I Parallelisation of self-consistent calculations of electronic-structure properties

I.1 First scaling tests

The first step in analysing the scaling of QUANTUM ESPRESSO is to perform a baseline scaling test without any optimisations appplied. In Fig. I.1 and **??** two scaling tests on the earlier mentioned benchmarking systems Si and TaS2 are pictured. The tests are run using QUANTUM ESPRESSO 7.0, compiled using the Fortran and C compilers in OpenMPI 4.1.0, without any of compilation or runtime optimisation parameters mentioned in section **??** used.

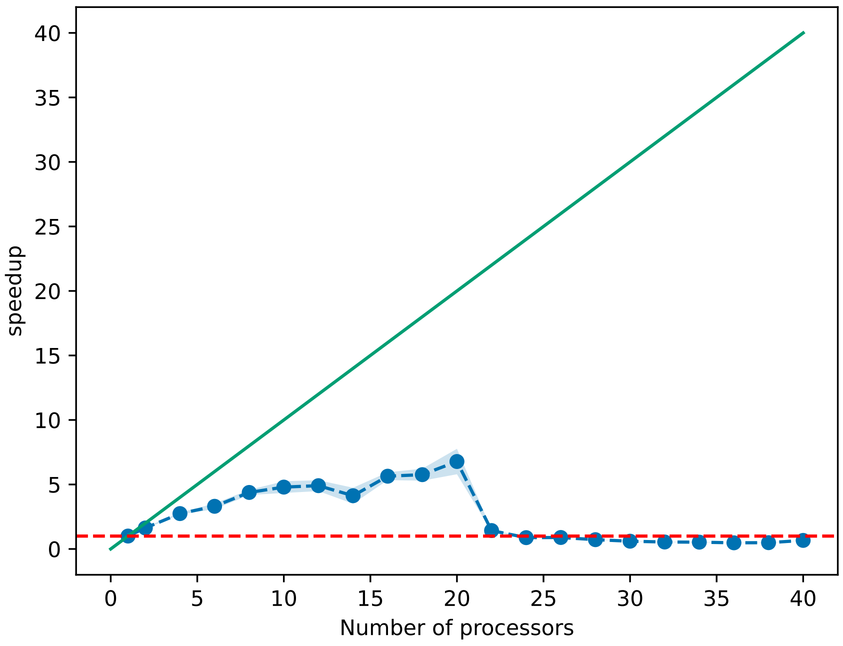


Figure I.1: *Baseline scaling test on the Si benchmarking system* QUANTUM ESPRESSO 7.0, OpenMPI 4.1.0, `nk 1` and `nd 1`

Three different metrics of scalability are pictured in **??**.

talk a bit about silicon there, then introduce TaS2

formulate differently with new figure structure!

- runtime: absolute runtime (walltime) of the compute job

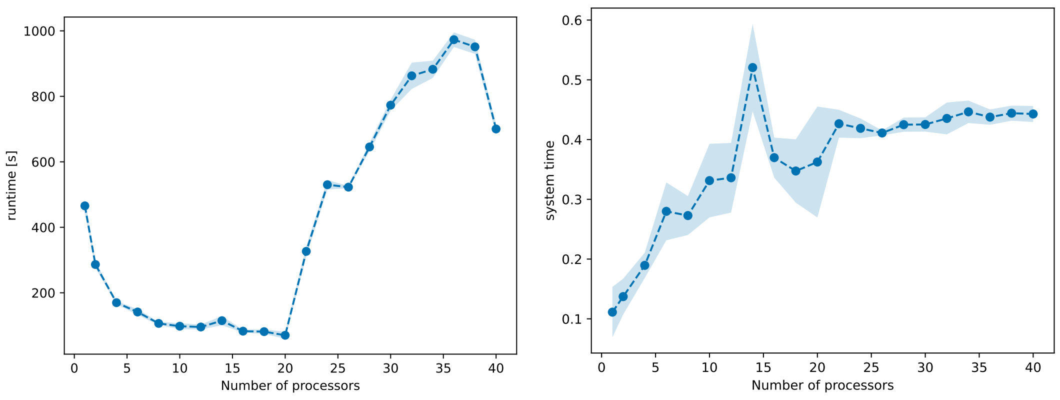


Figure I.2: *Baseline scaling test on the Si benchmarking system* QUANTUM ESPRESSO 7.0, OpenMPI 4.1.0, `nk 1` and `nd 1`

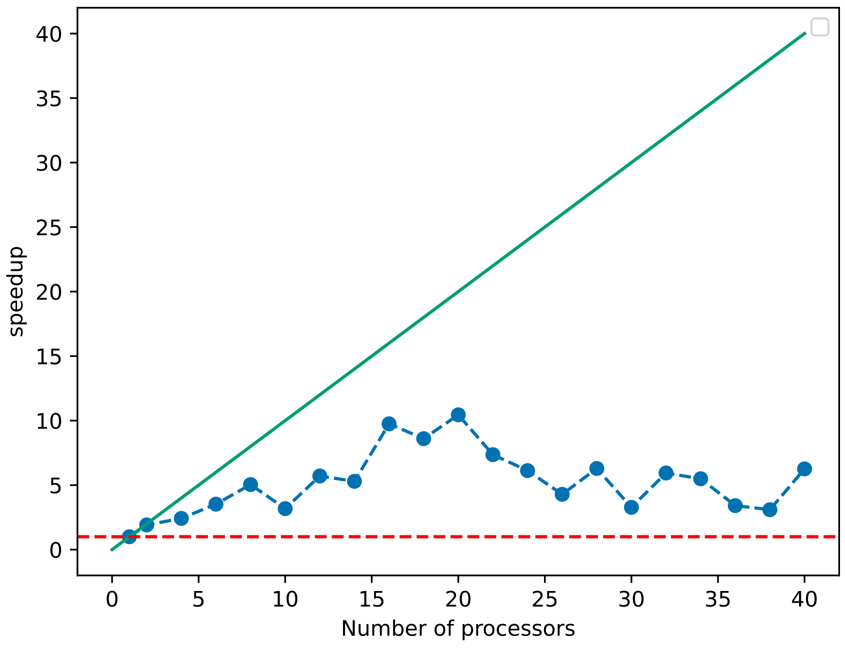


Figure I.3: *Baseline scaling test on the TaS2 benchmarking system* QUANTUM ESPRESSO 7.0, OpenMPI 4.1.0, `nk 1` and `nd 1`

- speedup: runtime divided by runtime of the job on a single core
- system time: percentage of wall time used by system tasks, e.g. writing to disk, etc. (calculated as (walltime - cputime) / walltime)

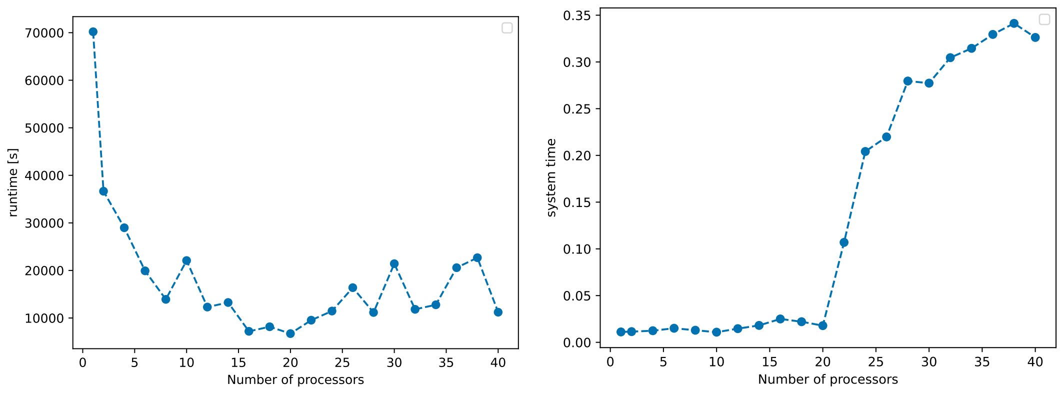


Figure I.4: *Baseline scaling test on the TaS2 benchmarking system* Quantum ESPRESSO 7.0, OpenMPI 4.1.0, `nk 1` and `nd 1`

For further analysis mainly speedup will be used as a metric of scalability, because it lends itself to easy interpretation: optimal scalability is achieved when the speedup scales linearly with the number of processors (with a slope of one), as discussed in ch. **??**. This on the one hand necessarily implies good parallelization and a lower runtime for more processors used, but the other two parameters should also always be considered.

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.

explanation of wait time

On a single node, both the Si and TaS2 calculations show good, but not perfect scaling behavior: the speedup does approximately scale linearly with the number of processors, but the slope is closer to $\frac{1}{2}$, than 1. Even though the scaling behavior is not perfect, there is just a small, almost constant amount of runtime used by system calls, this speaks for good parallelization. As discussed in sec. **??**, startup time is an unavoidable part of every parallel program, so a constant amount of time used not for calculations is expected, bad parallelization on the other hand shows itself by introducing waiting times between processors, which makes the waiting time in some way dependent on the number of processors.

When using more than one node, not only does the scaling get worse, the execution needs longer than on a single core for the Si system, with a marginally better performance for the TaS2 system. This is also seen in the plots of system time. The percentage of time used for tasks not directly related to calculations goes from a near constant value for under 20 processors to 50% of the execution time for the Si system and 35% for the TaS2 system.

These scaling tests pose now two questions to be answered:

- Is better scaling on a single node possible?

- How can acceptable scaling over more than one node be achieved?

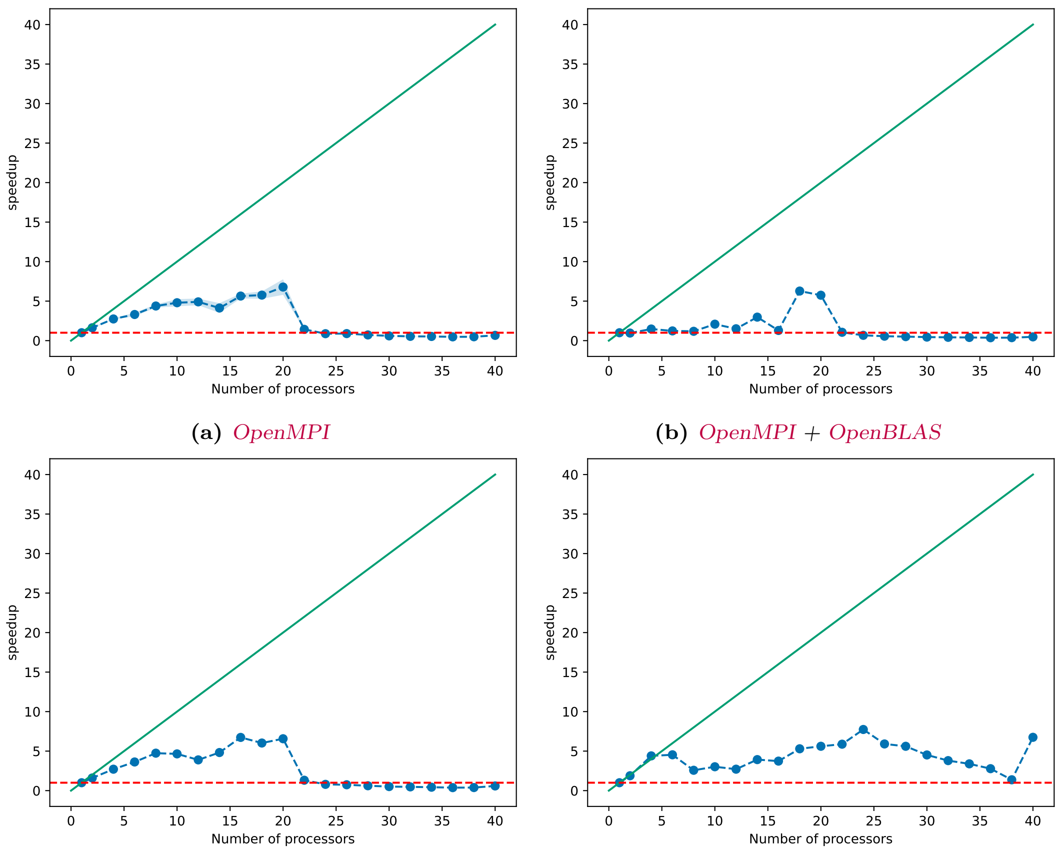
I.2 Testing different compilers and mathematical libraries

missing

A first strategy for solving issues with parallelization is trying different compilers and mathematical libraries. In the PHYSnet cluster a variety of software packages is available. As discussed in sec. **??**, for the compilation of Quantum ESPRESSO

For testing Quantum ESPRESSO will be compiled using the following software combinations:

- OpenMPI 4.1.0 and OpenBLAS
- OpenMPI 4.1.0 and Scalapack
- Intel OneAPI 2021.4 (includes Intel MPI, Fortran and C compilers as well as Intel MKL, a scalable mathematical library)



(a) *OpenMPI* **(b)** *OpenMPI + OpenBLAS*

(c) *OpenMPI + Scalapack* **(d)** *Intel OneAPI*

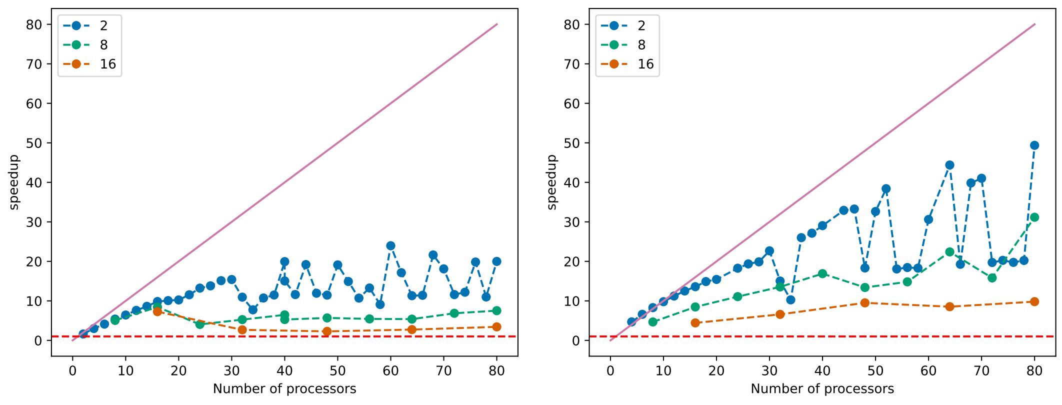
Figure I.5: *Baseline scaling test with different combinations of compilers and mathematical libraries*

analysis

I.3 Using the parallelisation parameters of Quantum ESPRESSO

As detailed in section ??, QUANTUM ESPRESSO offers ways to manage how the workload is distributed among the processors. In `pw.x` the default plane wave parallelization, k-point-parallelization and linear-algebra parallelization are implemented.

The benchmark pictured in I.6 is set up as follows: for a given number of processors N_p, the parameter N_k splits the N_p processors into N_k processors pools. As the number of processors in one pool has to be a whole number, only certain combinations of N_p and N_k are possible, for example $N_p = 32$ could be split into processor pools of size 2 with $N_k = 16$, size 8 with $N_k = 4$ or size 16 with $N_k = 2$. This leads to choosing the size of the processor pools as a variable, not the parameter `nk`. Fig. I.6 shows the scaling for poolsizes 2, 8 and 16 for QUANTUM ESPRESSO being compiled with OpenMPI/Scalapack and Intel oneAPI. This choice of pool sizes showcases the smallest pool size possibly (namely 2), as well as a bigger pool size with 16, that still gives rise to a few data points over the chosen range of processors.



(a) *OpenMPI + Scalapack* **(b)** *Intel OneAPI*

Figure I.6: *Benchmark with k-point parallelization for the Si benchmarking system with 3 different sizes of processor pools*

Fig. I.6 shows that using k parallelization with a pool size of 2 significantly improves the scaling behavior, not only on one node, but especially over more than one node.

analysis

Another important conclusion to draw out of fig. I.6 is the impact of using Intels compiler instead of OpenMPI, as that factor alone speeds up the calculation by a factor of 2 over the whole range of processors.

The same scaling test is applied to the TaS2 system in fig. I.7, with the same list of pool sizes, but over a wider range of processors.

Remarkably, the scaling behavior is swapped in comparison to I.6, as the pool size 2 saturates fast and the bigger pool sizes show way better scaling behavior. Furthermore, there are instances of better than linear scaling, which according to QUANTUM ESPRESSO docs can be attributed to better caching of data.

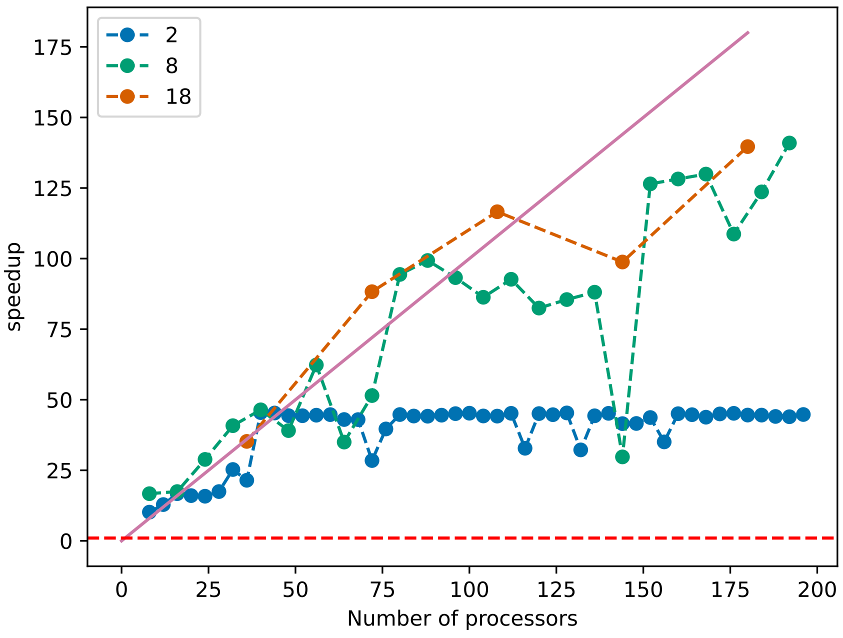


Figure I.7: *Benchmark with k-point parallelization for the TaS2 benchmarking system*

It can also be instructive to look at the idle time for this benchmark to judge the quality of parallelization.

Fig. I.8 shows a distribution of idle times between 1 % and 4 % of the whole wall time, without any kind of systemic increase over any range of processors. This means the parallelization is as good as possible for these

missing

Linear Algebra

I.3.1 Comparison with calculations on the HLRN cluster

I.3.2 Conclusion: Parameters for optimal scaling

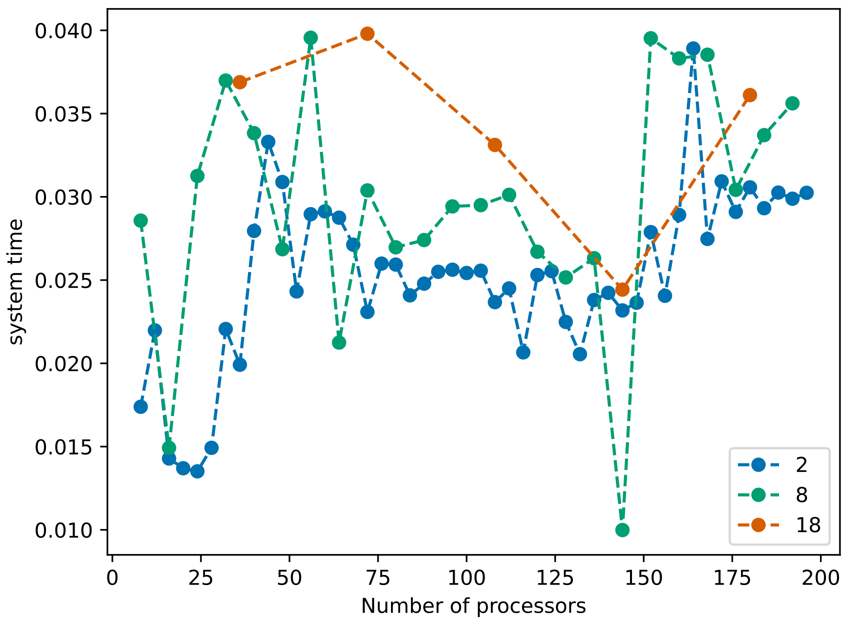


Figure I.8: *Idle time for the k point parallelization benchmark for the TaS2 system*

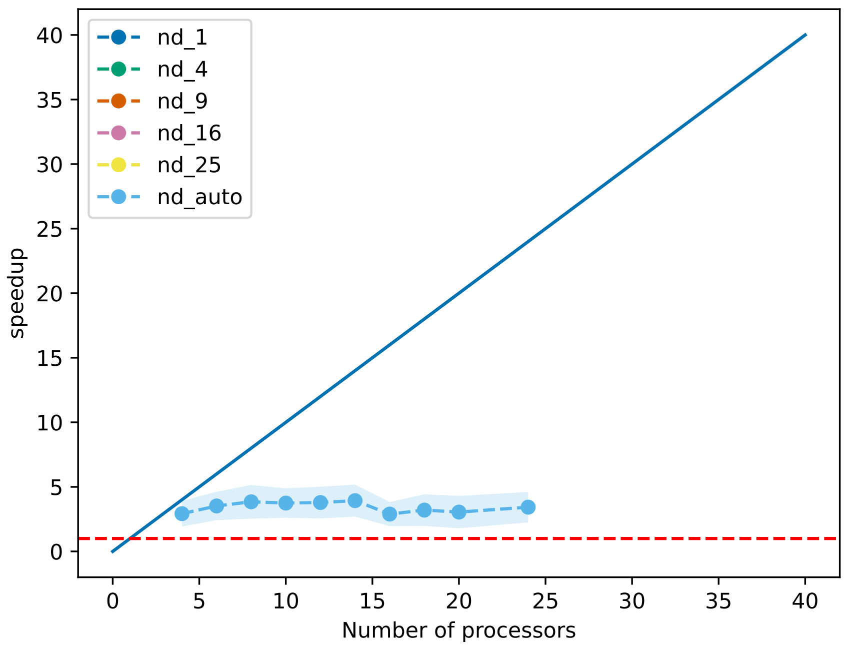


Figure I.9: *Benchmark with linear algebra parallelization for the silicon benchmarking system*

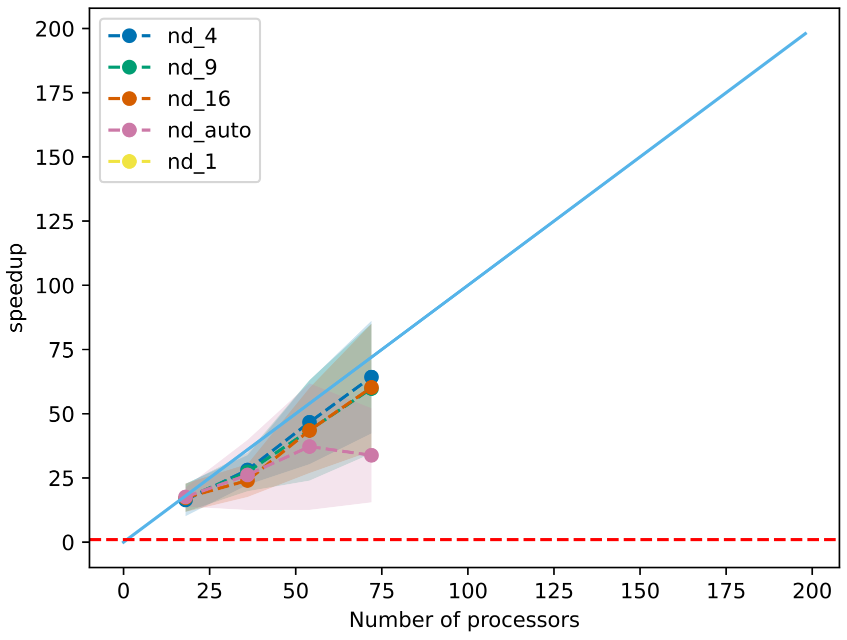


Figure I.10: *Benchmark with linear algebra parallelization for the TaS2 benchmarking system*