I. SCALING INFORMATION

A. Performance/Scaling of the B-Spline Atomic R-Matrix Code

Figure 1 provides some scaling information for major parts of the BSR package. While BSR_BREIT and BSR_MAT scale well until about 800 cores in the example shown, the performance of BSR_HD deteriorates rapidly when the number of cores exceeds about 400-500. This lack of scaling efficiency in the BSR_HD step is due to; 1) the SCALAPACK [1] diagonalization. and 2) the ALL-TO-ALL operation of required to collect the eigenvectors from the distributed matrix. By simply collecting only the surface amplitudes, the scaling would be improved significantly. This will be implemented in the codes during the upcoming year.

Specifically, runs were performed on the Knight's Landing (KNL) and Skylake (SKX) nodes on Stampede 2 with 256, 512, 768, and 1024 cores. We used 64 cores per node on the KNL nodes and 48 cores per node on the SKX nodes. On SuperMIC, 20 cores per node were used. If nodes rather than cores are a measure of performance then the KNL system on Stampede-2 and SuperMIC are comparable. The figure also shows that the SKX cores outperform the KNL cores by about a factor of three. This is disappointing but in line with what others have seen on the two systems. Unfortunately, we found that using more than one thread on the KNL cores slows the program down significantly. We currently do not have a clear explanation for this, but we intend to look into it in the next allocations period. Finally, while we realize that SuperMIC is no longer available to XSEDE, we trust that this information is still useful. It basically shows that we can expect to move rather easily to other machines, e.g., Comet and Bridges.

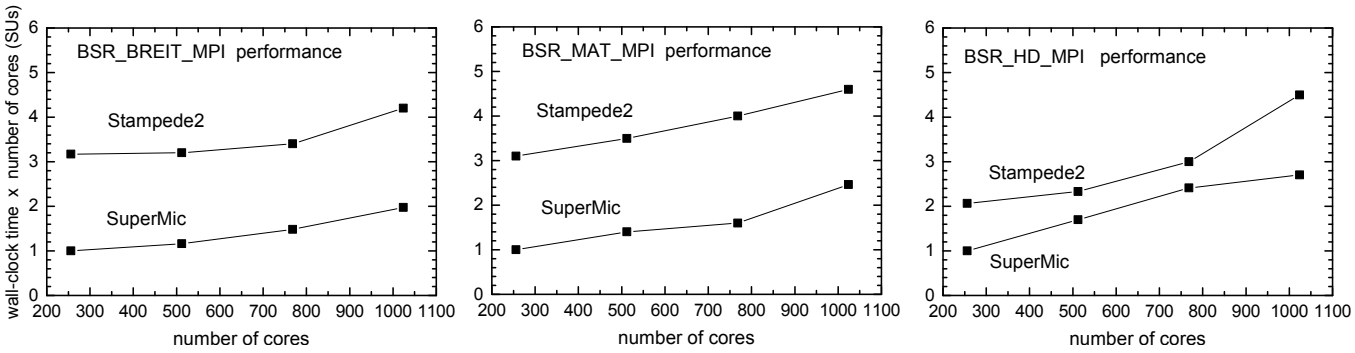

FIG. 1. The product of wall-time and the number of cores (normalized to the smallest number as a function of the latter) for `BSR_BREIT`, `BSR_MAT`, and `BSR_HD` on Stampede 2 and SuperMIC. The rank of the matrix was approximately 50,000.

We emphasize that in practice we do *not* waste SUs to the extent illustrated above, where doubling the number of cores from 512 to 1,024 increased the product of wall-time and the number of cores by a factor of $\approx 2 - 2.5$ on Stampede 2, i.e., it resulted in an increased wall-time as well. In fact, we always try to optimize the number of cores by using the following criteria:

- The matrix has to fit into the available memory, which sets the minimum number of cores.
- Increase the number of cores only to the extent that scaling does not deteriorate significantly while also ensuring that the maximum allowed wall-time is not exceeded.

This strategy ultimately sets the maximum size of the matrices that we can handle, and hence the size of the physical problems that we can tackle. To date, we have been able to deal with matrices of rank up to 400,000. Running `BSR_HD` represents the most expensive part of our work, and it has to be done for every partial-wave symmetry.

B. Performance/Scaling of the CCC Code

The CCC code has been parallelized using OpenMP on a node plus MPI between nodes. Recently, the code has been ported to GPU's using OpenACC. The manner in which the calculation scales is shown Figure 2 for the two major computational steps in the code, formation of the CCC matrix and the solution of the large set of linear equations after the matrix has been formed. The first step, matrix formation, an essentially parallelizable operation, shows strong scaling, while the solution of the set of linear equations via ScaLAPACK degrades as the number of nodes increases. In addition, the GPU version of the code shows a performance enhancement of about 10 over the CPU version.

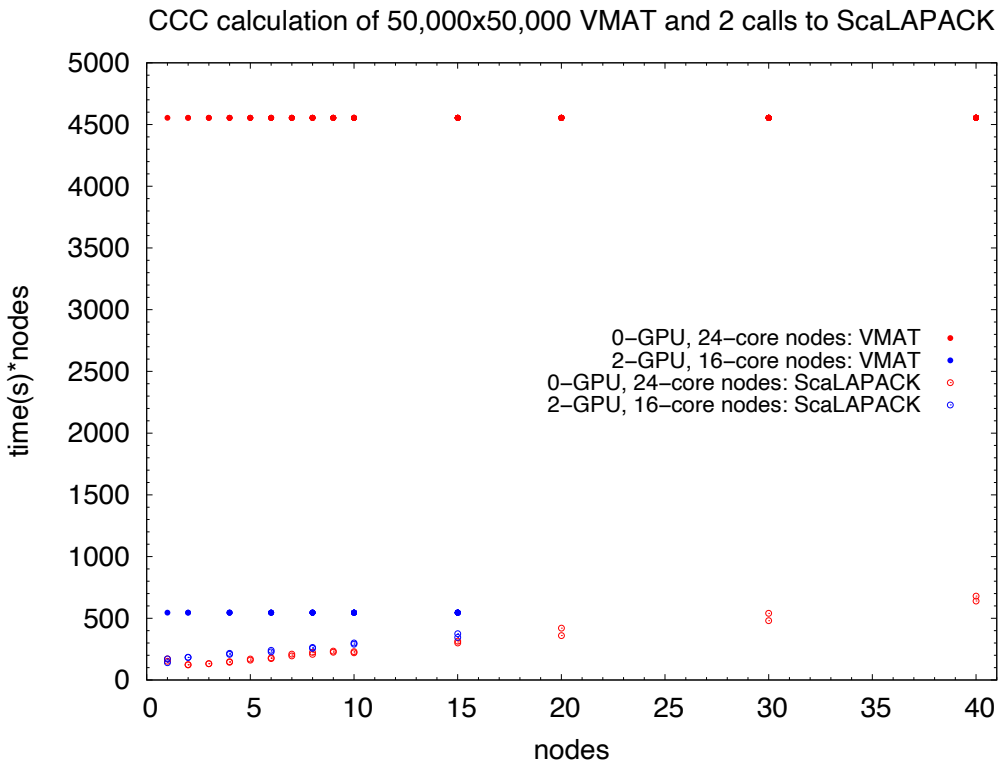

FIG. 2. The product of walltime and the number of cores (nodes) as a function of the latter for a CPU/GPU-based calculation on a 50,000 by 50,000 CCC matrix.

C. Performance/Scaling of the tRecX Code

Scaling is strongly application-dependent, mostly driven by the locality properties of the system's representation. Performance in tRecX is achieved by a highly structured representation of the operators. Most of the time is spent in time-propagation with the repeated application of block-sparse operators, where each block in itself is structured, e.g., as tensor products or singular-value representations. The linear dimension of the blocks varies from 10 to 400. The blocks determine the code's granularity and are distributed based on self-measured CPU load. The distribution algorithm also attempts a trade-off between load-balance and communication. On the block level LAPACK and EIGEN routines are used. Expansion of the full operator into a matrix in order to use parallel linear-algebra software, while possibly improving formal scaling, leads to the loss of tensor structure and dramatically increases the flop count.

Scaling is shown in Fig. 3 for calculations as in Ref. [2], where most of the load is used in the electron-electron interaction. The scaling data are for actual production-type runs including processing and output of intermediate results after each timestep, which involves non-local operations. It does not include setup times, as setup data can be reused among the runs of a parameter study. Results are for *strong scaling*, i.e., a problem of fixed size is distributed over an increasing number of nodes with a corresponding increase of the relative role of communication.

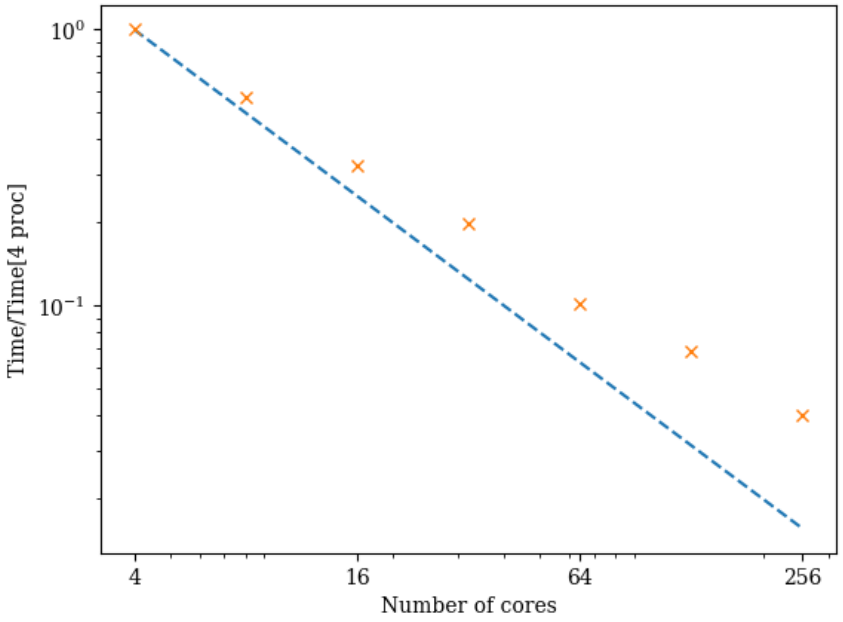

FIG. 3. Scaling of tRecX for double-ionization of Helium at laser wave length 400 nm. The results were obtained on the Munich Cool-MUC cluster. The specific section of Cool-MUC has nodes composed of two sockets with 16 cores each of an Intel(R) Xeon(R) Gold 6130 CPU @ 2.10GHz with 192GB RAM DDR4@2666MHz. The dashed line is ideal scaling.

D. Performance/Scaling of the XChem Code

The XChem code employs the OpenMolcas package [3] to compute the Gaussian integrals needed as input to XChem. The performance of OpenMolcas is highly dependent on the disk media used for storage. By employing the SLURM job manager, one obtains excellent parallelization between nodes. Since there are several parts of the calculation where the jobs can be sent independently, we employ the SLURM manager to initiate several jobs simultaneously. Prior to the use of XChem, it is necessary to perform a quantum chemical calculation using OpenMolcas [3]) to obtain the molecular orbitals needed for the target wavefunctions. These are then augmented with a large monocentric basis of Gaussian plus B-spline functions that are needed to describe the continuum.

In this way, the XChem code separates into modules with different performance:

- basis: Prepares all 1 electron integrals. This step is not parallelized.
- orbitals: Creates a set of linearly independent orbitals, where the target and monocentric orbitals are orthonormalized. This step is parallelized using OpenMP.
- integrals: This module is one of the most computationally demanding, but unfortunately it is barely parallelized at the OpenMP level. However, integral calculations can be easily parallelized by sending them to different nodes using SLURM. For a standard calculation of 22 monocentric Gaussian exponents, the module can be parallelized using up to 253 nodes.
- rdm: Calculates the Reduced Density Matrices of the different electronic states and can be efficiently parallelized using the SLURM manager. Moreover, this calculation is independent of the actual integrals and can be done in parallel with the integral evaluation.
- qci: Uses the RDM and the integrals to evaluate all matrix elements. It is parallelized using OpenMP. The use of different modules for RDM and the integrals allows the user to independently compute and use both variables with a minimum of recomputation.
- bsplines: Creates the B-spline basis. It is parallelized using OpenMP and the MKL library.

- ccm: Creates the Close-Coupling Matrix for the different channels and is parallelized using the SLURM job manager.
- boxstates: This is parallelized using OpenMP and the MKL library or using SCALAPACK [1].
- scattering states and dta: Evaluates the scattering and the photoelectron spectrum.

Depending on the system, the most demanding modules can be the integrals (i.e., when large angular momenta are required), rdm (if a big active space is employed) and ccm/boxstates (depending of the number of channels in the continuum). All these codes can be used in a highly parallelized manner by using the SLURM manager. As an example, Fig. 4 shows the performance of the code when several nodes (up to 32) are employed. In this case, the calculation was done for pyrazine, a medium-size molecule containing 10 atoms, maximum angular momentum of 4, a CASSCF calculation of 10 electrons in 8 orbitals with 7 channels. Due to the small number of channels and the modest active space, the time consuming step is integral evaluation. As seen in Fig. 4, the performance is quite good.

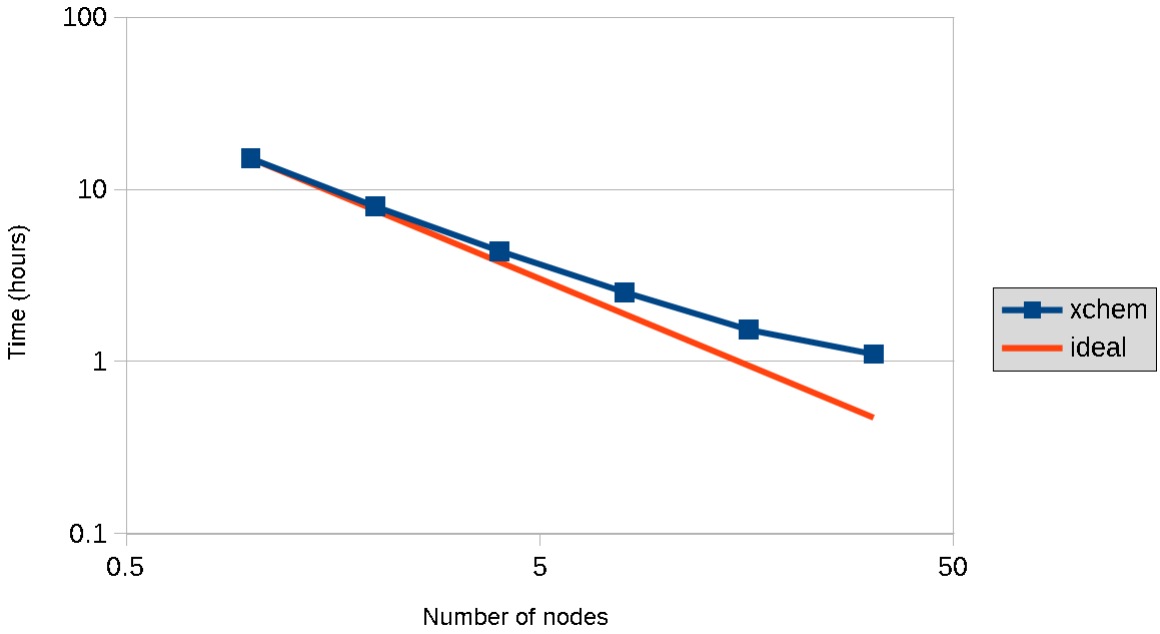

FIG. 4. Performance of the XChem code in pyrazine with different number of nodes

E. Performance/Scaling of the UKRMol Code

The UKRmol+ code is parallelized using a combination of OpenMP and MPI. Table I shows the scaling of the UKRmol+ code on ARCHER for the diagonalization of the Hamiltonian matrix in an electron-uracil calculation. The "Expected" scaling is given approximately by (Configurations)3/(tasks).

MPI tasks	Configurations	Time	Expected
32	19668	10m 48s	-
121	39725	19m 41s	23m 32s
2025	135835	37m 39s	56m 13s

TABLE I. UKRmol+ Hamiltonian diagonalization for electron-uracil calculation for the number of configurations indicated run on ARCHER. The 'Expected' scaling is given approximately by $(\text{Configurations})^3/(\text{tasks})$

Figure 5 shows the scaling of the construction of the Hamiltonian matrix by MPI-SCATCI for phosphoric acid. The linear scaling begins to deteriorate at about 32 cores.

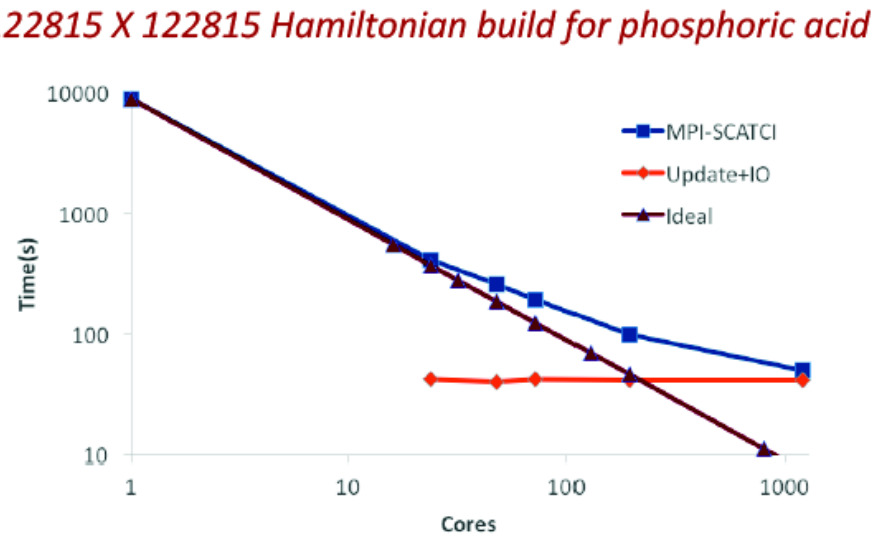

FIG. 5. UKRmol+ Hamiltonian construction for electron-scattering from phosphoric acid using the program MPI-SCATCI. The update+IO time is the time taken for MPI synchronization steps that include disk writes. The time taken by MPI-SCATCI includes the update+IO time.

F. Performance/Scaling of the RMT Code

RMT uses the R-matrix paradigm, partitioning the interaction region into an "inner' and an "outer" region with different numerical schemes utilized in each region. This approach facilitates efficient parallelization without sacrificing accuracy. Fig. 6 provide some information about the strong scaling of an entire calculation, while Fig. 7 shows the scaling for inner and outer regions, side-by-side.

Strong scaling cannot be trivially inferred from one calculation, as at some point the bottleneck switches between the two regions (inner/outer). It is relatively easy to demonstrate scaling if one region dominates, but the more difficult problem in setting up efficient calculations is finding the right balance between the two regions. Both regions need to be optimized in order to see full scaling, and the optimal arrangement is likely to be different for the two regions, and again different in each specific calculation.

Limited RMT scaling data are available on XSEDE machines. Most of the data below come from the ARCHER (Cray XC-30). Moreover, the most recent capabilities and performance enhancements in RMT have not yet been tested for scaling on any machine. These are both reasons for inclusion in the current proposal.

Table II displays the weak scaling behaviour of the finite-difference based RMT outer region. The number of cores used in the outer region determines the total size of the configuration space. Increasing the number of outer-region cores has a negligible effect on the average iteration time (i.e., the time needed to propagate the wave function one step forward in time). Thus efficient scaling of the outer region is possible to very large distances without increasing overheads.

Cores	620	750	1000	1500	2000	2500
Iteration Time (s)	1.135	1.08	1.134	1.135	1.135	1.148

TABLE II. Weak scaling of outer region: Argon, two target states, inner region fixed (144 cores) on ARCHER (CRAY XC30).

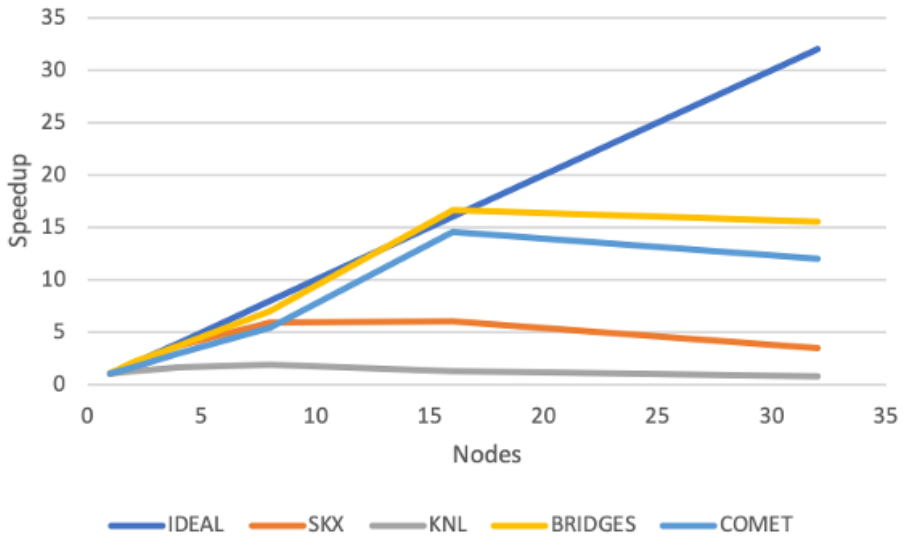

FIG. 6. Strong scaling of RMT in an entire calculation: Photoionization of Argon with circularly polarized light (XSEDE machines). Preliminary scaling data have been obtained for RMT on several XSEDE machines (Comet, Bridges and Stampede2). In this calculation the ratio between the number of inner region cores and outer region cores was kept fixed. Calculations were performed with the same level of optimization on all machines, although the number of cores per node varies from 48 cores/node on Stampede2 (SKX and KNL) to 28 and 24 cores/node on Bridges and Comet, respectively.

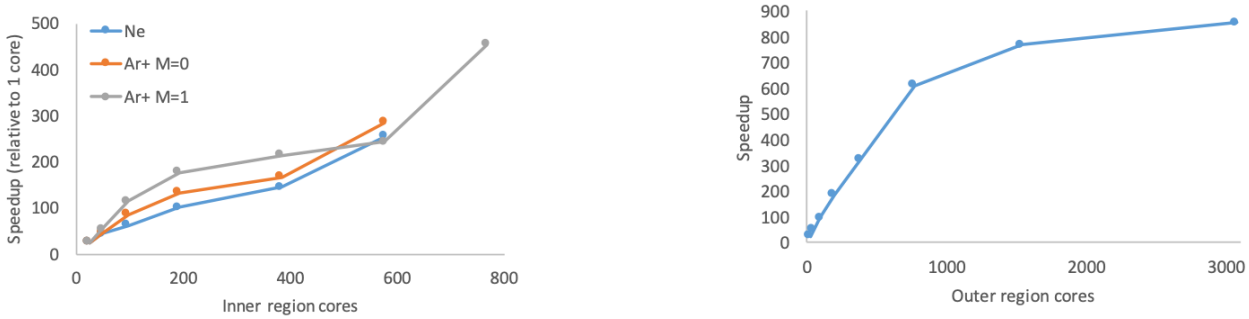

FIG. 7. Left. Strong scaling of RMT in the inner region: Various target atoms, outer region fixed (1200 cores) on ARCHER (CRAY XC30). The more complete the description of atomic structure, the larger the matrix-vector multiplication in the inner region. Calculations using non-linear laser polarisation will be even more intensive than the largest (Ar+ M=1) calculation shown here, and hence we expect that higher core counts will be possible.
Right. Strong scaling of RMT in the outer region: Argon, two target states, inner region fixed (144 cores) on ARCHER (CRAY XC30). For calculations comprising many electron emission channels, the outer region becomes the bottleneck. Performance is enhanced by reducing the size of the radial grid on each core. The levelling off above 1,000 cores shows that the inner region is becoming dominant. Additional performance could then be gained by adding cores to the inner region.

References

[1] http://www.netlib.org/scalapack/
[2] J. Zhu and A. Scrinzi, arxiv.org/abs/1912.09250.
[3] Galván *et al.*, *J. Chem. Theory Comput.* 15, 5925 (2019).