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 ground state density and 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 module 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.

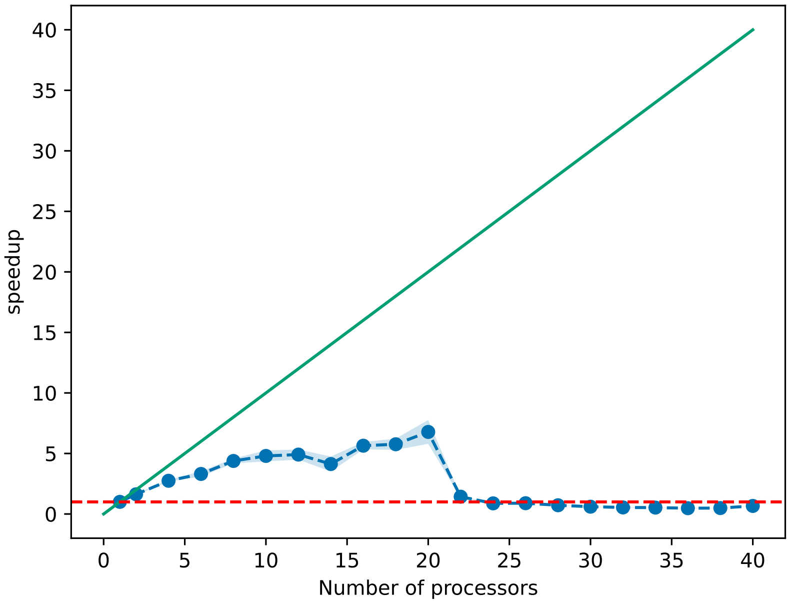


Figure I.1: *Baseline scaling test on the Si benchmarking system* Quantum ESPRESSO 7.0, OpenMPI 4.1.0, nk 1 and nd 1

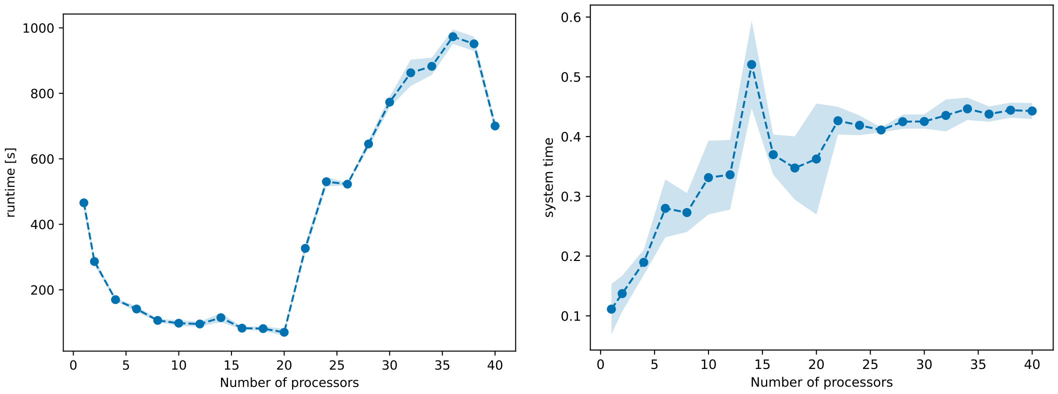


Figure I.2: *Baseline scaling test on the Si benchmarking system* QUANTUM ESPRESSO 7.0, OpenMPI 4.1.0, `nk 1` and `nd 1`

As discussed in sec. **??**, three different metrics of scalability can be deduced from the time data given by QUANTUM ESPRESSO :

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

These are pictured in fig. I.1 and I.2 for the silicon benchmarking system.

On a single node, the speedup does scale linearly with the number of processors until around 10 processors, but with a slope of $\frac{1}{2}$ instead of 1 (which would mean ideal scaling). Beyond that number, the slope decreases even more so that a maximal speedup of around 7 is achieved for 20 processors used. As discussed in sec. **??**, one node is equipped with 20 cores, so trying to scale the communication intensive calculations beyond that threshold makes the calculations run even slower than on a single core. Interestingly, the wait time plot in I.2 shows that a significant amount (10 % to 40 %) of runtime is taken up by wait time already for less than 20 processors. As discussed in sec. **??**, this hints to poor parallelization, which is a reason for the poor scaling seen in fig. I.1.

Pictured in fig. I.3 and I.4 are the same scaling test run for the TaS2 benchmarking system.

Both benchmarks share a key feature in the linear scaling in speedup with a slope of $\frac{1}{2}$ up to 10 processors, the tests on TaS2 actually continue this trend up until 20 processors, resulting in a maximal speedup of 10. Beyond 20 processors the scaling also gets worse, but there still is a speedup of 2 to 3. Looking at the wait time plot in fig. I.4 shows the quality of parallelization: for up to 20 processors, the wait time is independent of the number of processors, which is the unavoidable startup overhead for tasks like data distribution. After that, the wait time scales with the number of processors, which hints to communication being a factor limiting speedup here as discussed in sec. **??**.

In conclusion, systems with more electrons and by extension bigger matrices and longer iteration times seem to be parallelize better and as such profit more from using more processors than systems with just a few number of electrons.

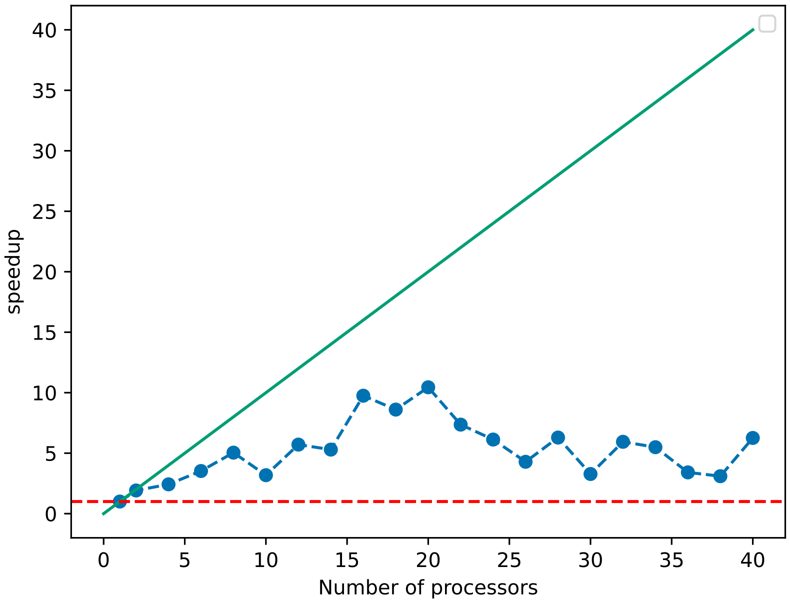


Figure I.3: *Baseline scaling test on the TaS2 benchmarking system* Quantum ESPRESSO 7.0, OpenMPI 4.1.0, `nk 1` and `nd 1`

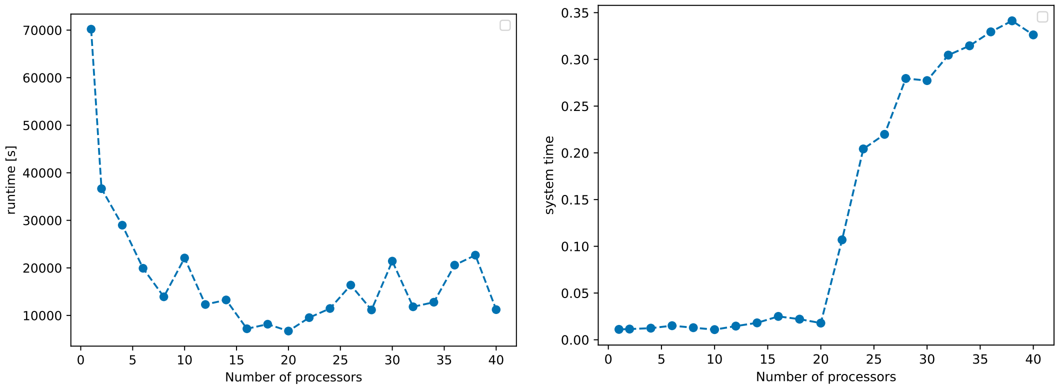


Figure I.4: *Baseline scaling test on the TaS2 benchmarking system* Quantum ESPRESSO 7.0, OpenMPI 4.1.0, `nk 1` and `nd 1`

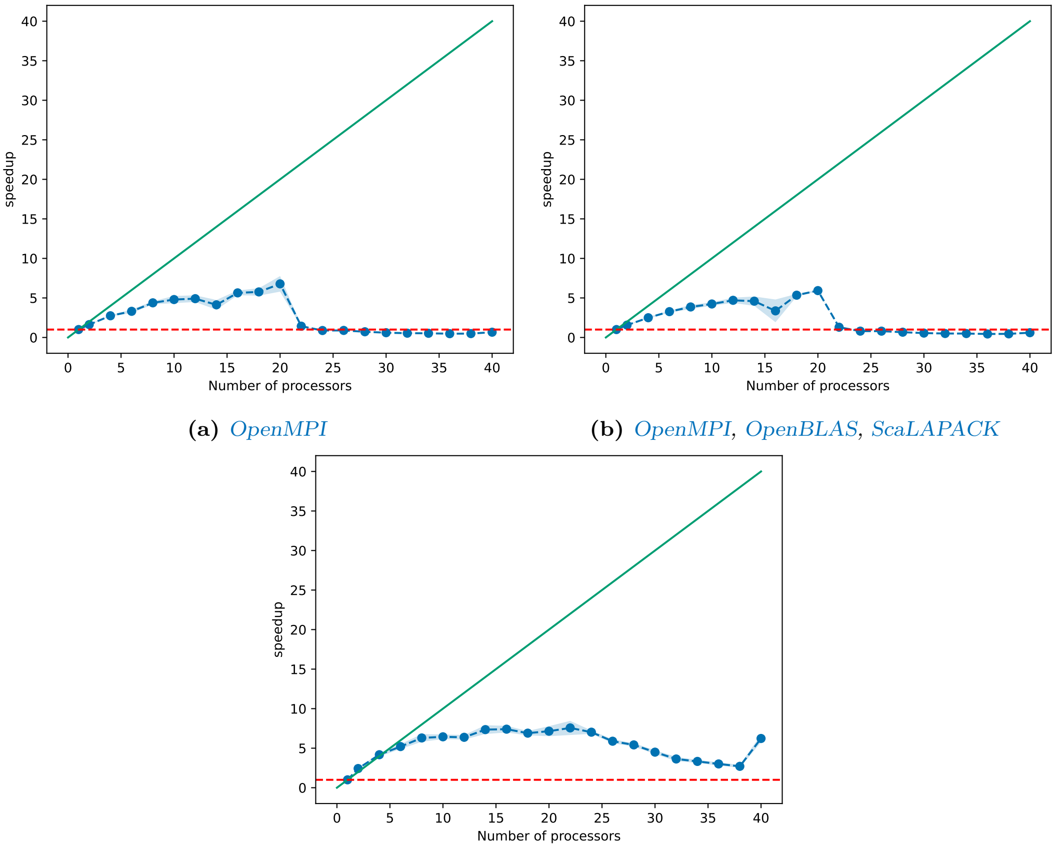
These scaling tests pose now two questions to be answered:

- 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, OpenBLAS 0.3.20 and ScaLAPACK 2.2.0
- Intel oneAPI 2021.4



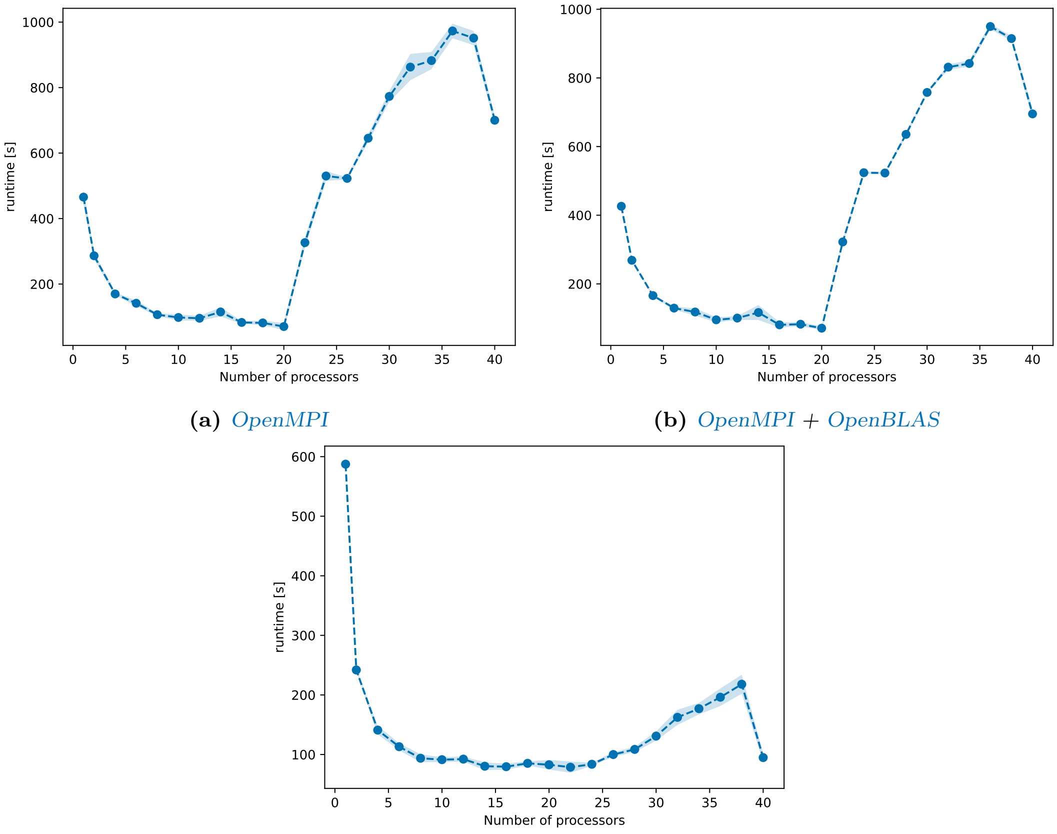
(a) *OpenMPI*

(b) *OpenMPI*, *OpenBLAS*, *ScaLAPACK*

(c) *Intel oneAPI*

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

Fig. I.5 shows that just dropping in another BLAS library (OpenBLAS in this case) does not change the scaling behavior, in contrast to using Intels Intel oneAPI packages. Here, optimal scaling behavior is seen for up to 6 processors, which means those calculations ran about twice as fast as the calculations with just OpenMPI.



(a) *OpenMPI*

(b) *OpenMPI + OpenBLAS*

(c) *Intel oneAPI*

Figure I.6: *Baseline scaling test Si benchmarking system with different combinations of compilers and mathematical libraries*

Fig. I.6 shows

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.

analyse absolute runtime -> speedup can be deceiving

why intel? seems to work best with the infiniband hardware, absolute runtime faster for like 2-4 processors? -> can hint to better perfomance with k point parallelization

efficiency maybe?

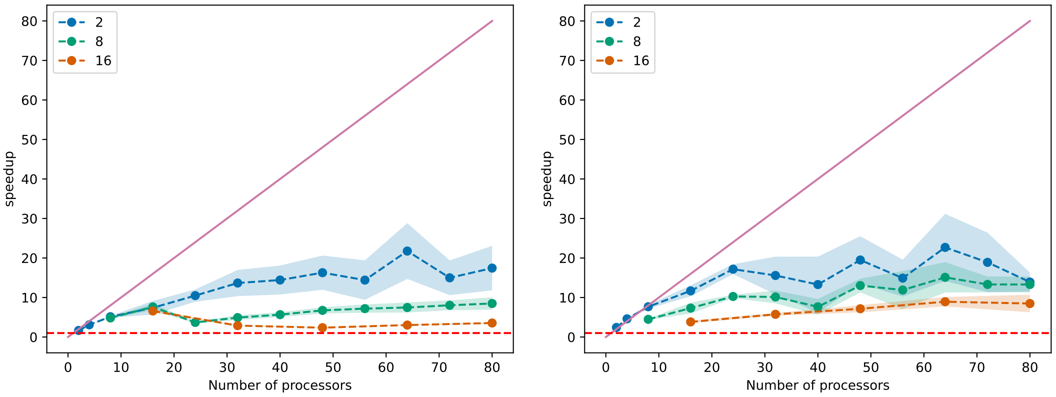
TaS2 intel scaling?

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.7 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.7 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*

(b) *Intel oneAPI*

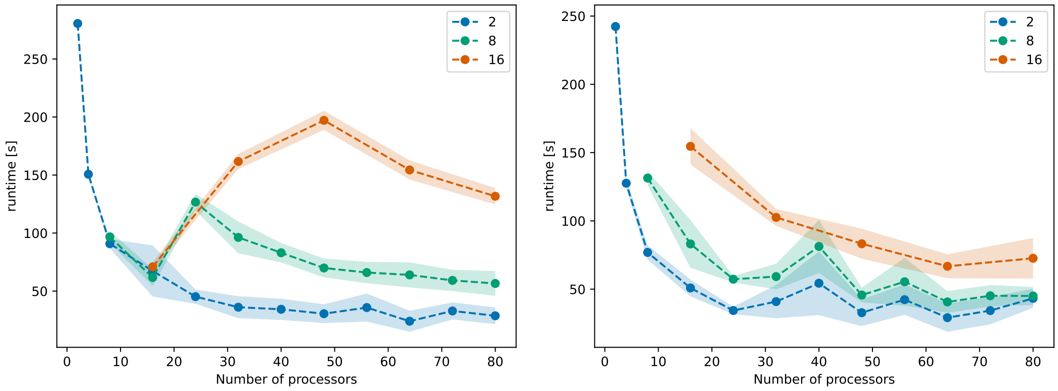
Figure I.7: *Benchmark with k-point parallelization for the Si benchmarking system with 3 different sizes of processor pools*

Fig. I.7 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.7 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.9, with a similar list of pool sizes, but over a wider range of processors.



(a) OpenMPI **(b)** Intel oneAPI

Figure I.8: *Benchmark with k-point parallelization for the Si benchmarking system with 3 different sizes of processor pools*

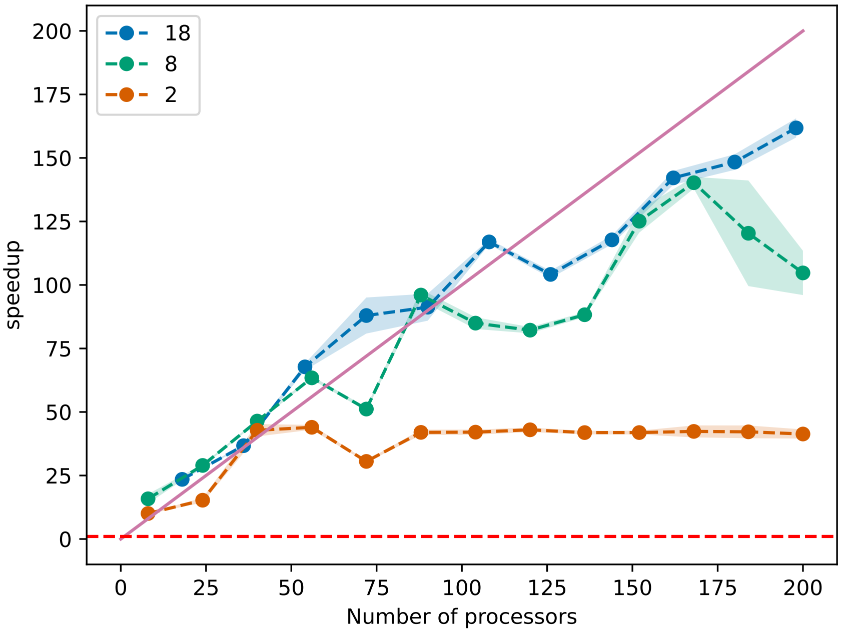


Figure I.9: *Benchmark with k-point parallelization for the TaS2 benchmarking system*

Remarkably, the scaling behavior is swapped in comparison to I.7, 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.

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

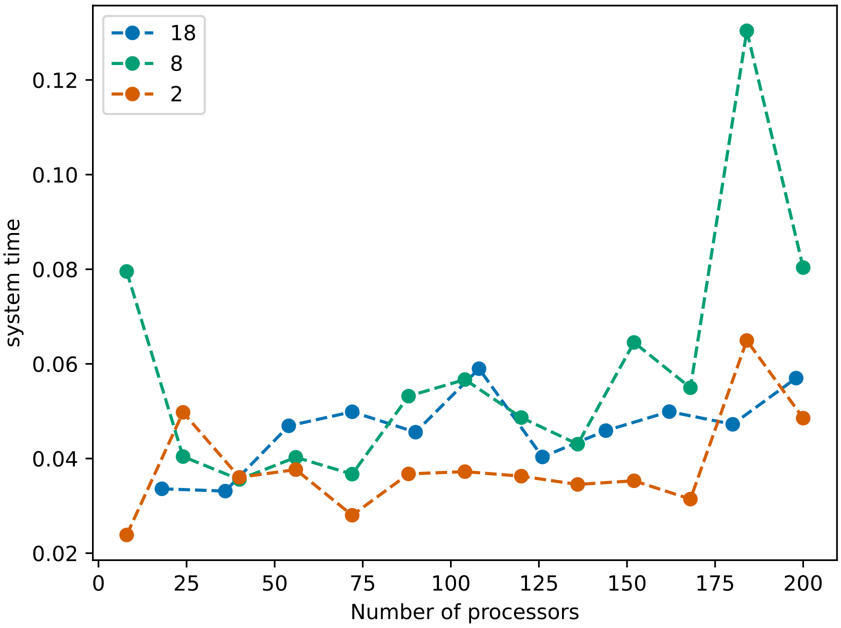


Figure I.10: *Idle time for the k point parallelization benchmark for the TaS2 system*

Fig. I.10 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

I.3.2 Linear algebra parallelization

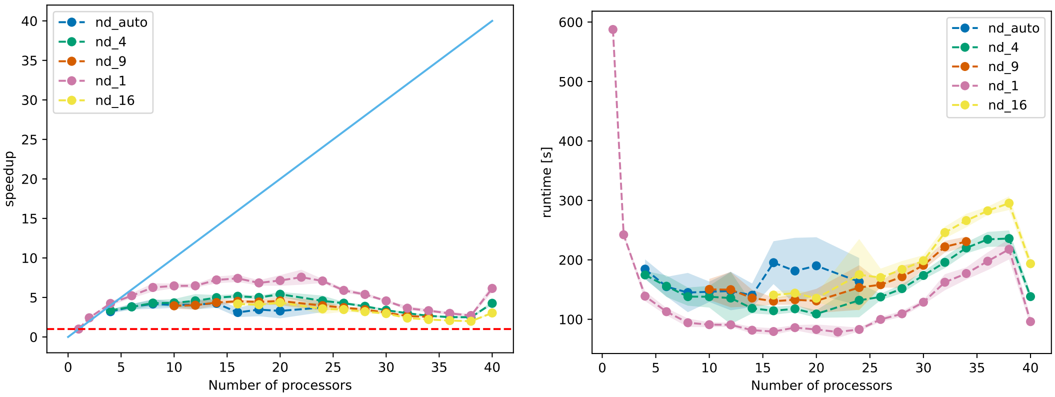


Figure I.11: *Benchmark with linear algebra parallelization for the silicon benchmarking system*

Fig. ?? shows the scaling behavior for different values of the parameter **nd**. Here, nd_auto means that no value for **nd** is specified so Quantum ESPRESSO automatically chooses the biggest square number smaller than the number of processors. It is clearly shown that using linear algebra parallelization slows the calculation down significantly for the silicon system.

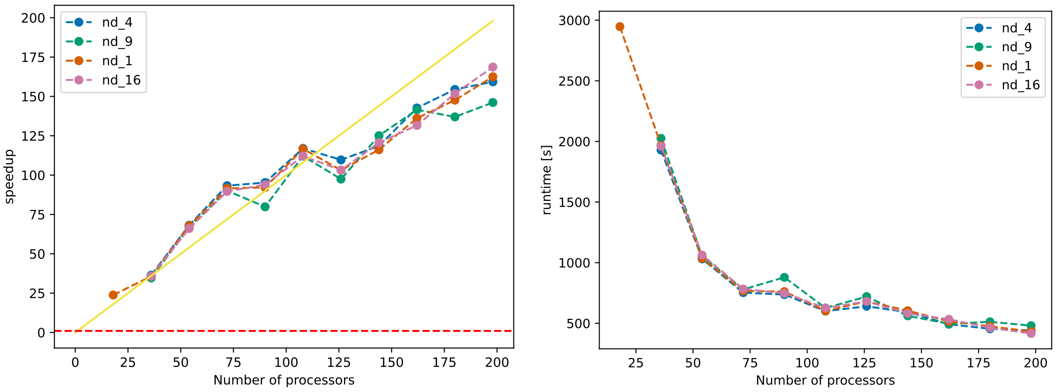


Figure I.12: *Benchmark with linear algebra parallelization for the TaS2 benchmarking system*

Interestingly, this again is not reproduced for the more expensive TaS2 benchmarking system. Fig. ?? shows a pretty much consistent times across all values for **nd**.

Those results are already hinted at in the `PWscf` user guide [1]. Here, in the guide for choosing parallelization parameters, using linear algebra parallelization is recommended when the number of KS states is a few hundred or more. The silicon system has 8 electrons and is as such described with 8 Kohn-Sham (KS) states, the TaS2 system has 153 electrons, so Quantum ESPRESSO uses 92 KS states (in case of metallic materials, the band occupation is smeared around the Fermi energy to avoid level crossings, so more KS states than $\frac{1}{2} *$ (number of electrons) are needed to account for that). Apparently, this number of KS states is on the edge of linear algebra parallelization actually speeding up calculations.

I.4 Comparison with calculations on the HLRN cluster

I.5 Conclusion: Parameters for optimal scaling

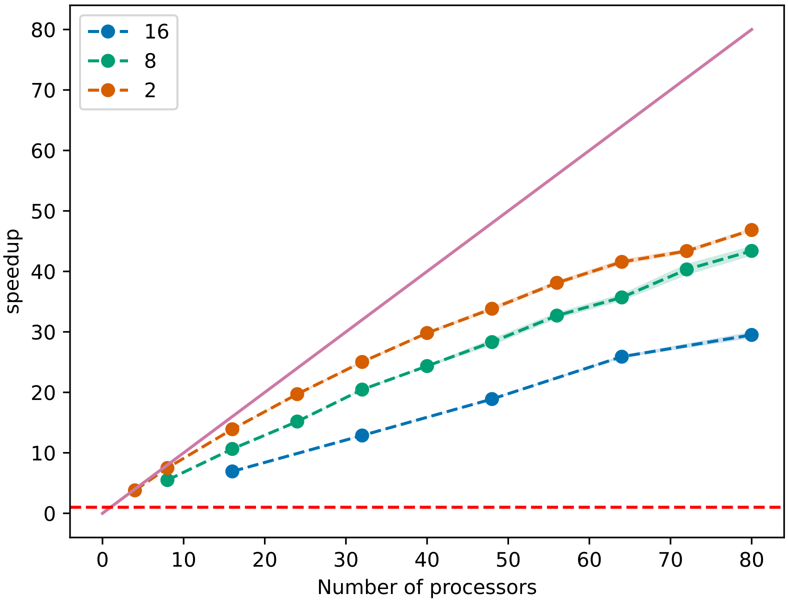


Figure I.13: *CAPTION*

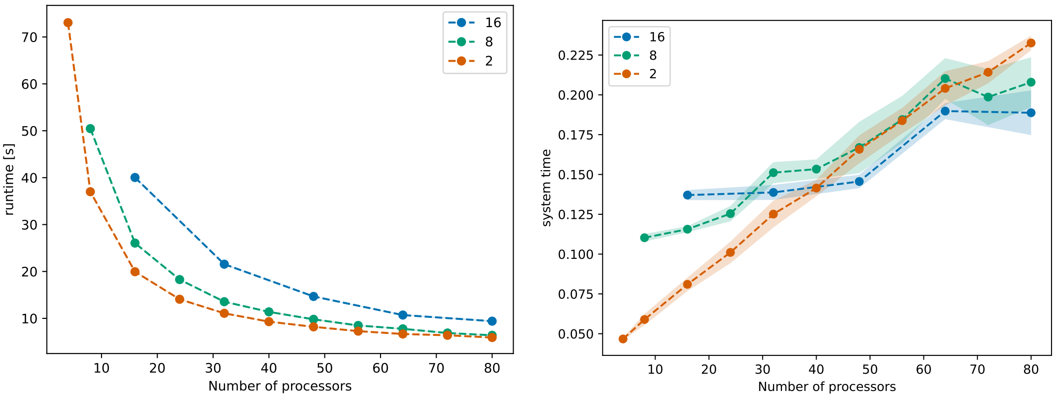


Figure I.14: *CAPTION*