Technical Report for eCSE03-5 “Unleashing the Potential for Superior Parallel Performance of the ONETEP Linear-Scaling Density Functional Theory Package on ARCHER2”

Background

The ONETEP Linear-Scaling Density Functional Theory (LSDFT) package [1,2] is a fully-featured ab initio electronic structure package suited to large-scale atomistic simulations of systems such as nanomaterials, crystalline interfaces, and biological materials. It has been developed over the last 15 years by a group of academics, the ONETEP Developers Group (ODG), based at leading UK universities including Warwick, Imperial, Cambridge and Southampton. It has a strong user community both in academia and industry, being made available both via a free academic license to UK academics, a low-cost license to other academics, and via DS Biovia’s Materials Studio package. The parallel scaling in the early days of ARCHER2, obtained by initial measurements utilising hybrid MPI/OpenMP parallelism, demonstrated that while the code exhibited what constituted respectable scaling on ARCHER, it was vital parallel algorithms were redeveloped to run efficiently on ARCHER2, and to take advantage of the large capacity of individual nodes with 128 cores.

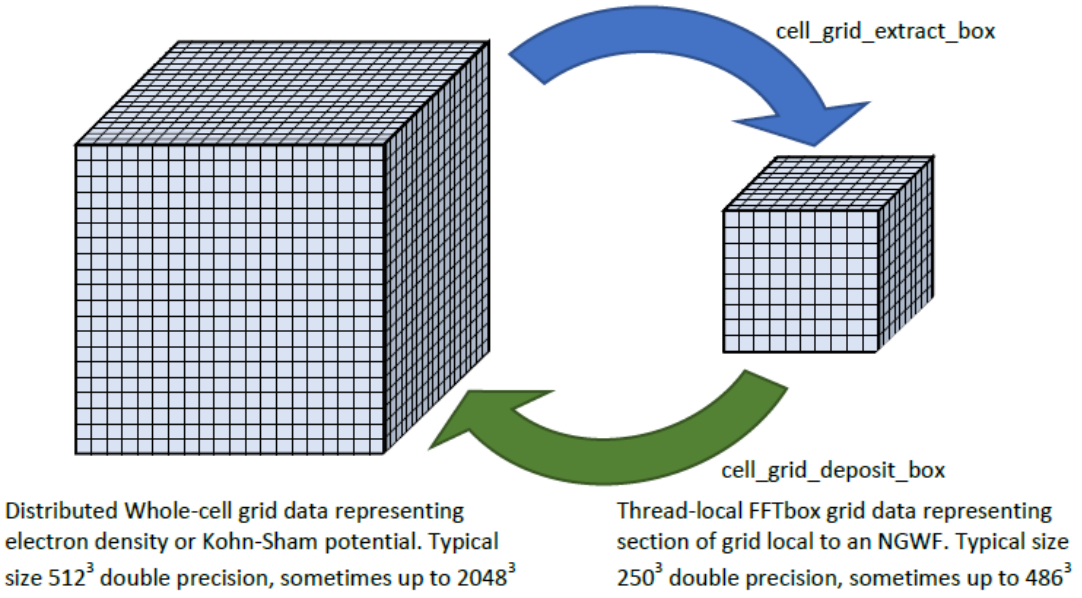

Figure 1 - Schematic of the density extraction and deposition routines which are used in converting whole-cell data to thread-local FFTbox grid data.

This proposal aimed to provide a dramatic upgrade to the parallel scaling of the ONETEP Theory code, when run on state-of-the-art parallel computing environments such as ARCHER2. Large nodes make novel demands of the parallel algorithms, exposing algorithmic issues that were not previously been problematic. Work done in the past, particularly around the time of first use on the ARCHER supercomputer [3,4], resulted in parallel scaling ONETEP that is, at first, reasonably good with total core count: a nearly-perfect scaling regime exists for a large (1000+ atom) job on up to around 1000 cores. However, beyond that, parallel efficiency began to drop rapidly, and we aimed to address this via work packages on

whole-cell grid (see Figure 1) and sparse matrix routines respectively. A further work package involved code sustainability and benchmarking activities including documenting best practice when running at very large scale with hybrid MPI/OpenMP parallelism.

As this technical report describes, while all the objectives relating to code implementation and dissemination were achieved, the parallel performance gains of the implemented approaches were not as high as might have been hoped, because of technical limitations of the One-Sided communications routines within the available MPI libraries on ARCHER2 and other available systems.

Development work was undertaken within a fork of the ONETEP code by Chris Brady and Heather Ratcliff. Three sections of ONETEP have had their communication approaches supplemented by MPI RDMA calls.

Density Deposition and Extraction

In this part of ONETEP quantities are defined on hexahedral mesh subdomains of the hexahedral mesh global domain. They are then either deposited (value is accumulated from the subdomains onto the global domain) or extracted (value on the global domain is copied onto the subdomains). The global domain is decomposed over all of the MPI ranks that ONETEP is running on but each subdomain is held entirely on a single rank. The subdomains that each rank holds are not related to the section of the global domain that the rank holds. This is show diagrammatically in Figure 2.

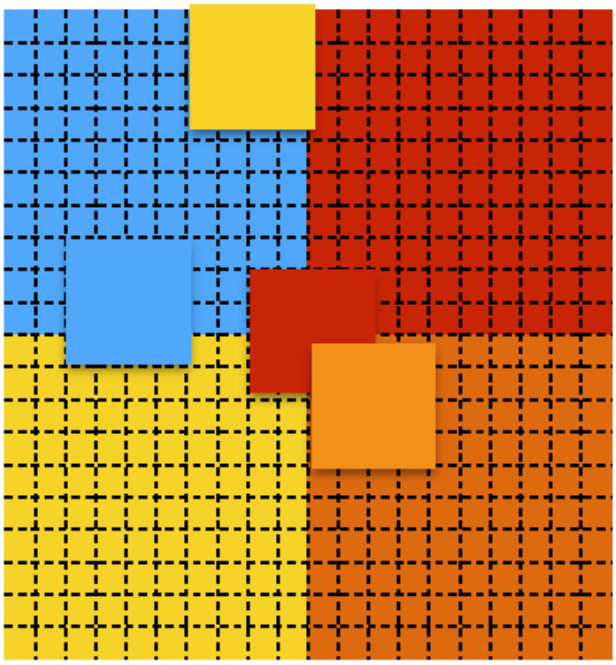

Figure 2 - 2D analogue of the global domains and subdomains in ONETEP. Colours indicate the rank that owns a part of the global domain or a given subdomain

In general remote accumulation to many destinations is needed for deposition and remote acquisition of data from multiple sources is needed for extraction. In deposition there is the further complication that each subdomain is calculated before deposition in a separate thread and the MPI communication is then run inside an OpenMP critical section In the existing approach to deposition in ONETEP a subdomain is prepared on each rank and then MPI_Alltoall is called so that the overlap of each selected subdomain with each other rank is known. Each rank then exchanges the relevant overlapping information with the other ranks using sends and receives, except in the case of subdomains that are located on the local part of the domain. The final accumulation of the result is by local addition. Extraction is the same basic algorithm but the final data is simply stored to the subdomain. Note that in this algorithm all of the information needed to calculate the overlap of the subdomain and each other rank's part of the domain is held on the processor that holds the subdomain. Since ONETEP has to put all MPI calls into OpenMP critical sections the requirement for synchronisation between

all MPI ranks causes thread level synchronisation across the entire job which limits scaling when using many threads and many MPI ranks. Using MPI RDMA calls eliminates this effective synchronisation between threads since each MPI rank is now able to act entirely independently of the others until all subdomains have been either deposited from or extracted to. The simplest approach to implementing MPI RMDA for both deposition and extraction creates an MPI window over the entire domain and then using MPI_Accumulate (for deposition) or MPI_Get (for extraction) to actually transfer the data. In the case of deposition there is a requirement that the source buffer must be available for reuse after the call to MPI_Accumulate. This means that either the access EPOCH must be started and ended immediately before and after the call to MPI_Accumulate or it must be valid to call MPI_Win_flush_local on the window after the call to MPI_Accumulate.

Both of these imply very strongly that passive target synchronisation must be used. While there are fewer restrictions on extraction the current implementation shares code between extraction and deposition and retaining this approach was simpler. Two approaches to implementing MPI RDMA were tested. In the first approach MPI_Win_lock_all was called immediately after the window was created and MPI_Win_unlock_all was called just before the window was freed. Immediately after the call to MPI_Accumulate, MPI_Win_flush_local was called to allow reuse of the source buffer. In the second approach MPI_Win_lock and MPI_Win_unlock are called immediately before and after the call to MPI_Accumulate or MPI_Get specifically for the target rank. After testing the first approach was rejected since it caused a substantial increase in the memory required which seemed to be due to the MPI layer choosing to deal with the call to MPI_Win_flush_local by copying the source buffer to MPI_Accumulate to a temporary buffer.

The second approach works as expected and produced initial evidence of an increase in scaling efficiency for small systems – see Figure 3. However, absolute times for the RDMA approach were slower than for the conventional approach. It can be hoped this may improve with future MPI implementations and there is still scope for improvement within the current approach.

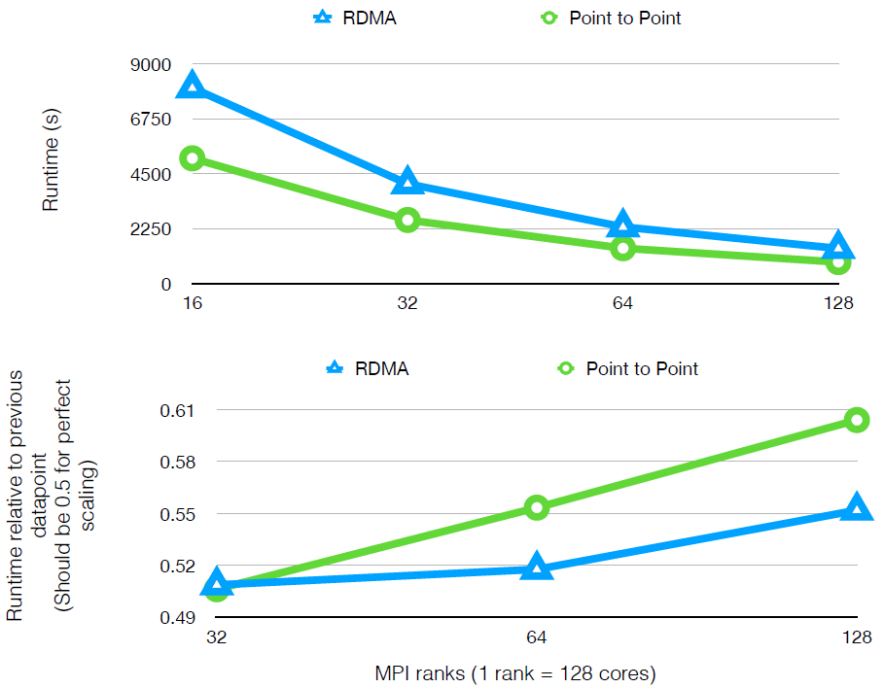

Figure 3 - Scaling of tests of the runtime associated with density extract/deposit operations

Sharing basis functions

The second part of ONETEP that was modified to use MPI RDMA was the section that deals with sharing basis functions across the processors. This section of ONETEP uses an asynchronous request response system overlapping communication with compute. The processor that wants a given basis function sends a request for that basis function asynchronously. The processor that has that basis function receives the request asynchronously and then asynchronously sends the response to the requesting processor which in turn asynchronously receives the returned basis function. The overlap of compute and communication is already written so this maps naturally onto MPI RDMA. Internal changes to the layout of memory were made so that all of the basis functions on a given MPI rank were stored contiguously but otherwise the changes were simple. An MPI window is created when the calculation involving the basis functions starts and also locked over all ranks at the same time and information about how the various basis functions are laid out across ranks is exchanged. The phases of data exchange in the existing ONETEP code are then followed but simplified. Where in the original code a request is made to a remote processor for a given basis function this is replaced with a call to MPI_Get, the intermediate step of responding to remote requests disappears since this is now handled by the MPI RDMA implementation and the step of responding to the asynchronous receive of the response becomes an MPI_Win_flush_local. Performance tests on this code as shown in Figure 4 suggests that it performs similarly but slightly better than the existing two sided implementation, but there is scope for more performance improvements as the MPI RDMA system matures.

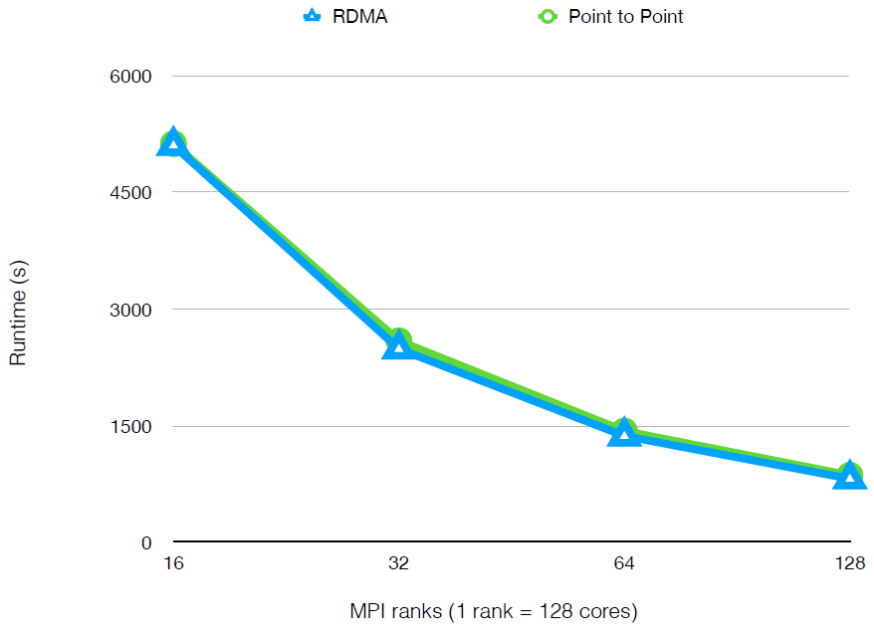

Figure 4 - Scaling of runtime associated with basis function comms operations on 16 to 128 MPI ranks. Performance of both approaches is very similar.

Final tests on total runtime

For acceptance testing of the changes made during this project, a set of realistic timings tests were performed on full-scale runs on a very large model of an Amyloid fibril, comprising 13696 atoms. These are shown in Figure 5 and demonstrate that the total time

in the revamped code achieved is comparable to the original code, without demonstrating significant speedup when applied at large scale. Scaling benefits of the new algorithms do not seem to transfer over to large-scale execution of the new code.

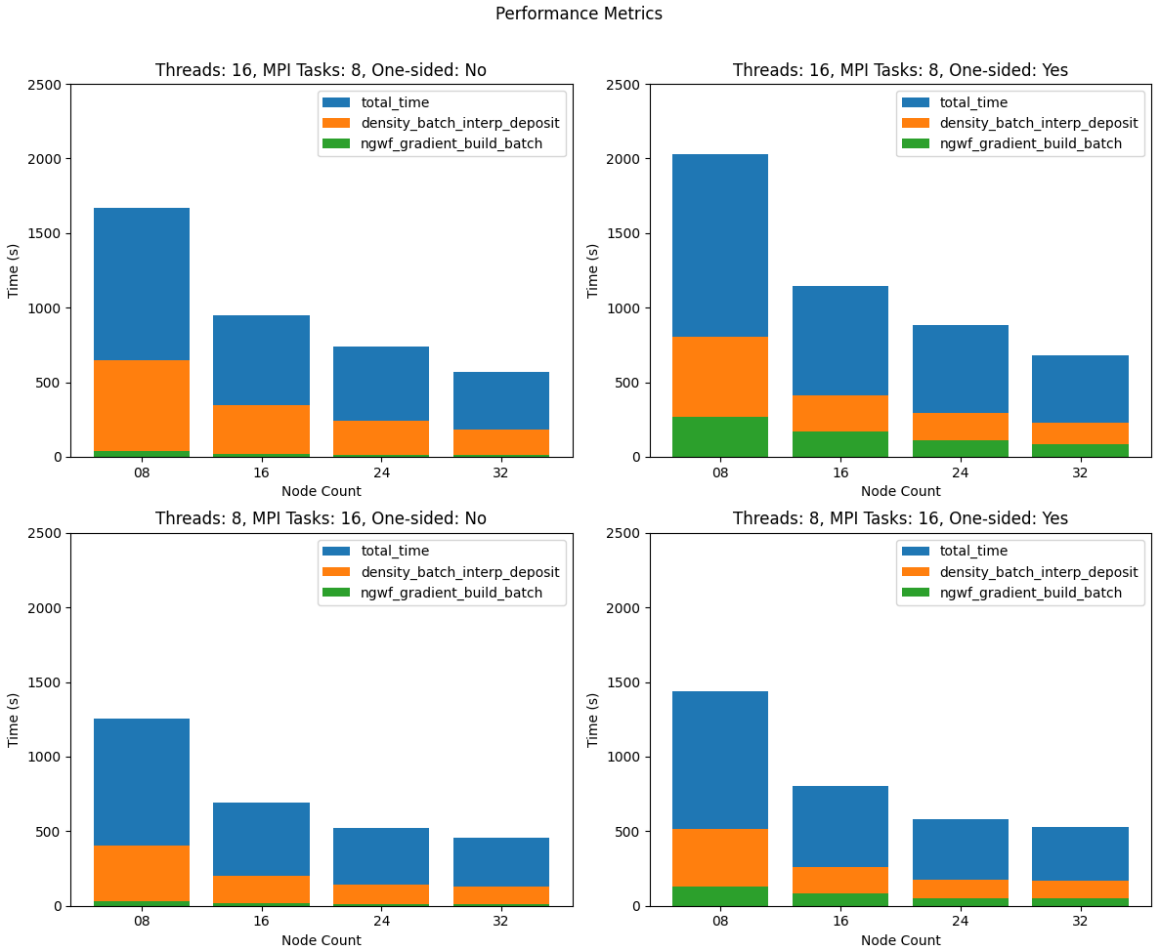

Figure 5 - Final runtime before and after implementation of RDMA operations under two combinations of OpenMP and MPI parallelism (16 threads x 8 MPI tasks and 8 threads x 16 MPI tasks), on 8-32 nodes of ARCHER2.

Conclusions

The work in this project successfully implemented “one-sided” RDMA operations with the goal of accelerating the scaling with MPI process count of some of the more challenging MPI communications routines in the ONETEP code. All the technical goals relating to implementation of the approach were achieved, though the performance of the resulting routines was no better than the original code. All of the objectives relating to making the code more useful to users through improved documentation, and ensuring automated recurrent testing were achieved, and the project fed well into further developments as part of the Software for Research Communities EPSRC project that took parallel scaling development in a different direction including developments aimed at porting to GPUs. This subsequent work has made great progress on optimising the same basic operations and

improving overall speed to the degree envisaged this eCSE project, but by a different approach (see, for example fig 21 at ref [5]).

This work was funded under the embedded CSE programme of the ARCHER2 UK National Supercomputing Service (http://www.archer2.ac.uk).

Bibliography

[1] Introducing ONETEP: Linear-scaling density functional simulations on parallel computers, C.-K. Skylaris, P. D. Haynes, A. A. Mostofi and M. C. Payne, J. Chem. Phys. 122, 084119 (2005).
[2] The ONETEP linear-scaling density functional theory program, J. C. A. Prentice, …, N. D. M. Hine, …, et al (36 authors), J. Chem. Phys. 152, 174111 (2020)
[3] Linear-scaling density-functional theory with tens of thousands of atoms: Expanding the scope and scale of calculations with ONETEP, N. D. M. Hine, P. D. Haynes, A. A. Mostofi, C.-K. Skylaris and M. C. Payne, Comput. Phys. Commun. 180, 1041 (2009)
[4] Hybrid MPI-OpenMP parallelism in the ONETEP linear-scaling electronic structure code: Application to the delamination of cellulose nano-fibrils, K. A. Wilkinson, N. D. M. Hine, and C.-K. Skylaris, J. Chem. Theory Comput. 10, 4782(2014).
[5] https://docs.onetep.org/developer_area.html#fast-density-calculation-for-developers

Authors: Dr Nicholas D M Hine, Dr Christopher S Brady, Dr Heather Ratcliffe (University of Warwick)