eCSE03-15

Future-proof parallelism for plane-wave DFT in CASTEP

Ben Durham[1], Ed Higgins[2], Matt Smith[1], Phil Hasnip[1]

August 2023

[1] School of PET, University of York, Heslington, York YO10 5DD, U.K.
[2] IT Services, University of York, Heslington, York YO10 5DD, U.K.

Abstract

The aim of this eCSE was to substantially improve the parallel performance of the CASTEP materials modelling program, reducing the time-to-science and extending the range of cores which may be used efficiently. This was to be accomplished by; re-engineering CASTEP's main distributed-memory parallel decomposition, and extending and optimising CASTEP's shared-memory parallelism.

1 Introduction

Density Functional Theory (DFT) is a simulation method using quantum mechanics, that has become one of the most widely used research tools for studying materials and systems of interest where the electrons dictate the behaviour. Modelling systems of interest exactly by solving the many interacting electron Schrödinger equation directly is not practical for systems of more than a few electrons, as the computer memory required for such problems scales exponentially with the number of electrons. Instead, the fundamental DFT theorems state that all ground state properties of the system (observables) are functionals of the electron density $\rho(r)$, which does not have a problematic memory scaling with the number of electrons. The energy functional is of particular interest, as by knowing the energy and its derivatives one can predict the lowest energy (ground state) structure of a material/molecule. The ground state of a system is the most stable state and therefore the state most likely to be found in nature. Hence, determining the ground state structure must usually be done before attempting to calculate more complicated properties of the material/molecule. The exact energy functional is unknown, but with some relatively simple approximations one can yield highly accurate results, with approximately 30,000 papers using DFT published each year.[1]

The CASTEP code is a UK flagship code specialising in material simulation using DFT [1], and is heavily used on ARCHER2 (over 100 active users). CASTEP uses periodic boundary conditions for its systems and as such uses plane waves as its basis set for the Kohn-Sham wavefunctions and densities and the interaction potentials. The system is represented using a periodic unit cell of finite size which restricts the plane wave which are permitted in the system to a discrete 3D grid, each point is defined by its position in reciprocal space, its "G-vector". A complete basis set would still be infinite however, therefore a sparse representation in Fourier space is used in which it is assumed that all Fourier components with G-vectors which have a magnitude larger than some user input, have zero weighting. This leads to a basis set made up of a sphere of G-vectors in reciprocal space.

CASTEP makes heavy use of Fast Fourier Transforms (FFTs) to transform between Fourier/reciprocal space and real space which is required to perform several operations, such as calculating the electron density from the wavefunctions, or applying the external potential due to an atom. When running large simulations on a parallel computer, the most common data decomposition is to distribute the Fourier wavevectors ("G-vectors") over MPI ranks, but the resultant parallel 3D FFT limits its strong scaling. This is due to the all-to-all communications required by the current FFT algorithm, which generate a large number of small simultaneous messages, which in turn places a large burden on the network. For N MPI processes in the G-vector group, all N processes exchange data so the total number of messages scales as N^2, and the message length scales as $\frac{1}{N^2}$. As N increases, the message size decreases and eventually the communication time is dominated by the interconnect latency; furthermore, the quadratic increase in the number of messages leads to

[1] See, for example, atomistic.software

contention for shared network resources, in particular the Network Interface Controllers (NICs). This causes issues on ARCHER2, which has 128 physical cores per node which all share two NICs.

G-vector parallelism may be combined with shared-memory parallelism (MPI+OpenMP) to reduce memory overhead and improve parallel scaling somewhat. The shared-memory parallelism of CASTEP has been implemented using OpenMP directives for several of the more expensive operations such as large matrix-matrix multiplications, however not all parts of the code have been threaded. The shared memory parallelism is therefore not presently efficient enough to compensate fully for the loss of MPI parallel efficiency towards the limit of the strong-scaling regime. This project will (1) implement a load-balanced process-grid-based distributed-memory parallel decomposition of the G-vectors in CASTEP to improve parallel efficiency and time-to-science; and (2) increase shared-memory parallelism across core routines to reduce the memory footprint and further improve parallel scaling.

2 eCSE Aims

2.1 Re-engineer CASTEP's main distributed-memory parallel decomposition

For large parallel CASTEP calculations the Fourier wavevectors ("G-vectors") are distributed across MPI ranks. A Fast Fourier Transform (FFT) requires information from all MPI ranks, and the present distribution necessitates global all-to-all communications which limit the parallel scaling. We will replace the existing G-vector parallelism with one based on a 2D logical process-grid to significantly reduce the time-to-science and increase the parallel efficiency of CASTEP simulations. The decomposition has been implemented in a prototype and shown to perform particularly well in the latency-dominated regime.

- **OB1a**: Implement the process-grid parallel decomposition in CASTEP.
- **OB1b**: Extend decomposition to allow non-rectangular process topologies.
- **OB1c**: Optimise the process placement within the nodes to improve performance.

2.2 Extend and optimise CASTEP's shared-memory parallelism

Computational time is not the only resource which must be considered; the memory requirements are also important and ARCHER2's memory per core is 25% less than ARCHER1. Some of CASTEP's data objects are currently replicated across G-vector MPI processes, and the storage required can be considerable for large calculations. This is particularly the case for calculations which make use of a non-local exchange-correlation (NLXC) functional, where each MPI process has a copy of the entire Kohn-Sham wavefunction, which is a very large memory object. Shared-memory is an effective way to reduce the memory footprint per node, by allowing multiple processes or threads to access a single shared copy. For a given core count, shared-memory parallelism can improve the MPI communication time by reducing the number of MPI processes per node, thus reducing contention for NICs, and using threads to occupy the remaining cores. However, the present OpenMP threaded implementation only scales well to around $4-6$ threads.

- **OB2a**: Reduce the memory overhead per node.
- **OB2b**: Extend and optimise the OpenMP parallelism to improve scaling with threads.

3 Work Done

3.1 New Parallel Memory Decomposition

3.1.1 Current Decomposition

Performing a Fourier transform from reciprocal space to real space requires doing a 3D FFT of the sparse G-vector decomposition. In order to perform a FFT along one of the dimensions, a single MPI process must have all the data which has the same co-ordinates in the other two dimensions, i.e for an FFT along the x dimension, a single MPI process should have a "pencil" of all the data with a particular (y, z) coordinate. In order to limit the communications required, the sphere of weighted data points are partitioned into "pencils" of data of differing "lengths", depending on how many points of the sphere they contain (i.e. all points with weight > 0). The pencils are ordered according to this length and then distributed among the MPI processes in a round robin fashion. The 3D FFT under CASTEPs current G-vector decomposition for N processes follows the following steps:

1. All processes perform a 1D FFT along x-direction for their x pencils.
2. All processes take part in an N-way all-to-all communication, with $\sim N^2$ messages of length $\sim \frac{1}{N^2}$
    - This transposes the data such that it is arranged as y pencils, not x pencils of data

3. All processes perform a 1D FFT along y-direction.
4. All processes take part in another N-way all-to-all communications, $\sim N^2$ messages of length $\sim \frac{1}{N^2}$
    - This transposes the data such that it is arranged as z pencils not y pencils of data
5. All processes perform a 1D FFT along z-direction.

In the current G-vector decomposition, the strong parallel scaling is severely limited by the communication cost of the all-to-all data transposes. There are $\sim N^2$ messages, but they are of size $\sim 1/N^2$ (constant total data exchanged) and in principle all N cores can send messages simultaneously – so in the bandwidth-limited regime this isn't too bad. When latency-limited, the data size is irrelevant so the number of messages is the key metric. This causes particular issues on systems with fat nodes, such as ARCHER2 where each node has 128 cores competing for only 2 NICs. This contention causes massive problems when these all-to-all communications are internode, because the sharing of NICs means the hardware cannot handle N messages in flight simultaneously, and consequently some of the communications are serialised. The slightly naïve choice to distribute the x pencils of data using a round robin approach also has performance implications. The round robin approach is guaranteed to lead to a load imbalance between MPI processes, i.e. some processes will have more work to do than others, meaning that at each communication point those with less work are forced to wait for those processes that are still working on their data, and hence the total run time is dictated by the MPI process with the largest amount of work.

3.1.2 'Regular' Process Grid Decomposition

As is pointed out above, the limiting factor to the strong parallel scaling of CASTEP is the communications cost of performing two all-to-all data transposes for each FFT. In this work, an alternative distribution was implemented based on a process-grid. In a simple 2D process grid, the processes are mapped directly to the 2D grid of G-vector columns. Each of the 3D FFT's data transpositions now only requires collective communications between processes in the same row or column of the process grid, avoiding global all-to-all communications.

The 3D FFT under the new G-vector decomposition for N processes which form a 'regular' $C \times R$ process grid follows the following steps:

1. All processes perform a 1D FFT along x-direction.
2. All processes take part in one R-way all-to-all communications, of which there are C in total. $\sim CR^2$ messages, of length $\sim 1/N^2$
    - These are performed between MPI processes in the same process grid column.
    - This transposes the data from being arranged such that x is the fastest index to y being the fastest index.
3. All processes perform a 1D FFT along y-direction
4. All processes take part in one C-way all-to-all communications, of which there are R in total. $\sim RC^2$ messages, of length $\sim 1/N^2$
    - These are performed between MPI processes in the same process grid row.
    - This transposes the data from being arranged such that y is the fastest index to z being the fastest index.
5. All processes perform a 1D FFT along z-direction

In these communication phases, each process only communicates with the processes in the same row (or column) of the process grid. When C and R are $\simeq \sqrt{N}$, this results in a total message count of $\sim N^{\frac{3}{2}}$, and the cost of the communications is improved by a factor of $\sqrt{N}$ from the original $\sim N^2$. A regular grid can be formed when the number of columns and rows in the process grid, C and R, can be chosen such that $CR = N$, and that C and R are $\sqrt{N}$, or $\sqrt{N}$ and $\sqrt{N}+1$, where C and R are both integers.

The $N^{\frac{3}{2}}$ scaling of the cost of the transpose of the logical process grid is achieved by constraining communications such that data is exchanged strictly only between those processes sharing a common column (row) coordinate of the process grid in the first (second) transpose phase of the 3D FFT. This introduces the further constraint that the G-vector pencils to be distributed over a given column (row) of the process grid must share some subset of FFT grid coordinates. Our domain decomposition thus proceeds in two steps: first, planes of FFT data are distributed to each column of the process grid; and second, the pencils which constitute each plane are then distributed to the processes in the given process grid column.

Load balancing of the columns and rows of the process grid, and subsequently of the processes therein, is a non-trivial task because the G-vector sphere leads to different weights (number of nonzero elements) for the pencils. Optimal load balancing in this context is a well-known deterministic scheduling problem: the non-preemptive assignment of independent tasks to a set of processors where the objective function is to minimise the makespan. The independent tasks are here the G-vector planes (the set of all data pencils sharing a single

co-ordinate value, e.g. z), and subsequently pencils; the set of processors is comprised of the columns (or rows) of the process grid, and subsequently the individual processes therein; the makespan is the wallclock time of the given CASTEP calculation. The problem is NP-hard, and there have been several polynomial time approximation schemes developed (see for example reference [2] for a summary of many of these). Among these, we chose to implement the generalised Karmarkar-Karp largest differencing method [3], as our testing found this to run in negligible wallclock time and be robustly performant for all problem sizes of interest.

3.1.3 'Irregular' Process Grid Decomposition

There are some numbers of processes, N, for which no efficient regular process grid can be formed. We define a 'regular' process grid as a rectangular grid where all the rows of the process grid have the same number of processes, and all the columns of the process grid have the same number of processes (but not necessarily the same number as in each row). The extreme example would be where N is a prime number, in which case the only regular process grids that could be made would be a $1 \times N$ or an $N \times 1$ process grid. Neither of these options allows the communication costs of the FFT to be reduced substantially. The first question which arises is: why would anyone choose to parallelise over a prime number of processes? In some of the calculation tasks that CASTEP can perform, the allowed symmetries of the system can change several times, leading to a change in the optimal parallel strategy. This change in parallelisation is handled "on the fly" by CASTEP, which means that what may have seemed to the user as a sensible choice for the total number of processes at the beginning of the calculation (given their initial system's symmetries), may end up leading to an inconvenient number of processes to form a process grid at later stages of the calculation. It is therefore necessary for the parallel distribution of the work to handle all numbers of processes and to do something performant in all cases.

In the initial implementation, we choose the number of process grid columns, C, to be $\sqrt{N}$ rounded down ($C = \lfloor\sqrt{N}\rfloor$). The number of processes per columns, R, is then $R = \lfloor\frac{N}{\lfloor\sqrt{N}\rfloor}\rfloor$ which will either be $\sqrt{N}$ or $\sqrt{N}+1$ and some remaining spare processes, S, due to the integer divison. If $S = 0$ we have a regular process grid, if $S > 0$ then we distribute 1 extra process to the first S process grid columns. In the case of irregular process grids, the FFT follows the following steps:

1. All processes perform a 1D FFT along x-direction for their x pencils.
2. All processes take part in one R-way all-to-all communications.
   - Performed between MPI processes in the same process grid column.
   - This transposes the data such that it is arranged as y pencils.
   - R here is the number of processes in the process grid column, which is no longer required to be the same for all columns.
   - Any of S spare processes at this point are assigned no y pencils, and do no work for the next section of the calculation.
3. Only processes that form a regular grid perform a 1D FFT along y-direction
4. All processes take part in one C-way all-to-all communications of which there are R. RC^2 messages, of length $1/N^2$
   - These are performed between MPI processes in the same process grid row.
   - This transposes the data to z pencils.
   - The S spare processes rejoin the calculation and are assigned z pencils.
   - R here is the number of processes in the process grid column which is not the same for all columns.
5. All processes perform a 1D FFT along z-direction

The robust performance of the Karmarkar-Karp load balancing algorithm relies on the assumption that the G-vector data is being distributed to *identical* processors, i.e. each and every column or row of the process grid has exactly the same processing speed with respect to any given element of data. This assumption is true for the regular process grid decomposition as, by definition, each column or row of the grid comprises an equal number of processes, and thus each indeed possesses equal processing speed. However, the assumption clearly does not hold for irregular process grids, where unequal numbers of processes are permitted to constitute each column or row. The load balancing problem here is hence one of *uniform* processors, i.e. the columns and rows of the irregular grid have different relative processing power, despite each of their constituent processes being identical. We thus implemented an additional load balancing algorithm for irregular grids, based largely on that described in Ref.[4] and optimised specifically for uniform processors. Our tests demonstrated that this algorithm performs robustly and efficiently over a wide range of irregular process grids and data sets of interest.

3.1.4 Performance Improvements

The aim of introducing the new parallel data decomposition is to reduce the cost the communications during the FFT stages of CASTEP execution. Figure 1 shows how the percentage of the total runtime spent performing communications for FFTs scales as the total number of MPI processes is increased for a medium-sized CASTEP benchmark (270 atom surface slab of Al_2O_3), comparing CASTEP's current parallel data distribution with the new process grid distribution. This benchmark uses two k-points, which are sampling points in the material's Brillouin zone, and CASTEP will automatically parallelise over these; this in this case the G-vectors for each k-point are distributed over half of the total MPI ranks. It can be seen in both cases that a significant portion of the run is spent on these communications, however for CASTEP's current distribution, above 256 processes (once the G-vector parallelism groups span more than 1 node of ARCHER2) the FFT communications dominate the calculation time. In contrast, for the process grid distribution the proportion of the calculation spent on the FFT communications never rises above ∼30%, and indeed it remains at a similar fraction even at high process counts. Clearly, using the implementation of the process grid is successful at reducing the cost of the FFT communications in CASTEP.

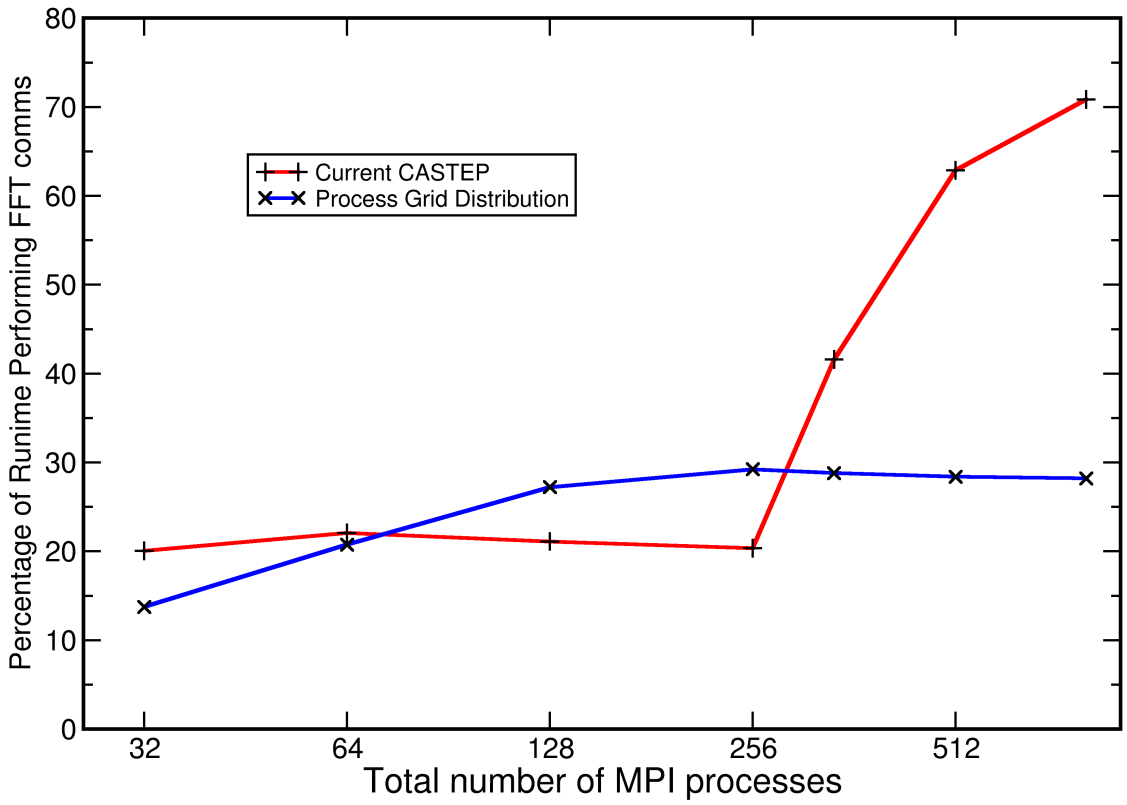

Figure 1: Scaling of the percentage of total runtime spent performing FFT communications as the total number of MPI processes is increased, comparing the current CASTEP decomposition and the Process grid decomposition for CASTEPs medium benchmark.

The performance of the FFT communication has been dramatically improved at higher process counts. The result of this is that CASTEPs parallel scaling as a whole is far better for G-vector parallelism and CASTEP exhibits good scaling to 4× as many MPI processes without getting slower from adding more processes. Figure 2 shows the parallel speed up of ≈ 14× when using 32× as many MPI processes when compared to a calculation using 32 MPI processes. Comparing the old distribution verses the new distribution, the new distribution is 6× faster on 1024 MPI processes.

Once the new parallel distribution was implemented and profiled, it became clear that other MPI calls were now starting to take a significant portion of the runtime (≃ 25%), specifically some calls to MPI_allreduce which were making use of a buffer. By switching these calls to use an in-place reduction, several memory copies can be avoided which improves performance even further. Now that MPI calls other than the all-to-alls have become more dominant, future parallel scaling optimisations should focus on ways to improve these, for instance by using asynchronous MPI calls.

Objectives **OB1a** and **OB1b** of our aims are defined with success metrics of achieving a 20% reduction in calculation time for both rectangular and non-rectangular process grids, particularly at 1024 cores of ARCHER2, which can be seen to have been achieved from figure 2. CASTEP passes the full test suite using

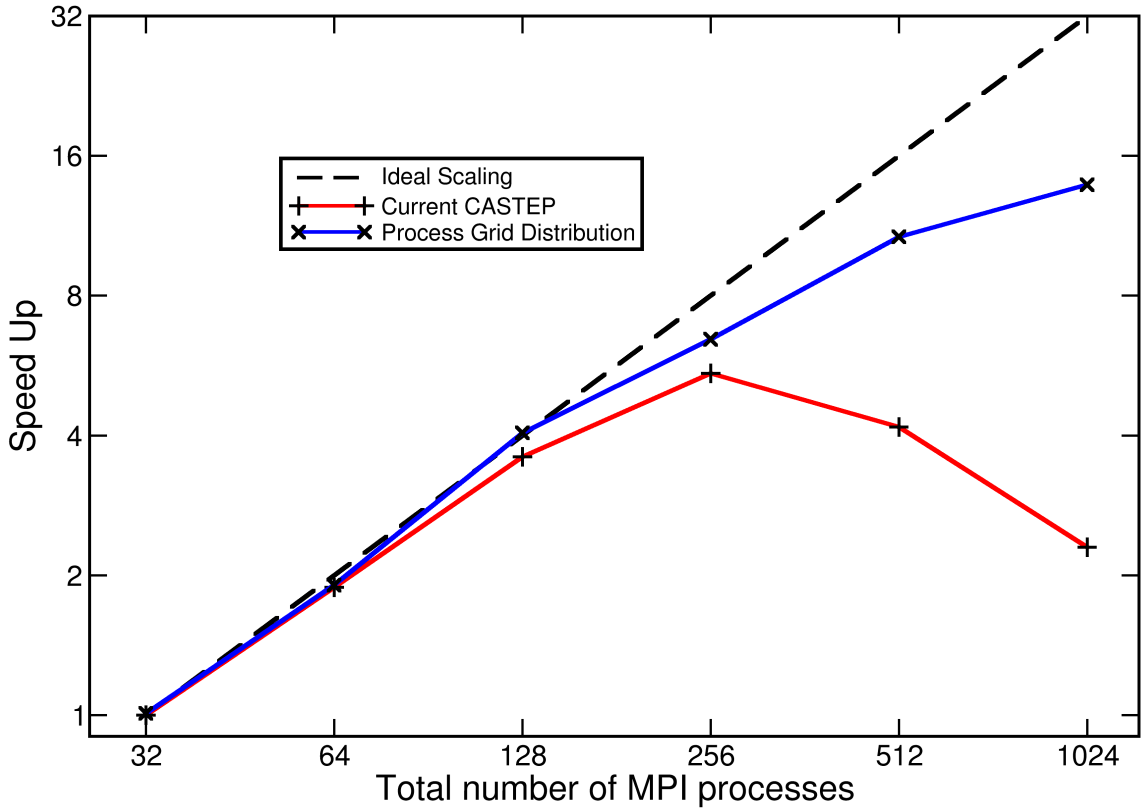

Figure 2: Parallel scaling of the average time per iteration for CASTEPs medium benchmark calculation, when comparing the current CASTEP decomposition and the Process grid decomposition.

the new parallel distribution for both rectangular and non-rectangular process grid using a variety compilers and optimisation levels. This verifies that the implementation is correct and has not broken any aspect of CASTEP functionality and was part of the success metric for objectives **OB1a** and **OB1b**. The final objective related to the process grid decomposition was to achieve 70% parallel efficiency on 4096 physical cores of ARCHER2 for the CASTEPs medium benchmark. Although a factor of 2 speed up was enabled by this work (when comparing using 8 threads) 70% parallel efficiency was not achieved. We attribute this partially to the improved threading performance which reduces the time spent performing work as well as the time spent in the communications and to communications other than those optimised in the FFT becoming more dominant. The other cause could be load balancing, using many MPI processes means that even with appropriate load balancing algorithms even an optimised solution is still imbalanced.

3.2 Shared Memory Parallelism

3.2.1 NLXC calculations

CASTEPs current shared memory parallelism is not yet efficient enought to provide comprable performance when compared with CASTEPs MPI parallelism. There are serialised parts of the code that are tightly wound up with communications, making them difficult to implement threading for such as the FFTs. When a CASTEP calculation use the NLXC functionals, the majority of the work is done either in the NLXC module or is dominated by FFTs, in both sections, the code has not been threaded. This leaves users with only CASTEPs MPI parallelism to parallelise calculations, which as perviously mentioned is problematic for NLXC calculations as all MPI procecess have a copy of all the Kohn-Sham wavefunctions, which for large systems consumes huge amounts of memory, limiting the number of MPI processes per node. This would not be an issue if shared memory parallelism was an effective parallelisation strategy however until the new G-vector distribution was implemented, there was very little benefit to be gained from using shared memory as the vast majority of the runtime for NLXC calculations was spent performing communications.

The main findings of our investigation into further exploiting shared memory parallelism was that once the communications involved with the FFTs were improved/reduced, there are large performance gains from threading the work of the FFTs. This was done by modifying the CASTEP interface to the FFTW3 library to use threading and using FFTW3s default behaviour. This had been trialed previously in CASTEP but at that point it was found that the majority of the cost was in the communications, therefore threading

the FFTs was not found to have a significant impact on performance. The improved performance of the communications involved with the FFTs means that threading the FFTs now has a much more significant effect on the CASTEP run time. In addition to this, at several points in CASTEPs NLXC module OMP workshare directives were added and these were found to further improve performance. Further attempts at threading more of the module were found to either generate inconsistent performance improvements or were found to be detrimental to performance, there could be many reasons for this, such as poor cache performance, large OMP performance overheads.

3.2.2 Performance improvements

The benchmark NLXC calculation uses Screened Hartree-Fock Exchange for 4 atom unit cell of Fe_2VAl. The calculation is already 10-way MPI parallel over a different CASTEP parallelisation strategy which is ordinarily found to give close to ideal scaling. It should be noted that although the benchmark is only 4 atoms, the 10-way parallel calculation takes around 5.5 hours to complete, highlighting the expensive nature of NLXC calculations. Prior to this eCSE, the performance gain due to threading these calculations would have been negligible as the majority of the work done in these calculations was unthreaded, therefore it can be assumed that any speed up due to threading in this calculation is a direct result of the work done as part of this eCSE.

Figures 3 and 4 show how the calculation scales in parallel speed up and memory usage per node when compared to the 10-way MPI parallel calculation. From Figure 3 it can be seen that the threaded parallelism now scales well up to around 8 threads (80 total cores used), outperforming the new G-vector parallel distribution by a factor of around 1.5×. At 16 threads / 16 MPI processes G-vector parallel the two parallelism methods are fairly comparable at around 7× parallel speed up. Beyond 16-way parallelism the threading scales very poorly, never achieving higher than a 7× speed up, this is unsurprising given that NUMA regions on ARCHER2 include 16 cores and the data is initialised on the main thread, therefore all the data first be touched by the main thread in the first NUMA region and any memory accesses from threads outside the first NUMA region will be more expensive and slow the whole calculation.

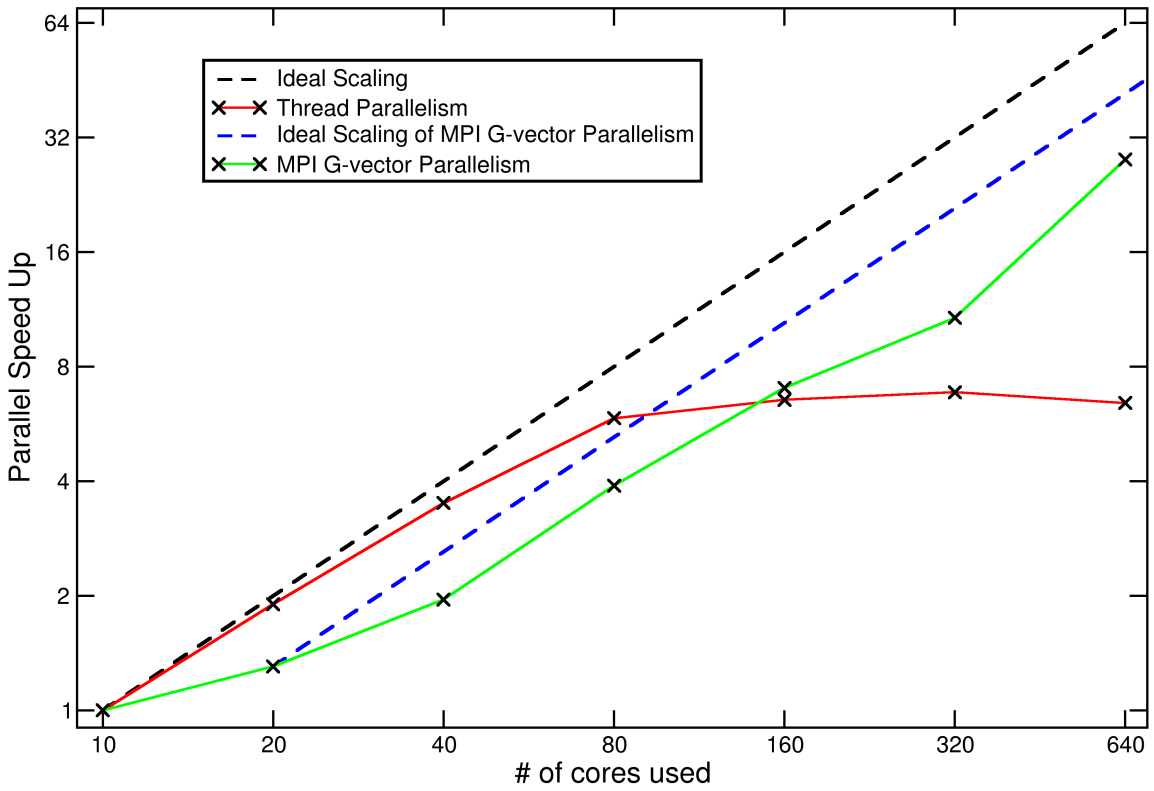

Figure 3: Comparison of performance when running the benchmark NLXC calculations using threading verses using MPI G-vector parallelism. The calculation is already 10-way parallel over a different CASTEP parallelisation strategy. The dashed blue line represents the ideal scaling of G-vector parallelism where it is more fair to compare scaling against a calculation which is already 2-way parallel as a less efficient FFT algorithm must be used when using any form G-vector parallelism. The G-vector parallelism used here is the new process grid distributions, not the original CASTEP distribution.

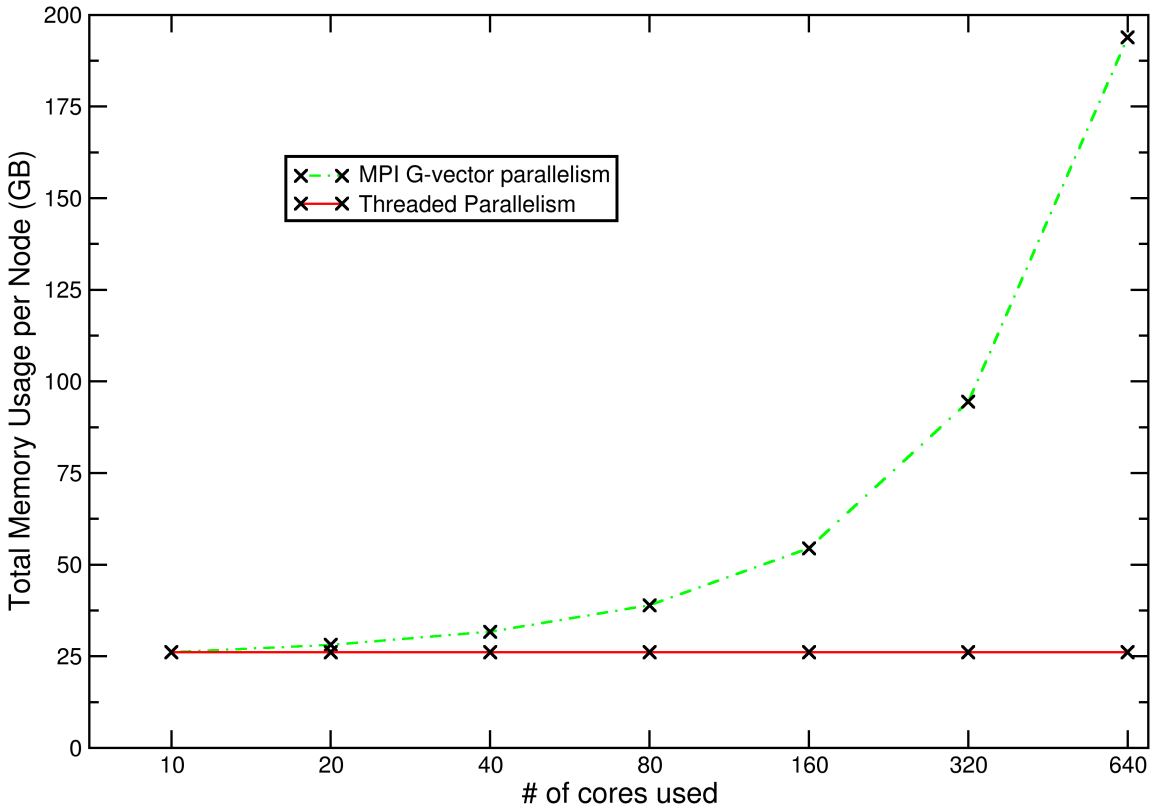

Figure 4: Comparison of CASTEPs memory usage per node when running the benchmark NLXC calculations using threading verses using MPI G-vector parallelism. Since each additional MPI process uses their own copy of the electron wavefunctions, the memory scales very poorly. The calculation is already 10-way parallel over a different CASTEP parallelisation strategy.

At higher levels of parallelism the MPI G-vector parallelisation scales far better, achieving a 28× speed up when 64-way G-vector parallel, far outstripping the 6.4× speed up of the 64-way threaded parallelism. Figure 4 shows the memory usage of two parallelisation strategies, the threaded parallelism uses a constant amount of memory, as is expected for a shared memory strategy, the MPI G-vector parallelism scales much worse as the number of cores is increased. The MPI parallelism scales poorly since each MPI processes uses its own duplicate copy of the electron wavefunctions to avoid a large amount of communications that would otherwise be required. This means that even for this small calculation, CASTEPs memory requirements prevent the calculation being run with more than 640 MPI processes without under populating nodes (each ARCHER2 node has 256 GB of memory available). Under populating nodes with MPI processes is a useful way to perform calculations with higher memory requirements but is inherently wasteful as several cores will be unused yet still drawing significant amounts of power and will use up budgets allocated to users. By implementing some amount of threading for these calculations, however limited the scaling may be, allow cores more of these cores to be utilised before a threshold for having to underpopulate nodes is reached.

4 Conclusion

As part of this work we have implemented a new parallel decomposition in CASTEP for its wavefunction representation which uses a process grid like communication topology. This has been extended for non-rectangular processes grid topologies which uses a multi-fit load-balancing algorithm to distribute the work among processes. The FFT has two communication stages, in order to avoid the chances of having one communication phase which is intra-node and one that is inter-node we do not distribute the MPI processes sequentially to the process grid instead distributing the odd ranks to the gird first then the even ranks.

The main shared memory optimisation achieved in this work was found to be enabling threading of the FFTs. This has so far only be implemented for CASTEPs interface for FFTW3, but this will be extended to other the library interfaces for which threading is available in time for the next CASTEP release. NLXC calculations now scale very well up to 8 threads on ARCHER2, beyond that using more threads yields very little performance gain, however allows far more cores to be used for no additional memory overhead unlike MPI G-vector parallelism.

The new code has been merged into the CASTEP development repository and will be included in the forthcoming 2024 release of the code, after which it will be widely available to ARCHER2 users. Key beneficiaries of the project include the CCP-NC, CCP9 and UKCP communities, where the new parallel decomposition and shared memory parallelism optimisations will greatly improve parallel scaling of all calculations but especially for large scale simulations which may be used run in ARCHER2.

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

References

[1] S. J. Clark, M. D. Segall, C. J. Pickard, P. J. Hasnip, M. J. Probert, K. Refson, and M.C. Payne. First principles methods using CASTEP. *Z. Kristall.*, 220:567–570, 2005.

[2] Ethan L Schreiber, Richard E Korf, and Michael D Moffitt. Optimal multi-way number partitioning. *Journal of the ACM (JACM)*, 65(4):1–61, 2018.

[3] Narendra Karmarkar and Richard M Karp. *The differencing method of set partitioning.* Computer Science Division (EECS), University of California Berkeley, 1982.

[4] Donald K. Friesen and Michael A. Langston. Bounds for multifit scheduling on uniform processors. *SIAM Journal on Computing*, 12(1):60–70, 1983.