

I Parallelization 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 analyzing the scaling of the `PWscf` module is to perform a baseline scaling test without any optimizations applied. In Fig. I.1 to I.4 two scaling tests on the earlier mentioned benchmarking systems Si and TaS₂ are depicted. 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 optimization parameters mentioned in section ?? used.

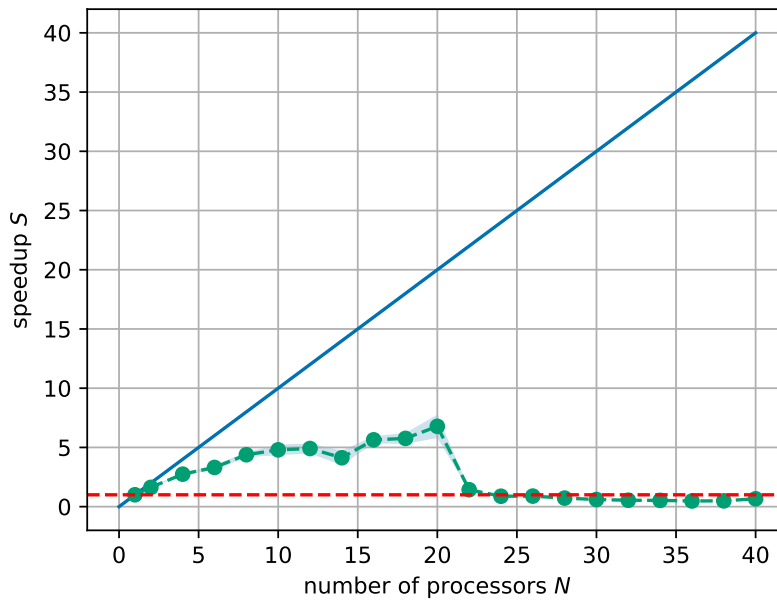


Figure I.1: Scalability for the Si benchmarking system, QUANTUM ESPRESSO 7.0, [OpenMPI 4.1.0](#), nk 1 and nd 1

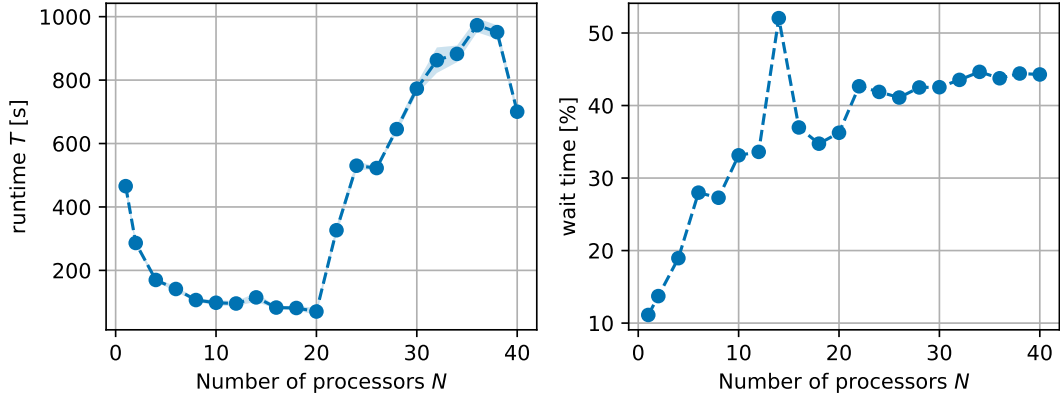


Figure I.2: Absolute runtime and wait time for the scalability test on the Si benchmarking system, QUANTUM ESPRESSO compiled with OpenMPI 4.1.0, `nk 1` and `nd 1`

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 depicted 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.

shown in fig. I.3 and I.4 are the same scaling test run for the TaS₂ 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 TaS₂ 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.

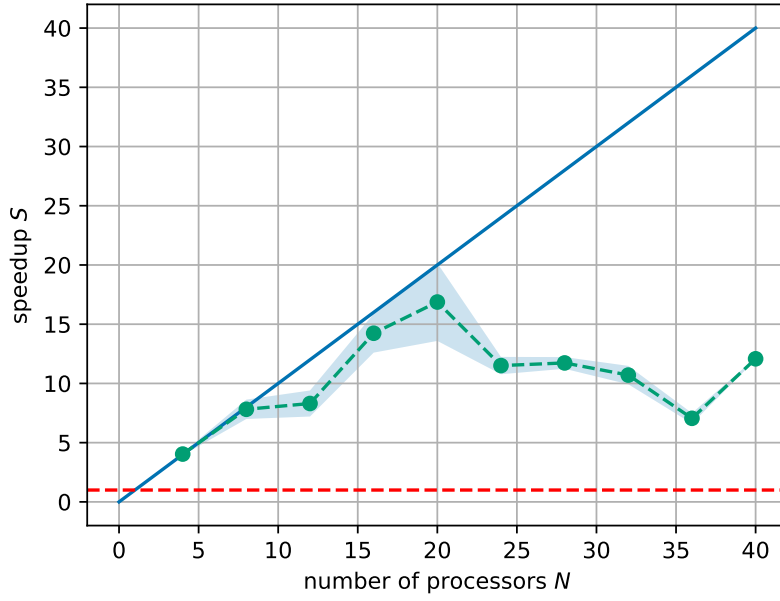


Figure I.3: Scalability for the TaS_2 benchmarking system, QUANTUM ESPRESSO 7.0, OpenMPI 4.1.0, $\text{nk } 1$ and $\text{nd } 1$

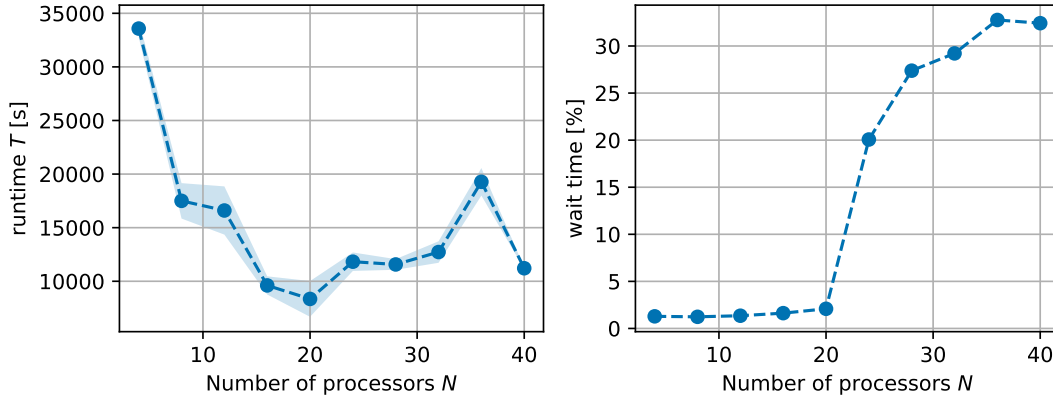


Figure I.4: Absolute runtime and wait time for the scalability test on the TaS_2 benchmarking system, QUANTUM ESPRESSO 7.0, OpenMPI 4.1.0, $\text{nk } 1$ and $\text{nd } 1$

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 now pose 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:

- (a) [OpenMPI](#) 4.1.0 and QUANTUM ESPRESSO provided [BLAS/LAPACK](#), so the baseline test discussed in sec. I.1
- (b) [OpenMPI](#) 4.1.0, [OpenBLAS](#) 0.3.20 and [ScaLAPACK](#) 2.2.0
- (c) [Intel oneAPI](#) 2021.4

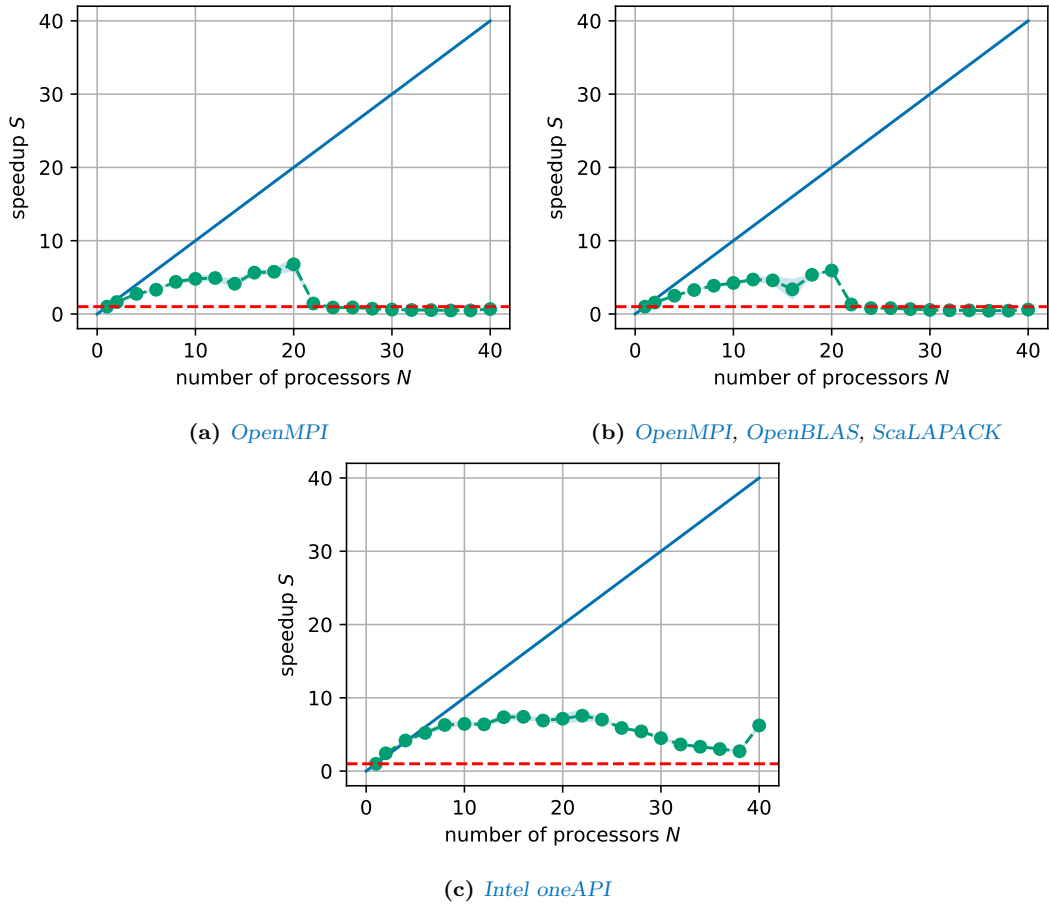


Figure I.5: Scalability for the Si benchmarking system with different combinations of compilers and mathematical libraries, `nk 1` and `nd 1`

Fig. ?? shows that just using another [BLAS/LAPACK](#) library ([OpenBLAS](#) in this case) with the same [MPI](#) version 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.

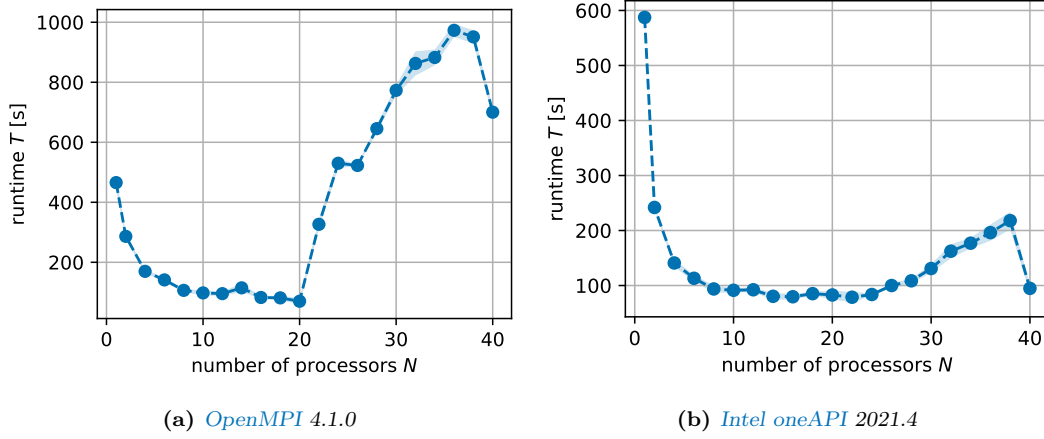


Figure I.6: Comparison of absolute runtimes between *QUANTUM ESPRESSO* compiled with [OpenMPI](#) and [Intel oneAPI](#) for the *Si* benchmarking system, $nk = 1$ and $nd = 1$

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 4.1.0](#) and [Intel oneAPI 2021.4](#) for the *Si* benchmarking system, $nk = 1$ and $nd = 1$

Number of processors	OpenMPI	Intel oneAPI
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 and I.8. For this system, the speedup roughly 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. This is remarkable and also a difference to the silicon benchmarking system, where 1 node is a definite upper bound for

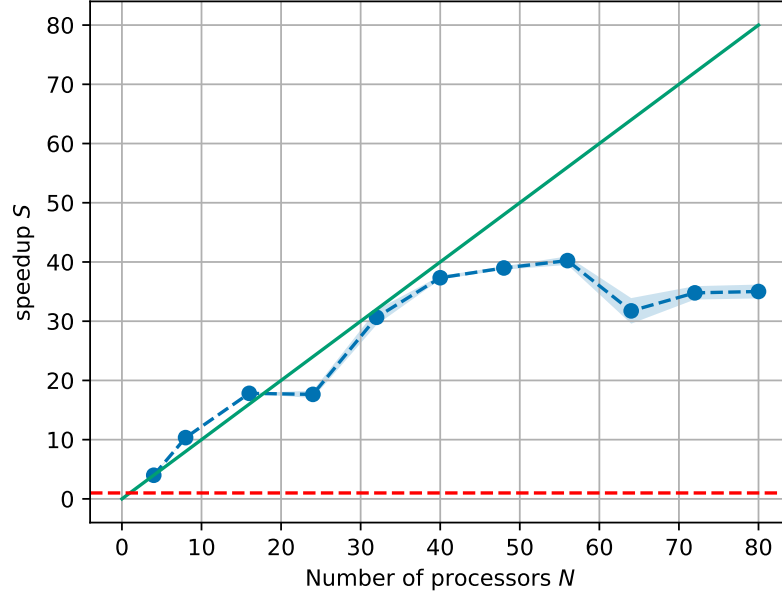


Figure I.7: Scalability for the TaS_2 benchmarking system, QUANTUM ESPRESSO 7.0 compiled with Intel oneAPI 2021.4, nk 1 and nd 1

scalability. An explanation for this behavior can be made with the help of Amdahl's law again. As discussed in sec. ??, the exact form of the speedup is not dependent on absolute times of parallelized and unparallelized parts of a calculations, but rather the proportion between these two (and can thus be characterized by just the purely serial part s). This means that for a more expensive system such as the TaS_2 benchmarking system, when the absolute time for communication, data distribution and collection stays roughly the same and the time for a single Kohn-Sham (KS) iteration (which can be parallelized) is way longer, the proportion of the purely serial part s gets smaller and the scaling behavior changes significantly.

Moreover, the absolute runtime shown in fig. I.8 shows that the calculations not just scale better than with OpenMPI, but they are significantly faster: whereas the minimum for the OpenMPI benchmark is around 1 h50 min for 20 processors, the Intel oneAPI benchmark averages around 40 min for 24 processors. While the benchmarks on the silicon benchmarking system do not seem to favor one set of compilers over the other, the tests on the TaS_2 benchmarking system clearly show the advantages of using Intel oneAPI on the Intel hardware in the PHYSnet cluster.

The observations on how many processors are optimal for certain kinds of systems not only stand for themselves as a statement about scaling on a single node or a small number of nodes, but also provide 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

wait time tas2

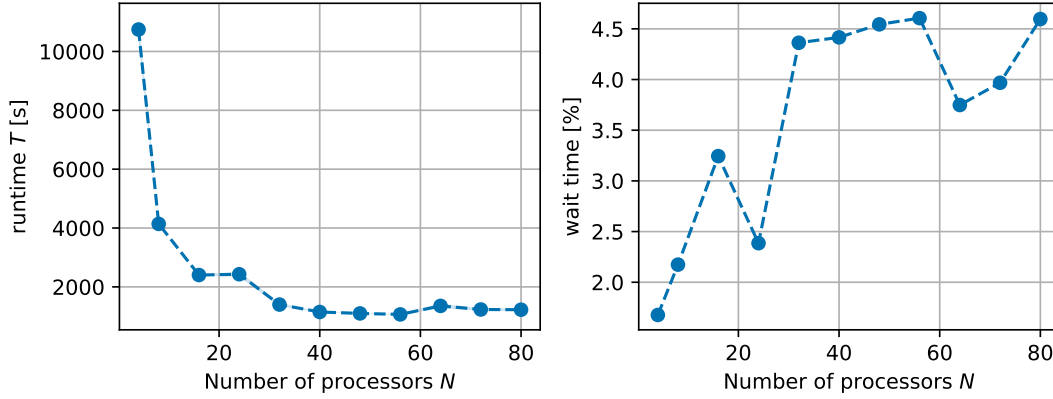


Figure I.8: Absolute runtime and wait time for the scalability test on the TaS₂ benchmarking system, QUANTUM ESPRESSO compiled with Intel oneAPI 2021.4, nk 1 and nd 1

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, an estimate for the quality of k point parallelization can already be made: For the silicon system, the size of pools should not be bigger than 6 processors for optimal scaling and for the TaS₂ 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. While plane wave parallelization is automatically applied, k point and linear algebra parallelization can be controlled and will be tested in this section.

I.3.1 k point parallelization

The benchmark shown 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 and Intel oneAPI.

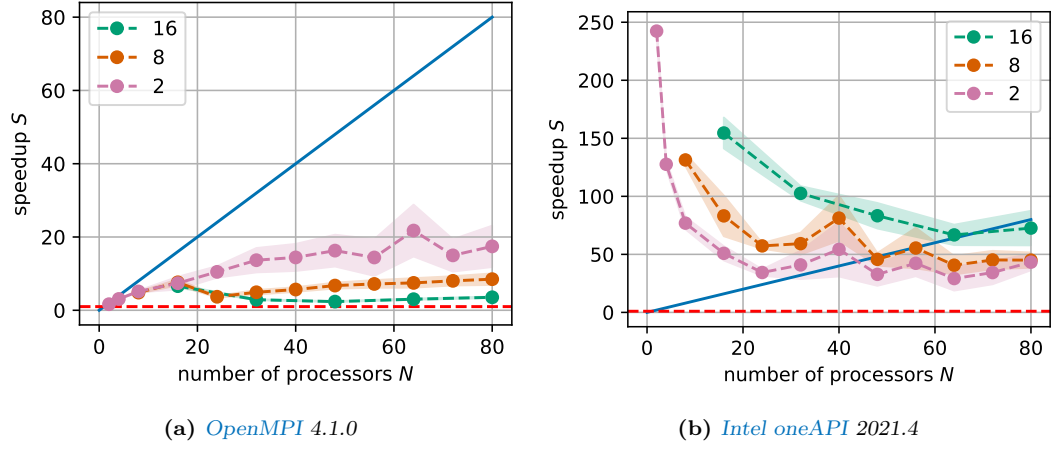


Figure I.9: Scalability utilizing k -point parallelization for the Si benchmarking system with 3 different sizes of processor pools. The size is determined by the parameter nk via size of pools = number of processors / nk

The speedup depicted in fig. I.9 shows that using k parallelization with a pool size of 2 improves the scaling behavior not only on one node, but especially over more than one node. While the speedup without k point parallelization hit a plateau when using more than 6 processors, k point parallelization enables scaling up until 24 processors. The bigger pool sizes do not scale as well, which is in agreement with the results of the benchmarks without k point parallelization presented in the previous section.

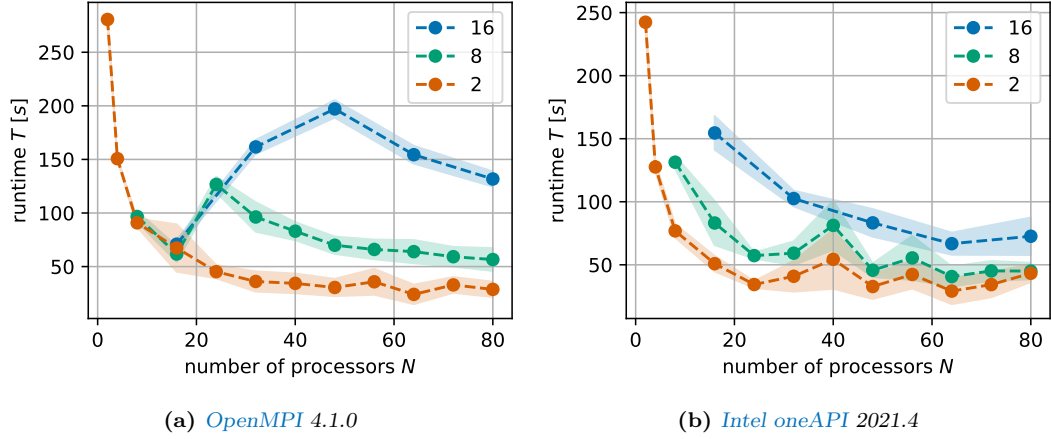


Figure I.10: Absolute runtime for the scalability test with k -point parallelization for the Si benchmarking system with 3 different sizes of processor pools, $nd = 1$

The runtime shown in fig. I.10 show that the choice between using *OpenMPI* and *Intel oneAPI* does not result in a big difference in total execution time, in accordance For both benchmarks, the ideal number of processors seems to be around 24, after that the runtime is subject to

i think there is stuff missing here

significant fluctuations and does not decrease in a predictable way. The average runtime for 24 processors are 45.2s (OpenMPI) and 34.4s (Intel oneAPI) respectively, so about a 10 seconds difference and also around half the minimal time (70.2s and 82.8s at 20 processors) in comparison to the benchmark without k point parallelization.

The same scaling test is applied to the TaS₂ benchmarking system in fig. I.11 and I.12.

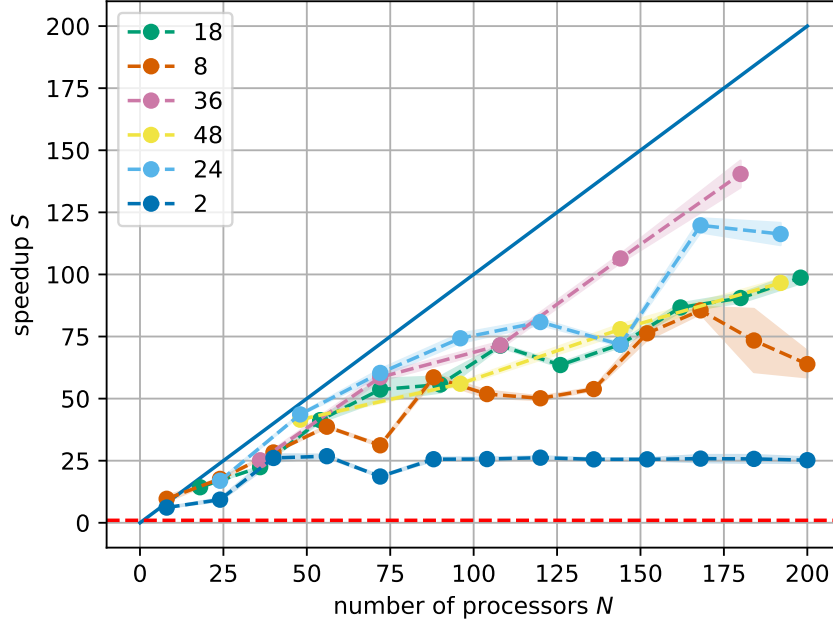


Figure I.11: Scalability utilizing k -point parallelization for the TaS₂ benchmarking system over a range of processor pools, QUANTUM ESPRESSO compiled with Intel oneAPI 2021.4, nd 1

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 TaS₂ system profit more from parallelization and as such scale better for bigger pool sizes up until 36 processors in one pool, which is around the upper limit established in the benchmark without k point parallelization.

Fig. ?? shows a distribution of wait times between about 4% and 6% of the whole wall time, with a slight overall increase when going over 160 processors. This suggests very good parallelization across the whole range of processors.

analyse runtimes
TaS2 nk

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.

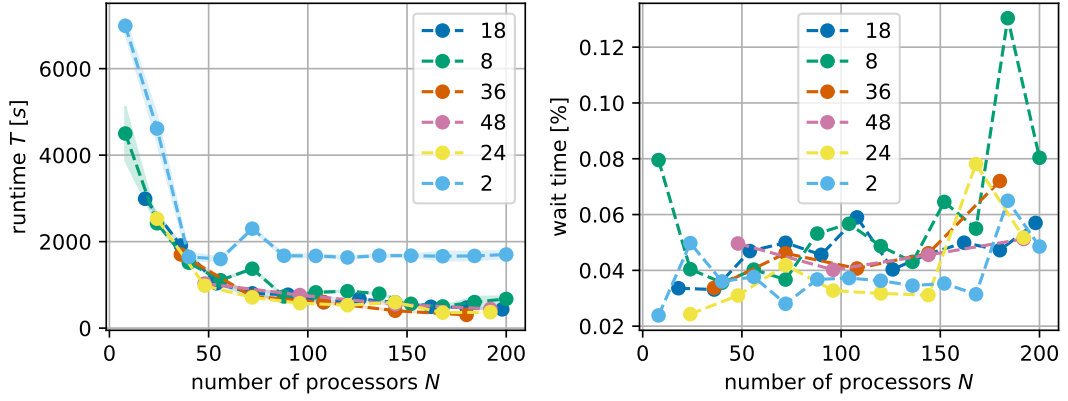


Figure I.12: Absolute runtime and wait time for the scalability test with k -point parallelization for the TaS_2 benchmarking system over a range of processor pools, QUANTUM ESPRESSO compiled with Intel oneAPI 2021.4, nd 1

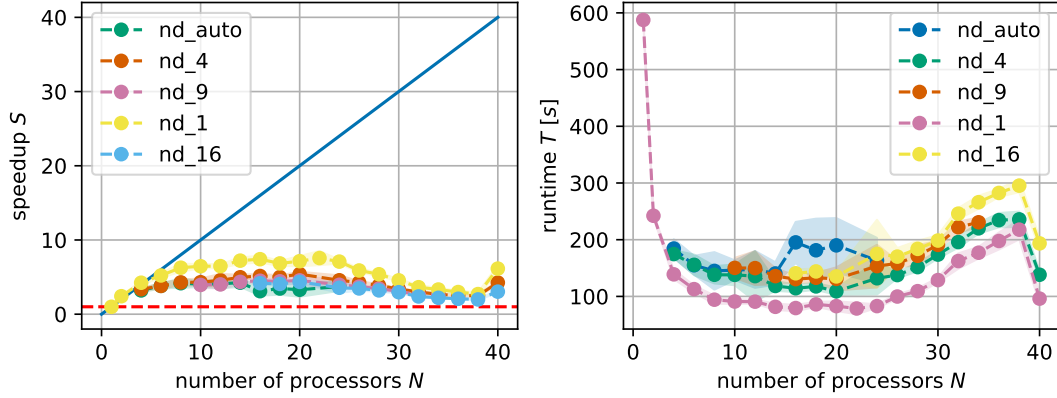


Figure I.13: Scalability and runtime utilizing linear algebra parallelization for the Si benchmarking system, QUANTUM ESPRESSO compiled with Intel oneAPI 2021.4, nk 1

Interestingly, this again is not reproduced for the more expensive TaS_2 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 `KS` states, the TaS_2 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.

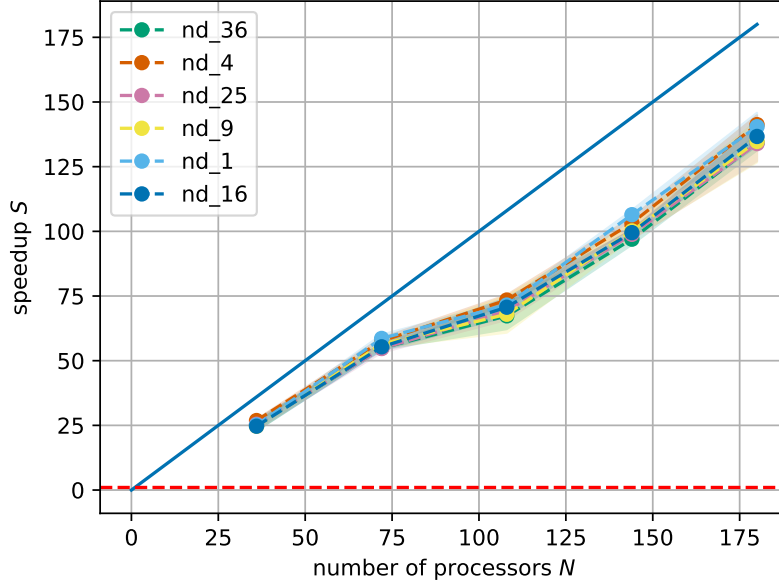


Figure I.14: Scalability and runtime utilizing linear algebra parallelization for the TaS₂ benchmarking system, QUANTUM ESPRESSO compiled with Intel oneAPI 2021.4, `nk` chosen such that pool size = 36

I.4 Comparison with calculations on the HLRN cluster

All calculations so far were exclusively run on the PHYSnet cluster and as such are limited by hardware and configuration present in the cluster. To assess this limitation, the k point benchmarks from sec. I.3.1 were run again on the another cluster, the HLRN cluster in particular. The North German Supercomputing Alliance (Norddeutscher Verbund für Hoch- und Höchstleistungsrechnen - HLRN) operates a distributed supercomputer system at the Georg-August-Universität Göttingen and the Zuse Institute Berlin. The current iteration HLRN-IV has nodes with 2 Intel Cascade Lake Platinum 9242 CPUs (48 cores each) and an Omni-Path (Intels proprietary Infiniband competitor) connection between nodes. QUANTUM ESPRESSO is compiled with Intel Parallel Studio XE Composer Edition 2019 Update 5 (which is the predecessor of Intel oneAPI, bundled without MPI on the cluster) and Intel MPI 2018.5.

Fig. I.15 and I.16 show the benchmarks for the silicon benchmarking system.

The scaling behavior has some striking differences in comparison with the same benchmarks run on the PHYSnet cluster. First of all, speedup and runtimes are very consistent across runs, with only a minimal variance across the whole range of processors. On a single node this is similar to the results from the benchmarks run on the PHYSnet, but on the HLRN cluster this also holds true across two nodes (for more than 96 processors). This is most likely due to the HLRN cluster being equipped with better communication hardware, which is also the reason for the speedup further increasing over two nodes, whereas in the benchmark on the PHYSnet cluster, the maximum speedup of around 20 was already achieved for 24 processors.

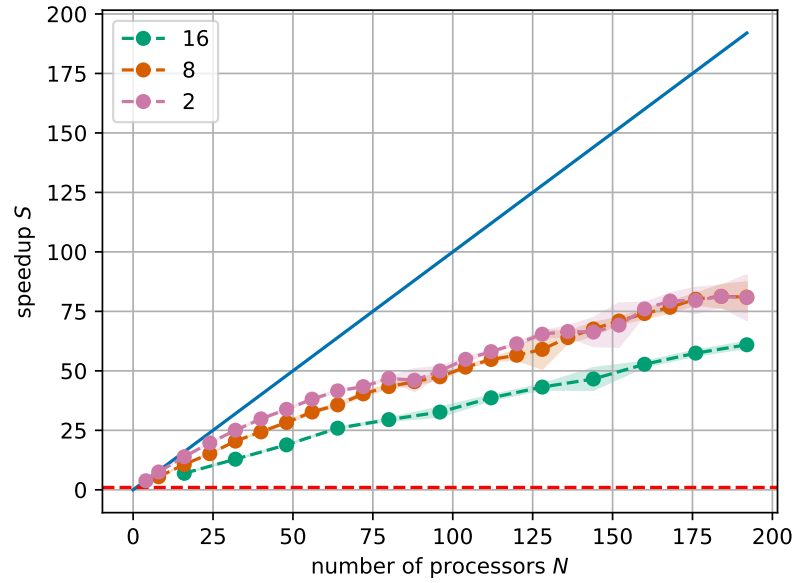


Figure I.15: Scalability utilizing k -point parallelization for the Si benchmarking system with 3 different sizes of processor pools run on the HLRN cluster, QUANTUM ESPRESSO compiled with Intel Parallel Studio 2019u5, Intel MPI 2018.5, nd 1

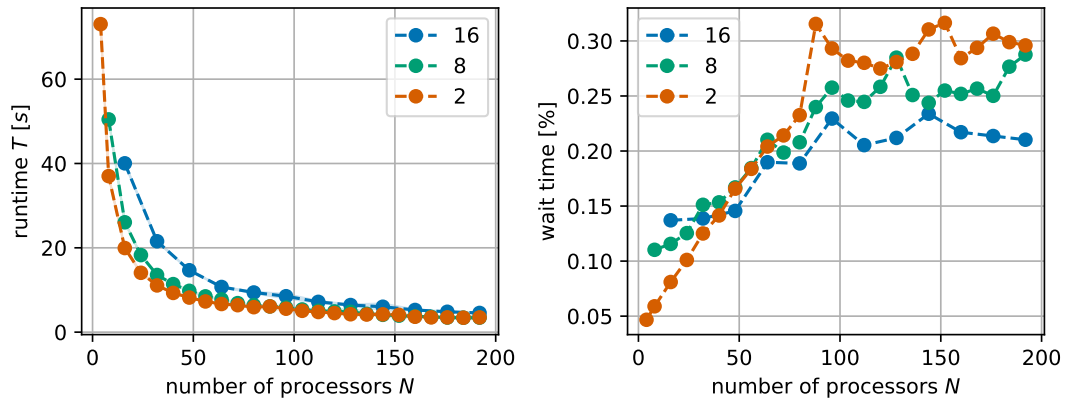


Figure I.16: Absolute runtime and wait time for the scalability test with k -point parallelization for the Si benchmarking system run on the HLRN cluster, QUANTUM ESPRESSO compiled with Intel Parallel Studio 2019u5, Intel MPI 2018.5, nd 1

wait time and run
time HLRN si

A distinction between jobs running across on or two nodes can be made through the wait time though.

Another difference lies in the fact that the pool sizes 2 and 8 show a similar speedup with pool size 16 being a bit worse. This fact shows that while recommendations for parallelization

parameters can be qualitatively made based on system size (more expensive system in terms of iterations, absolute runtime, etc. seem to benefit more from bigger k point pools, while smaller pools work best for smaller systems), the optimal size and processor range can vary depending on the compute cluster the calculations are run on.

Fig. I.17 and I.18 show the benchmarks for the TaS₂ benchmarking system. This benchmark was only run a single time, not averaged over multiple runs.

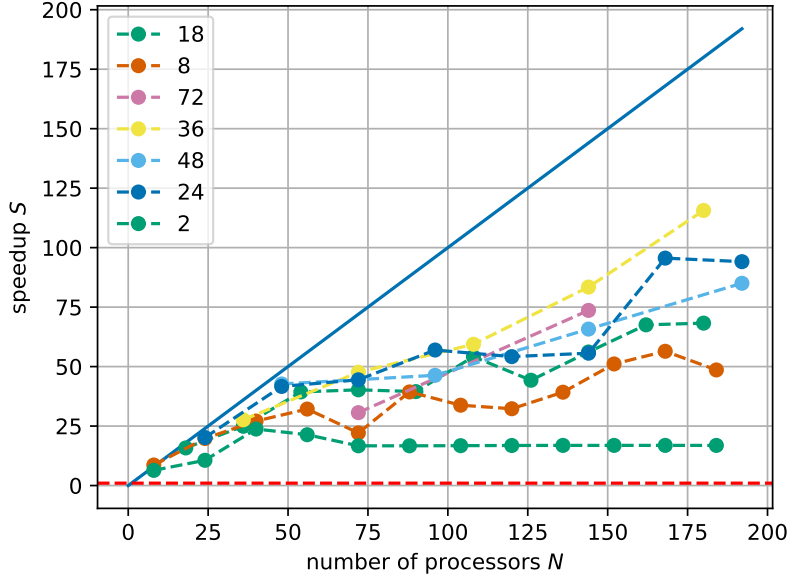


Figure I.17: Scalability utilizing k -point parallelization for the TaS₂ benchmarking system over a range of processor pool sizes run on the HLRN cluster, QUANTUM ESPRESSO compiled with Intel Parallel Studio 2019u5, Intel MPI 2018.5, `nd 1`

Interestingly, the upper bound for pool size is the same as for the calculations on the PHYSnet. In both cases, bigger pool sizes perform better up to 36 processors, with more processors per pool not showing better scaling. This suggests that this limit is not dependent on the compute cluster, but instead a property of the TaS₂ system: Just looking at the PHYSnet cluster, 36 as an upper limit might allude to an interpretation involving the size of nodes, where 36 processors is just under 2 full nodes and everything more will be spread over at least 3 nodes, with possibly more communication slowing down calculations. On the HLRN, the processor counts per CPU and node are different, so an analogous interpretation would suggest that the scaling gets better for pool sizes up to some multiple of 48 (the number of processors on one CPU). This is not the case, the processor pool of size 48 already shows worse scaling than the size 36.

This is an important result, as for the more expensive TaS₂ system the topology of the compute cluster seems to not be important for the scaling behavior. This opens up the possibility of using the PHYSnet not just for calculations but also for first finding the best parameters for a particular system (so just running a few iterations) and then doing the real calculation on a cluster like the HLRN.

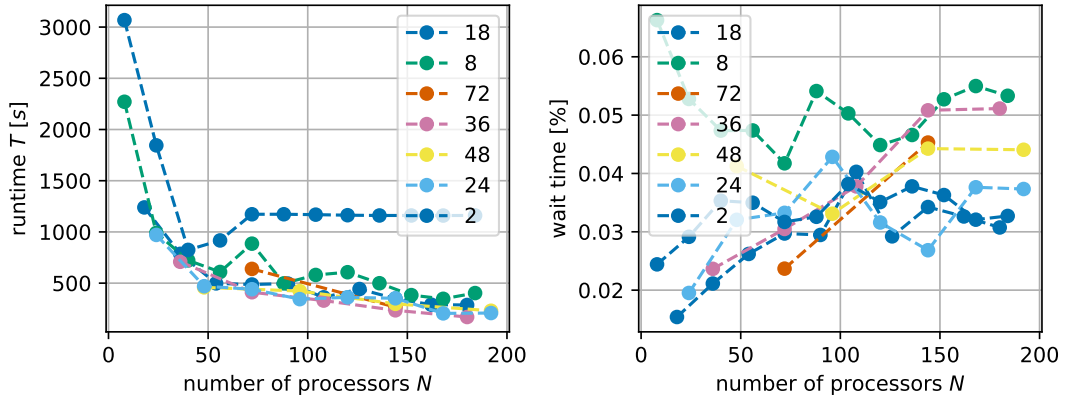


Figure I.18: Absolute runtime and wait time for the scalability test with k -point parallelization for the TaS_2 benchmarking system run on the HLRN cluster, QUANTUM ESPRESSO compiled with Intel Parallel Studio 2019u5, Intel MPI 2018.5, `nd 1`

analyse absolute
times and wait
times HLRN tas2

While the speedup plots look very similar between the PHYSnet and the HLRN, the absolute runtime in fig. I.18 shows a massive difference between the two clusters.

I.5 Conclusion: Parameters for optimal scaling

The benchmarks testing QUANTUM ESPRESSO's `Pwscf` module showed that the speedup gained from parallelization is highly dependent on the specific system. While calculations on the relatively inexpensive silicon benchmarking system gained performance just in a range up to 24 processors and were faster by a factor of 2, the more expensive TaS_2 benchmarking system benefitted more from parallelization itself and as such from all efforts improving parallelization.

Using the compilers and mathematical libraries from the Intel oneAPI package was shown to improve scalability beyond a single node, with the maximum speedup at around 32 processors for the TaS_2 benchmarking system. This value was then confirmed to be the best choice for the size of processor pools in k point parallelization not just on the PHYSnet cluster, but also on the more capable HLRN cluster.

A generalization for other systems beyond a qualitative assessment that more expensive system profit more from parallelization than smaller system is not possible, but this chapter gives a guide on how to examine a system in order to get the optimal parameters.