I Computational Details

I.1 Parallel computing and scalability

The following section will give an overview of the technical aspects of running computer code, such as QUANTUM ESPRESSO, on massively parallel computing environments. The information presented in this section follows closely the textbook on high-performance computing by Hager and Wellein [1].

In scientific computing, one can identify two distinct reasons for distributing 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 exceed the capabilities of a single core.

Parallelization of a task across multiple processors can be differentiated into two ways:

Single Program Multiple Data (SPMD) Every processor runs the same program, with data distributed among processors.

Multiple Program Multiple Data (MPMD) Every processor runs a different function, for example in a pipelining process where multiple consequent operations on the input data need to be done and data comes in chunks, so every step of the pipeline can run independent of the others.

The typical case in physics is SPMD. For example, many calculations require diagonalization of matrices, which can be iteratively done with algorithms requiring only knowledge of the data of the nearest neighbors for every matrix element in every step. This enables parallelization as the whole matrix can be distributed and communication is only required at the bordering regions for exchange of data in every iteration step.

why does that introduce waiting times?

In order to judge how well a task can be parallelized, a scalability 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 efficiently are the resources utilized?

In this thesis, the main concern is speeding up the execution of extensively expensive calculations with a fixed problem size, so the first metric will be used to judge the quality of parallelization. This metric is called speedup and is defined as

$$S = \frac{T_1}{T_N}\,, \tag{I.1}$$

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 and all processors need the same time for their respective workloads, the execution time on N processors would be T_1/N, so inserting this into eq. I.1 gives a speedup of

$$S = \frac{T_1}{\frac{T_1}{N}} = N\,. \tag{I.2}$$

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 program 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, *better caching* can lead to better scaling than $S = N$: as optimal performance per core is achieved when all the data can be kept in cache, reducing the data size per processor by distributing data among more processors can lead to each individual processor being faster than in the single core case.

A simple ansatz for modeling speedup with these limitations in mind was first derived by Gene Amdahl [2]. Assuming the work that needs to be done is split into a part which cannot be parallelized s and a part which can be parallelized ideally p, serial time can be normalized to 1:

$$T_1 = s + p = 1 \tag{I.3}$$

The time for solving the problem on N processors is then

$$T_N = s + \frac{p}{N}\,. \tag{I.4}$$

The speedup is now

$$S = \frac{T_s}{T_p} = \frac{1}{s + \frac{p}{N}} = \frac{1}{s + \frac{1-s}{N}}\,. \tag{I.5}$$

This equation is called *Amdahl's law*. It shows that even for $N \to \infty$, the speedup has an upper bound of $\frac{1}{s}$. Furthermore, the value of s determines the range of processors where the speedup is close to the ideal case, as shown in fig. I.1. It shows that for a bigger value of s, not only leads to the speedup saturating at a smaller constant, but also differing significantly from the ideal case even for a small number of processors. For $s = 0.01$, a speedup of around 16 for 20 processors used can be deemed acceptable in terms of how efficient the computing resources are used, whereas even using more than 6 processors in the case of $s = 0.2$ is not.

The weakness of Amdahl's law lies in the simplicity of it, as the different factors limiting parallelization are generally not independent of N. Communication overhead would need to

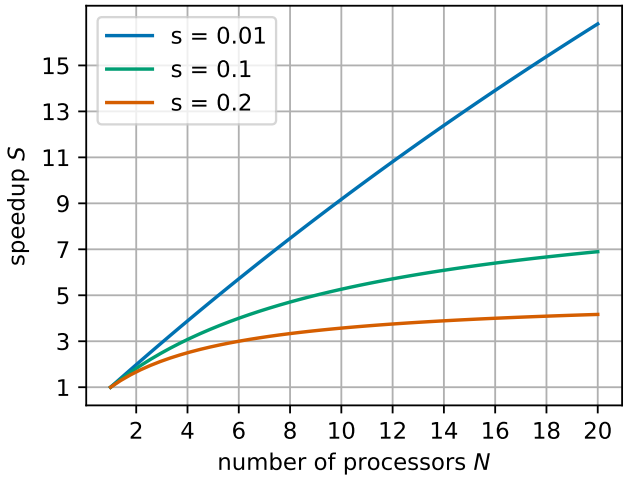

Figure I.1: *Speedup modeled by Amdahl's law for different portions of strictly serial workload s*

be accounted for with some kind of function $c(N)$ with the form depending on many factors like speed and bandwidth of the communication hardware or the way the data is distributed.

With just Amdahl's law, decomposition into the several factors limiting parallelization is not possible, so an assessment of how calculations can be improved in detail is also not possible. Regardless, Amdahl's law explains in simple ways how speedup can differ from the ideal case and can also be reasonable accurate when the costs for communication don't depend strongly on N.

I.2 Quantum ESPRESSO

Quantum ESPRESSO (opEn-Source Package for Research in Electronic Structure, Simulation, and Optimization) [3, 4] is a collection of packages implementing (among others) the techniques described in sec. **??** and **??** to calculate electronic structure properties (module `PWscf`) as well as phonon frequencies and eigenmodes (module `PHonon`).

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 and the used compilers influence significantly how Quantum ESPRESSO performs, a discussion of the way it is compiled is needed. The information in this section is taken from the Quantum ESPRESSO 7.0 user guide [5].

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 might improve performance, e.g. libraries included in Intel oneAPI, which are optimized for use on Intel CPUs.

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

- `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 a 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.

The output of the configuration script gives information about the detected libraries. In the following output, the Intel Intel oneAPI package was loaded, so BLAS and ScaLAPACK libraries from that package will be used, whereas the included FFT library will be used:

```
The following libraries have been found:
BLAS_LIBS=  -lmkl_intel_lp64  -lmkl_sequential -lmkl_core
LAPACK_LIBS=
SCALAPACK_LIBS=-lmkl_scalapack_lp64 -lmkl_blacs_intelmpi_lp64
FFT_LIBS=
```

I.2.2 Parallelization capabilities implemented in Quantum ESPRESSO

Quantum ESPRESSO is intended to be used in parallel environments and as such offers possibilities to manage how the work is parallelized. This section introduces the parallelization capabilities of the `PWscf` and `PHonon` modules and explores how they potentially affect the

scaling behavior of QUANTUM ESPRESSO. The information in this section stems from the user guides for the two modules [6, 7]

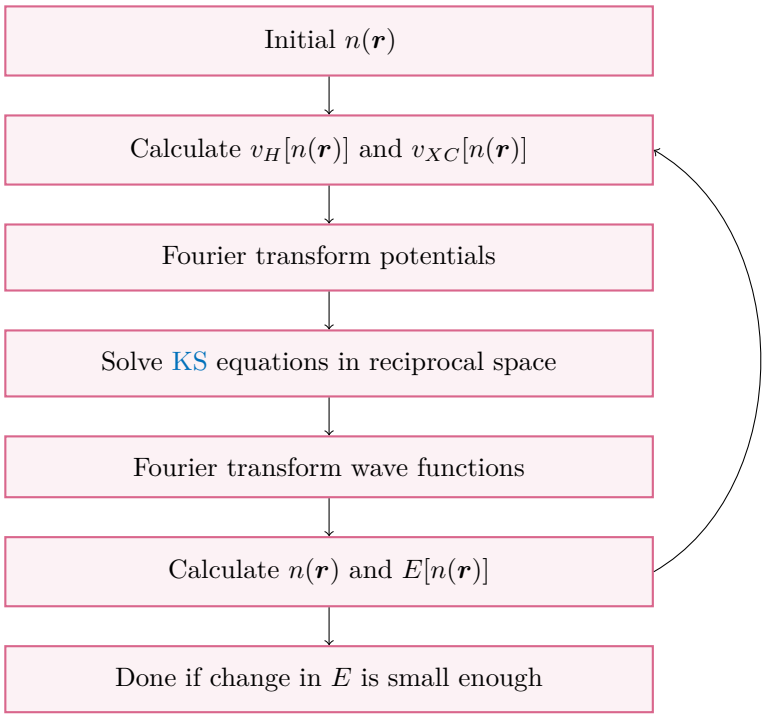

Figure I.2: *Flowchart of an iterative algorithm to iteratively solve the KS equations with the use of fourier transform. As the density $n(\boldsymbol{r})$ determines again the potentials going into the KS equations, steps 2-5 are run until self-consistency is reached.*

Fig. I.2 shows a possible approach to solving the KS equations. The algorithm is taken from the textbook by Martin [8], with the fourier transform steps added to establish in which representation every calculation step is run.

talk a bit about the figure

A few possibilities for parallelization of calculations can be derived from that.

First of all, the real and reciprocal space are discretized to allow for numerical treatment, these grids can be distributed among processors, meaning the wave functions in the plane-wave basis set as well as charges and densities. 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 this 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 and to the whole reciprocal/real grid in order to transform between them. This kind of parallelization is called *PW (plane-wave)* or *R&G (real & reciprocal)* parallelization.

As discussed in sec. **??**, the density in the plane-wave basis set is a sum over different k points, where the calculation for these are independent of each other until calculating the density $n(\boldsymbol{r})$.

In Quantum ESPRESSO this is implemented such that a separation of the total number of processors into smaller pools, each doing the calculations for a set of k points is possible. This is called *k point parallelization*. The CLI parameter `-nk <number of pools>` determines how many pools the total number of processor N is split into. Hence, the resulting number of processors in one pool is N/N_k. Within one k-point processor pool, the PW parallelization with its heavy communication is automatically applied.

In a level of parallelization independent of that, 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. This means that the number of processors in the linear-algebra group has to be a square number.

In the case of the `PHonon` module, the representation of states in a plane-wave basis set stays the same, so all three parallelization schemes mentioned for the `PWscf` module can also be employed. Furthermore, as calculations for two phonon wave vectors $\boldsymbol{q}, \boldsymbol{q}'$ are not coupled (as different wave vectors lead to different perturbations and as such independent self-consistent equations), they can be split up into images. The concept of image parallelization in Quantum ESPRESSO is actually more general than just for phonon calculations, as other kinds of independent iterative calculations can also be run with image parallelization. The parameter controlling image parallelization is `ni <number of images>`. Following this, the number of processors in one k-point pool is then given by $N/N_i/N_k$.

I.2.3 Evaluating the scalability of Quantum ESPRESSO calculations

In the Quantum ESPRESSO output, a time report is printed at the end. This time report includes CPU time and wall time. Three different metrics of scalability can be calculated from this:

- runtime: absolute runtime (wall time) of the compute job
- speedup: runtime on N processors divided by runtime on a single core
- wait time: percentage of wall time not used by Quantum ESPRESSO process, so writing to disk, waiting for IO devices or other processes, etc. (calculated as (wall time - CPU time) / wall time)

For analysis mainly speedup will be used to evaluate the scalability of Quantum ESPRESSO calculation. It makes comparing the scaling of calculations with different absolute runtimes easy: as discussed in sec. I.1, optimal scaling is achieved when the speedup has a slope of one, independent of the runtime.

Regardless, the other two parameters should also always be considered. In the end, absolute runtime is the most important factor and should govern the decision of how much computational resources should be used for solving a particular problem. For instance, a problem with a single core runtime of 600 s, a speedup of 100 would mean a runtime of 6 s, whereas a speedup of 200 would mean a runtime of 3 s. Even with optimal scaling, the 100 processors needed for

the speedup of 200 could be considered wasted for just 3 s of saved time. On the other hand, for a problem with a single core runtime of 2400 h, the difference between a speedup of 100 (runtime 24 h) and 200 (runtime 12 h) is the difference between needing to wait a whole day for the calculation and being able to let the job run overnight and continue working in the morning, so using 100 more processors for that can be considered a good use of resources.

As for the wait time, this metric can be used to separate the different factors of poor parallelization discussed in I.1. Startup overhead is easy to identify, as this should be a small, near constant percentage of the absolute runtime. This of course can vary depending on how complex data distribution is, but there should at least not be a strong dependence on the number of processors, as only a small amount of communication is needed. Communication and bottlenecks on the other hand both introduce wait time which depends on the number of processors. Differentiating between them relies on knowledge of the specific hardware of the system running the calculations. This means how many cores are on a single chip, motherboard or node, which resources are shared between how many cores, etc..

For this interpretation to be meaningful, the CPU and wall times reported by QUANTUM ESPRESSO have to be accurate. As an example for how errors could be overlooked, when executing programs on multiple processors in parallel, CPU time is measured per processors. This means some kind of information truncation is done when a single number (such as in QUANTUM ESPRESSO) is reported. Whether this is taking the average over all processors, just reporting the time for a single processor or any other kind of truncation is unclear.

However, the notion of using the difference between wall time and CPU time for evaluating the quality of parallelization is supported by the user guide for one of the QUANTUM ESPRESSO modules [6] (sec. 4.5), therefore it will also be used as a qualitative measure of good parallelization in this thesis.

I.3 Hardware configuration of the PHYSnet cluster

All calculations were run on a reserved subset of the `infinix` queue on the PHYSnet compute cluster with 20 nodes. As of time of writing, the nodes in this queue are equipped with two Intel Xeon E5-2680 CPUs, as such providing 20 cores per node, 10 per chip and 200 processors in the whole queue. The nodes are connected with with an Infiniband FDR 4x network.