I Computational Details

I.1 Parallel computing

The following section will give an overview of the technical aspects of running computer code (such as QUANTUM ESPRESSO) on massively parallel computing environments (such as the PHYSnet compute cluster). The information presented can be found in any textbook on parallel or high-perfomance computing [1].

functional/data parallelism?

I.1.1 On scalabilty

In scientific computing, one can identify two distinct reasons to distribute workloads to multiple processors:

- The execution time on a single core is not sufficient. The definition of sufficient is dependent on the specific task and can range from "over lunch" to "multiple weeks"
- The memory requirements grow outside the capabilities of a single core

In order to judge how well a task can parallelized, usually some sort of scalabilty metric is employed, for example:

- How fast can a problem be solved with N processors instead of one?
- What kind of bigger problem (finer resolution, more particles, etc.) can be solved with N processors?
- How much of the resources is used for solving the problem?

The speedup by using N workers to solve a problem instead of one is defined as $S = \frac{T_1}{T_N}$, where T_1 is the execution time on a single processor and T_N is the execution time on N processors. In the ideal case, where all the work can be perfectly distributed among the processors, all processors need the same time for their respective workloads and don't have to wait for others processors to finish their workload to continue, the execution time on N processors would be $\frac{T_1}{N}$, so the speedup would be $S = \frac{T_1}{\frac{T_1}{N}} = N$.

In reality, there are many factors either limiting or in some cases supporting parallel code scalability. Limiting factors include:

- *Algorithmic limitations*: when parts of a calculation are mutually dependent on each other, the calculation cannot be fully parallelized
- *Bottlenecks*: in any computer system exist resources which are shared between processor cores with limitations on parallel access. This serializes the execution by requiring cores to wait for others to complete the task which uses the shared resources in question

- *Startup Overhead*: introducing parallelization into a programm necessarily introduces an overhead, e.g. for distributing data across all the processors
- *Communication*: often solving a problem requires communication between different cores (e.g. exchange of interim results after a step of the calculation). Communication can be implemented very effectively, but can still introduce a big prize in computation time

On the other hand, faster parallel code execution can come from:

- *Better caching*: when the data the program is working with is distributed among processors (assuming constant problem size), it may enable the data to be stored in faster cache memory. Modern computers typically have three layers of cache memory, with level 1 cache being the smallest and fastest and level 3 being the largest and slowest, so smaller data chunks per processor can lead to the data not being stored in main memory, but completely in cache or in a faster cache level

more details strong/weak scaling?

I.2 Quantum ESPRESSO

Quantum ESPRESSO (opEn-Source Package for Research in Electronic Structure, Simulation, and Optimization) [2, 3] is a collection of packages implementing (among others) the techniques described in sec. **??** and **??** to calculate electronic structure properties as well as phonon frequencies and eigenvectors.

I.2.1 Compilation of Quantum ESPRESSO

As the core of this thesis is an in depth examination of the Quantum ESPRESSO software and ways its performance can be optimized, a discussion of the way is is compiled is needed. The information in this section is taken from the Quantum ESPRESSO 7.0 user guide [4].

The Quantum ESPRESSO distribution is packaged with everything needed for simple, non-parallel execution, the only additional software needed are a minimal Unix environment (a shell like `bash` or `sh` as well as the utilities `make` `awk` and `sed`) and a Fortran compiler compliant with the F2008 standard. For parallel execution, also MPI libraries and an MPI aware compiler need to be provided.

Quantum ESPRESSO needs three external mathematical libraries, BLAS and LAPACK for linear algebra as well as an implementation of FFT for fourier transforms. In order to make the installation as easy as possible, Quantum ESPRESSO comes with a publicly available reference implementation of the BLAS routines, the publicly available LAPACK package and an older version of FFTW (Fastest Fourier Transform in the West, an open source implementation of FFT). Even though these libraries are already optimized in terms of the algorithms implemented, usage of libraries implementing the same routines which can use more specific CPU optimizations significantly improves performance (e.g. libraries provided by Intel can use CPU optimizations not present on AMD processors).

On the PHYSnet cluster, a variety of software packages are available as modules. The benchmark in this thesis were made using the following modules:

- `openmpi/4.1.1.gcc10.2-infiniband`: OpenMPI 4.1.0 (implies usage of Quantum ESPRESSO provided BLAS/LAPACK)

- `openmpi/4.1.1.gcc10.2-infiniband openblas/0.3.20`: OpenMPI 4.1.0 and OpenBLAS 0.3.20
- `scalapack/2.2.0`: OpenMPI 4.1.0, OpenBLAS 0.3.20 and ScaLAPACK 2.2.0
- `intel/oneAPI-2021.4`: Intel oneAPI 2021.4

QUANTUM ESPRESSO offers an configuration script to automatically find all required libraries. As the default options of the `configure` script work well in the use case of this thesis, all compilations were made using the minimal commands

```
module load <module names>
./configure --with-scalapack=no|yes|intel
```

with the scalapack options yes (when using `scalapack/2.2.0`), intel (when using `intel/oneAPI-2021.4`) and no otherwise.

what to look out for in configure output: when are the internal copies of blas and lapack used?

I.2.2 Parallelization capabilities offered by Quantum ESPRESSO

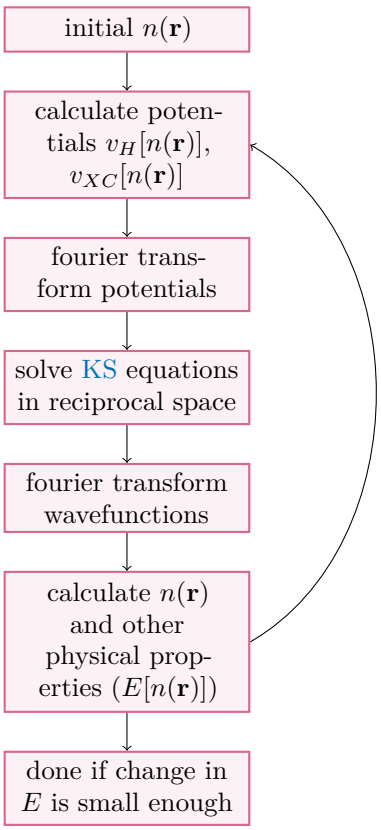

Figure I.1: *Flowchart of an algorithm to solve the KS equations*

Fig. ?? shows a possible approach to solving the KS equations. This opens a few possibilities for parallelization of calculations: first of all, the orbitals in the plane wave basis set as well as charges and densities can be distributed among processors. This distribution of data mainly works around memory constraints, as using more processors lowers the memory requirement for every single processor. Going further, QUANTUM ESPRESSO automatically parallelizes all linear algebra operations on real space/reciprocal grid. The price to pay for this parallelization is the need for communication between processors: as an example, fourier transforms always need to collect and distribute contributions from both representation in order to transform between them.

To remedy this problem,

k point parallelization

In another level of parallelization, QUANTUM ESPRESSO can use ScaLAPACK to parallelize among other things the iterative orthonormalization of KS states. This parallelization level is called *linear algebra parallelization* and is controlled by the CLI parameter `-nd <number of processors in linear algebra group>`. Importantly, this parameter sets the size for the linear algebra group in every k point processor pool, so the

number of processors in the linear algebra group has to be smaller than the number of processors in one pool. Furthermore, the arrays on which the calculations are performed on are distributed in a 2D grid among processors, so the number of processors in the linear algebra has to be n^2, where n is an integer.

short conclusion: which parameters, how can they be chose?

I.3 Hardware configuration of the PHYSnet cluster

is this good here?