Bachelor's Programme in Science and Technology

Parallel Simulations, Finite Bath

Parallelising Numerical Simulations for the Time Evolution of a Finite Spin Bath Coupled to a Qubit

Stergios Tsiormpatzis

Aalto University

BACHELOR'S THESIS

Aalto University
BACHELOR'S THESIS 2024

Parallel Simulations, Finite Bath

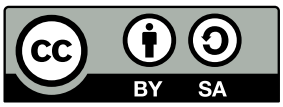
Parallelising Numerical Simulations for the Time Evolution of a Finite Spin Bath Coupled to a Qubit

Stergios Tsiormpatzis

Thesis submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Technology.
Otaniemi, 2 Sep 2024

Supervisor: Matti Raasakka
Advisors: Jukka Pekola
Ilari Mäkinen

Aalto University
School of Science
Bachelor's Programme in Science and Technology



Author
Stergios Tsiormpatzis

Title
Parallel Simulations, Finite Bath: Parallelising Numerical Simulations for the Time Evolution of a Finite Spin Bath Coupled to a Qubit.

School School of Science

Degree programme Bachelor's Programme in Science and Technology

Major Quantum Technology **Code** SCI3103

Supervisor Matti Raasakka

Advisor Jukka Pekola, Ilari Mäkinen

Level Bachelor's thesis **Date** 2 Sep 2024 **Pages** 28+28 **Language** English

Abstract

At low energies, the environment of most open quantum systems can be represented by baths of harmonic oscillators or spins. Modelling and simulating an open quantum system requires substantial computational resources, and simulations on personal computers are limited to small models. Parallel computing architectures may allow expansion of the model size and enhance simulation capabilities. This thesis aims to optimise existing simulation code of an open quantum model for parallel execution. The model consists of an open quantum system, represented by a qubit, which interacts with an environment containing a finite number of weakly and randomly coupled spins. To reduce the dimension of its Hilbert space, the environment is mapped into an oscillator bath, producing a quadratic Hamiltonian. The time evolution of all particles is tracked through exact diagonalisation. The code is written in MATLAB and good scientific computing practices are followed. Primary optimisation takes place in a local computing environment using MATLAB Parallel Computing Toolbox. The analysis of this legacy code facilitates its *modular* restructuring with distinct physical processes or computational tasks assigned to specific functions. The *modular* version, which has been vectorised, shows an approximate 35% reduction in time consumption for a small model with 1500 spins. Optimisation for parallelism continues with dividing the code further into *multicore* and *GPU* parallelisms, with the latter indicating improved performance of about 23% over the *modular* version. Next, the optimised code was tested in Aalto Triton supercomputing cluster, achieving an approximate 99% improvement in time performance between the *modular* and the *GPU* versions for a model of 10 000 spins, as well as an enlargement of the model by a factor of 20, to 30 000 spins. Having being tested in two different computational environments, the aimed optimisation is considered successful. The structure of the code makes its adaptation simple for simulation of different models, enabling further exploration of its potential.

Keywords Finite bath, MATLAB, parallelisation, qubit, simulation, spin bath

urn https://aaltodoc.aalto.fi

Contents

1. INTRODUCTION

Quantum mechanics, when considered at an introductory level, describes the dynamics of an isolated (closed) quantum system as defined by the Schrödinger equation. However, from an engineering perspective, an isolated quantum system is only an ideal model [1]. In the laboratory, all elements of quantum technology interact with their environment. In this setting, it is customary to employ a description that somewhat resembles the one of ensembles in thermodynamics, as introduced by Gibbs [2]. The microscopic or mesoscopic and spatially localised quantum system is called an open quantum system. It is considered coupled to a much larger environment, constituted of infinitely many degrees of freedom, called a reservoir or a bath when it is also in thermal equilibrium [3].

The interaction of the open quantum system with the reservoir induces changes in its dynamics; consequently, the time evolution of the system cannot be described by its Hamiltonian alone anymore. Instead, various approaches have been developed to follow the dynamics of an open quantum system, such as:

- supplementing its Hamiltonian with a non-Hermitian Hamiltonian that describes the effective interaction between the open quantum system and the reservoir [4], [5];

- working in the Schrödinger picture by using Master equations [3], [6] (e.g., the commonly utilised phenomenological Lindblad master equation or its physically concrete equivalent [7] Bloch-Redfield master equation) that describe the unitary evolution of the density operator as well as the non-unitary interaction with the reservoir (i.e., noise) [8];

- working in the Heisenberg picture with the Langevin equations [6] that

combine deterministic parts stemming from the open quantum system with stochastic parts representing the dissipative effect of the reservoir [9].

However, the introduction of infinite degrees of freedom makes the calculations intractable. To achieve useful calculations, one is usually limited to approximations, such as considering the weak-coupling and low-energy regimes. In those approximations, some systems can be described by models in which a qubit is coupled with oscillator or spin baths, usually infinite. Then, the qubit and the bath together constitute a pure state, i.e., the interaction with the wider environment is neglected. Videlicet, the open system and the reservoir are now assumed to be a single closed system [3]. To describe it visually, one can think of an open system in a manner much like the Matryoshka dolls. An open system is considered to be in contact with a specific environment, as part of a larger closed system. This larger closed system can be considered again an open system in contact with a larger environment. Thus, it constitutes a larger closed system, and so on, reaching the limits of the universe. As a result, the model can efficiently reduce the complexity of an infinite open quantum system to a manageable description of a closed system.

The usage of models in science is extensive and satisfies multiple purposes [10]. Scientists often simplify complex systems with mathematical tools in validatable models [11]. Those models can then become simulations that, in similar respect to experiments, can produce useful insights about the real system they are modelling [12].

With the development of digital computing during the last half century, computers have become an integral part of scientific and engineering practice, by expanding the modelling and simulating toolbox. In this context, scientific computing refers to the set of tools that scientists and engineers utilise to solve theoretical or technological problems [11]. Its practice has become equivalent to the traditional science laboratory and, similarly to a laboratory, requires compliance with specific protocols to maintain efficiency and reproducibility [13].

The more complicated the system to be modelled, the more computing speed and memory it needs. Although technological improvement in the miniaturisation of microelectronic components, which makes faster processors and larger memories, has managed for decades to satisfy the scientific thirst for increased computing power [14], technology has reached a "power

wall" limit [15]. Hence, the difficulties in affordably managing the growing power consumption and heat production of microelectronic components are no longer simply sustainable [16]. Thus, an engineering approach to handle the problem has appeared in the form of parallel computing architectures.

Parallelism is a set of computational techniques designed to exploit the resources of parallel computing architectures. Three types of parallelism exist: vector (or array) parallelism, shared memory (or multi-threaded/multiple processes) parallelism, and distributed memory (or message passing interface) parallelism [17], [18]. Vector parallelism denotes the technique of applying an operation simultaneously to all the coordinates of a vector, instead of applying it to each value separately. Shared memory parallelism refers to the usage of multiple central processing unit (CPU) cores in a single computer, with a common memory. Distributed memory parallelism is achieved with multiple supercomputer nodes, in which each node has its own memory. In addition, recent advancements in graphical processing unit (GPU) technology [19], [20] create an overcover that, based on vector and shared memory parallelism, creates the opportunity for an exponential acceleration of computing performance [17], [21].

Thus, the aim of the present thesis is to optimise for parallelisation computing code that has been recently used to simulate the time evolution of a qubit coupled to a finite spin bath [22], [23], in order to improve time performance and to make possible the expansion of the numerical simulations in a larger number of spins. To achieve this goal, the following steps are undertaken in the research:

1. The code used in the previous studies is studied and analysed.

2. Optimisation for parallelisation is attempted in a local environment.

3. Implementation of the parallelised code is tested in the Triton scientific computing cluster available at Aalto University.

The rest of the thesis has the following structure. Section 2 provides an essential theoretical background for the simulated model. Section 3 describes the detailed methods used for the optimisation. Section 4 presents the main results achieved. Section 5 discusses the results, indicates limitations of the approach used, and suggests ideas for further research.

2. ESSENTIAL THEORETICAL BACKGROUND

The following section provides a theoretical background for the simulated model. It opens with a brief introduction to the canonical models used in open quantum systems. This is followed by an attempt to provide a model Hamiltonian for a qubit coupled to a finite spin bath, and a description of the approximations used in the simulation. Finally, it presents concepts related to the time-evolution of the model.

2.1 *Canonical Models*

Even if there is an infinite variety of open quantum systems in nature, it has been found that most of them, when restricted in the low-energy regime (e.g.,

$$k_B T << \Delta E$$

where

k_B	the Boltzmann constant;
T	temperature in Kelvin;
ΔE	the energy gap between the quantum states of the system.),

can be modelled by a small number of canonical models [24]. Those canonical models are built around the usage of two elements to model the system and the environment: harmonic oscillators and spins [25]. Oscillator baths correspond to an environment of delocalised modes (i.e., bosonic field modes) with the wave function of each harmonic oscillator spreading to a large area. Spin baths correspond to an environment of localised modes (i.e., fermionic modes, such as paramagnetic or nuclear spins, material impurities, and defects) with the wave function of each of them limited in a very small area [26], [27].

Studies have indicated how in many cases the quantum environment can be mapped to an oscillator bath [28], [29]; however, at low energies the local modes dominate and are usually restricted to two-level states, making the 1/2-spin bath a more appropriate model [24], [27]. The localised nature of the wave functions of the spins in the bath make the interaction and coupling between the spins of the bath weak, leaving space for stronger system-environment interaction [30] compared to the oscillator case.

2.2 *A Qubit Coupled to a 1/2-Spin Bath Model*

A qubit (quantum bit) is the basic carrier of information for quantum computing, much like the bit is for the conventional computer [31]. Several approaches to developing qubits have been implemented, such as atomic ion traps [32], utilisation of nuclear magnetic resonance and nitrogen vacancies in diamond [33], semiconductor qubits [34], [35], superconducting qubits [36], [37]. Abstractly, a qubit is a paradigmatic quantum two-level system model [38], and as such, it can be described by the Hamiltonian of a fictitious 1/2-spin particle under the effect of a static magnetic field [32], [39, p. 413]. Despite its abstract simplicity (i.e., spin up, spin down), the description of a real 1/2-spin particle in its environment is a fairly complicated task [40], requiring considerations of the physical properties of the materials that can be explored experimentally [41]. Especially when considering many particles, the quantum field theory concept and method of second quantisation are able to simplify the description and calculations [42], [43]. Thus, the Schrödinger picture of the model is subsequently described first in the Pauli operator, followed by the matrix, and finally the second quantisation formalisms.

The fermionic qubit–1/2-spin bath model consists of N microscopic 1/2-spin particles that behave as the bath and are independently coupled to a qubit (modelled by a central 1/2-spin particle) which is the actual open quantum system. The whole model can be characterised by the Hamiltonian below:

$$\hat{H} = \hat{H}_S + \hat{H}_B + \hat{H}_{int},$$

where

$\hat{H}_S$	self-Hamiltonian of the qubit;
$\hat{H}_B$	self-Hamiltonian of the bath;
$\hat{H}_{int}$	Hamiltonian of the qubit–bath interaction/coupling.

To represent high levels of complexity in an experimental setting, the qubit is assumed to be placed in a weak, $z-$oriented, and static (i.e., time-independent) magnetic field B_z, and an oscillating (i.e., time-dependent) magnetic field B_x, perpendicular to the $z-$axis, along the $x-$axis, similarly to [44], [45]. With such a configuration and at low temperatures, we work in the low-energy regime [46, p. 5].

Then, the self-Hamiltonian of the qubit can be written as

$$\hat{H}_S = \frac{1}{2}\hbar\omega_0\hat{\sigma}_z + \frac{1}{2}\Delta\hat{\sigma}_x.$$

Here, the difference in energy between the qubit's ground state and excited state due to the Zeeman effect [47] is $\hbar\omega_0$. The angular frequency of the oscillation between the two, ω_0, depends on the strength of the magnetic field B_z by the relation $\omega_0 = \gamma B_z$, where γ is the gyromagnetic ratio [48, p. 40] of the qubit, which for 1/2-spin particles is equivalent to the product of the $g-$factor with the Bohr magneton: $\gamma = \frac{g\mu_B}{\hbar}$. Accordingly, Δ is the tunnelling amplitude between the two levels due to the oscillating magnetic field, physically flipping spin up and spin down. While $\hat{\sigma}_{x,y,z}$ are the Pauli x, y, z matrices [49]:

$$\hat{\sigma}_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \doteq |0\rangle\langle 1| + |1\rangle\langle 0|$$

$$\hat{\sigma}_y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \doteq -i\,|0\rangle\langle 1| + i\,|1\rangle\langle 0|$$

$$\hat{\sigma}_z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \doteq |0\rangle\langle 0| - |1\rangle\langle 1|,$$

where

$|0\rangle$ $= \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ is the ground state in the standard basis;

$|1\rangle$ $= \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ is the excited state in the standard basis;

i $= \sqrt{-1}$ is the imaginary unit.

In the matrix form, the spin Hamiltonian is:

$$\hat{H}_S = \frac{1}{2}\begin{bmatrix} \hbar\omega_0 & \Delta \\ \Delta & -\hbar\omega_0 \end{bmatrix}.$$

In the second quantisation formalism for fermions [50], with creation operator $\hat{c}^\dagger$, annihilation operator $\hat{c}$, and commutation relations

$$\{\hat{c}_k, \hat{c}_l^\dagger\} = \delta_{k,l} = \hat{I}$$

$$\{\hat{c}_k, \hat{c}_l\} = \{\hat{c}_k^\dagger, \hat{c}_l^\dagger\} = 0,$$

where k and l correspond to the k and l particles, the Pauli's matrices are represented as

$$\hat{\sigma}_x = \hat{c}^\dagger + \hat{c}$$

$$\hat{\sigma}_y = i(\hat{c} - \hat{c}^\dagger)$$

$$\hat{\sigma}_z = 2\hat{c}^\dagger\hat{c} - 1,$$

making the spin Hamiltonian

$$\hat{H}_S = \hbar\omega_0\left(\hat{c}^\dagger\hat{c} - \frac{1}{2}\right) + \frac{1}{2}\Delta\left(\hat{c}^\dagger + \hat{c}\right).$$

The complexity of the energy dynamics of a bath consisting of N $1/2$-spin particles requires multiple approximations, and finding a proper analytical model for a Hamiltonian can require multiple elements of quantum field theory and relativistic quantum mechanics, as explained in detail in [51]. The interactions between the spins in the bath, which in practice are defined by the properties of the material [52], play an essential role in the dynamics of the whole system [53]. Randomness in the strength of interspin coupling can deeply influence the decoherence [54]. Thus, non-interacting spin models, such as the star network described in [55], would not be realistic candidates for a spin bath.

Probably the most widely adopted models of interspin interactions are the Ising model, which was developed first by Ising and his teacher Lenz [56], and the Heisenberg model, which considers more complicated interactions.

The Hamiltonian of the Ising model of interspin interactions is given by

$$\hat{H} = -J\sum_{n\neq m}\sigma_z^{(n)}\sigma_z^{(m)},$$

where the summation is over n, m neighbouring spin pairs and J is the exchange parameter [57], [58], i.e., the factor defining the strength of the interaction/coupling. Since the model considers neighbouring spin pairs interactions in one axis only, it is called anisotropic.

The Hamiltonian of the Heisenberg model of interspin interactions is given by

$$\hat{H} = -J \sum_{n \neq m} (\sigma_x^{(n)} \sigma_x^{(m)} + \sigma_y^{(n)} \sigma_y^{(m)} + \sigma_z^{(n)} \sigma_z^{(m)}),$$

where the summation is over n, m neighboring spin pairs and J is the exchange parameter[57], [58]. The model is isotropic, i.e., considers the interactions in all axes equally (with the same exchange parameter too).

There are modifications or generalisations of the two models, and it has been found that they belong to two universality classes (or Landau classification) [58] of Q-state Potts model and n-vector model, with the Ising model being an intersection of both [59]. Each of the models can be more accurate for different parameters and configurations [60], [61].

Similarly to the previous handling of the qubit's self-Hamiltonian, and considering the intention of computer-based numerical simulation, the self-Hamiltonian of the bath under the influence of a weak, $z-$oriented, and static magnetic field B_z can be written using a modification of the one–dimensional Ising model as

$$\hat{H}_B = \sum_{n=1}^{N} \frac{1}{2} \hbar \omega_n \sigma_z^{(n)} - \sum_{n \neq m} J_{nm} \sigma_z^{(n)} \sigma_z^{(m)}.$$

Here, the energy gap between the two states of the n^{th} spin is $\hbar\omega_n$, with ω_n the angular frequency of the oscillation between them. The second term represents the interspin interactions in the bath, with the summation over all n, m spin pairs. However, instead of a common exchange parameter J, a different exchange parameter J_{nm} is used for each pair. The "matrix" representation takes the form of tensor products:

$$\hat{H}_B = \sum_{n=1}^{N} \frac{1}{2} \hbar \omega_n \left(I \otimes \cdots \otimes \sigma_z \otimes \cdots \otimes I \right) - \sum_{n \neq m} J_{nm} \left(I \otimes \cdots \otimes \sigma_z \otimes \cdots \otimes \sigma_z \otimes \cdots \otimes I \right),$$

where $I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ is the identity matrix. In the second quantisation formal-

ism the Hamiltonian becomes

$$\hat{H}_B = \sum_{n=1}^{N} \hbar\omega_n \left(\hat{c}^\dagger \hat{c} - \frac{1}{2}\right) - \sum_{n \neq m} J_{nm} \left(\hat{c}^\dagger \hat{c} - \frac{1}{2}\right)^{(n)} \left(\hat{c}^\dagger \hat{c} - \frac{1}{2}\right)^{(m)}.$$

Finally, the interaction Hamiltonian can be written as

$$\hat{H}_{int} = \frac{1}{2}\sigma_z \otimes \sum_{n=1}^{N} l_n \sigma_z^{(n)}.$$

Here l_n is the coupling strength between the qubit and the n^{th} spin in the bath. In "matrix" form as a tensor product it is

$$\hat{H}_{int} = \frac{1}{2}\sigma_z \otimes \sum_{n=1}^{N} l_n \left(I \otimes \cdots \otimes \sigma_z \otimes \cdots \otimes I\right),$$

while in second quantization formalism

$$\hat{H}_{int} = \frac{1}{2} \sum_{n=1}^{N} l_n (2c^\dagger c - 1)(2c^{\dagger(n)} c^{(n)} - 1).$$

2.3 *Further Approximations*

Up to now, in the previously described Hamiltonian, the following assumptions are made to reduce the system in a plausible model: low temperatures and thus low energies, as well as weak and random inhomogeneous intra-bath spin coupling in an one–dimensional modified Ising model. Although the model might be able to describe in abstract terms an experimentally feasible open quantum system, further approximations are needed to make computational numerical simulation easier.

Even if it is generally understood how the behaviour of oscillator baths and spin baths is very different [24], [62], it has been shown that at the weak-coupling limit, spin baths can be efficiently mapped in oscillator baths [25], [63]. It has also been suggested that this might be the case even at the strong-coupling limit [64], even if this is not supported elsewhere in the literature. In this context, the work of Pekola et al. [22] is considered, to construct the quadratic effective Hamiltonian mapped to an oscillator bath, as previously tested in [65], and given in the second quantisation formalism as

$$\hat{H} = \hbar\omega_0 \hat{c}^\dagger \hat{c} + \sum_{n=1}^{N} \hbar\omega_n \hat{c}^\dagger \hat{c} + \sum_{n \neq m} J_{nm} \hat{c}^\dagger \hat{c} + \sum_{n=1}^{N} l_n (c c^{\dagger(n)} + c^{(n)} c^\dagger).$$

The first term is the Hamiltonian of the qubit, the second term is the Hamiltonian of the bath, the third term is the intrabath coupling, and the fourth is the coupling between the qubit and the bath. It is hypothesised that there is no alternating magnetic field ($\Delta = 0$) and the model is considered in the rotating wave approximation (i.e., ignoring the fast rotating terms). However, Mäkinen [23] has already shown, in terms of the generalised Gibbs ensemble (GGE), that the rotating wave approximation for the model can be dropped.

Due to the exponential growth of the matrix size of the tensor products for N bath particles (leaving in a 2^N-size Hilbert space), quantum field methods such as the algorithms of the density-matrix renormalisation group, the numerical renormalisation group or the matrix product states have been developed [66], [67]. Quantum processors and algorithms are also experimentally used to achieve similar aims [68]. However, these approaches are beyond the scope of this thesis. Instead, similar to the symmetry reduced example in [69] and the Schrödinger picture implementation of quadratic Hamiltonian by Pekola et al. [22] as justified in their appendices, the 2^N-size Hilbert space of the Hamiltonian used in this thesis is reduced to a N-size Hilbert space.

2.4 *Time Evolution*

Usually in open quantum models the evolution of the bath is "integrated out" providing an average of all spins to focus on the central qubit [27]. In contrast, exact diagonalisation [70] provides a way to also follow each individual spin of the bath [71]. First, the model is set in an initial state, which in the case of Pekola et al. [22] is with the qubit excited and the bath in the ground state. Then, the time evolution of the system and the bath is given by applying to the density-matrix of the total Hamiltonian, $\rho(t)$, the time evolution operator, $\hat{U}(t)$, as solution of the Liouville-von Neumann equation

$$\frac{d\rho}{dt} = -\frac{i}{\hbar}[\hat{H}, \rho].$$

The solution is given by:

$$\rho(t) = \hat{U}(t)\rho(0)\hat{U}^\dagger(t),$$

where $\rho(0)$ is the density-matrix at time $t = 0$ (i.e., the initial state). With respect to a time-independent Hamiltonian, the time evolution operator is denoted as:

$$\hat{U}(t) = e^{-\frac{i\hat{H}t}{\hbar}}.$$

As the model evolves, the qubit is expected to exchange energy and information with the bath. As a result, two distinct but related processes emergence:

- decoherence [72], which is the process by which the state of the qubit (being in superposition or not) becomes entangled with the various states of the particles in the bath. This implies spreading of the qubit's information to the bath.

- dissipation [73], which is the loss of energy of the qubit towards the bath, leading to relaxation (i.e., transitioning from an excited state to a lower energy state) and thermalisation [74], [75], [76] (i.e., coming into thermal equilibrium with the bath).

Following the system as well as each particle in the bath, Pekola et al. demonstrated thermalisation of the qubit, while the particles within the bath are moving towards a non-thermal distribution (i.e. relaxation) [22]. In addition, GGE can provide an alternative but equivalent description [74], [77] in more general terms that allows to drop the rotating wave approximation used in the Pekola et al. model [23]. This equivalence provides another indication of memory-preserving equilibration [78], [79], violating assumptions required for the Markovian scenario, even if it has been found to give equivalent results [22].

3. METHODS

This section outlines the process followed to optimise for parallelisation the code of the simulation of the model with the characteristics outlined in the preceding section. First, it describes the programming language and the hardware specifications. Then it gives the steps taken to follow good scientific computing practices. In the end, it presents the utilised parallelisation techniques.

3.1 *Programming Language*

The code is written in MATLAB [80], an interpreted programming language focused on vector and matrix calculations. Its syntax, which closely resembles actual mathematical notation, makes it a good choice for proof-of-concept scripting in science and engineering [81]. In addition, when the approach reaches a stage ready to transition from prototyping to broader deployment, the MATLAB Coder package [82] can generate compiled code for C and C++ programming languages. This can provide a more efficient solution in terms of execution speed and resource usage. Furthermore, the Parallel Computing Toolbox [83] provides a set of tools that simplify the process of parallelising MATLAB code without the need to dive into the programming of the CPUs and GPUs themselves.

3.2 *Hardware and Operating System Specifications*

For this thesis, two different environments are used: a local and a remote.

- The local environment is based on an AMD FX-8120 64-bit processor, at 3100 MHz with 4 physical cores and 8 logical processors, in a Gigabyte GA-990FXA-UD3 motherboard, with 40 GB DDR3 SDRAM and NVIDIA

GeForce GTX 1660 SUPER GPU with 1408 CUDA threads at 1830 MHz and 6 GB GDDR6 with 192-bit architecture.

The operating system is Microsoft Windows 10 Education, Version 10.0.19045 Build 19045 [84].

- The remote environment is known as Triton cluster in Aalto University. The specifications of its nodes and its GPUs are available in its documentation [85].

  The operating system is Red Hat Enterprise Linux 9 [86], with SLURM [87] as the scheduler and batch system.

3.3 *Good Scientific Computing Practices*

The thesis is designed to follow relevant principles of good scientific computing as described by Wilson et.al. [13].

3.3.1 *Data Management*

The legacy code that had been used in the previous studies [22], [23] was kindly shared in personal communication by Ilari Mäkinen. It is stored without changes as legacy code in a MATLAB Project that is created locally and backed up in a git repository [88] for the purpose of this thesis. Henceforth, the term 'repository' will denote the local MATLAB project along with the associated git repository as detailed in this paragraph.

3.3.2 *Software*

The legacy code [89] was provided in Live Code file format (.mlx) [90]. It is transformed into plain format (.m) due to known performance issues [91]. The code is analysed from a physics perspective to understand its structure and the purposes of its parts. To understand its performance and identify bottlenecks in its initial form, the help of the native MATLAB Profiler application [92] is utilised in the local environment, in the following way:

```
profile on
% Code to profile
profile viewer
```

The code is decomposed into functions to create a modular and reusable

form, with an overarching `run_all.m` script. The names of the functions and of the variables are given in a meaningful way. Documentation and usage examples are added for each function. A `README.md` as well as a `requirements.txt` files are included in the root directory of the repository.

3.3.3 *Collaboration*

While the code created for the present thesis is the work of a single researcher, it is designed in a way that makes easy joining of future collaborators. The `README.md` file as well as this thesis report provide an overview of the project which can help future users to navigate it. In the root directory of the repository are also included a `LICENSE` file describing the permissions of the code and the report, and a `CITATION` file with details about how to cite them.

3.3.4 *Project Organisation*

The project is organised in a structure reflecting the parallelisation approaches. Inside the main `src` folder the source code is organised in three directories: `modular` (only vector parallelisation), `GPU` (parallelisation using GPU), and `multicore` (shared memory parallelisation), with the running script `run_all.m` at the root of `src`. The report of the thesis is written using the cloud LaTeX [93] editor Overleaf [94]. The references are managed using the open-source application Zotero [95]. The flowchart is designed with yEd Graph Editor [96]. The present report, as well as all other documentation such as profiler reports, are stored to a `doc` folder in the root directory of the repository. The output of the code as experimentally performed is stored in the `output` folder, using meaningful names (for example `simulation_30000_25_GPU.txt` and `time_evolution_30000_25.png` reflecting $N = 30\,000$ spins and $Nr = 25$ iterations).

3.3.5 *Changes Tracking*

Before anything else, a GIT repository at the version control system of Aalto was created to host the project [88]. The GIT repository is mirrored locally and all changes are PGP signed. A manual changelog is kept during the entire length of the project and a `changelog` file is included in the root folder, to keep track of the changes that might follow after the first "stable" version of the code.

3.4 *Vector Parallelisation*

During the modular decomposition of the code, improvements regarding the vectorisation are implemented. Those include removing unnecessary operations, reducing the usage of for loops, and improving the efficiency of matrix construction.

3.5 *Shared Memory Parallelisation*

Distributed arrays are implemented in relevant functions for shared memory parallelisation. Gathering of the data to the main memory is kept to a minimum to avoid data transfer overhead. In addition, the iterations of the calculations are spread in multiple parallel workers with the usage of the parallel for loop `parfor`.

3.6 *GPU Parallelisation*

`gpuArray`'s are used for GPU parallelisation, with minimal gathering of the data similar to the shared memory parallelisation. In addition, `arrayfun` is used to implement vectorised calculations in GPU when needed.

3.7 *Combined Shared Memory and GPU Parallelisation*

Even if it is possible to implement a version using both shared memory and GPU parallelisation, initial tests highlight memory limitations. Those occur with models that can simulate successfully using separate shared memory or GPU parallelisation. Thus, the combined approach is not considered further.

3.8 *Distributed Memory Parallelisation*

While it is possible to use MATLAB code with a distributed memory parallelisation using MATLAB Parallel Server [97], this is not possible to test, since Triton does not currently have a license [98].

4. RESULTS

This section presents the main results of the thesis. First, it provides an analysis of the legacy code. Next, it describes the optimisation steps taken to create a modular code and to improve it compared to the legacy code. Then it gives performance comparisons for different parallelisation approaches in the local environment, as well as results of the testing in the cluster node. The section includes graphs and comparison tables.

4.1 *Analysis of the Legacy Code*

Running the legacy script [89] with the profiler in the local environment, with $N = 1500$ spins and $Nr = 25$ iterations, took ≈ 135 seconds. Table 4.1 shows the lines that took the longest time, as identified by the profiler. The summary of the line-by-line profiling analysis of the script, with the number of times each line is called and the time each line took to run, is available in the `doc` folder of the code repository. From the profiler analysis, it is evident that, as expected, the manipulation of large matrices (diagonalisation and multiplications) is the most intense part of the code and should be limited as much as possible. It is also possible to recognise time-consuming lines not justified by the physics or the flow of the script (for example, the bath Hamiltonian diagonalisation in line 56). Moreover, it can be seen that the physically important randomness in the construction of the total Hamiltonian and the iterations required for the statistics is partly compromised, since in fact the pseudorandom number generator is constantly using the same seed. Additionally, although the code is somewhat vectorised, there are structural aspects that could be enhanced concerning vectorisation.

4.2 *Modular Optimisation*

Based on the results of the profiling and the analysis of the underlying physics, the legacy code is split into separate functions used together with the running script (`run_all.m`). The functions are:

- `total_hamiltonian`, which constructs analytically the total Hamiltonian of the bath coupled to the qubit;
- `diagonal`, which diagonalises the total Hamiltonian;
- `time_evolution`, which time-evolves the initial density matrix and calculates the resulting populations;
- `GGE`, which calculates the numerical GGE prediction for the populations;
- `analytical`, which calculates an analytical solution for the GGE.

Figure 4.1 shows a logical flowchart of the modular code as utilised with the running script.

4.3 *General Optimisation*

Besides restructuring the script in modular format, further optimisation is carried out as considered necessary (e.g., implementing a new seed for the pseudorandom number generator in each run of the simulation, vectorising for loops), and the code is extensively commented inline. The optimised modular version, without shared memory or GPU parallelisation, runs with the profiler in the local environment for $N = 1500$ spins and $Nr = 25$ iterations in ≈ 88 seconds; therefore, an improvement at the level of $1/3, \approx 35\%$, compared to the legacy code.

Similarly to legacy code profiling, table 4.2 shows the lines required the most time, while the summary of the profiling is available in the `doc` folder of the repository. In the same folder are stored the profile summaries of the called functions.

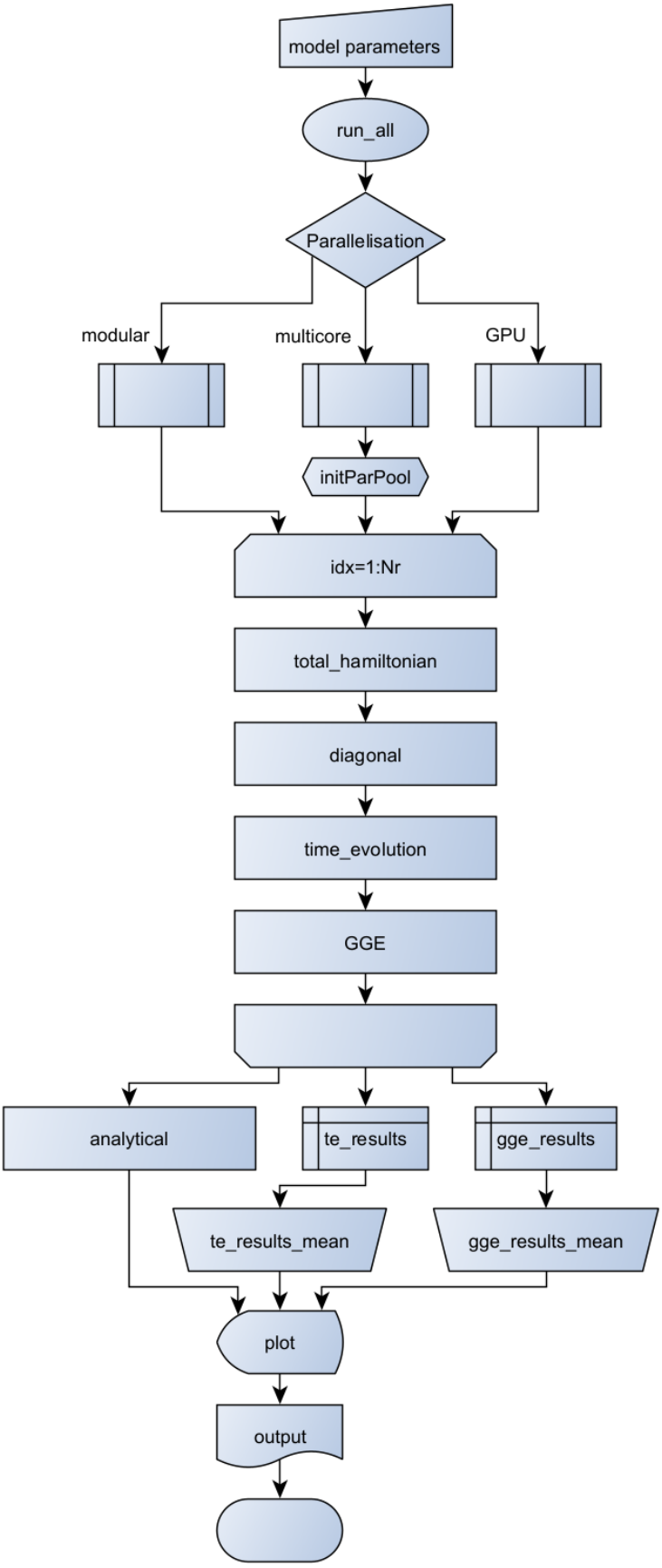


Figure 4.1. Logical flowchart of the optimised code

4.4 *Parallelisation*

The code is further optimised for parallel computation using multicore CPU and GPU. The functions `total_hamiltonian` and `analytical` are common for all types of parallelisation. The functions `diagonal`, `time_evolution`, and `GGE` are different and available in separate folders (`modular`, `multicore`, `GPU`). The running script `run_all.m` contains an "adapter" that chooses the version used in the simulation, as described in the inline comments.

Repeating the simulation for $N = 1500$ spins and $Nr = 25$ iterations, it takes ≈ 67 seconds with GPU parallelisation, representing a further $\approx 23\%$ improvement compared to the modular version. In contrast, with multicore parallelisation, the performance goes down to ≈ 165 seconds for the first run, and settles to ≈ 110 seconds for the next runs. As previously, profile summaries are available in the repository documentation. It is also important to note that, testing with a larger number of spins, a combination of multicore and GPU parallelisation does not seem to provide a benefit. This is due to fastly occurring `out of memory` errors from the side of the GPU, which is unable to handle the high feed from the parallel cores.

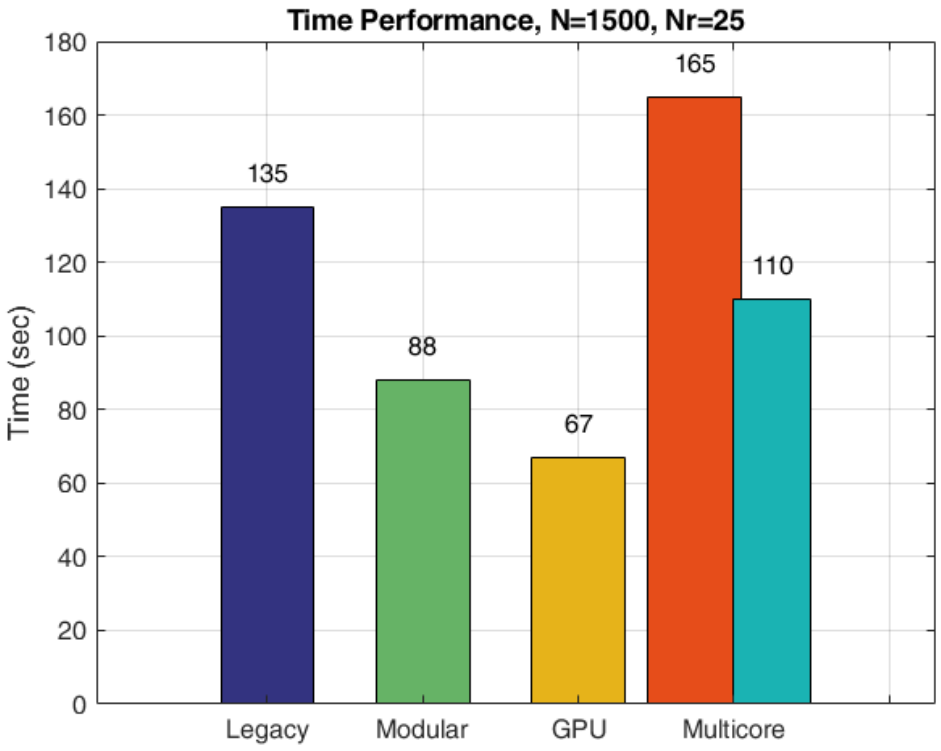


Figure 4.2. Time performance comparison for $N = 1500$, $Nr = 25$, local environment

4.5 *Triton Testing*

The testing of the simulation in Aalto's cluster Triton requires further optimisation of the code. For example, it is necessary to include a function `initParPool.m` to define how the parallel workers pool is created in multicore parallelisation.

A comparison of performance and resource usage for a simulation with $N = 10\,000$ spins and $Nr = 25$ iterations using the three different parallelisation options is presented in figure and table 4.3. GPU parallelisation is the most efficient approach, achieving $\approx 99\%$ improvement in time performance between the modular and the GPU versions. The single node limitation of the cluster, restricts the application to a single GPU, thus the limit is the GPU memory available to be utilised. Similarly, the multicore parallelisation can employ at best up to 80 cores and 2 TB of memory (in the `fn3` node). However, even with just 25 cores, the multicore implementation is unable to achieve the maximum number of spins that the GPU implementation can. A comparison of performance and resource usage for a simulation with $N = 20\,000$ spins and $Nr = 25$ iterations using multicore and GPU parallelisation is presented in table 4.4. Another comparison for a simulation with $N = 30\,000$ spins and $Nr = 25$ iterations, which utilise maximum GPU resources and leads to `out of memory` error in multicore parallelisation, is presented in table 4.5. Figure 4.4 shows the simulation results for $N = 20\,000$ spins and $Nr = 25$ iterations with multicore parallelisation, and Figure 4.5 with GPU parallelisation. Figures 4.6, 4.7, and 4.8 show the results of the GPU parallelisation simulations for $N = 22\,000$, $N = 25\,000$, and the maximum achieved $N = 30\,000$ spins, always with $Nr = 25$ iterations.

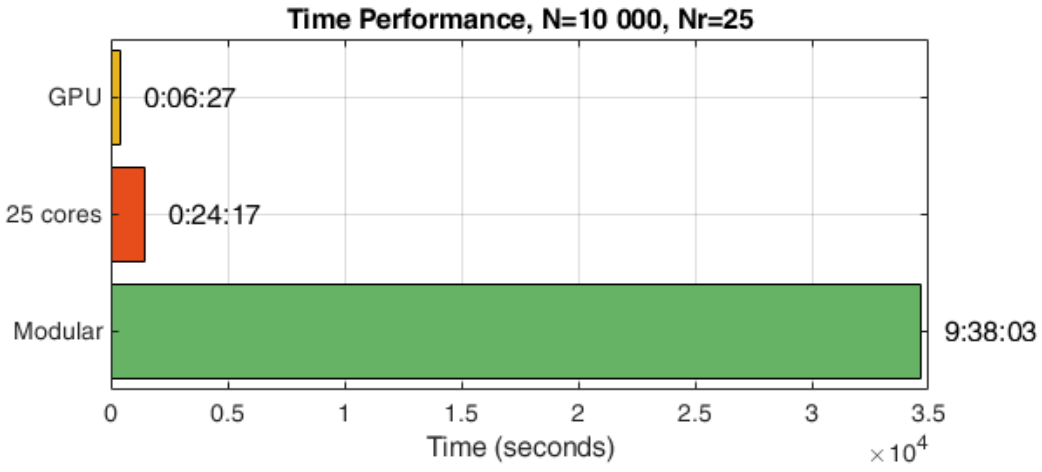


Figure 4.3. Time performance comparison for $N = 10\,000$, $Nr = 25$, Triton node

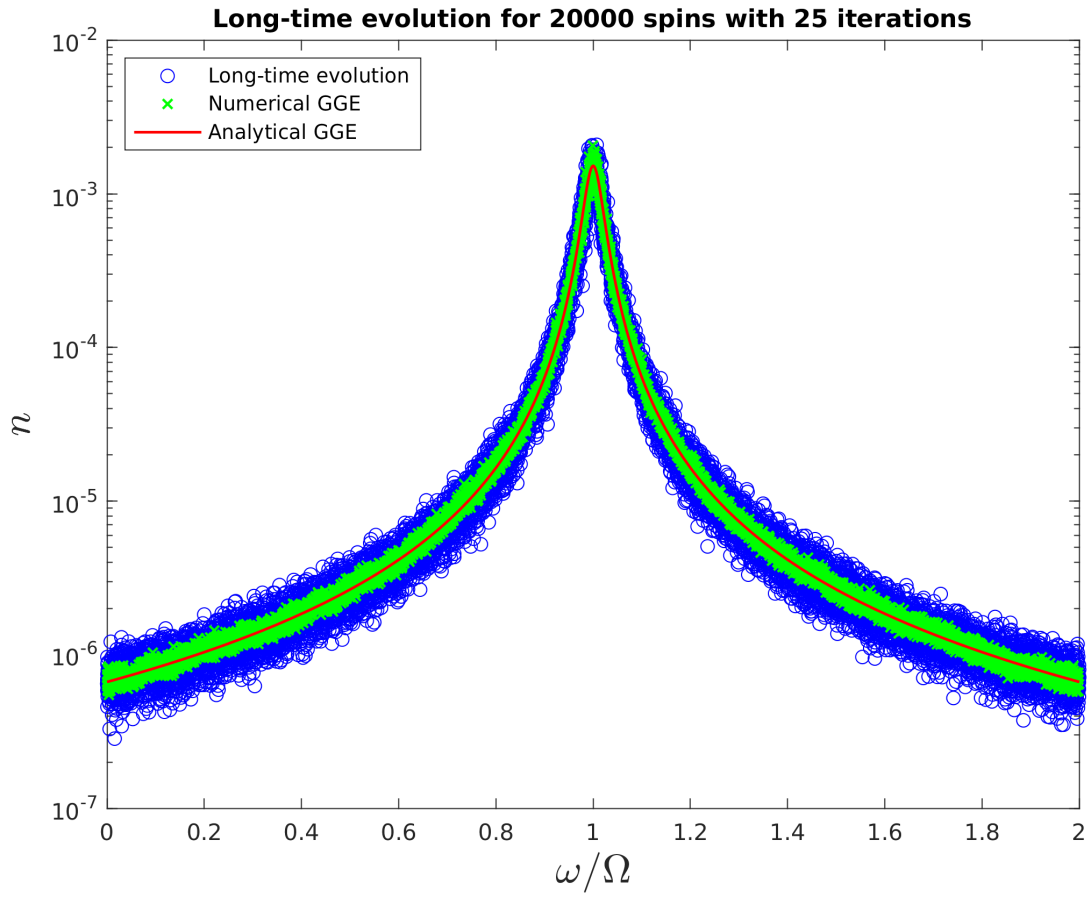


Figure 4.4. $N = 20\,000$, $Nr = 25$, multicore parallelisation

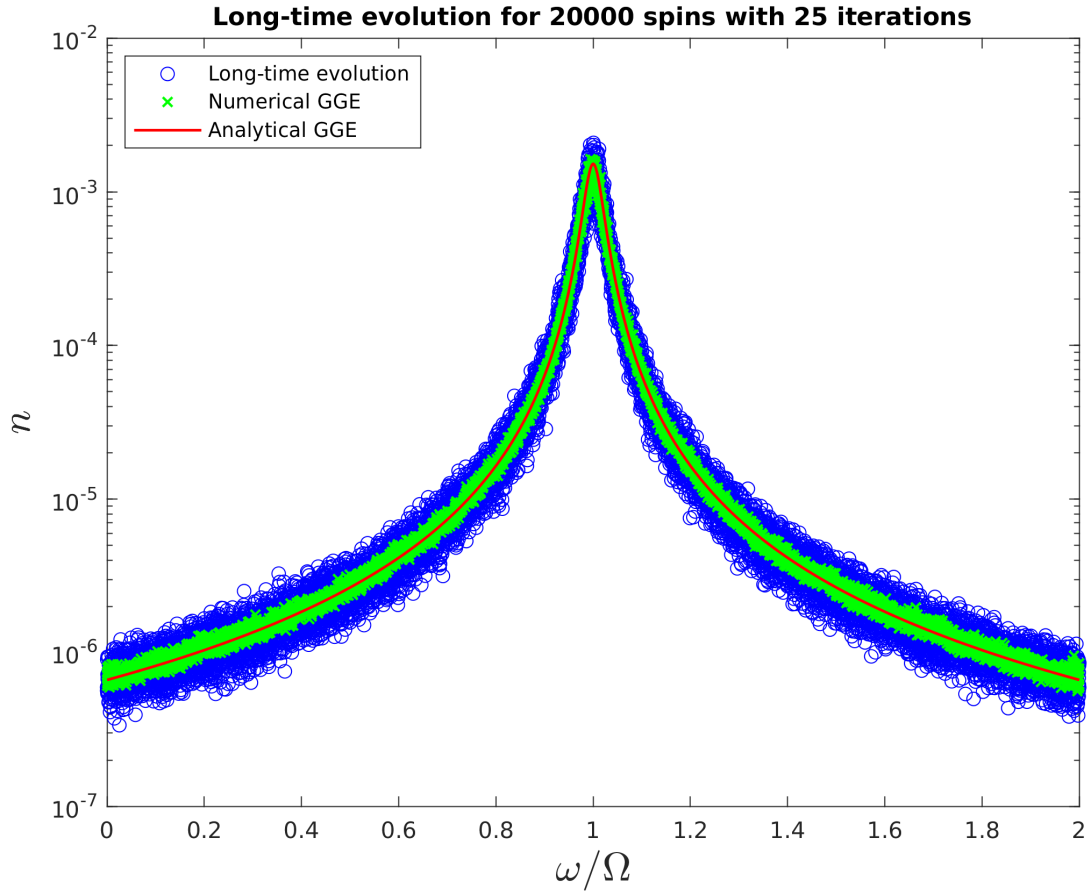


Figure 4.5. $N = 20\,000$, $Nr = 25$, GPU parallelisation

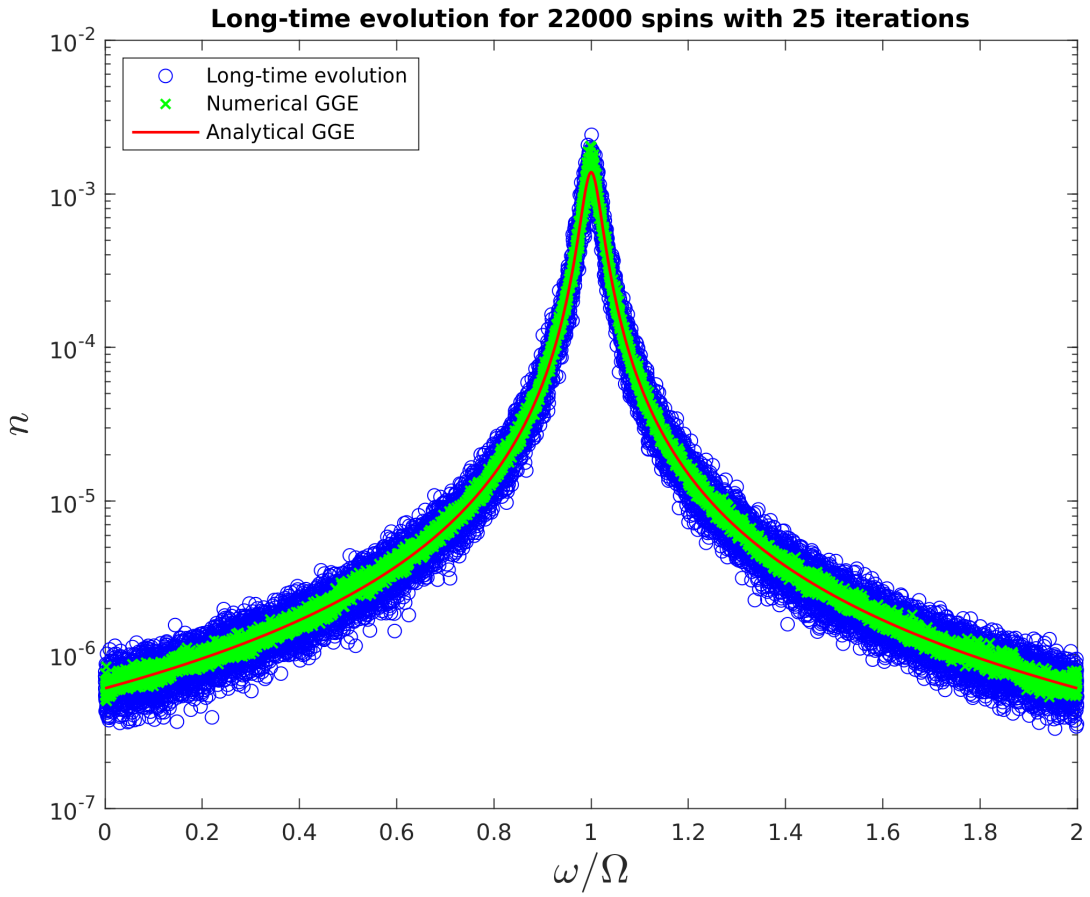


Figure 4.6. $N = 22\,000$, $Nr = 25$, GPU parallelisation

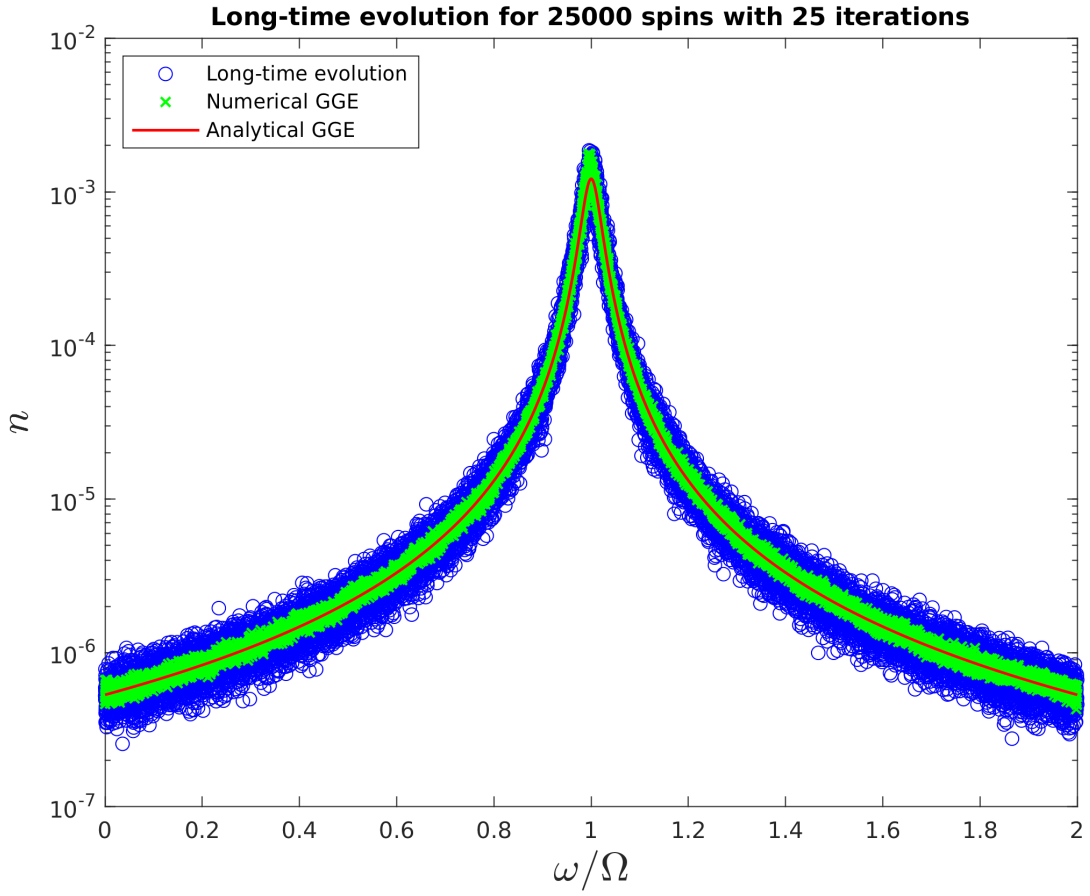


Figure 4.7. $N = 25\,000$, $Nr = 25$, GPU parallelisation

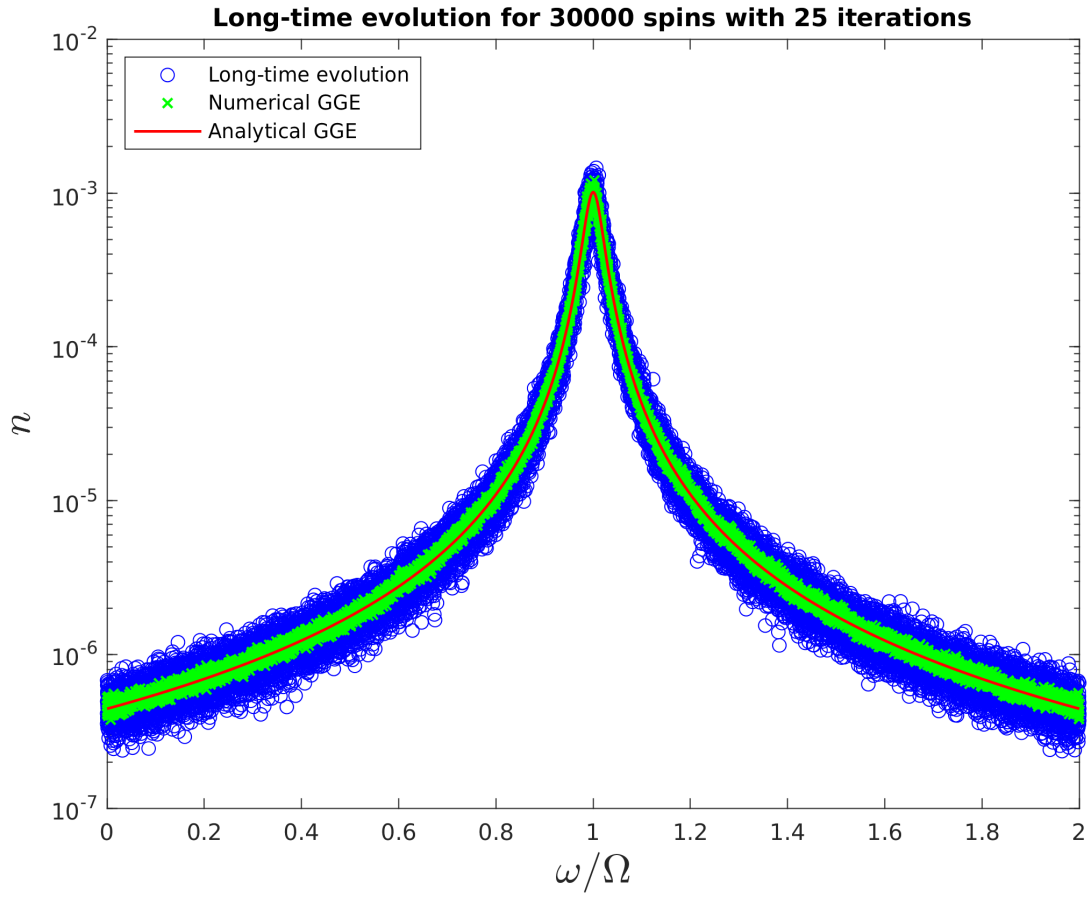


Figure 4.8. $N = 30\,000$, $Nr = 25$, GPU parallelisation

Line Number	Code	Calls	Total Time (s)	% Time
91	xi = (Uj')*xi0*Uj;	50	43.574	32.3%
90	Uj = vel*Ul*(vel');	50	34.778	25.8%
74	[vel, el] = eig(H);	25	23.789	17.6%
54	[vek1, ek1] = eig(H1);	25	23.752	17.6%
148	a1 = semilogy(dia1, te_result, 'o', "Color", 'b');	1	1.561	1.2%
All other lines			7.596	5.6%
Totals			**135.050**	**100%**

Table 4.1. Lines that took the most time in the original plain script at the local environment

Line Number	Code	Calls	Total Time (s)	% Time
105	E1 = time_evolution (N, hbar, tmax, vel, el, rho0);	25	56.154	63.8%
104	[vel, el] = diagonal (H);	25	24.980	28.4%
103	H = total_hamiltonian (N,w,mutual,gamma,omega$_j$);	25	2.746	3.1%
125	a1 = semilogy(omega_j, te_result$_{mean}$,'o',"$Color$",'b');	1	2.144	2.4%
134	legend([a1(1), a2(1), a3(1)], 'Long-time evolution', ...;	1	1.211	1.4%
All other lines			0.743	0.8%
Totals			**87.978**	**100%**

Table 4.2. Lines that took the most time in `run_all.m` at the local environment with a single core

Modular Job ID 1411817					
Node	**CPU**	**Cores Used**	**CPU Efficiency**	**Memory Utilised**	**Memory Requested**
pe15	2x12 core Xeon E5 2680 v3 2.50GHz	1	99.10%	9.52 GB (DDR4-2133)	64 GB
GPU	**GPU Memory**	**MATLAB Timer**	**Total Time**	**Triton Billing**	**Time Requested**
-	-	34642 sec	9:38:03	6	12 hours
Multicore Job ID 1416291					
Node	**CPU**	**Cores Used**	**CPU Efficiency**	**Memory Utilised**	**Memory Requested**
milan16	2x64 core AMD EPYC 7713 @2.0 GHz	25	90.11%	273.50 GB(DDR4-3200)	384 GB
GPU	**GPU Memory**	**MATLAB Timer**	**Total Time**	**Triton Billing**	**Time Requested**
-	-	1421 sec	0:24:17	76	4 hours
GPU Job ID 1413829					
Node	**CPU**	**Cores Used**	**CPU Efficiency**	**Memory Utilised**	**Memory Requested**
gpu17	2x24 core AMD EPYC 7413 @ 2.65GHz	1	96.38%	5.12 GB (DDR4-3200)	16 GB
GPU	**GPU Memory**	**MATLAB Timer**	**Total Time**	**Triton Billing**	**Time Requested**
Tesla A100 (7936 threads)	12 / 80 GB	355 sec	0:06:27	85	1 hour

Table 4.3. Simulation Performance and Resources Usage for $N = 10\,000$ spins and $Nr = 25$ iterations

Multicore Job ID 1454878					
Node	**CPU**	**Cores Used**	**CPU Efficiency**	**Memory Utilised**	**Memory Requested**
fn3	4x20 core Xeon Gold 6148 2.40GHz	25	88.07%	1004.81 GB(DDR4-2666)	2 TB
GPU	**GPU Memory**	**MATLAB Timer**	**Total Time**	**Triton Billing**	**Time Requested**
-	-	5324.7778 sec	1:29:21	400	6 hours
GPU Job ID 1428172					
Node	**CPU**	**Cores Used**	**CPU Efficiency**	**Memory Utilised**	**Memory Requested**
gpu48	2x48 core Xeon Platinum 8468 2.1GHz	2	83.64%	13.06 GB (DDR5-4800)	24 GB
GPU	**GPU Memory**	**MATLAB Timer**	**Total Time**	**Triton Billing**	**Time Requested**
Tesla H100 (16896 threads)	34 / 80 GB	1377.1732 sec	0:23:14	99	2 hour

Table 4.4. Simulation Performance and Resources Usage for $N = 20\,000$ spins and $Nr = 25$ iterations

Multicore Job ID 1458397 - FAILURE					
Node	**CPU**	**Cores Used**	**CPU Efficiency**	**Memory Utilised**	**Memory Requested**
fn3	4x20 core Xeon Gold 6148 2.40GHz	25	88.07%	1.95 TB(DDR4-2666)	2 TB
GPU	**GPU Memory**	**MATLAB Timer**	**Total Time**	**Triton Billing**	**Time Requested**
-	-	-	6:05:22	400	12 hours
GPU Job ID 1429067					
Node	**CPU**	**Cores Used**	**CPU Efficiency**	**Memory Utilised**	**Memory Requested**
gpu48	2x48 core Xeon Platinum 8468 2.1GHz	2	81.03%	28.02 GB (DDR5-4800)	48 GB
GPU	**GPU Memory**	**MATLAB Timer**	**Total Time**	**Triton Billing**	**Time Requested**
Tesla H100 (16896 threads)	79 / 80 GB	3329.074 sec	0:55:47	99	2 hour

Table 4.5. Simulation Performance and Resources Usage for $N = 30\,000$ spins and $Nr = 25$ iterations

5. DISCUSSION AND CONCLUSION

The aim of this thesis was to optimise the existing code that simulates the time evolution of a model consisting of a qubit coupled to a finite spin bath, in order for it to be used in parallel computing architectures and, thus, handle larger models. Using a local computing environment, legacy code was analysed and restructured in a modular format. This facilitated parallelisation optimisation, permitting broader vectorisation, and enabled exploitation of the parallel computing capabilities of the employed programming language. Three versions of code were produced, one using only vector parallelisation, one using multiple cores, and one using GPU.

Although promising, the initial tests in the local environment were mostly relevant to code optimisation, since the performance of the local environment can hardly benefit from parallelisation. In addition, those results were only indicative of the small size of spins in the bath, as was shown in the followed implementation in the cluster. When tested in the remote cluster environment, GPU parallelisation was found to be the most efficient, achieving the simulation of a model with ≈ 20ply number of spins in the bath compared to the legacy code. This is not surprising since GPU parallelisation is the most favourable approach for large matrices manipulation.

It is worth noting the considerable reduction in speed caused by the required build-in and overhead time needed for the initialisation of the workers pool when multicore parallelisation is used. This speed decrease makes the multicore approach less favourable for small baths, while the GPU parallelisation performs more efficiently compared to the other methods in every tested bath size.

Although, after being tested in two different computing environments, optimisation can be considered an overall success, this thesis has some considerable limitations. The lack of a licence for the MATLAB Parallel

Server in the remote cluster limits the application of the simulation in a single node only. This limitation restricts the size of the bath according to the maximum memory available in a node. Both multicore and GPU parallelisations are affected by this limitation. Similarly, it is not possible to test the applicability of the optimised code in a multinode environment. In addition, even if the modular structure of the code can theoretically facilitate its usage for the simulation of different models, the scope, as well as the time restrictions of this thesis, do not permit the actual testing of the code in models different from the one simulated in the legacy code.

Apart from the achievement of its declared aim, this thesis shows cases in which the computational modelling of quantum systems does not yet exploit the full potential of the existing classical computing capabilities, especially those of the parallel computing architectures. Future research could aim to investigate the extent of this potential, further expanding its limits. Different models could be attempted to be simulated by supplementing the code with additional functions or modifications. For example, the assumptions of the model and the approximations used could be modified, leading to different total Hamiltonian structures (e.g., with complex coupling coefficients). A model consisting of multiple qubits coupled to a single bath could be considered or a qubit coupled to multiple baths. Maybe the extreme test could be the consideration of a non-quadratically reduced model, requiring construction of a tensor-product-based full Hamiltonian. Finally, it remains to be seen how efficient this simulation could be when translated to compiled C or C++ code.

References

[1] M. L. Bellac, “Open quantum systems,” in *Quantum physics*, Cambridge, UK: Cambridge University Press, 2006, pp. 507–551.

[2] S. J. Blundell and K. M. Blundell, “Temperature and the Boltzmann factor,” in *Concepts in Thermal Physics*, 2nd ed., Oxford, UK: Oxford University Press, 2009, pp. 32–46. DOI: 10.1093/acprof:oso/9780199562091.003.0004.

[3] H. P. Breuer and F. Petruccione, “Quantum master equations,” in *The Theory of Open Quantum Systems*, Oxford, UK: Oxford University Press, 2007, pp. 109–218. DOI: 10.1093/acprof:oso/9780199213900.003.03.

[4] I. Rotter and J. P. Bird, “A review of progress in the physics of open quantum systems: Theory and experiment,” *Reports on Progress in Physics*, vol. 78, no. 11, Nov. 2015, Art. no: 114001. DOI: 10.1088/0034-4885/78/11/114001.

[5] I. Rotter, “A non-Hermitian Hamilton operator and the physics of open quantum systems,” *Journal of Physics A: Mathematical and Theoretical*, vol. 42, no. 15, Apr. 2009, Art. no: 153001. DOI: 10.1088/1751-8113/42/15/153001.

[6] U. Weiss, “Diverse limited approaches: A brief survey,” in *Quantum Dissipative Systems*, 4th ed., Singapore, Singapore: World Scientific, 2012, pp. 5–19. DOI: 10.1142/9789814374927_0002.

[7] D. Fernández de la Pradilla, E. Moreno, and J. Feist, “Recovering an accurate Lindblad equation from the Bloch-Redfield equation for general open quantum systems,” *Physical Review A*, vol. 109, no. 6, Jun. 2024, Art. no: 062225. DOI: 10.1103/PhysRevA.109.062225.

[8] F. Campaioli, J. H. Cole, and H. Hapuarachchi, “Quantum master equations: Tips and tricks for quantum optics, quantum computing,

and beyond," *PRX Quantum*, vol. 5, no. 2, Jun. 2024, Art. no: 020202. DOI: 10.1103/PRXQuantum.5.020202.

[9] M. J. de Oliveira, "Quantum Langevin equation," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2020, no. 2, Feb. 2020, Art. no: 023106. DOI: 10.1088/1742-5468/ab6de2.

[10] E. Ippoliti, T. Nickles, and F. Sterpetti, "Modeling and inferring in science," in *Models and Inferences in Science*, Cham, Switzerland: Springer International Publishing AG, 2016, pp. 1–9. DOI: 10.1007/978-3-319-28163-6_1.

[11] G. H. Golub and J. M. Ortega, "The world of scientific computing," in *Scientific computing and differential equations: an introduction to numerical methods*, 2nd ed., Cambridge, MA. USA: Academic press, 1992, pp. 1–14.

[12] A. Tolk, "Learning something right from models that are wrong: Epistemology of simulation," in *Concepts and Methodologies for Modeling and Simulation*, L. Yilmaz, Ed., Cham, Switzerland: Springer International Publishing AG, 2015, pp. 87–106. DOI: 10.1007/978-3-319-15096-3_5.

[13] G. Wilson, J. Bryan, K. Cranston, J. Kitzes, L. Nederbragt, and T. K. Teal, "Good enough practices in scientific computing," *PLOS Computational Biology*, vol. 13, no. 6, Jun. 2017, Art. no: e1005510. DOI: 10.1371/journal.pcbi.1005510.

[14] D. Padua, A. Ghoting, J. A. Gunnels, *et al.*, "Moore's law," in *Encyclopedia of Parallel Computing*, D. Padua, Ed., Boston, MA, USA: Springer US, 2011, pp. 1177–1184. DOI: 10.1007/978-0-387-09766-4_81.

[15] B. Leasure, D. J. Kuck, S. Gorlatch, *et al.*, "Power wall," in *Encyclopedia of Parallel Computing*, D. Padua, Ed., Boston, MA, USA: Springer US, 2011, pp. 1593–1608. DOI: 10.1007/978-0-387-09766-4_499.

[16] M. Själander, M. Martonosi, and S. Kaxiras, "Introduction," in *Power-Efficient Computer Architectures*, Cham, Switzerland: Springer International Publishing AG, 2015, pp. 1–10. DOI: 10.1007/978-3-031-01745-2_1.

[17] T. Weinzierl, "Moore myths," in *Principles of Parallel Scientific Computing*, Cham, Switzerland: Springer International Publishing AG, 2021, pp. 11–17. DOI: 10.1007/978-3-030-76194-3_2.

[18] Aalto Scientific Computing. “Parallel computing: Different methods explained,” Aalto Scientific Computing. (2024), [Online]. Available: https://scicomp.aalto.fi/triton/tut/parallel/ (visited on 06/11/2024).

[19] J. Peddie, “What is a GPU?” In *The History of the GPU - Steps to Invention*, Cham, Switzerland: Springer International Publishing AG, 2022, pp. 333–345. DOI: 10.1007/978-3-031-10968-3_7.

[20] G. Barlas, “GPU programming: CUDA,” in *Multicore and GPU Programming: an integrated approach*, 2nd ed., Burlington, MA, USA: Morgan Kaufmann, 2023, pp. 389–581. DOI: 10.1016/B978-0-12-814120-5.00015-9.

[21] Aalto Scientific Computing. “GPU computing,” Aalto Scientific Computing. (2024), [Online]. Available: https://scicomp.aalto.fi/triton/tut/gpu/ (visited on 06/11/2024).

[22] J. P. Pekola, B. Karimi, M. Cattaneo, and S. Maniscalco, “Long-time relaxation of a finite spin bath linearly coupled to a qubit,” *Open Systems & Information Dynamics*, vol. 30, no. 2, Jun. 2023, Art. no: 2350009. DOI: 10.1142/S1230161223500099.

[23] I. Mäkinen, *Generalized Gibbs ensemble in quadratic quantum systems with application to a finite spin bath*, unpublished.

[24] N. V. Prokof’ev and P. C. E. Stamp, “Theory of the spin bath,” *Reports on Progress in Physics*, vol. 63, no. 4, pp. 669–726, Apr. 2000. DOI: 10.1088/0034-4885/63/4/204.

[25] P. C. E. Stamp, “Quantum environments: Spins baths, oscillator baths, and applications to quantum magnetism,” in *Tunneling in Complex Systems*, S. Tomsovic, Ed., Singapore, Singapore: World Scientific, 1998, pp. 101–197. DOI: 10.1142/9789812796332_0003.

[26] M. Schlosshauer, “A world of spins and oscillators: Canonical models for decoherence,” in *Decoherence and the Quantum-To-Classical Transition*. Heidelberg, Germany: Springer, 2007, pp. 171–241. DOI: 10.1007/978-3-540-35775-9_5.

[27] M. Dubé and P. Stamp, “Mechanisms of decoherence at low temperatures,” *Chemical Physics*, vol. 268, no. 1, pp. 257–272, Jun. 2001. DOI: 10.1016/S0301-0104(01)00303-2.

[28] R. Feynman and F. Vernon, “The theory of a general quantum system interacting with a linear dissipative system,” *Annals of Physics*, vol. 24, pp. 118–173, Oct. 1963. DOI: 10.1016/0003-4916(63)90068-X.

[29] A. Caldeira and A. Leggett, “Quantum tunnelling in a dissipative system,” *Annals of Physics*, vol. 149, no. 2, pp. 374–456, Sep. 1983. DOI: 10.1016/0003-4916(83)90202-6.

[30] P. C. E. Stamp, “Environmental decoherence versus intrinsic decoherence,” *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 370, no. 1975, pp. 4429–4453, Sep. 2012. DOI: 10.1098/rsta.2012.0162.

[31] M. A. Nielsen and I. L. Chuang, “Quantum bits,” in *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge, UK: Cambridge University Press, 2012, pp. 13–17. DOI: 10.1017/CBO9780511976667.005.

[32] J. C. Bardin, D. H. Slichter, and D. J. Reilly, “Microwaves in quantum computing,” *IEEE Journal of Microwaves*, vol. 1, no. 1, pp. 403–427, Jan. 2021. DOI: 10.1109/JMW.2020.3034071.

[33] F. Jazaeri, A. Beckers, A. Tajalli, and J.-M. Sallese, “A review on quantum computing: From qubits to front-end electronics and cryogenic MOSFET physics,” in *2019 MIXDES - 26th International Conference* "*Mixed Design of Integrated Circuits and Systems*", Rzeszów, Poland: IEEE, Jun. 2019, pp. 15–25. DOI: 10.23919/MIXDES.2019.8787164.

[34] A. Chatterjee, P. Stevenson, S. De Franceschi, A. Morello, N. P. De Leon, and F. Kuemmeth, “Semiconductor qubits in practice,” *Nature Reviews Physics*, vol. 3, no. 3, pp. 157–177, Feb. 2021. DOI: 10.1038/s42254-021-00283-9.

[35] W. Uddin, B. Khan, S. Dewan, and S. Das, “Silicon-based qubit technology: Progress and future prospects,” *Bulletin of Materials Science*, vol. 45, Mar. 2022, Art. no: 46. DOI: 10.1007/s12034-021-02621-0.

[36] P. Krantz, M. Kjaergaard, F. Yan, T. P. Orlando, S. Gustavsson, and W. D. Oliver, “A quantum engineer’s guide to superconducting qubits,” *Applied Physics Reviews*, vol. 6, no. 2, Jun. 2019, Art. no: 021318. DOI: 10.1063/1.5089550.

[37] P. Krantz, M. Kjaergaard, F. Yan, T. P. Orlando, S. Gustavsson, and W. D. Oliver, *A quantum engineer’s guide to superconducting qubits*, 2021. arXiv: 1904.06560v5.

[38] W. Dür and S. Heusler, “Visualization of the invisible: The qubit as key to quantum physics,” *The Physics Teacher*, vol. 52, no. 8, pp. 489–492, Nov. 2014. DOI: 10.1119/1.4897588.

[39] C. Cohen-Tannoudji, B. Diu, and F. Laloë, *Quantum mechanics: Basic Concepts, Tools, and Applications, vol.1*, 2nd ed. Weinheim, Germany: Wiley-VCH Verlag GmbH & Co, 2020.

[40] I. Bialynicki-Birula and T. Sowiński, "Quantum electrodynamics of qubits," *Physical Review A*, vol. 76, no. 6, Dec. 2007, Art. no: 062106. DOI: 10.1103/PhysRevA.76.062106.

[41] J. S. Smart, "Evaluation of exchange interactions from experimental data," in *Spin Arrangements and Crystal Structure, Domains, and Micromagnetics*, ser. Magnetism: A Treatise on Modern Theory and Materials, vol. 3, G. T. Rado and H. Suhl, Eds., Cambridge, MA. USA: Academic Press, 1963, pp. 63–114. DOI: 10.1016/B978-0-12-575303-6.50009-9.

[42] P. R. Surján, "Introduction," in *Second Quantized Approach to Quantum Chemistry*. Heidelberg, Germany: Springer, 1989, pp. 1–5. DOI: 10.1007/978-3-642-74755-7_1.

[43] P. R. Surján, "Advantages of second quantization—illustrative examples," in *Second Quantized Approach to Quantum Chemistry*. Heidelberg, Germany: Springer, 1989, pp. 40–45. DOI: 10.1007/978-3-642-74755-7_6.

[44] M. Majeed and A. Z. Chaudhry, "Effect of initial system – environment correlations with spin environments," *The European Physical Journal D*, vol. 73, Jan. 2019, Art. no: 16. DOI: 10.1140/epjd/e2018-90416-0.

[45] R. P. Feynman, R. B. Leighton, and M. L. Sands, "The Hamiltonian of a spin one-half particle in a magnetic field," in *Quantum mechanics*, ser. The Feynman lectures on physics, vol. 3, Boston, MA, USA: Addison-Wesley, 2007, pp. 10.22–10.26.

[46] K. W. H. Stevens, "Spin Hamiltonians," in *Magnetic ions in insulators, their interactions, resonances and optical properties*, ser. Magnetism: A Treatise on Modern Theory and Materials, vol. 1, G. T. Rado and H. Suhl, Eds., Cambridge, MA. USA: Academic Press, 1963, pp. 1–23.

[47] T. A. Littlefield and N. Thorley, "The Zeeman effect," in *Atomic and Nuclear Physics*, Boston, MA, USA: Springer US, 1979, pp. 183–195. DOI: 10.1007/978-1-4684-1470-7_13.

[48] P. Grivet and L. Malnar, "Measurement of weak magnetic fields by magnetic resonance," *Advances in Electronics and Electron Physics*, vol. 23, pp. 39–151, 1967. DOI: 10.1016/S0065-2539(08)60060-8.

[49] R. P. Feynman, R. B. Leighton, and M. L. Sands, “The Pauli spin matrices,” in *Quantum mechanics*, ser. The Feynman lectures on physics, vol. 3, Boston, MA, USA: Addison-Wesley, 2007, pp. 11.1–11.9.

[50] Y. V. Nazarov and J. Danon, “Second quantization for fermions,” in *Advanced quantum mechanics: A practical guide*, Cambridge, UK: Cambridge University Press, 2013, pp. 74–79.

[51] R. M. White, “The magnetic Hamiltonian,” in *Quantum Theory of Magnetism*, Heidelberg, Germany: Springer, 2007, pp. 33–83. DOI: 10.1007/978-3-540-69025-2_2.

[52] L.-M. Duan, E. Demler, and M. D. Lukin, “Controlling spin exchange interactions of ultracold atoms in optical lattices,” *Physical Review Letters*, vol. 91, no. 9, Aug. 2003, Art. no: 090402. DOI: 10.1103/PhysRevLett.91.090402.

[53] S. Camalet and R. Chitra, “Effect of random interactions in spin baths on decoherence,” *Physical Review B*, vol. 75, no. 9, Mar. 2007, Art. no: 094434. DOI: 10.1103/PhysRevB.75.094434.

[54] J. Lages, V. V. Dobrovitski, M. I. Katsnelson, H. A. De Raedt, and B. N. Harmon, “Decoherence by a chaotic many-spin bath,” *Physical Review E*, vol. 72, no. 2, Aug. 2005, Art. no: 026225. DOI: 10.1103/PhysRevE.72.026225.

[55] A. Hutton and S. Bose, “Mediated entanglement and correlations in a star network of interacting spins,” *Physical Review A*, vol. 69, no. 4, Apr. 2004, Art. no: 042312. DOI: 10.1103/PhysRevA.69.042312.

[56] S. G. Brush, “History of the Lenz-Ising model,” *Reviews of Modern Physics*, vol. 39, no. 4, pp. 883–893, Oct. 1967. DOI: 10.1103/RevModPhys.39.883.

[57] G. Grosso and G. P. Parravicini, “Magnetic ordering in crystals,” in *Solid State Physics*, Cambridge, MA. USA: Academic Press, 2000, pp. 619–662. DOI: 10.1016/B978-012304460-0/50017-7.

[58] L. Roelofs, “Phase transitions and kinetics of ordering,” in *Physical Structure*, ser. Handbook of Surface Science, W. Unertl, Ed., vol. 1, Amsterdam, The Netherlands: Elsevier Science, 1996, pp. 713–807. DOI: 10.1016/S1573-4331(96)80018-7.

[59] H. E. Stanley, “Scaling, universality, and renormalization: Three pillars of modern critical phenomena,” *Reviews of Modern Physics*, vol. 71, no. 2, S358–S366, Mar. 1999. DOI: 10.1103/RevModPhys.71.S358.

[60] H. Kawamura, “Two models of spin glasses — Ising versus Heisenberg,” *Journal of Physics: Conference Series*, vol. 233, Jun. 2010, Art. no: 012012. DOI: 10.1088/1742-6596/233/1/012012.

[61] K. Huang, “The Ising model,” in *Statistical mechanics*, 2nd ed., Hoboken, NJ, USA: Wiley, 1987, pp. 341–367.

[62] D. Gelman, C. P. Koch, and R. Kosloff, “Dissipative quantum dynamics with the surrogate Hamiltonian approach. A comparison between spin and harmonic baths,” *The Journal of Chemical Physics*, vol. 121, no. 2, pp. 661–671, Jul. 2004. DOI: 10.1063/1.1759312.

[63] A. O. Caldeira, A. H. Castro Neto, and T. Oliveira de Carvalho, “Dissipative quantum systems modeled by a two-level-reservoir coupling,” *Physical Review B*, vol. 48, no. 18, pp. 13 974–13 976, Nov. 1993. DOI: 10.1103/PhysRevB.48.13974.

[64] S. M. H. Halataei, *Caldeira-Leggett oscillator bath simulation of the Prokof’ev-Stamp spin bath in strong coupling limits*, 2021. arXiv: 2108.04915.

[65] J. P. Pekola and B. Karimi, “Ultrasensitive calorimetric detection of single photons from qubit decay,” *Physical Review X*, vol. 12, no. 1, Feb. 2022, Art. no: 011026. DOI: 10.1103/PhysRevX.12.011026.

[66] F. Verstraete, V. Murg, and J. Cirac, “Matrix product states, projected entangled pair states, and variational renormalization group methods for quantum spin systems,” *Advances in Physics*, vol. 57, no. 2, pp. 143–224, Mar. 2008. DOI: 10.1080/14789940801912366.

[67] U. Schollwöck, “The density-matrix renormalization group in the age of matrix product states,” *Annals of Physics*, vol. 326, no. 1, pp. 96–192, Jan. 2011. DOI: 10.1016/j.aop.2010.09.012.

[68] N. Yoshioka, M. Amico, W. Kirby, *et al.*, *Diagonalization of large many-body Hamiltonians on a quantum processor*, 2024. arXiv: 2407.14431.

[69] D. Gitman, M. C. Baldiotti, and M. M. Santos, “Two and three coupled spins in magnetic field,” in *Proceedings of Third International Satellite Conference on Mathematical Methods in Physics — PoS(ICMP 2013)*, Londrina - PR (Brazil), Apr. 2014. DOI: 10.22323/1.200.0009.

[70] A. Weiße and H. Fehske, “Exact diagonalization techniques,” in *Computational Many-Particle Physics*, H. Fehske, R. Schneider, and A. Weiße, Eds., Heidelberg, Germany: Springer, 2008, pp. 529–544. DOI: 10.1007/978-3-540-74686-7_18.

[71] A. S. Aramthottil, T. Chanda, P. Sierant, and J. Zakrzewski, “Finite-size scaling analysis of the many-body localization transition in quasiperiodic spin chains,” *Physical Review B*, vol. 104, no. 21, Dec. 2021, Art. no: 214201. DOI: 10.1103/PhysRevB.104.214201.

[72] M. Schlosshauer, “The basic formalism and interpretation of decoherence,” in *Decoherence and the Quantum-To-Classical Transition*, Heidelberg, Germany: Springer, 2007, pp. 13–114. DOI: 10.1007/978-3-540-35775-9_2.

[73] U. Weiss, “Introduction,” in *Quantum Dissipative Systems*, 4th ed., Singapore, Singapore: World Scientific, 2012, pp. 1–4. DOI: 10.1142/9789814374927_0001.

[74] M. Rigol, V. Dunjko, and M. Olshanii, “Thermalization and its mechanism for generic isolated quantum systems,” *Nature*, vol. 452, no. 7189, pp. 854–858, Apr. 2008. DOI: 10.1038/nature06838.

[75] G. Kurizki and A. G. Kofman, “Equilibration of large quantum systems,” in *Thermodynamics and Control of Open Quantum Systems*, Cambridge, UK: Cambridge University Press, 2021, pp. 3–13. DOI: 10.1017/9781316798454.002.

[76] T. Mori, T. N. Ikeda, E. Kaminishi, and M. Ueda, “Thermalization and prethermalization in isolated quantum systems: A theoretical overview,” *Journal of Physics B: Atomic, Molecular and Optical Physics*, vol. 51, no. 11, Jun. 2018, Art. no: 112001. DOI: 10.1088/1361-6455/aabcdf.

[77] M. Rigol, V. Dunjko, V. Yurovsky, and M. Olshanii, “Relaxation in a completely integrable many-body quantum system: An *Ab Initio* study of the dynamics of the highly excited states of 1d lattice hard-core bosons,” *Physical Review Letters*, vol. 98, no. 5, Feb. 2007, Art. no: 050405. DOI: 10.1103/PhysRevLett.98.050405.

[78] F. H. L. Essler and M. Fagotti, “Quench dynamics and relaxation in isolated integrable quantum spin chains,” *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2016, no. 6, Jun. 2016, Art. no: 064002. DOI: 10.1088/1742-5468/2016/06/064002.

[79] S. Sotiriadis, “Memory-preserving equilibration after a quantum quench in a one-dimensional critical model,” *Physical Review A*, vol. 94, no. 3, Sep. 2016, Art. no: 031605. DOI: 10.1103/PhysRevA.94.031605.

[80] The MathWorks Inc., *MATLAB*, version 24.1.0.2603908 (R2024a) Update 3, 2024. [Online]. Available: https://www.mathworks.com.

[81] D. Stanescu and L. Lee, “Introduction,” in *A Gentle Introduction to Scientific Computing*, New York, NY, USA: Chapman and Hall/CRC, 2022, pp. 1–4. DOI: 10.1201/9780429262876.

[82] The MathWorks Inc., *MATLAB coder*, version 24.1 (R2024a), 2024. [Online]. Available: https://mathworks.com/help/coder/index.html.

[83] The MathWorks Inc., *Parallel computing toolbox*, version 24.1 (R2024a), 2024. [Online]. Available: https://se.mathworks.com/help/parallel-computing/index.html.

[84] Microsoft Corporation, *Microsoft windows 10 education*, version 10.0.19045 Build 19045, 2024. [Online]. Available: https://www.microsoft.com/.

[85] Aalto Scientific Computing. “Triton quick reference,” Aalto Scientific Computing. (2024), [Online]. Available: https://scicomp.aalto.fi/triton/ref/ (visited on 06/11/2024).

[86] Ivan Tervanto. “Triton major upgrade / spring 2024 (#1593),” GitLab. (Apr. 16, 2024), [Online]. Available: https://version.aalto.fi/gitlab/AaltoScienceIT/triton/-/issues/1593 (visited on 06/12/2024).

[87] SchedMD, *Slurm workload manager - documentation*, version 24.05, 2024. [Online]. Available: https://slurm.schedmd.com/documentation.html.

[88] Tsiormpatzis Stergios, *ParallelSimulations_finitebath · GitLab*, Jul. 15, 2024. [Online]. Available: https://version.aalto.fi/gitlab/tsiorms1/parallelsimulations_finitebath (visited on 07/15/2024).

[89] Ilari Mäkinen, *Spontaneous emission · GitLab*, Jul. 15, 2024. [Online]. Available: https://version.aalto.fi/gitlab/tsiorms1/parallelsimulations_finitebath/-/blob/main/original/spontaneous_emission_ORIGINAL_.mlx.

[90] The MathWorks Inc. “Live code file format (.mlx).” (), [Online]. Available: https://mathworks.com/help/matlab/matlab_prog/live-script-file-format.html (visited on 07/15/2024).

[91] The MathWorks Inc. “Code in live script run much slow,” MATLAB Answers. (), [Online]. Available: https://mathworks.com/matlabcentral/answers/312039-code-in-live-script-run-much-slow (visited on 07/15/2024).

[92] The MathWorks Inc., *Profiler*, version R2024a, 2024. [Online]. Available: https://mathworks.com/help/matlab/ref/profiler-app.html.

[93] The LaTeX Project, *LaTeX*, 2024. [Online]. Available: https://github.com/latex3/latex2e.

[94] Digital Science UK Ltd, *Overleaf*, version Server Pro 5.0.3, 2024. [Online]. Available: https://www.overleaf.com/.

[95] Corporation for Digital Scholarship, *Zotero*, Jul. 15, 2024. [Online]. Available: https://github.com/zotero/zotero.

[96] yWorks GmbH, *yEd graph editor*, version 3.24, 2024. [Online]. Available: https://www.yworks.com/products/yed.

[97] The MathWorks Inc., *MATLAB parallel server*, 2024. [Online]. Available: https://se.mathworks.com/products/matlab-parallel-server.html.

[98] Aalto Scientific Computing. "Matlab," Aalto Scientific Computing. (2024), [Online]. Available: https://scicomp.aalto.fi/triton/apps/matlab/ (visited on 06/12/2024).

A. Common Code

A.1 run_all.m

```
% A script that can be used to run the simulation of N spins bath with
% various types of parallelisation (modular vectorised, GPU parallel, multicore
% CPU parallel). It should run from inside the src folder.

% Reset the system
clearvars
close all
clc

% Enable long format for higher accuracy in the calculations
format long

% Initialize the pseudo-random number generator based on the current time
rng("shuffle");

% Define parallelisation type. Accepted values are 'modular', 'GPU',
% 'multicore'.
type = 'modular';

% Add the folders of the chosen parallelisation in the path
addpath(fullfile(pwd, type));

% Begin timing
tic;
```

```
% Defining example variables of the problem

% The total number of two level systems (TLSs) in the bath.
% The intially excited state, the qubit, is not considered to be
% part of the bath. Therefore N+1 is the overall number of TLSs
N = 1500;

% Number of independent, random iterations, to get a statistic
Nr = 25;

% The frequency of the qubit.
% Take it normalized to 1 for simpler calculations
w = 1;

% The reduced Planck's constant.
% Take it normalized to 1 for simpler calculations
hbar = 1;

% A flag that indicates the consideration of internal
% couplings of the TLSs in the bath. Use 0 for no
% internal coupling, 1 to include internal coupling
mutual = 1;

% Sets the magnitude of the internal coupling strength.
% For weak coupling regime, smaller of the frequency of the qubit.
% Taken to be w/(5*sqrt(2)) in the example case.
gamma = w/(5*sqrt(2));

% The final time at which the populations are calculated.
tmax = 8000000000;

% Construct a N-by-1 column vector with (sorted) uniformly distributed
% random numbers in [0, 2*hbar*w]. It will be the diagonal elements of
% the bath Hamiltonian, representing the energy levels hbar*frequencies
% of the spins of the bath hbar*omega_j where j is in [1, N].
% The energy levels are sorted to reflect the ordered energy spectrum of
% the physical systems.
% This is a constant random vector during the iterations.
```

```
omega_j = sort(2*hbar*w*rand(N,1));

% The initial state of the system with the bath in the ground state
% and the qubit excited
rho0 = zeros(N+1);
rho0(N+1, N+1) = 1;

% The array to collect the results of long time evolution
te_results = zeros(N, Nr);

% The array to collect the results of the GGE prediction
gge_results = zeros(N+1, Nr);

% Decision based on the parallelisation type choosen at line 18.
if strcmp(type, 'multicore')
    % Initiate the parallel poll. The default is for usage in Triton.
    % In local environment uncomment the next line...
    % parpool
    % ...and comment the next line.
    initParPool()
    % Initialize the pseudo-random number generator with the Multiplicative
    % lagged Fibonacci generator, for multiple workers in parallel
    s = RandStream.create('mlfg6331_64','NumStreams', Nr,'Seed',...
    'shuffle', 'CellOutput',true);
    % Iterrate Nr times with Nr separate parallel workers
    parfor idx = 1:Nr
    % Different pseudo-random generator seed for each parallel worker
    RandStream.setGlobalStream(s{idx});
    % Construct total Hamiltonian
    H = total_hamiltonian (N,w,mutual,gamma, omega_j);
    % Diagonalise it
    [vel, el] = diagonal (H);
    % Time evolution
    E1 = time_evolution (N, hbar, tmax, vel, el, rho0);
    % Numerical GGE prediction
    nau = GGE (N, vel);
    % Results per iteration in the arrays
    te_results(:, idx) = E1;
```

```
    gge_results(:, idx) = nau;
    end
else
    % Iterrate Nr times
    for idx = 1:Nr
    % Construct total Hamiltonian
    H = total_hamiltonian (N,w,mutual,gamma, omega_j);
    % Diagonalise it
    [vel, el] = diagonal (H);
    % Time evolution
    E1 = time_evolution (N, hbar, tmax, vel, el, rho0);
    % Numerical GGE prediction
    nau = GGE (N, vel);
    % Results per iteration in the arrays
    te_results(:, idx) = E1;
    gge_results(:, idx) = nau;
    end
end

% Get the statistic (mean of the iterations)
te_results_mean = sum(te_results, 2) / Nr;
gge_results_mean = sum(gge_results, 2) / Nr;

% The analytical GGE prediction for the populations
[nl, omega] = analytical (N, w, gamma);

% Plotting
% (i) Numerical long-time evolution
% (ii) Numerical GGE
% (iii) Analytical

a1 = semilogy(omega_j, te_results_mean, 'o', "Color", 'b');
hold on
a2 = plot(omega_j, gge_results(1:N), 'x', "LineWidth", 1.1, "Color","g");
a3 = plot(omega, nl, "LineWidth", 1.2, "Color", "r");

out1 = sprintf('Long-time evolution for %d spins with %d iterations', N, Nr);
xlabel("$\omega/\Omega$", 'Interpreter',"latex", 'FontSize',18)
```

```
ylabel("$n$", 'Interpreter',"latex", 'FontSize',18)
title(out1);
legend([a1(1), a2(1), a3(1)], 'Long-time evolution', 'Numerical GGE', ...
    'Analytical GGE', 'location', "northwest")
%ylim([0.5*10^(-5),10^(-1)])
hold off

% Save the image
relativeFolder = 'output';
filename = sprintf('time_evolution_%d_%d.png', N, Nr);
fullFolderPath = fullfile(pwd, relativeFolder);
fullFilePath = fullfile(fullFolderPath, filename);

% Ensure the directory exists
if ~exist(fullFolderPath, 'dir')
    mkdir(fullFolderPath);
end

% Define characteristics for the image
exportgraphics(gcf, fullFilePath, 'Resolution', 300);

% If multicore at a local environment
% uncoment the following line to delete the workers pool
% delete(gcp('nocreate'));

% Output display
disp('The simulation for')
disp(out1)
disp(['was completed in:', ' ', num2str(toc), ' seconds'])
disp(['using parallelisation type', ' ', type])
disp(['with', ' ', getenv('SLURM_CPUS_PER_TASK'), ' ', 'CPUs'])
```

A.2 total_hamiltonian.m

```
% A function that construct the total Hamiltonian of a single
% qubit with energy gap hbar*w coupled to a finite bath formed of N spins
% with energy gaps ranging between 0 and 2*hbar*w.
```

```
%
% Input variables:
%
% N         The total number of two level systems (TLSs) in the bath.
%           The intially excited state, the qubit, is not considered to be
%           part of the bath. Therefore N+1 is the overall number of TLSs
%
% w         The frequency of the qubit.
%           Take it normalized to 1 for simpler calculations
%
%
% mutual    A flag that indicates the consideration of internal
%           couplings of the TLSs in the bath. Use 0 for no
%           internal coupling, 1 to include internal coupling
%
% gamma     Sets the magnitude of the internal coupling strength.
%           Taken to be w/(5*sqrt(2)) in the example case.
%
% omega_j   a vector with the energy levels of the spins
%
% Output:
%
% H         the total hamiltonian matrix


function H = total_hamiltonian (N, w, mutual, gamma, omega_j)

% BATH

% Constructs an upper triangular N-by-N matrix of uniformly distributed
% random numbers between -(gamma/sqrt(N)) and (gamma/sqrt(N)).
% It represents the normalized by 1/sqrt(N) coupling strength between the
% spins on the bath, the off-diagonal elements of the Hamiltonian matrix.
% Thanks to the symmetry of the coupling between two spins and randomness,
% it is enough to take only the upper triangular matrix.
% g is relevant only if internal coupling is to be considered for the bath
% model (mutual=1), otherwise (mutual=0) it becomes zero.
g = mutual*(triu(-(gamma/sqrt(N)) + 2*(gamma/sqrt(N))*rand(N),1));
```

```
% Constructs a symmetric matrix of coupling strengths, by taking the sum of
% the upper triangular coupling strength matrix and its transpose. The
% (Hermitian this way) Hamiltonian is constructed with diagonal elements
% being zero.
H1 = g+g';

% Correct the Hamiltonian by replacing its diagonal elements with the
% energy gaps of the spins of the bath.
H1(1:N+1:end) = omega_j;

% BATH AND QUBIT

% Generates a column vector of uniformly distributed random numbers
% between -(gamma/sqrt(N)) and (gamma/sqrt(N)). It represents the
% normalized by 1/sqrt(N) coupling strength between the qubit and the
% spins on the bath.
% That sets the coupling between the spins and the coupling of the qubit
% with the spins at the same level.
lambda = -(gamma/sqrt(N)) + 2*(gamma/sqrt(N))*rand(N,1);

% Build the total Hamiltonian by concatenating lambda and its transpose
% (due to symmetry) as the last (N+1) column and last (N+1) row, while the
% last diagonal element is the frwquency of the qubit.
H = [H1, lambda; lambda', w];

end
```

A.3 analytical.m

```
% A function that calculates an analytical solution for the GGE

function [nl, omega] = analytical (N, w, gamma)

omega = linspace(0,2*w,1000000);
gavg = (gamma^2)/(3*N);
Omega = w;
```

```
nu0 = N/(2*Omega);

rate = pi*nu0*gavg;

nl = 2*gavg./((omega-Omega).^2+(2*rate)^2);


end
```

B. Modular Code

B.1 diagonal.m

```
% MODULAR VERSION

% A function that diagonalize the total hamiltonian
% vel:  the unitary matrix whose columns are the eigenvectors of H
% el:   the diagonal matrix of the eigenvalues (energies) of H

function [vel, el] = diagonal (H)

[vel, el] = eig(H);

end
```

B.2 time_evolution.m

```
% MODULAR VERSION

% A function that time-evolve the initial density matrix and calculates
% the resulting populations.
%
% Input variables:
%
% N:    The total number of two level systems (TLSs) in the bath.
%       The intially excited state, the qubit, is not considered to be
%       part of the bath. Therefore N+1 is the overall number of TLSs
```

```
% tmax: The final time at which the populations are calculated. Taken to be
%       8000000000 in the example case
% vel:  a matrix with column eigenvectors (from diagonal)
% el:   diagonal matrix of eigenvalues (from diagonal)
% rho0: The initial state of the system, with bath in the ground state
%       and qubit excited in the example case
%
% Output:
%
% E1:   the result of time evolution


function E1 = time_evolution (N, hbar, tmax, vel, el, rho0)

% Time-evolution operator U(t)=exp(-iHt/hbar)
% in the eigenbasis of the Hamiltonian
U_t = expm((-1i/hbar)*tmax*el);

% Spectral decomposition of the time-evolution operator
U_op = vel*U_t*(vel');

% Formal solution of Liouville-von Neumann equation
% rho(t) = U(t)*rho(0)*U(t)^dagger
rho_t = U_op*rho0*(U_op');

% A column (N+1) vector with the diagonal elements (probabilities of
% occupying the eigenstates) of the evolved density matrix
e1 = diag(rho_t);

% The part of the bath only, i.e. N
E1 = e1(1:N);

end
```

B.3 GGE.m

```
% MODULAR VERSION

% Calculate the numerical GGE prediction for the populations,
% which is to be compared with the long-time evolution.
% Basically, a convolution formula.
%
% Input variables:
% N:    The total number of two level systems (TLSs) in the bath.
%       The intially excited state, the qubit, is not considered to be
%       part of the bath. Therefore N+1 is the overall number of TLSs
% vel:  a matrix with column eigenvectors (from diagonal)
%
% Output
% nau:  the result of GGE

function nau = GGE (N, vel)

nau = zeros(1, N+1);
ujt = abs(vel(N+1,:)).^2;
uki = abs(vel).^2;
nau = sum(ujt .* uki, 2)';

end
```

C. Multicore Code

C.1 initParPool.m

```
% TRITON MULTICORE VERSION

% A function that creates a parallel pool on the cluster using the correct
% number of workers based on the SLURM_CPU_PER_TASK environment variable.
% Based on material provided by Triton documentation.

function initParPool()

% Check, whether there is already an open parallel pool,
% in order to avoid creating a new one.
parpoolOn = ~isempty(gcp('nocreate'));

% Try-catch expression that quits the Matlab session if the code crashes
if  ~parpoolOn
    % the number of workers based on the available cores
    num_workers = str2double(getenv('SLURM_CPUS_PER_TASK'));

    % Initialize the parallel pool
    c=parcluster();

    % Create a temporary folder for the workers working on this job,
    % in order not to conflict with other jobs.
    t=tempname();
    mkdir(t);
    c.JobStorageLocation=t;
```

```
    % start the parallel pool
    parpool(c,num_workers);
end
```

C.2 diagonal.m

```
% MULTICORE VERSION

% A function that diagonalize the total hamiltonian
% vel:  the unitary matrix whose columns are the eigenvectors of H
% el:   the diagonal matrix of the eigenvalues (energies) of H

function [vel, el] = diagonal (H)

H = distributed(H);

spmd
    [vel, el] = eig(H);
end

end
```

C.3 time_evolution.m

```
% MULTICORE VERSION

% A function that time-evolve the initial density matrix and calculates
% the resulting populations.
%
% Input variables:
%
% N:    The total number of two level systems (TLSs) in the bath.
%       The intially excited state, the qubit, is not considered to be
%       part of the bath. Therefore N+1 is the overall number of TLSs
```

```
% tmax: The final time at which the populations are calculated. Taken to be
%       8000000000 in the example case
% vel:  a matrix with column eigenvectors (from diagonal)
% el:   diagonal matrix of eigenvalues (from diagonal)
% rho0:  The initial state of the system, bath in the ground state
%       and qubit excited in the example case
%
% Output:
%
% E1:   the result of time evolution


function E1 = time_evolution (N, hbar, tmax, vel, el, rho0)

spmd

    % Time-evolution operator U(t)=exp(-iHt/hbar)
    % in the eigenbasis of the Hamiltonian
    U_t = expm((-1i/hbar)*tmax*el);

    % Spectral decomposition of the time-evolution operator
    U_op = vel*U_t*(vel');

    % Formal solution of Liouville-von Neumann equation
    % rho(t) = U(t)*rho(0)*U(t)^dagger
    rho_t = U_op*rho0*(U_op');

    % A column (N+1) vector with the diagonal elements (probabilities of
    % occupying the eigenstates) of the evolved density matrix
    e1 = diag(rho_t);

end

% The part of the bath only, i.e. N
E1 = gather(e1(1:N));

end
```

C.4 GGE.m

```
% MULTICORE VERSION

% Calculate the numerical GGE prediction for the populations,
% which is to be compared with the long-time evolution.
% Basically, a convolution formula.
%
% Input variables:
% N:    The total number of two level systems (TLSs) in the bath.
%       The intially excited state, the qubit, is not considered to be
%       part of the bath. Therefore N+1 is the overall number of TLSs
% vel:  a matrix with column eigenvectors (from diagonal)
%
% Output
% nau:  the result of GGE

function nau = GGE (N, vel)

nau = distributed.zeros(1, N+1);
spmd
    ujt = abs(vel(N+1,:)).^2;
    uki = abs(vel).^2;
    nau = sum(ujt .* uki, 2)';
end

nau = gather(nau);

end
```

D. GPU Code

D.1 diagonal.m

```
% GPU VERSION

% A function that diagonalize the total hamiltonian
% vel:  the unitary matrix whose columns are the eigenvectors of H
% el:   the diagonal matrix of the eigenvalues (energies) of H

function [vel, el] = diagonal (H)

H_gpu = gpuArray(H);
[vel, el] = eig(H_gpu);

% UNCOMMENT if you need to gather the results from the GPU
% vel = gather(vel);
% el = gather(el);

end
```

D.2 time_evolution.m

```
% GPU VERSION

% A function that time-evolve the initial density matrix and calculates
% the resulting populations.
%
```

```
% Input variables:
%
% N:    The total number of two level systems (TLSs) in the bath.
%       The intially excited state, the qubit, is not considered to be
%       part of the bath. Therefore N+1 is the overall number of TLSs
% tmax: The final time at which the populations are calculated. Taken to be
%       8000000000 in the example case
% vel:  a matrix with column eigenvectors (from diagonal)
% el:   diagonal matrix of eigenvalues (from diagonal)
% rho0:  The initial state of the system, bath in the ground state
%       and qubit excited in the example case
%
% Output:
%
% E1:   the result of time evolution


function E1 = time_evolution (N, hbar, tmax, vel, el, rho0)

% Time-evolution operator U(t)=exp(-iHt/hbar)
% in the eigenbasis of the Hamiltonian
U_t = expm((-1i/hbar)*tmax*el);

% Spectral decomposition of the time-evolution operator
U_op = vel*U_t*(vel');

% Formal solution of Liouville-von Neumann equation
% rho(t) = U(t)*rho(0)*U(t)^dagger
rho_t = U_op*rho0*(U_op');

% A column (N+1) vector with the diagonal elements (probabilities of
% occupying the eigenstates) of the evolved density matrix
e1 = diag(rho_t);

% The part of the bath only, i.e. N
E1 = gather(e1(1:N));

end
```

D.3 GGE.m

```
% GPU VERSION

% Calculate the numerical GGE prediction for the populations,
% which is to be compared with the long-time evolution.
% Basically, a convolution formula.
%
% Input variables:
% N:    The total number of two level systems (TLSs) in the bath.
%       The intially excited state, the qubit, is not considered to be
%       part of the bath. Therefore N+1 is the overall number of TLSs
% vel:  a matrix with column eigenvectors (from diagonal)
%
% Output
% nau:  the result of GGE

function nau = GGE (N, vel)

nau = gpuArray(zeros(1, N+1));
ujt = abs(vel(N+1,:)).^2;

uki = abs(vel).^2;
nau = arrayfun(@(k) dot(ujt, uki(k,:)), 1:(N+1));

nau = gather(nau);

end
```