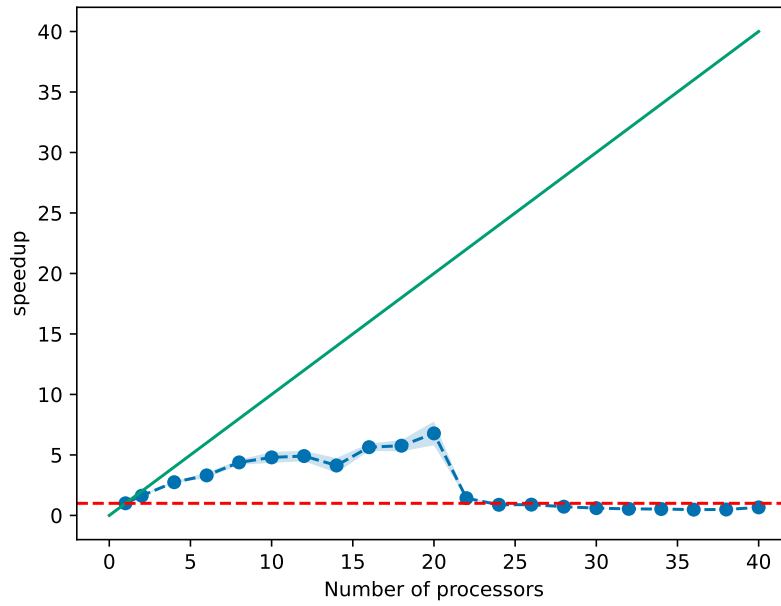


# I Parallelisation of electronic-structure calculations

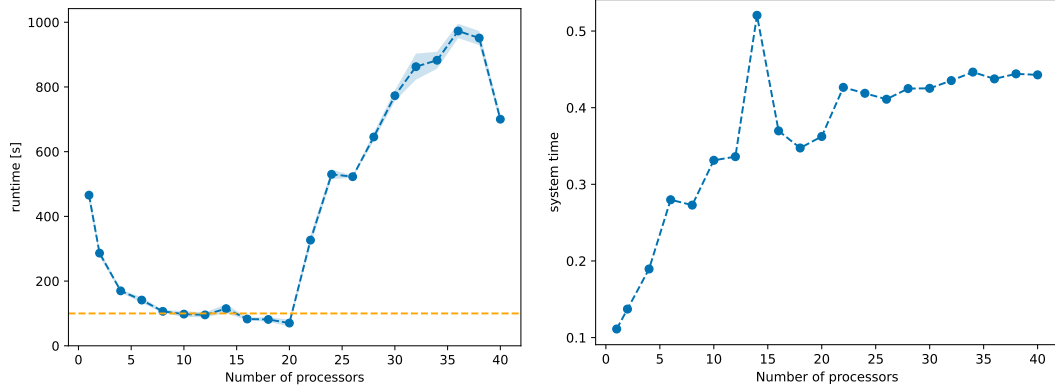
The `PWscf` (Plane-Wave Self-Consistent Field) package is one of the core modules of QUANTUM ESPRESSO, as many other modules need ground state density and total energy as input. This chapter deals with examining the best ways to run `PWscf` calculations in the `scf` mode.

## 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 applied. In Fig. I.1 to I.4 two scaling tests on the earlier mentioned benchmarking systems Si and TaS<sub>2</sub> 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:** Scalability for the Si benchmarking system, QUANTUM ESPRESSO 7.0, OpenMPI 4.1.0, nk 1 and nd 1



**Figure I.2:** Absolute runtime and wait time for the scalability test on the Si benchmarking system, QUANTUM ESPRESSO compiled with 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 :

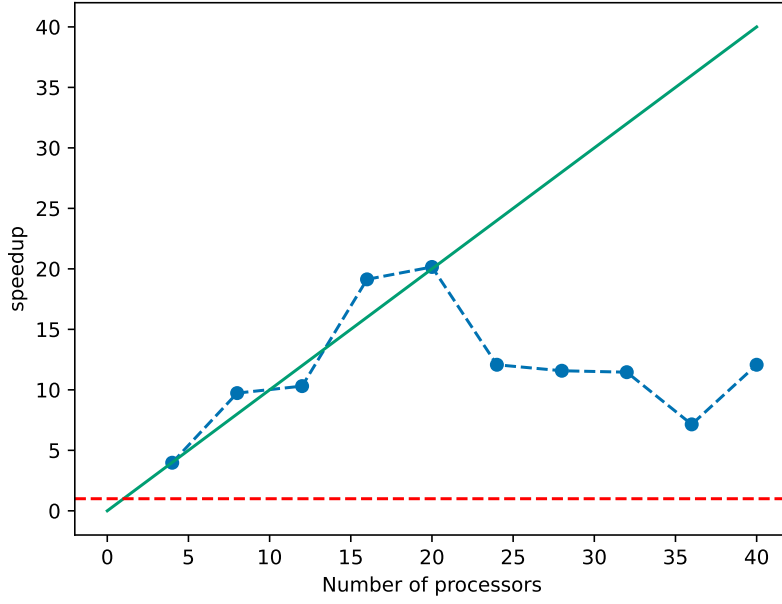
- runtime: absolute runtime (walltime) of the compute job
- speedup: runtime divided by runtime of the job on a single core
- wait time: percentage of wall time used by system tasks, e.g. writing to disk, etc.

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

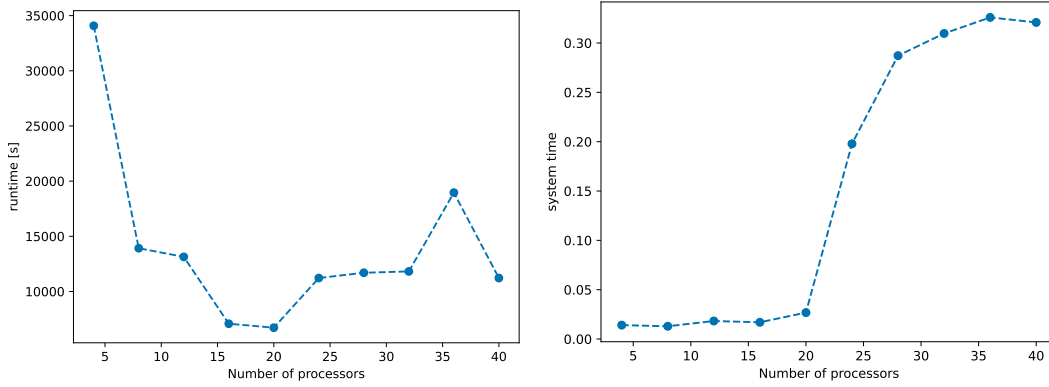
On a single node, the speedup does scale linearly with the number of processors until around 10 processors, but with a slope of  $\frac{1}{2}$  instead of 1 (which would mean ideal scaling). Beyond that number, the slope decreases even more so that a maximal speedup of around 7 is achieved for 20 processors used. One compute 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 is a sign of poor parallelization, which can explain the poor scaling seen in fig. I.1.

Pictured in fig. I.3 and I.4 are the same scaling test run for the TaS<sub>2</sub> benchmarking system. Here, the speedup is not taken as runtime divided by runtime on a single core, as the memory required is more than what can be accessed by a single core. Instead, an estimate of the single core runtime is made by multiplying the runtime of the job running on 4 cores by 4. This assumes perfect scaling for 1-4 processors, but the relative scaling is accurate, no matter the accuracy of this assumption.

The scaling test on the TaS<sub>2</sub> system shows much better scaling. For up to 20 processors, the speedup follows the ideal scaling with a stark decline with more processors. This is also reflected in the wait time in fig. I.4, as it goes from a small constant value for under 20 processors to some kind of dependence of the number of processors, which hints to communication or bottlenecks being a limiting factor here.



**Figure I.3:** Scalability for the  $TaS_2$  benchmarking system, QUANTUM ESPRESSO 7.0, OpenMPI 4.1.0, `nk 1` and `nd 1`



**Figure I.4:** Absolute runtime and wait time for the scalability test on the  $TaS_2$  benchmarking system, QUANTUM ESPRESSO 7.0, OpenMPI 4.1.0, `nk 1` and `nd 1`

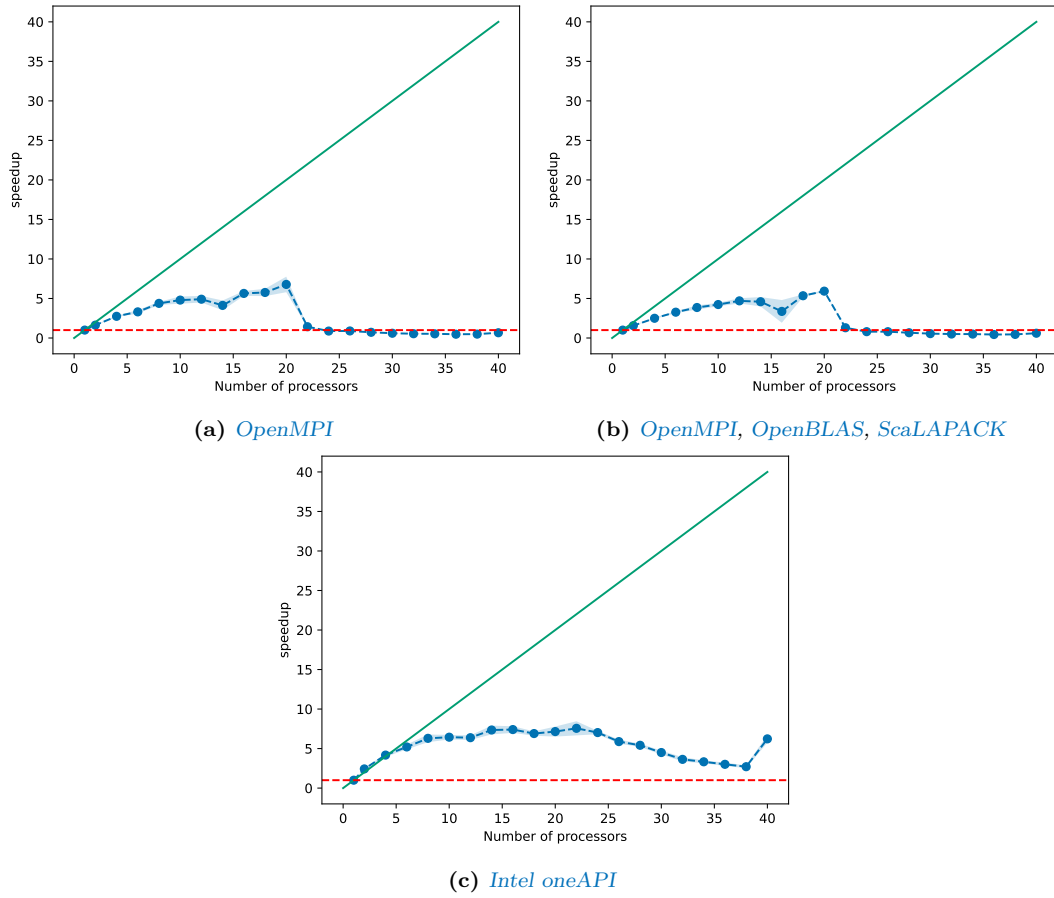
In conclusion, systems with more electrons and by extension bigger matrices and longer iteration times seem to be parallelize better and as such profit more from using more processors than systems with just a few number of electrons.

These scaling tests poses now the question how better scaling over more than one node can 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 ?? are run with the following software combinations:

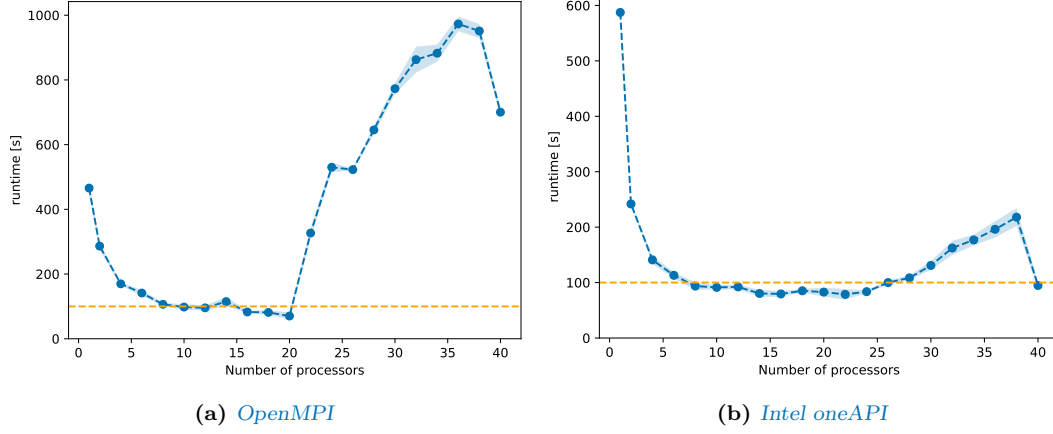
- OpenMPI 4.1.0 and QUANTUM ESPRESSO provided BLAS/LAPACK, so the baseline test discussed in sec. I.1
- OpenMPI 4.1.0, OpenBLAS 0.3.20 and ScaLAPACK 2.2.0
- Intel oneAPI 2021.4



**Figure I.5:** Scalability for the Si benchmarking system with different combinations of compilers and mathematical libraries, `nk 1` and `nd 1`

Fig. ?? 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. It is however important to also look at the total runtime in this context.



**Figure I.6:** Comparison of absolute runtimes between *QUANTUM ESPRESSO* compiled with *OpenMPI* and *Intel oneAPI* for the *Si* benchmarking system, *nk* 1 and *nd* 1

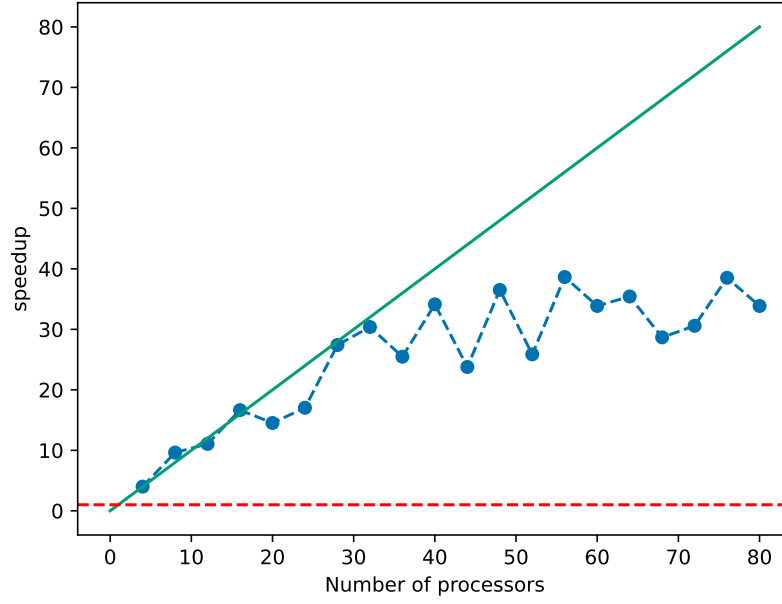
Fig. ?? shows the absolute runtime for both the *OpenMPI* and *Intel oneAPI* benchmarks. This explains the difference in scaling seen in the speedup plots: the runtime on a single core is significantly higher for the *Intel oneAPI* benchmark, so even though the runtime between both benchmarks is about the same starting from around 10 processors there is a difference in speedup. To assess this more quantitatively, tab. I.1 lists the average runtime for some selected number of processors. Importantly, the runtime for the *Intel oneAPI* benchmark is faster for smaller numbers of processors (except 1), but only 15 % for 2 cores and even smaller differences for more cores, with the *OpenMPI* calculation being even a little faster for 20 processors.

**Table I.1:** Selected absolute runtimes of *QUANTUM ESPRESSO* compiled with *OpenMPI* and *Intel oneAPI* for the *Si* benchmarking system, *nk* 1 and *nd* 1

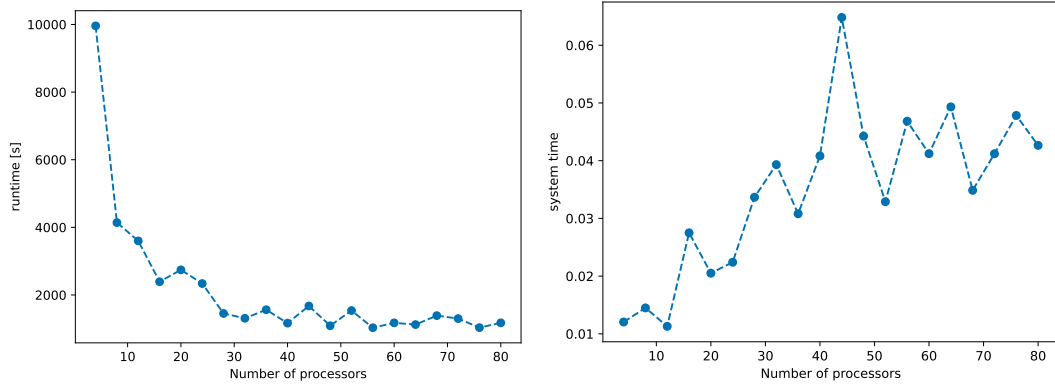
Number of processors	<i>OpenMPI</i>	<i>Intel oneAPI</i>
1	466 s	587 s
2	286 s	242 s
4	170 s	141 s
10	97.9 s	91.3 s
20	70.2 s	82.8 s

The same benchmark with the *Intel oneAPI* compiled version of *QUANTUM ESPRESSO* is shown in fig. I.7.

For this system, the speedup follows Amdahl's law, discussed in sec. ?? with a linear growth in speedup up to 32 processors with a saturation and only a small gain in speedup with more processors. In contrast to the benchmark with just *OpenMPI* (fig. I.3) there is no drop in speedup after 20 processors.



**Figure I.7:** Scalability for the  $\text{TaS}_2$  benchmarking system, QUANTUM ESPRESSO 7.0 compiled with Intel oneAPI 2021.4, `nk 1` and `nd 1`



**Figure I.8:** Absolute runtime and wait time for the scalability test on the  $\text{TaS}_2$  benchmarking system, QUANTUM ESPRESSO compiled with Intel oneAPI 2021.4, `nk 1` and `nd 1`

TaS2 intel scaling

Moreover, the absolute runtime shown in fig. I.8

This result not only stands for itself as a statement about scaling on a single node, but also provides a basis for scaling beyond the respective optimal ranges of processors for both systems: 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. Keeping the results of this section in mind, at least an for the quality k point parallelization can already be made: For the silicon system, the size of pools should be bigger than 6 processors for optimal scaling and for the TaS<sub>2</sub> system they should not be bigger than 32 processors.

## I.3 Using the parallelization parameters of Quantum ESPRESSO

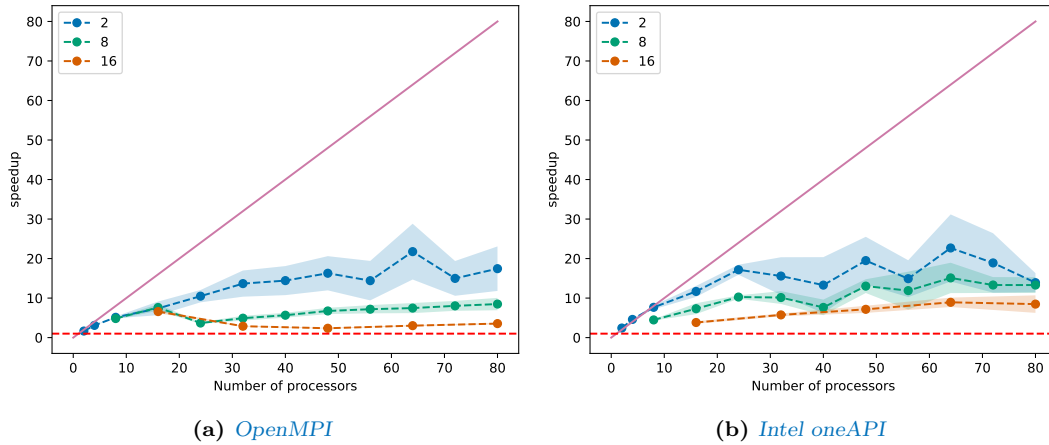
As detailed in section ??, *QUANTUM ESPRESSO* offers ways to manage how the workload is distributed among the processors. In `pw.x` the default plane wave parallelization, k-point-parallelization and linear-algebra parallelization are implemented.

already spoke about k point in the last section, maybe have a better transition here?

### I.3.1 k point parallelization

The benchmark pictured in I.9 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.9 shows the scaling for poolsizes 2, 8 and 16 for *QUANTUM ESPRESSO* being compiled with OpenMPI/Scalapack and Intel oneAPI.

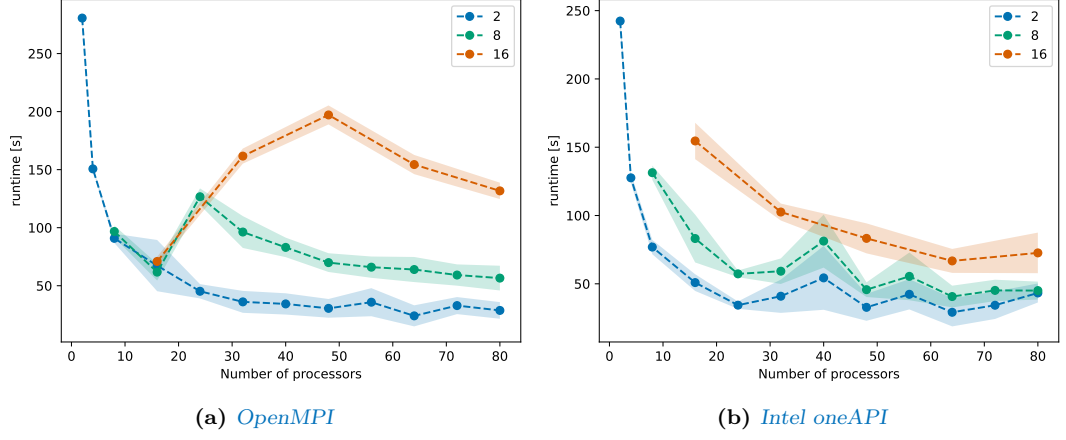


**Figure I.9:** Scalability utilizing *k*-point parallelization for the Si benchmarking system with 3 different sizes of processor pools. The size is determined by the parameter `nk` via size of pools = number of processors / `nk`.

Fig. I.9 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.

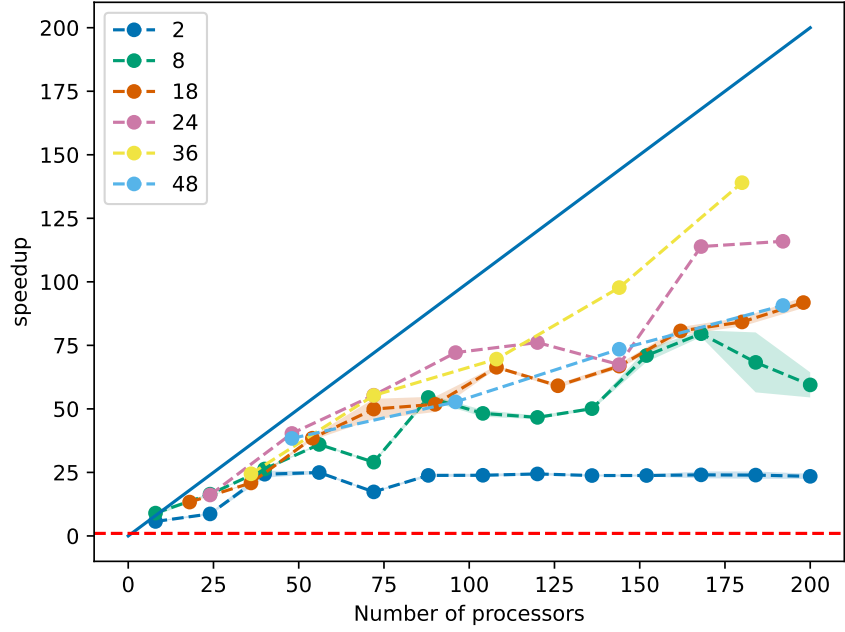
more analysis: difference between poolsizes

analyse absolute runtimes



**Figure I.10:** Absolute runtime for the scalability test with  $k$ -point parallelization for the Si benchmarking system with 3 different sizes of processor pools, `nd 1`

The same scaling test is applied to the TaS<sub>2</sub> system in fig. I.11.



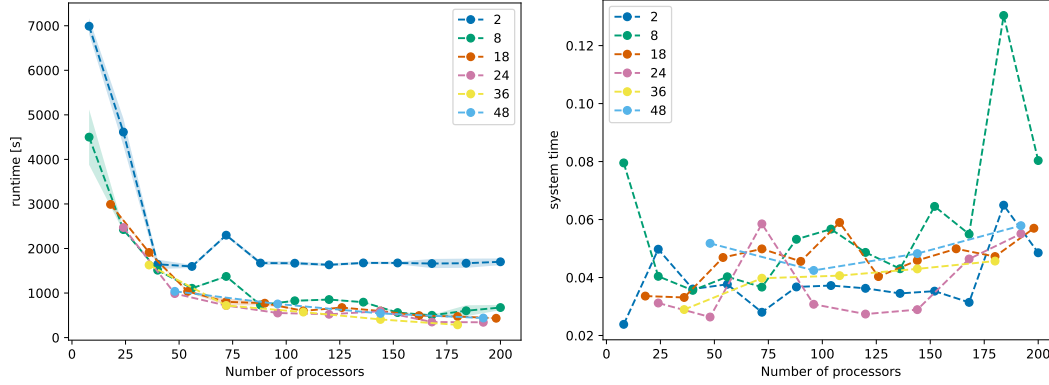
**Figure I.11:** Scalability utilizing  $k$ -point parallelization for the TaS<sub>2</sub> benchmarking system, QUANTUM ESPRESSO compiled with Intel oneAPI 2021.4, `nd 1`

Remarkably, the scaling behavior is swapped in comparison to fig. I.9, as the pool size 2 saturates and the bigger pool sizes show way better scaling behavior. As already alluded to



in sec. I.2, the calculations on the TaS<sub>2</sub> system profit more from parallelization and as such scale better for bigger pool sizes up until 36 processors in one pool, which is the upper limit established in the benchmark just over the number of processors.

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



**Figure I.12:** Absolute runtime and wait time for the scalability test with  $k$ -point parallelization for the TaS<sub>2</sub> benchmarking system, QUANTUM ESPRESSO compiled with Intel oneAPI 2021.4, nd 1

Fig. ?? 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.

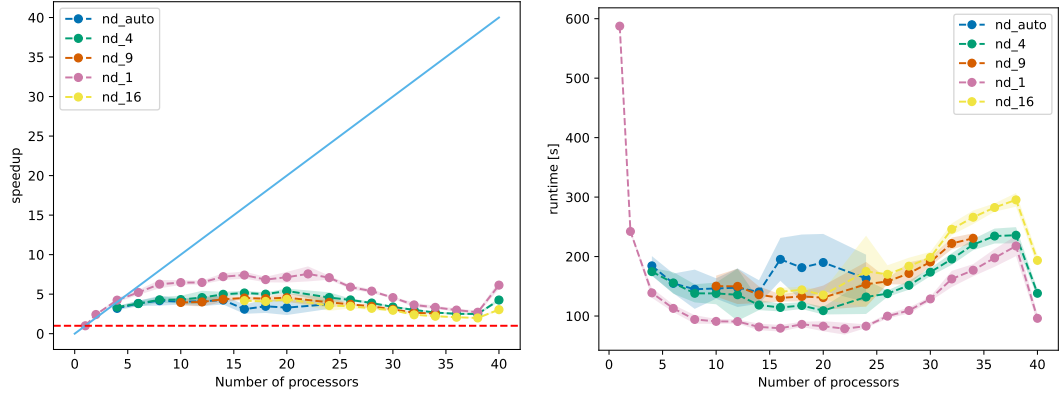
more analysis:  
difference between pool sizes

### I.3.2 Linear algebra parallelization

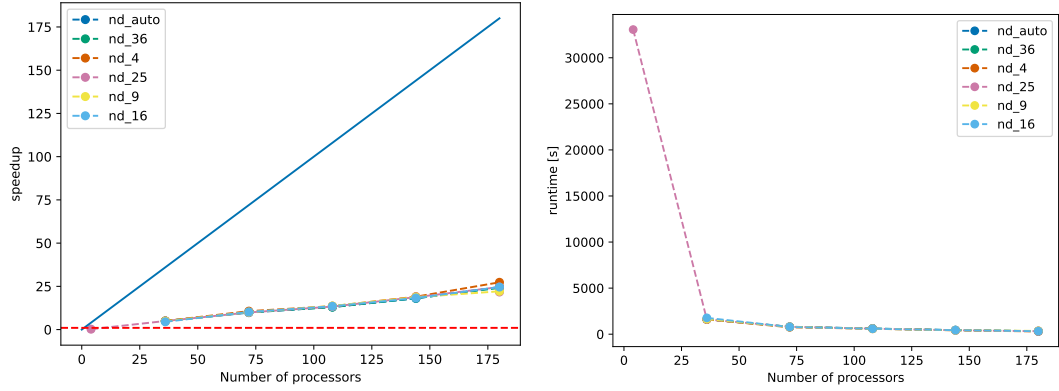
Fig. ?? shows the scaling behavior for different values of the parameter `nd`. Here, `nd_auto` means that no value for `nd` is specified so QUANTUM ESPRESSO automatically chooses the biggest square number smaller than the number of processors. It is clearly shown that using linear algebra parallelization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

Interestingly, this again is not reproduced for the more expensive TaS<sub>2</sub> 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 4 **Kohn-Sham (KS)** states, the TaS<sub>2</sub> 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  $\frac{1}{2} * (\text{number of electrons})$  are needed to account for that). Evidently, this number of **KS** states is on the edge of linear algebra parallelization actually speeding up calculations.



**Figure I.13:** Scalability and runtime utilizing linear algebra parallelization for the *Si* benchmarking system, QUANTUM ESPRESSO compiled with Intel oneAPI 2021.4, `nk 1`



**Figure I.14:** Scalability and runtime utilizing linear algebra parallelization for the *TaS<sub>2</sub>* benchmarking system, QUANTUM ESPRESSO compiled with Intel oneAPI 2021.4, `nk` chosen such that pool size = 36

## I.4 Comparison with calculations on the HLRN cluster

## I.5 Conclusion: Parameters for optimal scaling

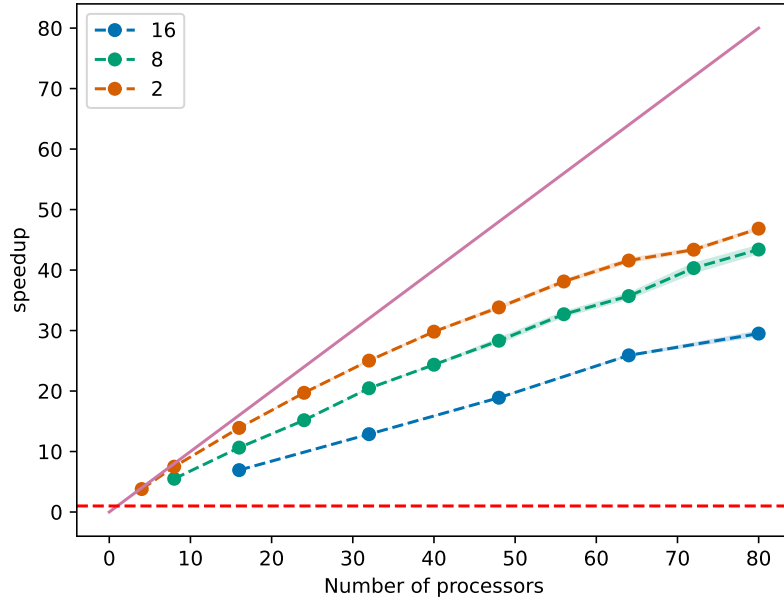


Figure I.15: CAPTION

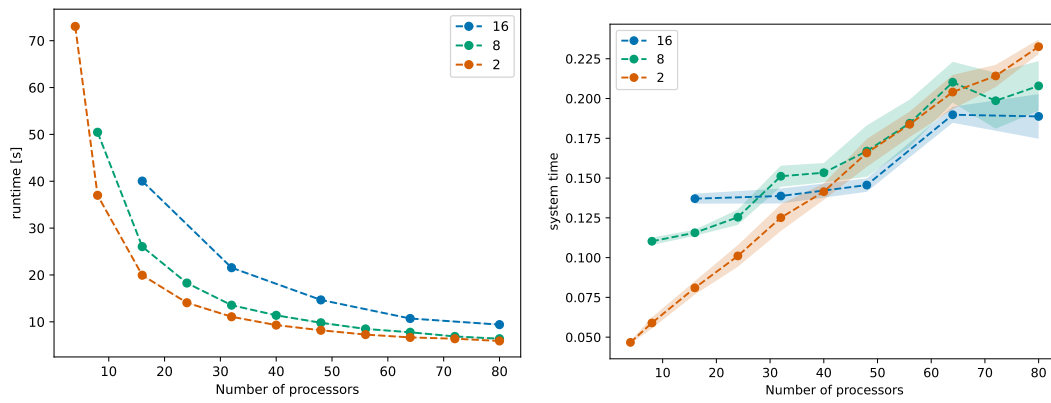


Figure I.16: CAPTION

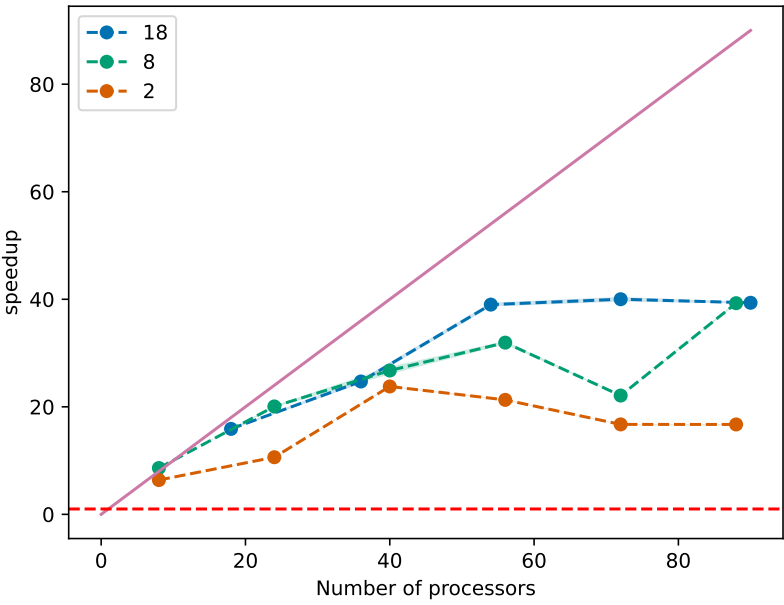


Figure I.17: *CAPTION*

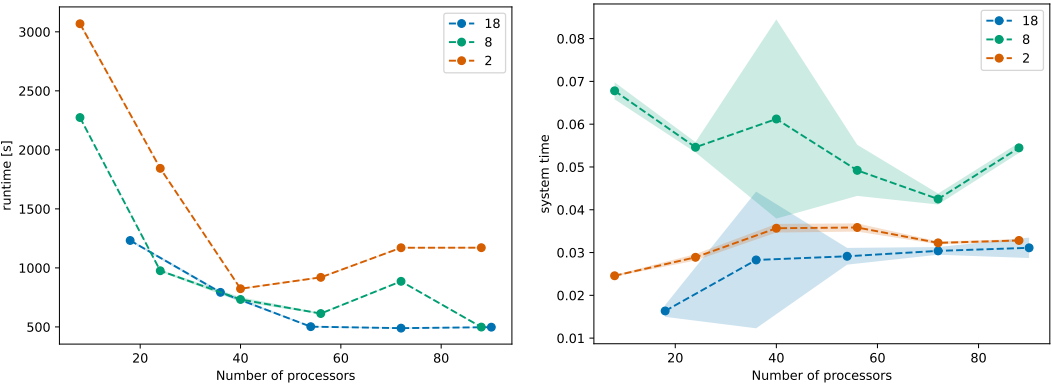


Figure I.18: *CAPTION*