Boosting the scaling performance of CASTEP: enabling next generation HPC for next generation science

Dominik Jochym[1], Jolyon Aarons[2], Keith Refson[1], Phil Hasnip[2] and Matt Probert[2]

[1]*Science and Technology Facilities Council,*
Rutherford Appleton Laboratory, Harwell Science and Innovation Campus, Didcot,
Oxfordshire, OX11 0QX, U.K.

[2]*Department of Physics, University of York, Heslington,*
York, YO10 5DD, U.K.

April 02, 2012

Abstract

This report describes three areas of development to the CASTEP density functional theory code. The main objectives of the work are to: improve I/O performance, produce a parallel efficiency report for users at run-time and further develop the band-parallel capability of CASTEP. In particular, more details will be given on the following developments:

- MPI collectives to replace MPI point-to-points for the wave function I/O routines. For a test case aluminium oxide '2x2' slab (al2x2) containing 5 k-points, 40000 G-vectors and 288 bands, faster reading and writing times will be demonstrated.
- A parallel efficiency report which is now written at the end of every CASTEP run. In addition to providing the basic parallel efficiencies, the report provides information regarding the parallel decomposition used. This information may also be used to see whether any further optimisations might be possible. The report is also capable of providing details on any aspects of the calculation that are particularly important to the parallelisation (e.g. the k-point distribution and G-vector communications). This will help CASTEP users to take full advantage of the parallel capability of the application and will enable more efficient use on HECToR and other high-end HPC architectures
- The band-parallel capability of CASTEP was improved by implementing an upgraded "triangular matrix" algorithm. This will provide a useful speedup to all band-parallel calculations. In particular, the improved method is 1.08 times faster with 256 HECToR Phase 3 cores and 1.16 times faster with 1024 cores.

Currently, the I/O improvements and parallel efficiency report have been incorporated into the main CASTEP 6.1 source repository; the band-parallel improvements will be available from CASTEP 7.0.

Contents

1 MPI buffer memory optimisation

1.1 Introduction

A large number of point-to-point MPI communications from many processes to, say, a master process during a checkpoint operation, can lead to an MPI "out of unexpected buffer space" error. If the MPI_receive calls are not pre-posted, the MPI implementation stores any "unexpected" messages in memory until the receive call is reached. One way to avoid this error is to increase the size of the unexpected buffer. This is undesirable as this memory is reserved throughout the job at the expense of that available for the intended purpose of the job. In this section we will discuss a solution to this problem using MPI collectives, in the context of the electronic structure code, CASTEP [1].

For most uses of CASTEP, the largest single piece of data is the wave function, essentially represented as a four dimensional, double precision complex array. The four dimensions are over plane waves (G-vectors), bands, k-points and spin. In a parallel HPC environment it is currently possible for CASTEP to distribute the wave function over k-points, bands and G-vectors. Typical "large" CASTEP calculations have fewer than 10 k-points, often only one (certain classes of phonon calculations require 1000s of k-points); the order of 100-1000s of bands; G-vectors numbering 104-105; and spin is always 1 or 2 in the colinear case. The wave function is written to disk as part of the CASTEP checkpoint process, understandably labelled a '.check' file, which can reach many Gb in size. During some CASTEP calculations auxiliary files containing wave functions are also used in order to make jobs restorable from a partially completed point or for further analysis.

The routines in CASTEP that perform the wave function I/O are appropriately named wave_read and wave_write. In the current 5.5 release of CASTEP, the disk I/O of the write operation is done by what we will refer to as the root node. (Internally, CASTEP refers to processes running on separate nodes. This does not necessarily mean they are running on physically separate compute nodes.) Each node sends its wave function data to the master in its band group, which in turn sends its data to its G-vector master and then passes it on to the root node for disk output. All this is done using MPI point-to-point communications.

As an aside, we would like to discuss why MPI-IO is not used. The CASTEP .check file is an unformatted fortran file that uses big endian byte order. This is to provide portability across any environment in which CASTEP is used. It was specified by the CASTEP developer group that backwards compatibility with the existing format of the .check file is to be maintained. Using MPI-IO (or NetCDF or HDF5) would allow each MPI process to access the .check file in parallel. NetCDF and HDF5 also provide architecture neutral files. (MPI-IO does allow for machine independent files, but vendors rarely implement this.) However, none of the above methods are compatible with unformatted fortran records and therefore break compatibility with existing CASTEP .check files.

1.2 Implementation

Often, the greatest amount of distribution in a CASTEP job is over G-vectors. Because of this, our initial prototype was to rewrite wave_write to use an MPI gather over G-vectors. Our approach is to collect the data for each band at a particular k-point and spin and then pass each band or block of bands up to the nodes that are both G-vector and band masters. In turn the data is to be passed to the root node for output. The figures in Table 1 show timings, in seconds, for wave_write from the CASTEP 5.5 release and our prototype code that uses collectives.

These benchmarks were run on HECToR Phase 2b with the Gemini interconnect. It should be noted that the time spent in disk I/O for all runs in Table 1 is approximately 6.4s, as deter-

No. processing elements	Version 5.5 write	Collectives write
24	7.09	7.63
48	7.23	7.72
96	9.85	7.88
192	19.49	8.13
384	70.48	8.42

Table 1: Benchmark times (in seconds) for wave_write.

mined by the internal trace function of CASTEP. The test case was based on an aluminium oxide '2x2' slab (subsequently referred to as al2x2) that contains 5 k-points, 40000 G-vectors and 288 bands. Only G-vector parallel distribution was used for the runs. The code has been checked for correctness with combinations of all three parallel distribution strategies, producing a correctly formatted CASTEP .check file that is compatible with existing analysis tools. For comparison, CASTEP's ground state SCF calculation on the al2x2 system takes approximately 1300s on 120 processing elements (5-way k-point distribution and 24-way G-vector) and approximately 16s is spent in 2 calls to wave_write.

The same was done for wave_read, rewriting it such that the data is distributed with an MPI scatter over G-vectors. The wave_read routine also contains extra machinery to determine how to distribute the data if the parallel data distribution has changed. Our initial prototype wave_read, provided substantial improvement over that in CASTEP 5.5. However, the collectives version of wave read was taking more than 4 times longer than the corresponding wave_write in some cases (see Table 2).

Further investigation, helped greatly by CASTEP's internal trace profiling routines showed some inefficiencies that could easily be rectified. First was that one of the internal arrays for reading in the wave function data was specified as (a block of) bands by G-vectors. Most fortran implementations store arrays in contiguous memory in array element order. The extra stride through memory during the fortran read statement for each bands set of G-vectors was causing slow running of the code. This was made more efficient by simply switching the order of the array indices so that G-vectors came first. The second optimisation came from rewriting the data handling for reordering the wave function data if the G-vector distribution has changed between runs. In the existing code a conditional statement is evaluated for every plane wave in the read-in data, nested within loops over the nodes in the G-vector group for the current run and the number of bands. The mapping for redistributing the data over G-vectors is independent of bands, so can be done outside of the band loop. In our optimised version of the routine, a many-one vector subscript is used to store the mapping between the read-in data and the distribution for the current run vector subscript. The array of data to be scattered over G-vectors is then assigned in a single statement with an indirect index for the read-in data array.

No. processing elements	Version 5.5 read	Prototype collectives read	Optimised collectives read	Collectives write
24	16.78	14.74	9.15	7.63
48	22.03	15.97	9.20	7.72
96	32.90	18.57	9.23	7.88
192	57.61	23.94	9.36	8.13
384	113.47	34.80	9.60	8.42

Table 2: Benchmark times (in seconds) for wave_read and, for comparison, wave_write.

Table 2 lists timings for wave_read in its original, prototype and optimised forms. The tests were performed using the al2x2 system, as outlined in 1.2. Time spent in disk I/O in each wave_read was approximately constant at a total of 7.0s. Note that it is expected that wave_read takes longer than the corresponding wave_write because all the handling for reordering data for changed parallel distributions is done in the read routine.

In analogy to the work on wave_read and wave_write, the corresponding read/write routines for the electron density and potential were modified to use MPI collectives. This task was straightforward as these objects are independent of k-points and bands.

The routine wave_apply_symmetry is an essential part of phonon calculations. It transforms a wavefunction from one k-point set to another one related by symmetry operations. For certain classes of large phonon calculations, the version of wave_apply_symmetry in CASTEP version 5.5 fails on HECToR phase 2a with an MPI unexpected buffer error. This error occurs because of the many point-to-point MPI communications that are present. To avoid use of the unexpected buffer, it was decided to use MPI gather over G-vectors for each band's data before applying the symmetry operation. The data is then scattered back to the appropriate nodes. It is possible that a symmetry transformation maps data between k-points that are stored on different nodes. In this case the gathered band data is sent to the node that holds that k-point, where the symmetry operations are applied before scattering.

The modified version of wave_apply_symmetry was then applied in situations where the unexpected buffer error had been reproduced. As expected, the jobs ran to completion without having to alter the MPICH_UNEXPECTED_BUFFER variable. This opens up the ability to perform large phonon calculations without having to dedicate unfeasible amounts of RAM to the unexpected buffer.

At the request of Phil Hasnip of the CASTEP Developers Group, the lessons learned in modifying the above routines were also applied to wave_reassign. This routine is called under the assumption that the k-point and band distribution are unchanged. It takes the data from an existing wavefunction and maps it onto a different plane-wave basis set. It is mostly used during variable cell geometry optimisations and molecular dynamics. The routine approximates the process of wave_write followed by wave_read but without any disk I/O. Performance gains in line with those detailed above were obtained.

1.3 Summary

In summary we have taken a number of CASTEP routines that made use of a large number of point-to-point MPI communications and modified them to make use of MPI collectives where appropriate. For some parallel data distribution strategies messages the size of a band or block of bands are still sent, but with the MPI receive call is pre-posted where possible to avoid unexpected buffer errors. A mechanism has been put in place to force pre-posting for large messages if requested. The user can also set the size of the block of bands being sent at run time.

The above code modifications make certain CASTEP calculations more feasible on HECToR and other HPC environments. Specifically, checkpoints and restarts are more efficient and restarts are now possible when a band parallel data distribution is used. Classes of phonon calculations that can use thousands of cores are now feasible without having to tweak MPI environment variables to sacrifice RAM from being used for the scientific computations. The optimisations are also of benefit for the checkpoint and restart of time-dependent density-functional-theory (TDDFT) calculations (the subject of another CASTEP dCSE [4]), where each TDDFT eigenvector is a wavefunction-sized object.

These changes were available initially as a beta version to HECToR users, and have been subsequently incorporated into the main CASTEP source code and released worldwide in CASTEP 6.0.

2 Parallel Efficiency Report

2.1 Outline

One of the difficulties scientists face in running any code on a massively parallel HPC machine is to exploit the parallelism efficiently. There is usually no means to determine in advance, nor to evaluate in retrospect the efficiency of any given parallel run. A series of benchmarking runs on different processor counts may be used to evaluate parallel efficiency and suggest a useful processor count for production runs. While somewhat burdensome, this approach is useful in the case where many separate near-identical runs are to be performed. In other cases, where the runs belonging to a project differ substantially, it is impractical. The problem is particularly acute for plane-wave DFT calculations where several distinct parallelisation schemes (FFT, band, kpoint) are used simultaneously and the efficiency is far from a regular monotonic function of processor count.

For this part of the work, CASTEP's various operations have been grouped into "communication" and "calculation" classes, and using the internal timing data a parallel efficiency report is written at the end of every run.

2.2 Implementation

CASTEP contains a "trace" module which provides, amongst other functionality, timing data via the "trace_entry" and "trace_exit" subroutines. There is a call to trace_entry at the start of most CASTEP subroutines, and a corresponding call to trace_exit at each exit point. Internally the trace module times how long CASTEP spent between the entry and exit point, and associates that time with the subroutine. The trace module also supports the association of a subroutine with a class of operation, allowing the large amount of timing data to be reported per class of operation rather than per individual subroutine.

By defining the generic class "comms", and the specific sub-classes "comms_farm", "comms_gv", "comms_kp" and "comms_bnd", each communication subroutine could be associated with one or more communication class. At the end of the CASTEP calculation this aggregate communication time is compared to the total time for the calculation, and from the resultant comms:compute ratio a parallel efficiency is estimated. The subclasses (comms_gv, etc.) allow the efficiency to be broken down further to demonstrate the efficiency of each type of parallelism; this allows a user to evaluate the particular efficiency of each aspect of the present parallel distribution. An example of the CASTEP parallel efficiency report is given below, for the standard "al3x3" benchmark:

```
Overall parallel efficiency rating: Satisfactory (61%)

Data was distributed by:-
Gvector (128way); efficiency rating: Satisfactory (61%)

kpoint (2way); efficiency rating: Excellent (99%)
```

2.3 Output

In addition to providing the basic parallel efficiencies, the parallel efficiency report code reports the parallel decomposition used and also examines the parallelisation to see whether any further optimisations may be suggested. The report is also capable of providing comments on any aspects of the calculation that are particularly important to the parallelisation. In particular there are notes included if:

(a) If the k-points were not fully distributed (k-point parallelisation is extremely efficient);

(b) G-vector communications took advantage of CASTEP's SMP optimisations;

(c) If the calculation time was too short to give a meaningful estimate.

2.4 Summary

The parallel efficiency report implemented provides not only an estimate of the efficiency of a CASTEP calculation, but also comments and suggestions based on the parallelisation chosen. It should prove a strong aid to all CASTEP users on HECToR in selecting a suitable processor count, and will not only promote efficient use of the machine, but will encourage highly parallel runs using the improvements described in Section 1 , Section 3 and previous dCSE projects for CASTEP([2], [3], [4] and [5]). This will enable scientists to fully exploit the service and extend the range of cutting-edge science. In addition it will aid in other benchmarking and optimisation exercises.

These changes have been incorporated into the main CASTEP source code, and released world-wide in CASTEP 6.1.

3 Band Parallelism

3.1 Outline

The major bottleneck in a band-parallel calculation is the application of a band-by-band transformation matrix to the wavefunction, so optimising these subroutines was the focus of this section of the work. When band-parallelism is enabled in these transformations, there are two main drawbacks compared to the alternative G-vector distribution:

(a) No support for triangular matrices - Several key transformations are triangular, and whilst this is exploited in a G-vector parallel calculations it is not in a band-parallel calculation.

(b) Transformations require all-to-all communication amongst the cores in a band-group - Such communications are not only time consuming, but scale as n^2 for n-way band-parallel calculations.

Optimising these two aspects of the transformation is the key to obtaining good performance.

In general the transformation proceeds according to the following algorithm:

1. At entry: each core holds its share of the wavefunction's bands, and the section of the transformation matrix that is required to transform those bands; however in general the transformation will generate contributions to *all* the bands of the wavefunction, not just the local share. These contributions must be summed and communicated to the appropriate core.
2. Each core selects a "client" core, and applies the subset of the transformation that generates the contribution of the local bands to the client's transformed bands.
3. The transformed data is sent to the client core (not necessarily directly).

4. The core receives transformed data from other cores. All contributions to its local share of the transformed data are summed.
5. Each core selects a different client core, and repeats steps 2-5 until all of the transformation has been applied and the data exchanged with the other cores.
6. At exit: each core holds its share of the transformed wavefunction's bands

3.2 Triangular Matrix Optimisations

Whenever the set of bands has to be orthonormalised, the transformation matrix is in fact triangular; this is exploited in G-vector parallel calculations, which use the dtrmm and ztrmm BLAS subroutines in this case, rather than the general dgemm and zgemm ones, to gain a significant boost in performance.

Because the bands are distributed in a round-robin fashion amongst the cores, transforming a wavefunction by a (global) triangular matrix leads to each distributed transformation in step (2) of the algorithm also being triangular. Optimising this operation is relatively straightforward, provided care is taken for the case where the local number of bands is not the same as that of the client core; this case may be handled easily with a simple zero-padding. The results are shown in Table 3

3.3 Communications

In order to improve the communication time, two major changes were implemented. Firstly the original, naive communication pattern was replaced with a systolic loop (a ring topology); secondly the blocking, synchronous communications were replaced with non-blocking, asynchronous communications. The systolic algorithm for a three-way band communicator is shown in Figure 1.The final (or initial) stage is move the data one step around the systolic loop, to its corresponding node.

The performance of the initial implementation of the systolic loop was disappointing. In this initial implementation core n in the band group has neighbours in the systolic loop that were cores n-1 and n+1, modulo the total number of cores in the band group (as shown in Figure 2). Unfortunately this means for N total cores in the band group, each loop communication phase involves communications between core N and core 1, which is likely to be the longest route in the actual hardware topology; since the systolic algorithm is essentially synchronous, it is limited in speed by the longest communication.

To improve the performance, the systolic loop was reordered so that all the odd-numbered cores have odd neighbours, and the even cores have even neighbours, with the exception of the lowest and highest numbered cores (as shown in Figure 3). This increases the average distance between neighbours, but decreases the longest distance; in fact almost all communications are between cores that are 2 apart in the band-numbering. By reducing the time of the worst-performing communication, each communication phase of the systolic loop was much quicker.

Total cores	Band parallelism	Original code	Optimised for triangular matrices
512	8	816s	807s

Table 3: Comparison between the original and optimised triangular matrix operation for the al3x3 benchmark and CASTEP 6.1.

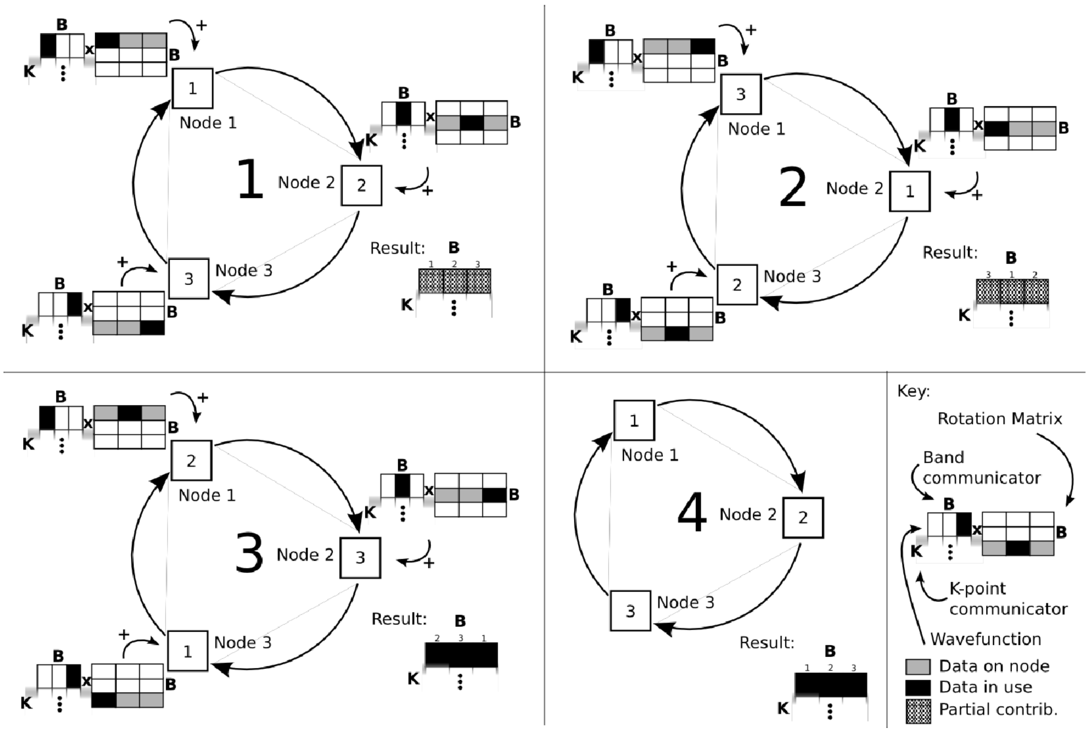

Figure 1: Diagram of the systolic algorithm for a three-way band communicator.

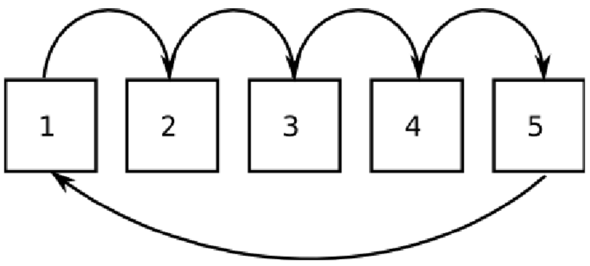

Figure 2: Initial implementation of the systolic loop.

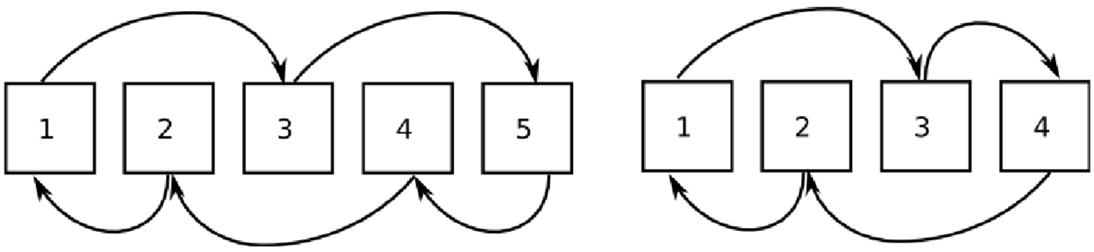

Figure 3: Improved implementation of the systolic loop.

Total cores	Band parallelism	Original code	New code
256	4	865s	804s
512	8	619s	607s
1024	16	647s	557s

Table 4: Comparisons of the improved algorithm for the al3x3 benchmark and CASTEP 6.1 (All with num_proc_in_smp : 2).

After testing, it was also found that reordering the gv- and bnd-groups was beneficial, so that consecutive MPI threads were in the same bnd-group rather than the same gv-group.

This data was collected for a system with 768 bands as shown in Table 4, and serves to show that the scaling is better with the new algorithm. The scaling advantage will come into its own when much larger systems are simulated, as there will be far more work to do per core and far more data to pass around. Triangular matrix optimisations were turned off to show the effect of the systolic algorithm alone.

3.4 Summary

As CASTEP calculations are run with an ever greater number of bands, the memory requirements per core, for storing the wavefunction data necessitate band-parallel calculations. The systolic algorithm implemented in this work will serve to benefit the scaling of the largest contemporary CASTEP calculations on HECToR and become ever more useful as the size of calculations increases into the future. The triangular matrix work will however provide a useful speedup to all band-parallel calculations, no matter their size.

These band-parallel improvements are available upon request to any HECToR user of CASTEP. They have been incorporated into the main CASTEP source code, and will be released worldwide in CASTEP 7.0.

4 Conclusions

The improvements described in Section 1 and Section 2 of this dCSE project have already been incorporated into the main CASTEP source repository and released to users; the remaining improvements described in this section will be available in the forthcoming 7.0 release.

Acknowledgements

This project was funded under the HECToR Distributed Computational Science and Engineering (CSE) Service operated by NAG Ltd. HECToR - A Research Councils UK High End Computing Service - is the UK's national supercomputing service, managed by EPSRC on behalf of the participating Research Councils. Its mission is to support capability science and engineering in UK academia. The HECToR supercomputers are managed by UoE HPCx Ltd and the CSE Support Service is provided by NAG Ltd. `http://www.hector.ac.uk`.

References

[1] `http://www.castep.org/`.

[2] `http://www.hector.ac.uk/cse/distributedcse/reports/castep/`.

[3] `http://www.hector.ac.uk/cse/distributedcse/reports/castep02/`.

[4] `http://www.hector.ac.uk/cse/distributedcse/reports/castep03/`.

[5] `http://www.hector.ac.uk/cse/distributedcse/reports/castep-geom/`.