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].

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

functional/data parallelism?

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 efficiently are the resources utilized?

In this thesis, the main concern is speeding up the calculation of very time 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 $S = \frac{T_1}{T_N}$, 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

A simple ansatz for modeling speedup was first derived by Gene Amdahl. 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, we can normalize the serial time to 1:

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

The time for solving the problem on N processors is then

$$T_N = s + \frac{p}{N} \tag{I.2}$$

The speedup is now

$$S = \frac{T_s}{T_p} = \frac{1}{s + \frac{p}{N}} = \frac{1}{s + \frac{1-s}{N}} \tag{I.3}$$

This equation is called *Amdahl's law* and is plotted in fig. **??** over a range of processors for a few different values of s.

I.1.2 Evaluating the scalability of Quantum ESPRESSO calculations

In the Quantum ESPRESSO output, a time report listing the whole module as well as on a per-function level is always printed at the end. This time report includes cpu time and wall time, from those three different metrics of scalability can be calculated:

- runtime: absolute runtime (wall time) of the compute job
- speedup: runtime divided by runtime of the job 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 further analysis mainly speedup will be used to evaluate the scalability of Quantum ESPRESSO calculation, because it makes comparing the scaling of calculations with different

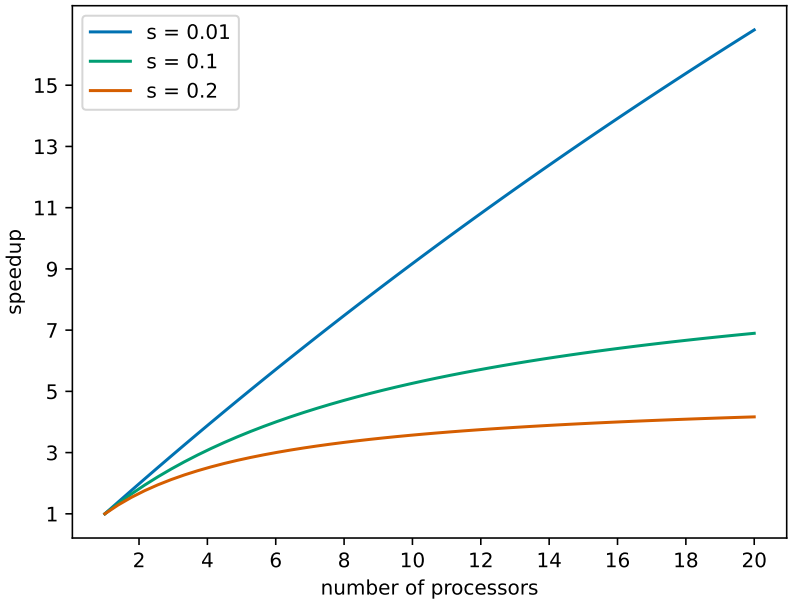

Figure I.1: *Amdahl's law for different values of s*

absolute runtimes easy: optimal scaling is achieved when the speedup has a slope of one, as discussed in sec. I.1.1, independent of the runtime.

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 resources are used for solving a particular problem. As an example, for 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.1.1.

how do different factors show?

Importantly, for this interpretation to be meaningful, the cpu and wall times reported by Quantum ESPRESSO have to accurate. When executing programs on multiple processors in parallel, cpu time is measured per processor, so some kind of 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 [2] (sec. 4.5), so it will be used a such in this thesis.

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 eigenvectors (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, 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 [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 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.

The output of the configuration script gives information about the libraries it found. 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 offered by 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 .

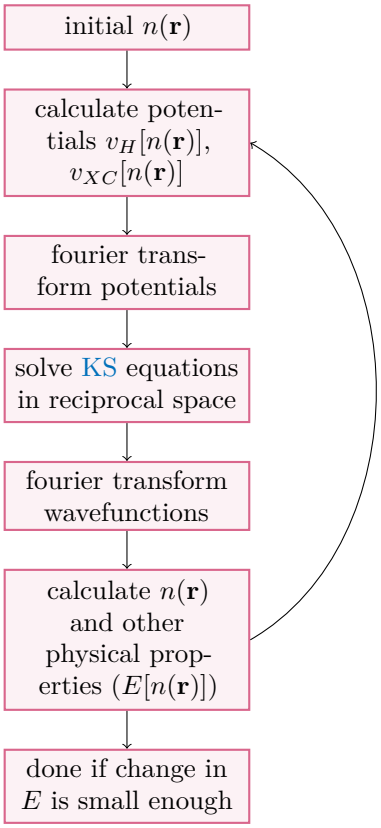

Figure I.2: *Flowchart of an algorithm to solve the KS equations*

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 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 $n(\mathbf{r})$. QUANTUM ESPRESSO can use this fact and separate the total number of

processors into smaller pools, each doing the calculations for a set of k points, so called *k point parallelization*. The CLI parameter `-nk <number of pools>` determines how many pools the total number of processor N is split into, so the resulting number of processors in one pool is $\frac{N}{N_k}$. Within one k point processor pool, the PW parallelization with its heavy communication is automatically applied.

In an 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, so the number of processors in the linear algebra group has to be a square number.

parallelization ph.x

I.3 Hardware configuration of the PHYSnet cluster

All calculations were run on the `infinix` queue on the PHYSnet compute cluster. 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.

Knowing the specific hardware configuration the calculations are run on is important, as that provides a scale of different communication speeds:

- between processors on one chip:
- between processors on one motherboard:
- between processors on different nodes:

speed comparison between processors on one chip, on one motherboard, between nodes (infiniband)

This gives guidelines on how well calculations can be parallelized on different numbers of processors. As an example, any calculation with a massive need for communication, for example the calculations one the PW