)

print(”This backend is called –0˝, and is on version –1˝. It has –2˝
 ↪   qubit–3˝. It ”
      ”–4˝ OpenPulse programs. The basis gates supported on this
        ↪   device are –5˝.”
      ””.format(config.backend˙name,
                config.backend˙version,
                config.n˙qubits,
                '' if config.n˙qubits == 1 else 's',
                'supports' if config.open˙pulse else 'does not support',
                config.basis˙gates))

                for job in jobs:
    job˙monitor(job)
    try:
        if job.error˙message() is not None:
            print(job.error˙message())
    except:
        pass

        # Compute the state tomography based on the st˙qcs quantum
          ↪   circuits and the results from those ciricuits
def state˙tomo(result, st˙qcs):
    # The expected final state; necessary to determine state
      ↪   tomography fidelity
    target˙state = (OneˆOneˆZero).to˙matrix()  # DO NOT MODIFY
      ↪   (—q˙5,q˙3,q˙1¿ = —110¿)
    # Fit state tomography results
    tomo˙fitter = StateTomographyFitter(result, st˙qcs)
    rho˙fit = tomo˙fitter.fit(method='lstsq')
    # Compute fidelity
    fid = state˙fidelity(rho˙fit, target˙state)
    return fid

# Compute tomography fidelities for each repetition
fids = []
for job in jobs:
    fid = state˙tomo(job.result(), st˙qcs)
    fids.append(fid)

print('state tomography fidelity = −:.4f″ “u00B1
↪   −:.4f″'.format(np.mean(fids), np.std(fids)))
```

CONCLUSIONS

This was an attempt to summarize as much as possible a short introduction to the field of quantum computation, an overview to the actual state of the art, at the time of writing, of quantum simulation and at the end the explanation of the solution provided for the competition.

Long story short, results are not extremely exciting and/or revolutionary. Many aspects of this solution could be improved:

NOISE SUPPRESSION: a main point which was missed mostly due to time limitation and lack of knowledge in this specific area. Even if noise suppression it is not and it can't be a valid replacement of a good decomposition of the time operator, it can definitely help.

ERRORS IN DIFFERENT GATES: exploiting the different errors in the gates in the decomposition was another big miss, but everything tried did not seems to give an improvement.

MORE ADVANCED DECOMPOSITIONS: exploiting specific decomposition tied to the tipe of qubit used in Jakarta it is also something worth exploiting, even if it does not seems to be too much literature about the subject.

ANCILLA QUBITS: ancilla qubits were not exploited, which lead to 3 qubit being de facto not used during all the computation, the involvement of those qubits could have played a major role, especially in noise suppression.

MORE EXPERIMENTS: a huge drawback was played by the long queue in accessing the device, resulting in a limited tuning of the parameters based on experimental results, this variable was obviously not under our control.

However, based on the various experiments, we are able to conclude the following:

WE ARE STILL FAR FROM A LOGICAL QUBIT: a logical qubit is an abstract qubit that performs as specified in a quantum algorithm and has a long enough coherence time to be usable by quantum logic gates. At the time of writing, we are still dealing with a physical qubit, i.e. a device that behaves as a two-state quantum system, but that carry with itself all the limitations of its implementation. That means that when implementing an algorithm, unlike classical computation, we are not only dealing with the algorithm itself, but mostly with the hardware (i.e. the physical qubits) used to implement it. Error-correction is essential, and very low level control of the hardware is needed to produce something useful. Different implementations of the physical qubit play a crucial role in how a specific algorithm or a specific decomposition will perform, something which is not present in classical computation.

SIMULATORS ARE ALMOST USELESS: the amount of randomness in the simulator of quantum backends on classical computer was astonishing, but it does make sense connecting experience with theory. Quantum computers are inherently more powerful for certain operations and simulating them in an accurate way is not possible. This however was and still is a big limitation, as access to new and reliable hardware is a bit of a challenge itself.

ENTANGLEMENT PLAYS A CRUCIAL ROLE: all the power of the state of the art decomposition of H_{Heis2} comes from entanglement. Citing [Mic11] 'entanglement is iron to the classical world's bronze age'. However, there is as yet no complete theory of it and it is not always clear how it can be used. Being able to exploit it in a decomposition however seems to drastically overcome any other type of decomposition.

MACHINE LEARNING DOES NOT SEEM A GOOD FIT: we tried (mostly adapted solutions found online) a machine learning approach to the decomposition, paired with a noisy simulator, but they didn't seem to work, at all. It could have been a fault of implementation, however this seems one of those few cases where pen and paper outperform brute force techniques.

SINGLE MEASUREMENTS ARE NOT ENOUGH: unless very rare cases, it is necessary to run the algorithm several times in order to produce a sample with statistical significance to work on, as a single measurement is not enough to establish the exact result, giving the large delta caused by the noise. This could of course be improved with error-correction techniques, but rarely to the point when a single execution is enough.

In the end, quantum computation and quantum simulation is an exciting field which is evolving rapidly and it is definitely very promising. This was a first and a small attempt to familiarize with it in a real-world scenario and with an open challenge. Even if results are not even near to the actual state of the art, the amount of knowledge acquired during this long journey was definitely worth the effort.

BIBLIOGRAPHY

[Ale+20] Thomas Alexander et al. "Qiskit pulse: programming quantum computers through the cloud with pulses". In: *Quantum Science and Technology* 5.4 (Aug. 2020), p. 044006.

[Ben+97] Charles H. Bennett et al. "Strengths and Weaknesses of Quantum Computing". In: *SIAM Journal on Computing* 26.5 (1997), pp. 1510–1523.

[BMK10] Katherine L. Brown, William J. Munro, and Vivien M. Kendon. "Using Quantum Computers for Quantum Simulation". In: *Entropy* 12.11 (2010), pp. 2268–2307. ISSN: 1099-4300.

[BV17] Stefano Bonzio and Paola Verrucchi. "The Rhythm of Quantum Algorithms". In: *Soft Comput.* 21 (2017), p. 1515.

[Chi+19] A. Chiesa et al. "Quantum hardware simulating four-dimensional inelastic neutron scattering". In: *Nature Physics* 15.5 (Mar. 2019), pp. 455–459.

[doc22a] Qiskit documentation. *Hamiltonian Tomography*. https://qiskit.org/textbook/ch-quantum-hardware/hamiltonian-tomography.html. 2022.

[doc22b] Qiskit documentation. *Measurement Error Mitigation*. https://qiskit.org/textbook/ch-quantum-hardware/measurement-error-mitigation.html. 2022.

[DS14] Ish Dhand and Barry Sanders. "Stability of the Trotter-Suzuki decomposition". In: (Mar. 2014). DOI: 10.1088/1751-8113/47/26/265206.

[GAN14] I. M. Georgescu, S. Ashhab, and Franco Nori. "Quantum simulation". In: *Rev. Mod. Phys.* 86 (1 Mar. 2014), pp. 153–185. DOI: 10.1103/RevModPhys.86.153. URL: https://link.aps.org/doi/10.1103/RevModPhys.86.153.

[Gri04] David J. Griffiths. *Introduction to Quantum Mechanics*. 2nd. Pearson Prentice Hall, 2004.

[JHG19] E.R. Johnston, N. Harrigan, and M. Gimeno-Segovia. *Programming Quantum Computers: Essential Algorithms and Code Samples*. O'Reilly Media, 2019. ISBN: 9781492039655.

[Kay07] Phillip Kaye. *An introduction to quantum computing*. Oxford: Oxford University Press, 2007.

[Ma+11] Xiao-song Ma et al. "Quantum simulation of the wavefunction to probe frustrated Heisenberg spin systems". In: *Nature Physics* 7.5 (Feb. 2011), pp. 399–405.

[Mic11] Isaac L. Chuang Michael A. Nielsen. *Quantum Computation and Quantum Information*. Cambridge, 2011.

[Ott+17] J. S. Otterbach et al. *Unsupervised Machine Learning on a Hybrid Quantum Computer*. 2017.

[Qis22] Qiskit. *Introduction to Transmon Physics*. https://qiskit.org/textbook/ch-quantum-hardware/transmon-physics.html. 2022.

[Sal+15] Y. Salathé et al. "Digital Quantum Simulation of Spin Models with Circuit Quantum Electrodynamics". In: *Phys. Rev. X* 5 (2015), p. 021027.

[Sha11] R. Shankar. *Principles of Quantum Mechanics*. Springer US, 2011.

[Ste+21] John P. T. Stenger et al. "Simulating the dynamics of braiding of Majorana zero modes using an IBM quantum computer". In: *Physical Review Research* 3.3 (Aug. 2021).

[Tac+19] Francesco Tacchino et al. "Quantum Computers as Universal Quantum Simulators: State-of-the-Art and Perspectives". In: *Advanced Quantum Technologies* 3.3 (Dec. 2019).

[VD04] G. Vidal and C. M. Dawson. "Universal quantum circuit for two-qubit transformations with three controlled-NOT gates". In: *Physical Review A* 69.1 (Jan. 2004).

[Wik22] Wikipedia contributors. *Quantum Heisenberg model — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Quantum_Heisenberg_model&oldid=1068863987. 2022.

[Wor22] Physics World. *Superconducting quantum bits*. 2022.