I Parallelisation of electronic-structure calculations

The `PWscf` (Plane-Wave Self-Consistend Field) package is one of the core modules of QUANTUM ESPRESSO , as many other modules need the variables ground state density or total energy as input. This chapter deals with examining the best ways to run `PWscf` calculations in the `scf` mode.

I.1 First scaling tests

The first step in analysing the scaling of the `PWscf` is to perform a baseline scaling test without any optimisations appplied. In Fig. I.1 to I.4 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 the compilation or runtime optimisation parameters mentioned in section **??** used.

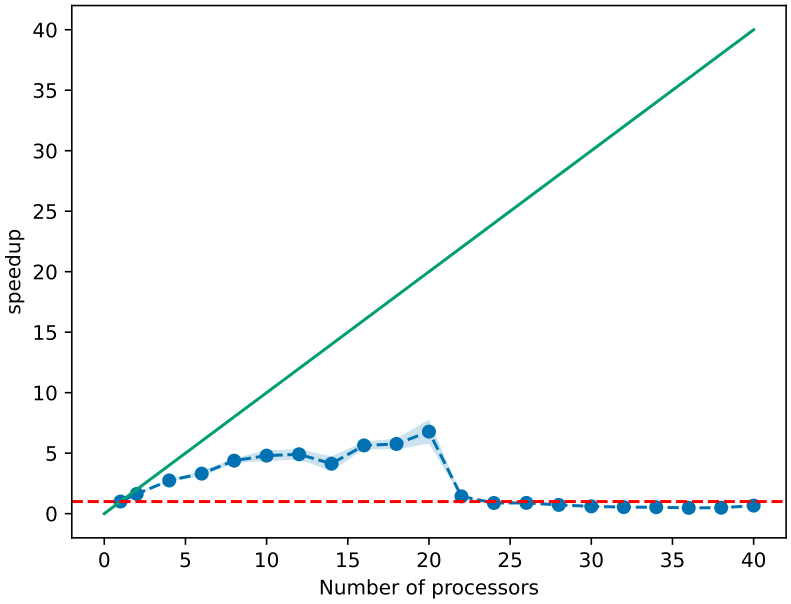


Figure I.1: *Baseline scaling test on the Si benchmarking system* QUANTUM ESPRESSO 7.0, OpenMPI 4.1.0, `nk 1` and `nd 1`

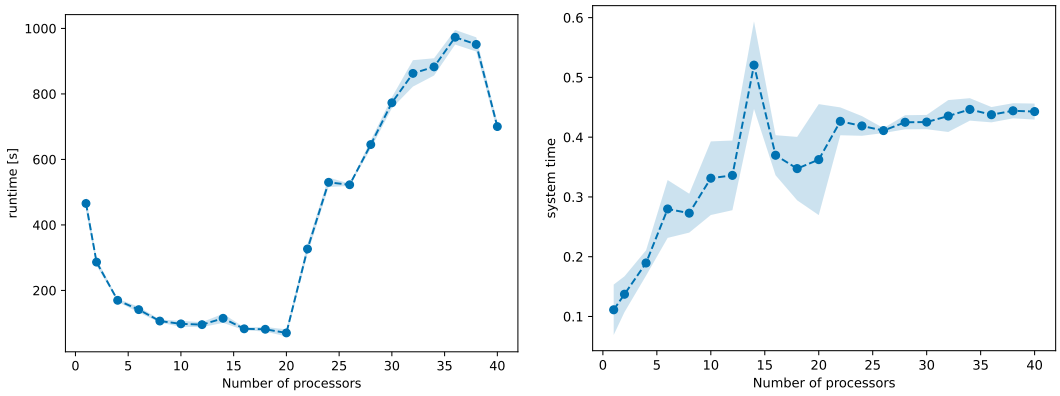


Figure I.2: *Baseline scaling test on the Si benchmarking system* QUANTUM ESPRESSO 7.0, OpenMPI 4.1.0, `nk 1` and `nd 1`

Three different metrics of scalability are pictured in I.1 and I.2.

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

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, also this all would probably fit better in the computation chapter? so there is a central reference?

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.

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 (resulting in a speedup smaller than 1), with a marginally better, but still worse performance than on a single node for the TaS2 system. The

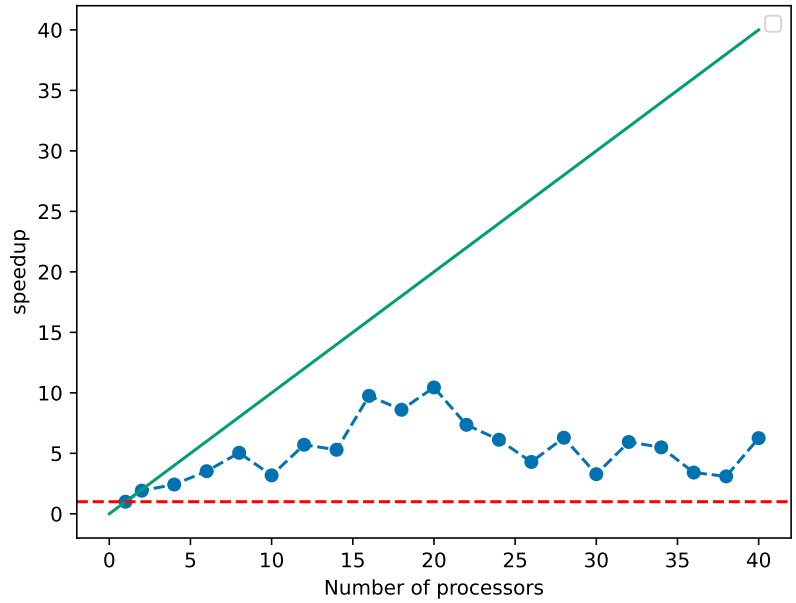


Figure I.3: *Baseline scaling test on the TaS2 benchmarking system* Quantum ESPRESSO 7.0, OpenMPI 4.1.0, `nk 1` and `nd 1`

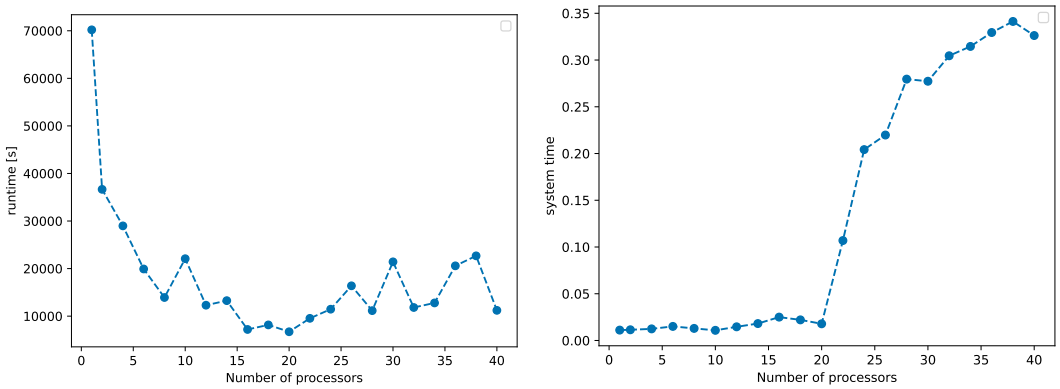


Figure I.4: *Baseline scaling test on the TaS2 benchmarking system* Quantum ESPRESSO 7.0, OpenMPI 4.1.0, `nk 1` and `nd 1`

reason for this can be seen in the plots for system time, as bad parallelization shows itself by introducing waiting times between processors, which makes the system time in some way grow with the the number of processors, as can be seen especially well in I.4. Here the percentage of time used for tasks not directly related to calculations goes from a near constant value for under 20 processors to 35% of the execution time 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

A first strategy for solving issues with parallelization is trying different compilers and mathematical libraries. As discussed in sec. **??**, Quantum ESPRESSO can make use of a variety of software packages available on the PHYSnet cluster. The benchmarks in I.5 are run with the following software combinations:

- OpenMPI 4.1.0 and Quantum ESPRESSO provided BLAS/LAPACK, so the baseline test discussed in sec. I.1
- OpenMPI 4.1.0 and OpenBLAS 0.3.20
- OpenMPI 4.1.0, OpenBLAS 0.3.20 and ScaLAPACK 2.2.0
- Intel oneAPI 2021.4

analysis with right plots for ompi, ompi+openblas, maybe kick out scalapack? it should be the same

Fig. I.5 shows that using Intels Intel oneAPI packages provides optimal scaling for up to round about 6 processors, which means those calculations ran about twice as fast as the calculations with just OpenMPI. This result not only stands for itself as a statement about scaling on a single node, but also provides a basis for scaling beyond this apparent optimal range of 6 processors The k point parallelization explained in sec. **??** can distribute the workload in such a way that processor pools of sizes within this range work on individual k points and as such can provide optimal scaling within one pool while also not losing performance because the pools do not need to communicate with each other in the same order of magnitude as the pools have to communicate within themselves.

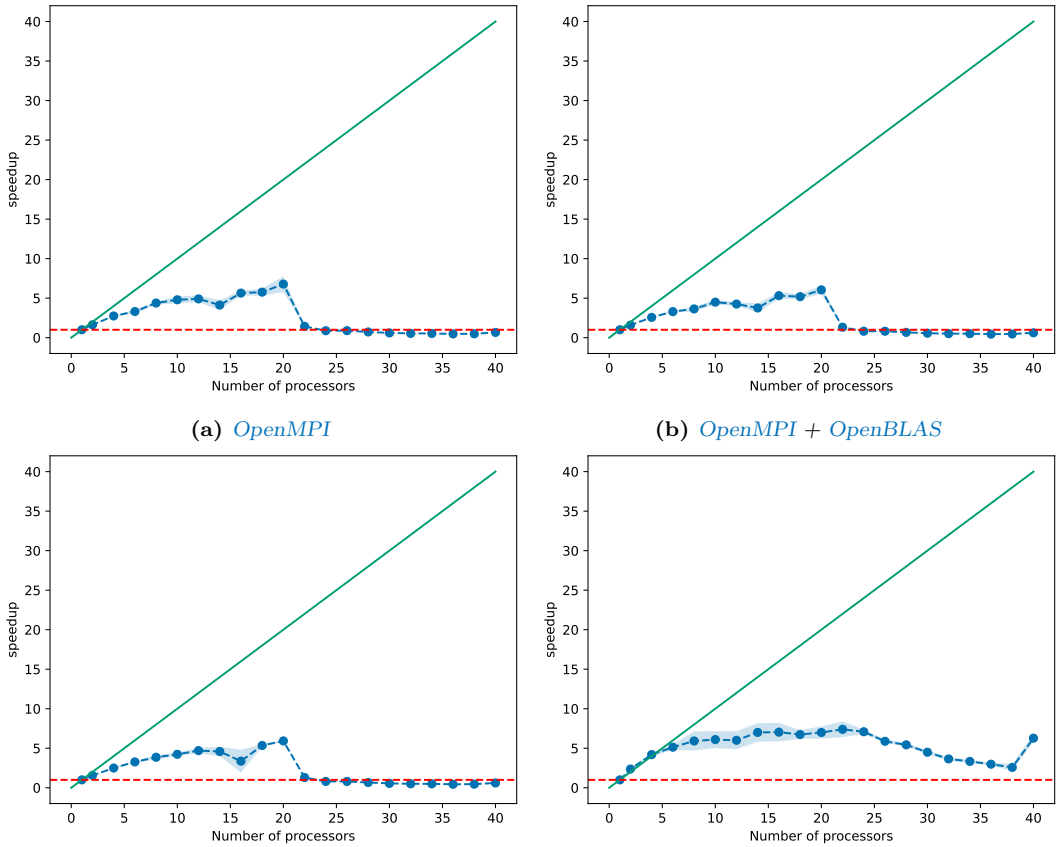
absolute runtimes! so comparison is possible! (singlecore runtimes could also be differen!)

I.3 Using the parallelization 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.

I.3.1 k point parallelization

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



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

(c) *OpenMPI + ScaLAPACK* **(d)** *Intel oneAPI*

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

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.

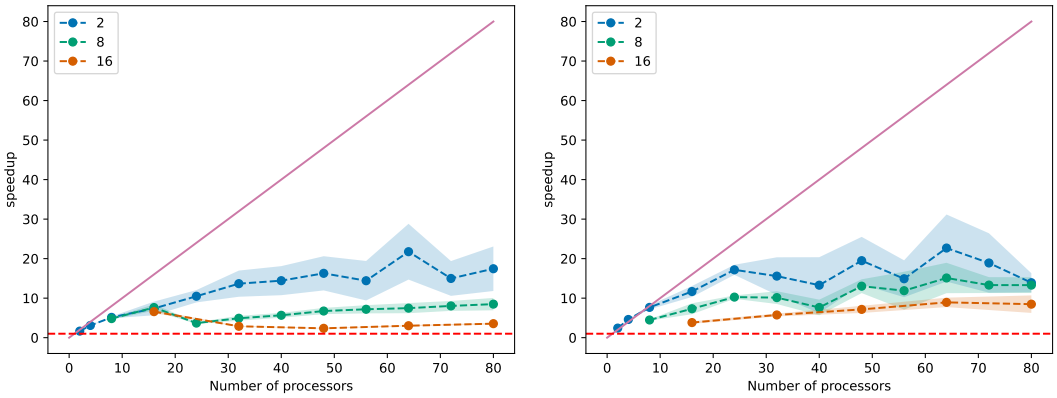
Fig. I.6 shows that using k parallelization with a pool size of 2 significantly improves the scaling behavior, not only on one node, but especially over more than one node.

more analysis: difference between poolsizes

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 a similar 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.



(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*

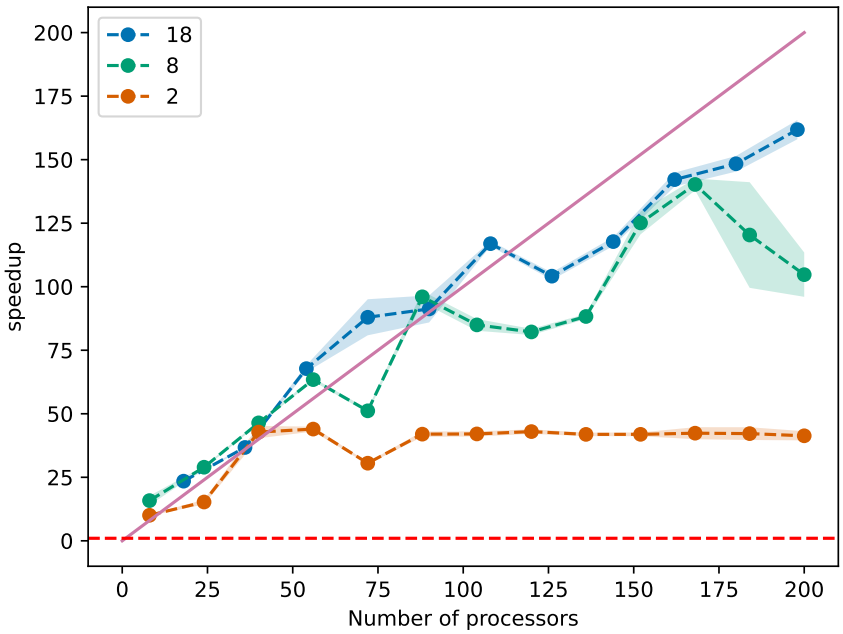


Figure I.7: *Benchmark with k-point parallelization for the TaS2 benchmarking system*

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 about 4 % and 6 % of the whole wall time, without any kind of systemic increase over any range of processors.

more analysis: difference between poolsizes

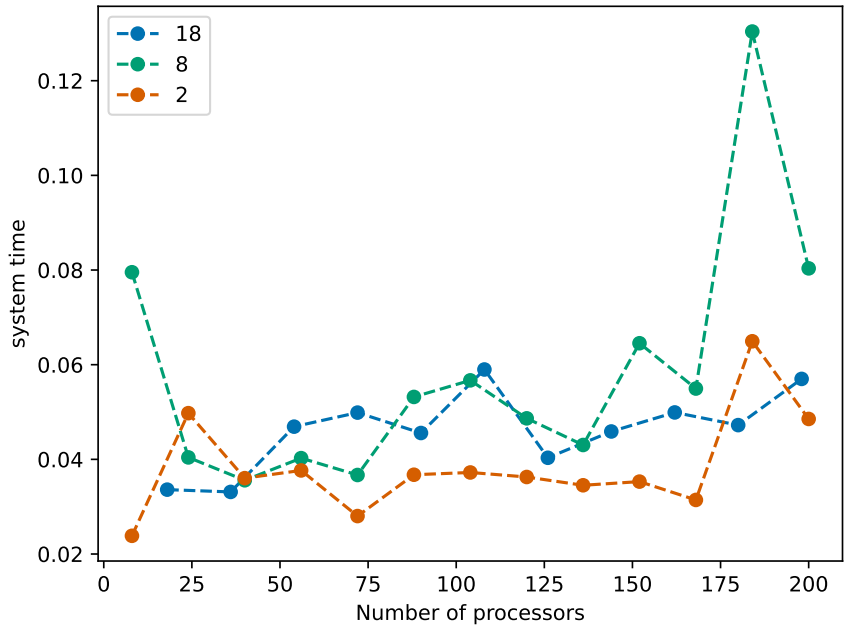


Figure I.8: *Idle time for the k point parallelization benchmark for the TaS2 system*

I.3.2 Linear algebra parallelization

Linear Algebra

I.4 Comparison with calculations on the HLRN cluster

I.5 Conclusion: Parameters for optimal scaling