

Towards capability computing with Castep

M Plummer[†] and K Refson[‡]

[†]CCLRC Daresbury Laboratory, Daresbury, Warrington, Cheshire, WA4 4AD, UK

[‡]CCLRC Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire OX11 0QX,
UK

Abstract

We describe the current status of the Castep code on HPCx and performance improvements made to the code over the last year or so. These are a mixture of source code improvements and operating system hardware and software improvements. We give performance examples and present a brief guide to users on how to maximize performance of Castep jobs. We indicate how further code optimizations and new developments will improve Castep performance in the near future.

This is a Technical Report from the HPCx Consortium.

Report available from
<http://www.hpcx.ac.uk/research/publications/HPCxTR0507.pdf>

© UoE HPCx Ltd 2005

Neither UoE HPCx Ltd nor its members separately accept any responsibility for loss or damage arising from the use of information contained in any of their reports or in any communication about their tests or investigations.

| | | |
|----------|--|-----------|
| <i>1</i> | <i>Introduction</i> | <i>1</i> |
| <i>2</i> | <i>The treatment of MPI_AllToAllV</i> | <i>2</i> |
| <i>3</i> | <i>General optimization of Castep</i> | <i>4</i> |
| <i>4</i> | <i>Performance examples</i> | <i>6</i> |
| <i>5</i> | <i>How to maximize performance of HPCx Castep jobs</i> | <i>9</i> |
| <i>6</i> | <i>The near future: upcoming terascaling features</i> | <i>10</i> |
| | <i>Acknowledgements</i> | <i>12</i> |
| | <i>References</i> | <i>12</i> |

1 Introduction

The Materials Science code Castep [1] performs quantum mechanical (density functional) calculations of bulk materials, surfaces and interacting molecules. Castep can give information about total energies, forces and stresses on an atomic system, as well as calculating optimum geometries, band structures, optical spectra and much more. It can also perform molecular dynamics simulations. Castep is distributed commercially by Accelrys Plc [1]. Academic licences (free to UK Academics) are distributed by DL staff (Dr A Wander, Mr D R Jones) on behalf of the UKCP consortium [1] who may be contacted at ukcp@dl.ac.uk. Dr Wander and Mr Jones also control permissions to use Castep executables on HPCx and the CSAR service. There is also a Castep discussion group [1].

Castep was originally created by Prof. M.C. Payne but since January 2003 the official version of Castep has been a completely new modular code written, developed and maintained by members of the Castep Developers Group (CDG) [2]. While HPCx has been in service, the supported version of Castep has changed from version 2.0 through versions 3.0-3.02 to version 3.1 to match the official academic releases, although with additional optimization features to improve HPCx performance. HPCx staff, particularly Dr M Plummer, work with the CDG, represented within CCLRC Materials Science by Dr K Refson, to optimize the code. Dr Refson has also optimized the code to run competitively on the CSAR SGI Altix 3700/1300 machine *newton* [1].

To understand the objectives and problems associated with maximizing Castep performance on HPCx, it is useful to remind ourselves of 'Amdahl's law' [3]. In its simplest form, we state:

$$T(N) = T(1) ((1 - \beta) + \beta / N)$$

for execution time T , N processors and 'parallel fraction' of the code β (ie the fraction of the code which is parallelizable). However this expression ignores overheads introduced by the parallel communications which reduce parallel performance and can make execution time increase with N beyond a large enough value. It also ignores memory accessibility issues. If the parallelization strategy includes data distribution (as it ideally should) then more processors implies smaller blocks of data per processor which for cache-based systems means much faster execution if the blocks of data being worked on fit into cache. The equation also assumes perfect 'load-balancing', ie all processors work equally and in synchronization so that no processors are idle while other processors work.

Castep expands electronic wave-functions in plane waves with many associated transformations between real and reciprocal space during a minimization of the electronic energy. The code uses three-dimensional Fast Fourier Transforms (3d-FFTs) of the 'g-vectors' on a distributed grid, performed using MPI_AllToAllV combined with serial FFTs of the distributed data [2]. These electronic minimization calculations occur in all the various scientific applications of the code. Castep uses a 'column' distribution of the grid which maximizes load-balancing throughout the program, resulting in two MPI_AllToAllV calls per 3d-FFT [2].

Castep also has a higher level parallelism in which processors are first distributed among Brillouin zone sampling k-points. The 3d-FFT may then be distributed among processors associated with a particular k-point. On 'fast processor with relatively slow communication' machines such as HPCx, k-point parallelism should first be performed whenever possible as the intensity of communications is much lower than for the g-vector parallelism. Both distributions have a high parallel fraction but g-vector parallelism has a high communications overhead. Following k-point parallelization the subsequent g-vector parallelism then involves fewer processors and hence lower communications overhead than 'pure' g-vector parallelism. Pure g-vector parallelism places much higher demands on the bandwidth (communication speed) and latency (individual communication start-up time) of the network. Thus the simple version of Amdahl's law above becomes more accurate for the mixed k-point and g-vector parallelization. More detailed studies of latency and bandwidth on HPCx may be found in several Technical Reports [4].

Although the Castep 3d-FFT distribution maximizes load-balancing, it has an apparent disadvantage on HPCx compared with other distributions as collective communications, in particular `MPI_AllToAllV`, have a heavy communications overhead. The quantum mechanical molecular dynamics simulation code CPMD [5], for example, uses a 'plane' distribution of the grid which requires one `all_to_all` communication per 3d-FFT, compared to two `MPI_AllToAllV` calls per 3d-FFT in Castep. With collective communication a major HPCx bottleneck, we may expect Castep to perform poorly in the 'terascaling' ~1000 processor regime for problems with few k-points. Thus, as well as general optimization of the code, optimization of the column 3d-FFT and hence `MPI_AllToAllV` becomes a priority, especially given the frequency of the 3d-FFT calls. In the next section we summarize how this problem has been dealt with on HPCx. We then consider other useful optimizations, present some data on current performance levels and a guide to users on how to find the ideal number of processors for a given job. Finally we preview ongoing optimizations and code changes which will be reported on more fully in the near future.

2 The treatment of `MPI_AllToAllV`

HPCx is of course a clustered SMP (shared memory processing or programming) system: for details of the HPCx Phase 2 IBM p690+ system with 32-processor SMP logical partitions or LPARs and how these are made up of four 8-processor 'MCM units' see, for example, [6]. These references also give details of the L1, L2 and L3 cache and main memory structure and sharing on the MCMs and LPARs. Note that as a further complication to Amdahl's law as quoted above, for clustered SMP units the reference number of processors used for scaling comparisons should take account of how basic memory is shared if we wish to tune or parameterize the equation for the specific system. On HPCx the reference number should be at least $T(2)$ rather than $T(1)$ as two cpus share L2 cache. Arguably for massively parallel scaling comparisons the reference level could be, for example, $T(32)$.

Within each LPAR we have fast communication either using shared-memory MPI (environment variable `MP_SHARED_MEMORY=yes`) or OpenMP. The CPMD code [5] has been carefully optimized as a mixed mode code for IBM architecture. Castep

is a pure MPI code and thus we must optimize MPI_AlltoAllV for the clustered SMP architecture. Substantial optimizations have been made that have improved performance of Castep on HPCx by ensuring that the communications involved in the g-vector data redistributions (the MPI_AlltoAllV calls surrounding the serial FFTs) are 'SMP-aware'. This was originally done for the HPCx Phase 1 system as described in an earlier Technical Report [7]. Processors within an LPAR are divided into groups and the MPI_AlltoAllV is transformed into an MPI_GatherV within a group, a rearrangement followed by MPI_AlltoAllV between group 'leaders' and finally an MPI_ScatterV within each group. The size of the group is a factor of the number of processors in an LPAR and is chosen dynamically to maximize performance as discussed in [7] with a competition between reducing the number of leader processors and maintaining a 'grand' message size generally within the buffered memory 'eager limit' [7] as Castep 3d-FFT grids are usually of moderately large rather than very large size while the 3d-FFT is performed many times. Technical Report HPCxTR0409 [8] describes a different possible SMP-aware breakdown of MPI_AlltoAllV, not specifically tuned and optimized for Castep, which may be implemented in other codes.

The HPCx Phase 2 system came into full user service on 29 April 2004. The Phase 2 upgrade (effectively a completely new system) doubled the overall capability of HPCx with a larger number of more powerful processors and the introduction of IBM's new High Performance Switch (HPS, formerly known as 'Federation') together with appropriate new operating system software. HPS has greatly increased bandwidth and reduced message latency compared to the old 'Colony' switch of HPCx Phase1 [9].

Following the introduction of the HPS switch there have been important upgrades to the HPS software. The first took place on 28 July 2004 as part of an upgrade to AIX known as Service Pack 7 (SP7) and delivered significant improvements to the point-to-point message latency and the message transfer times, especially in the mid-range of message lengths (around 64kB). The second upgrade with Service Pack 9 (SP9) had less effect on MPI performance but contained a significant performance improvement to some LAPI functions, affecting those codes which call LAPI directly, for example via the Global Array tools. SP9 and succeeding upgrades also provide important safety features in the event of systems failures which general users should ideally never actually knowingly encounter as well as other background features. Full details are available in [9] including references for all the technical terms above

A more detailed discussion of the customized Castep MPI_AlltoAllV for general SMP cluster systems will be available in the near future [10]. Reference [10] will also describe how Dr Plummer and Dr Refson have studied alternative successful optimizations of MPI_AlltoAllV for large SMP units such as *newton*. Despite the HPS and associated software upgrades, communication within LPARs is still noticeably faster than between LPARs. The algorithmic optimization is more complicated for HPCx Phase 2 than for HPCx Phase 1. HPCx Phase 2 replaced 8-way LPARs with 32-way LPARs. However each 32-way LPAR is physically constructed from 4 8-way units (MCMs) resulting in heterogeneous inner structure and 'links' between 32-way LPARs. This meant that the LPAR customization algorithm needed to be modified so that the group leader communications were evenly distributed among the links. Castep benchmark timings, initially slower on Phase2 than on Phase1 due to link 'traffic jams', were greatly improved.

The IBM software operating within LPARs which controls memory and process binding (currently `MP_TASK_AFFINITY=MCM` and `MEMORY_AFFINITY=MCM`) added to the performance gain achieved by our local updated LPAR-aware algorithm such that many of the benefits that were expected to occur only following the SP7 upgrade were in fact achieved during the initial Phase2 operating period. The Phase2 customized LPAR-aware code remains much faster than the original code for `MPI_AllToAllV` across several LPARs even after the system software upgrades. The examples in section 4 show that the combination of code optimization and system software and environment upgrades results in Castep 3d-FFTs scaling reasonably to 64 processors and bearably (or better) to 128 processors depending on the size of the problem.

3 General optimization of Castep

With the `MPI_AllToAllV` ‘under control’ we may now consider other ways of improving performance and scalability. Calculations with many k-points will scale well as described above. We will mention a further high level form of parallelization under development in section 6, but for the moment scalability will improve if (a) other MPI operations are speeded up and (b) the serial fraction of the code is reduced. Enhanced performance (and thus reduction) of the parallel fraction of the code is welcome as a general improvement but will result in poorer scaling without associated reduction of the communications overhead. General performance will also be improved by memory optimization.

The most important general optimization for Castep is the 1D-FFT routine. Castep’s provided FFT routine has been carefully optimized to take account of the L1-cache structure of HPCx (and can be and has been so optimized for other cache-based systems [11]). It is at least as fast as, for example, FFTW [12]. The routine can also be optimized for vector machines [11]. This optimization has greatly improved general performance but as the 1D-FFT is part of the parallel fraction it does not improve scaling.

As with other applications codes on HPCx, performance analysis tools are useful here: for Castep the tools **gprof**, **hmdlib** and **mpitrace** have proved useful. We note that due to the high level of communication that takes place in Castep, the visual analysis tool **vampir** [13] is not as useful as the other tools in this case.

gprof [13] shows cpu time spent in each subroutine (among other useful data) for each processor separately. Apart from highlighting ‘bottleneck’ subroutines, it can also highlight whether routines are in the serial or parallel fraction of the code via comparison of output files for test runs with different numbers of processors. For Castep, two serial-fraction routines were highlighted this way and subsequently optimized by a mixture of do-loop reordering and more use of BLAS routines. One of these, part of the electronic minimization part of the code involving function orthogonalization, was almost halved in execution time. The other, not involved in the electronic minimization but in subsequent population analysis, has had its execution time reduced to about 0.4 of its original value. The **gprof** output is

currently dominated by lower-level (for example 'move' and 'copy') routines rather than user-coded routines.

mpitrace [13] provides a detailed analysis of time spent in MPI routines. For Castep, MPI_AllToAllV is dominant (for the customized code, the optimized MPI_AllToAllV, MPI_GatherV, MPI_ScatterV combination, with MPI_AllToAllV still dominant within this triumvirate) with MPI_AllReduce in a clear second place. Tests on MPI_AllReduce show that the IBM implementation behaves in an LPAR-aware manner. Ongoing work on Castep is looking at ways of minimizing the occurrence of MPI_AllReduce and as far as possible restricting messages to an optimum length (to be reported on in the future). Finally we note that the **mpitrace** profile of Castep with MPI_AllToAllV and MPI_AllReduce clearly dominant over other MPI routines will be noticeably different from that of CPMD, with its plane 3d-FFT decomposition as described above. Similarly the **mpitrace** profile shown for the plane-wave materials code VASP in HPCx Technical Report TR0414 [14] shows a different 3d-FFT procedure with MPI_AllToAll dominant followed by MPI_AllToAllV and MPI_Bcast then MPI_AllReduce and MPI_Recv.

hmdlib [13] allows optimizers to be aware of overall allocated memory between user-chosen points in the code. It is very useful for detecting memory leaks (ie memory which is continuously needlessly allocated) over loops of code and for detecting large allocations of memory, for example for unparallelized arrays that seem harmless in small test jobs but become too big for large scale jobs submitted for massively parallel runs. These arrays may then be modified or considered for distribution over processors. Since HPCx runs much faster if data for particular operations is kept in cache, **hmdlib** is an effective tool for examining and improving larger calculations. **hmdlib** data has been used to discuss various options for general memory optimization of basic coding of Castep with the Developers' Group and has been useful for various optimizations for HPCx for the supported executables in /usr/local/packages/castep/ including, for example, avoiding the creation of over-large MPI buffers in fundamental routines. Examples of specific tasks optimized following memory analysis include the calculation of stress following an electronic minimization and the read-in of data on restart. As Castep has many diverse functional features, particular memory problems for large jobs are brought to our attention by users with specific concerns and **hmdlib** is a very useful tool in solving these problems.

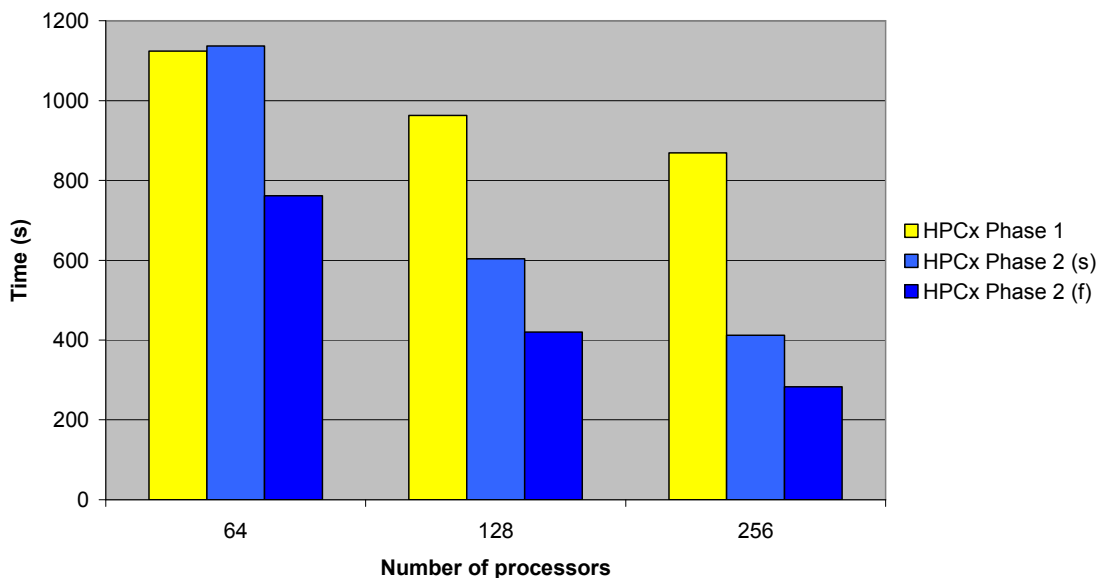
An extremely useful way of optimizing code is for the optimizer to work closely with the code developers. Apart from the collaboration between Dr Plummer (HPCx) and Dr Refson (CDG) which gave rise to the LPAR-customized MPI_AllToAllV, one further important optimization for electronic energy minimization of metals (or systems that may be classified as metals for Castep purposes) using the 'density mixing' method [2] has been made by modifying an algorithm in the minimization process. Careful consultation between Dr Plummer and the CDG (Dr P Hasnip and Dr Refson) ensured that this modification does not alter the physics and accuracy of the minimization process but does provide a noticeable performance improvement in most cases. Since the modification is algorithmic it requires knowledge of what the program is trying to do, which needs the optimizer either to be a developer or work closely with developers. The first job of Castep is of course to produce results that are accurate and reliable using carefully implemented algorithms and methods: performance 'improvements' must not compromise this.

The HPCx `/usr/local/packages/castep/` executable `castep_31f` includes this modification while executable `castep_31s` does not ('f' and 's' stand for 'fast' and 'standard' respectively: advice on which executable to use is given in section 5). Note that although the modification is optimizing the parallel fraction of the code, it does so in a way that should remove a large quantity of high-intensity communications overhead and thus improve performance without harming scaling. As a postscript to this HPCx modification, Dr Hasnip has incorporated a flexible modification based on similar ideas into the current development version of Castep which will appear as an official release in the near future.

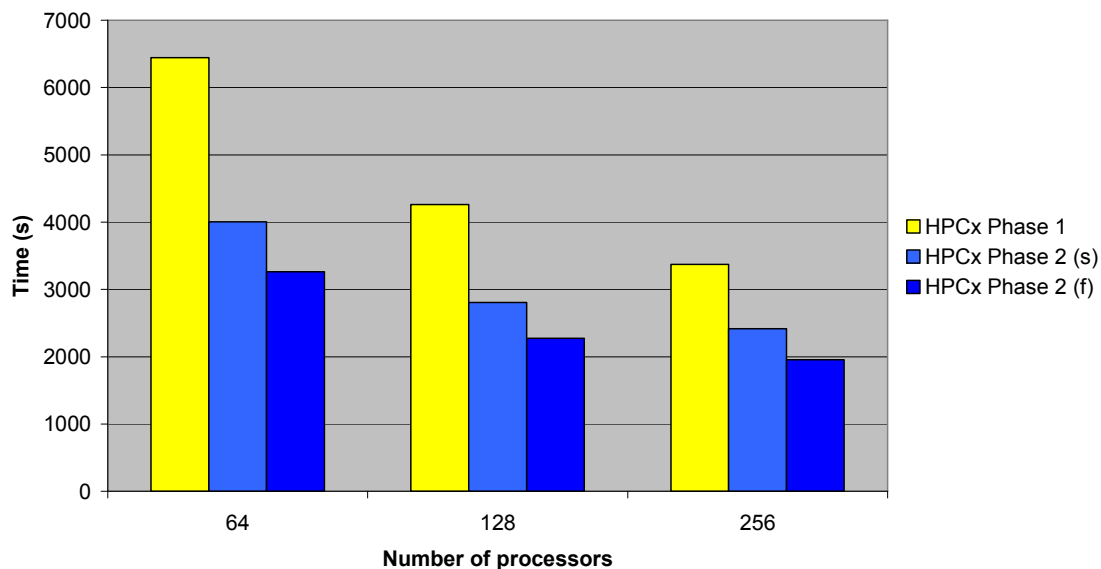
4 Performance examples

We present electronic energy minimization examples for two systems. Case TiN is of a 33 atom cell of titanium nitride with a hydrogen defect and has 8 k-points. Cases Al_2O_3 are of a 120 atom slab cell of aluminium oxide with 5 k-points. However while case $\text{Al}_2\text{O}_3(\text{kG})$ uses the full k-point plus 3d-FFT parallelism of Castep, case $\text{Al}_2\text{O}_3(\text{G})$ does not take advantage of the k-point parallelism and the 3d-FFT (and hence `MPI_AllToAllV`) is split over all processors. Thus this benchmark is designed to show the limiting performance of the 3d-FFT for large numbers of processors. Figures 1 and 2 show timings for cases TiN and $\text{Al}_2\text{O}_3(\text{G})$ for 64, 128 and 256 processors only, however note that in case TiN the 3d-FFT is split over 8, 16, and 32 processors respectively.

Figure 1: TiN

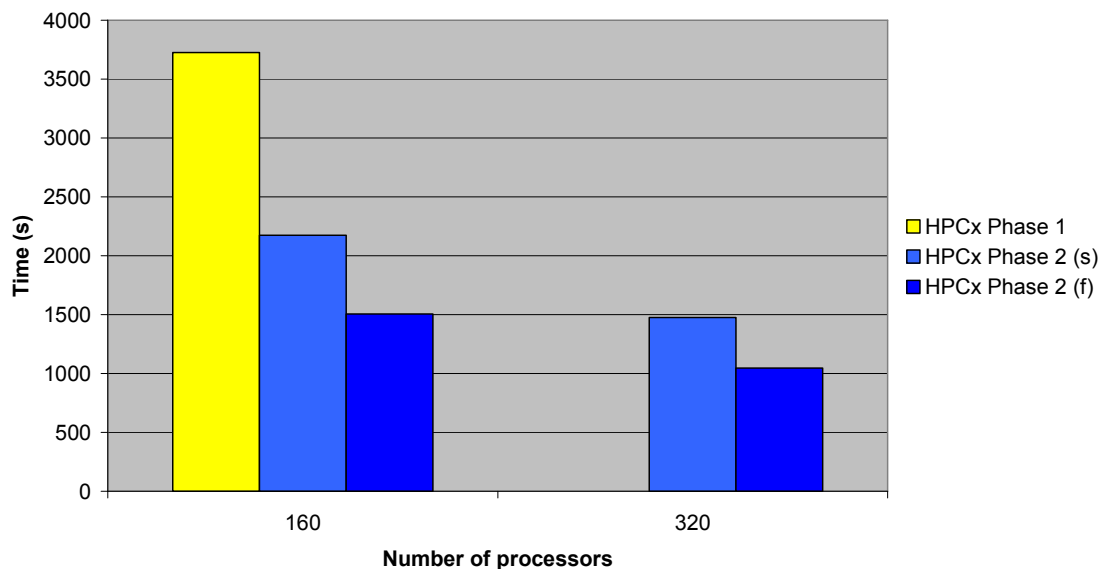


The results shown are the best results obtained on HPCx Phase 1 and results for HPCx Phase 2 using the executables `castep_31s` and `castep_31f`. Both these executables have the customized `MPI_AllToAllV`, but `castep_31s` does not have all the algorithmic optimizations as described in section 3.

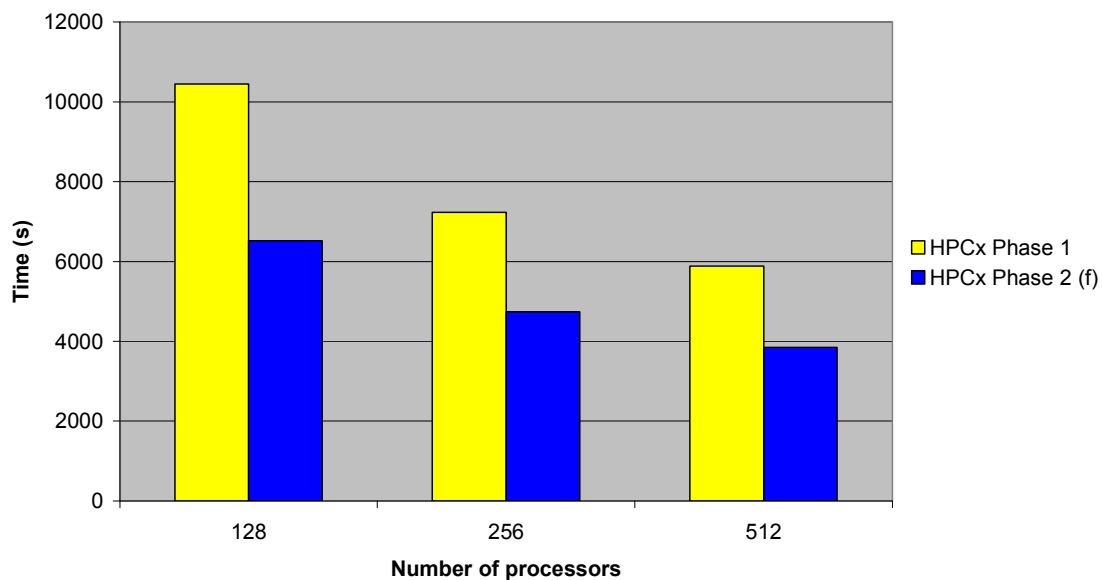
Figure 2: $\text{Al}_2\text{O}_3(\text{G})$ 

Note that on HPCx Phase 2 case TiN shows excellent scaling as may be expected as the 3d-FFT is contained within an LPAR. Case $\text{Al}_2\text{O}_3(\text{G})$ shows that for this size of problem (120 atoms) the 3d-FFT scales reasonably up to a limit of around 128 processors.

Figure 3 shows results for case $\text{Al}_2\text{O}_3(\text{kG})$. This case has been converged to much greater accuracy than case $\text{Al}_2\text{O}_3(\text{G})$ and has performed twice as many self-consistent-field single-point-energy (SCF-SPE) iterations, namely 24 cf 12. We present results for this case for two reasons, firstly to confirm that Castep can scale reasonably to large numbers of processors for appropriate calculations and secondly to show the importance of taking full advantage of the k-point parallelism. The number of processors and also the number of LPARs are multiples of the number of k-points.

Figure 3: $\text{Al}_2\text{O}_3(\text{kG})$ 

Finally, in figure 4 we present castep_31f results for a larger case, again Al_2O_3 but now using a 270 atom slab giving rise to two k-points. Here the larger size of the problem means that fair scaling up to 256 processors is possible, the 3d-FFT scaling to 128 processors.

Figure 4: Al_2O_3 (large)

5 How to maximize performance of HPCx Castep jobs

Details of how to run Castep on HPCx are provided in the file README in directory /usr/local/packages/castep, which is available to licensed users. This directory also contains the HPCx-optimized executables castep_31f and castep_31s plus a basic unoptimized executable castep_31exe which may be used as a back-up executable in case of unforeseen problems with the recommended executables. Please report any such problems to helpdesk@hpcx.ac.uk. As described above, Castep has the ability to parallelize over both electronic k-points and g-vectors and a combination of the two. k-point parallelism should be given priority whenever possible because g-vector parallelism places much higher demands on the bandwidth and latency of the network. Using 32 processors per k-point means that up to 32-way g-vector parallelism is accommodated within a single LPAR, which will give high performance.

Even 32-way g-vector parallelism may become communications-bound (ie not give a great speedup) if the number of plane-waves is small, ie a small cell, number of electrons and/or low cutoff energy. In this case {16, 8, 4} processors per k-point may give better processor usage. On the other hand, if your job is large or you have few k-points then you should experiment with 32, 64 and 128 (optimistically 256) processors per k-point. Depending on the size of your calculation/cell 64 processors per k-point should scale reasonably, 128 probably less-reasonably compared to 32 processors per k-point. HPCx staff and the Castep Developers' Group are continuing to look at ways to improve scaling.

The /usr/local/packages/castep directory also includes a file SAMPLE_SCRIPT which is available for you to adapt. The recommended environment parameters are currently:

```
#@ shell = /bin/ksh
#@ job_type = parallel
#@ network.MPI = csss,shared,US
#@ output = out.%(jobid).txt
#@ error = err.%(jobid).txt
#@ wall_clock_limit = 06:00:00
#@ tasks_per_node = 32
#@ cpus = 128
#@ account_no = <your account>
#@ bulkxfer = yes
#@ notification = never
#@ node_usage = not_shared
#@ queue

pwd
echo $LOAD_PROCESSOR_LIST
export MP_SHARED_MEMORY=yes
export MP_EAGER_LIMIT=65536
export MP_POLLING_INTERVAL=2000000000
export MP_SINGLE_THREAD=yes
export MEMORY_AFFINITY=MCM
export MP_USE_BULK_XFER=yes
export MP_TASK_AFFINITY=MCM
```

Note that you specify '\$@ cpus = <>', the total number of required processors. 'tasks_per_node' has a default setting of 32 and is thus optional unless you require a lower value. This will be the case for small jobs for which 32 processors per k-point

is communications bound, in order to have k-points spread evenly over LPARs. It will also be the case for very large jobs for which 32 processors per LPAR does not provide enough memory per processor: available memory per processor is currently set as 26.96GByte / 'tasks_per_node'.

To get maximum performance from the LPAR customization of MPI_AllToAllV described below, the process and memory binding environment is required (MP_TASK_AFFINITY, MP_MEMORY_AFFINITY). MP_EAGER_LIMIT is set to its maximum to make full use of the LPAR-customization [7,10]. '#@ bulkxfer = yes' enables the relatively new RDMA environment introduced in service pack upgrades SP12 and SP13 which will speed up very long messages [15]. It will most probably only affect the largest Castep jobs at present but does no harm to smaller jobs.

We recommend that your <job>.param input file includes the line 'opt_strategy = speed' to keep all operations in memory rather than reading and writing temporary scratch files, again to maximize performance. In the rare cases where this is not possible (ie you run out of memory even with reduced 'tasks_per_node') you should define your own \$TMPDIR as the default /tmp space is very limited and ideally restricted to systems files. Please submit a query to helpdesk@hpcx.ac.uk if you have any problems of this nature and are unsure what to do. Finally we also recommend that your '<job>.param' input file includes the line 'num_dump_cycles = 0' if you do not require '<job>.<nnnn>.wvfn' files to be dumped to your working directory.

Feedback on use of castep_31f executable is welcome: it is generally noticeably faster than castep_31s except possibly for simple systems for which the SCF calculation normally converges in, say, fewer than 8 iterations. Please report any problems with castep_31f and try castep_31s in the unlikely instance of SCF convergence problems. Further details of Accelrys Materials Studio [1] compatibility, available 'pseudopotentials' and other, older executables from Castep 3.02 are given in the README file.

6 The near future: upcoming terascaling features

We have demonstrated that despite the high communications intensity and hence overhead of the 3d-FFT, Castep will scale to reasonable numbers of processors and in some cases to large numbers depending on the details of the job. For example, recent successful studies of gold chains grown on copper surfaces have efficiently used 512 (or more) processors [16]. The performance gains at this level are due to a mixture of careful code optimization and improvements in IBM communications management (hardware and software). The Castep LPAR-customized MPI_AllToAllV is flexible enough to cope with future IBM upgrades and should only need to be fine-tuned when the next Phase of HPCx replaces the current Phase 2. Similarly, if the current LPARs are replaced with, for example, LPARs based on IBM Power5 technology [6], this should improve Castep performance as L3 cache is much more closely linked to processors in the Power5 design. Early testing is showing that a forthcoming compiler upgrade (xlf v9 rather than the current xlf v8) will produce a modest but noticeable improvement in both performance and scaling for larger jobs. One other IBM development currently being explored is the use of the RDMA environment [15] which speeds up very large messages. Most Castep 3d-FFT message sizes are not large enough to take advantage of this and we

concentrate on keeping message size within the 'fast-buffered' regime as noted in section 2 and [7,10], However for very large cells and grids this environment could become advantageous.

Apart from the ongoing gradual improvement in HPCx Castep optimization there are at least two major developments in Castep that will improve performance and scaling for large numbers of processors, aside from ongoing new scientific improvements and functionality which are beyond the scope of this report. Firstly, a near-future release of Castep will include improved code to treat 'gamma-point' calculations, ie calculations with one particular 'zero-coordinate' k-point [2] with a more efficient use of the 3d-FFT grid than is currently programmed. The modular philosophy of Castep allows this treatment to be chosen automatically by the program when it detects the calculation is of 'gamma-point' type. The modular philosophy of Castep makes sure that high-level scientific 'functional' modules of interest to users are clearly written with transparent (approaching object-oriented) interfaces to the lower-level 'fundamental' and 'utility' modules of interest to developers and optimizers. This optimization should greatly reduce the number of 3d-FFTs required for the real-space/reciprocal space transformations and thus improve scaling for this type of calculation.

The second development relates to our earlier discussion of communication overheads affecting Amdahl's law and the differences between k-point and g-vector parallelism. The ideal terascaling code is one for which the communication is essential but minimal as minimal communication implies minimal communication overhead. On the other hand zero communication, though allowing a parallel fraction of 1 if each processor performs a different calculation, is not a good use of the machine unless all the final data is absolutely needed simultaneously. The work could be performed much less expensively on serial machines, or local clusters if the 'terascale' job is actually a set of, for example, 32-processor jobs that do not communicate with each other. If, however, a calculation requires a group of 'similar but different' sub-calculations and operates relatively infrequently on the output data of these to set off new sub-calculations or produce global output, we have a useful terascaling calculation. This is the difference between 'intelligent' task-farming which includes feedback and makes use of sub-calculations and non-intelligent task-farming which disguises several few-processor calculations as one terascale calculation. Provided the sub-tasks involve similar amounts of work in order to maintain load-balancing, intelligent task-farming calculations can scale extremely well.

Dr M Probert and colleagues [17] have introduced new intelligent task-farming MPI communicators into the lower-level modules of Castep in a way which does not disturb the existing k-point and g-vector parallelization. So far these options are restricted to certain quantum molecular dynamics calculations in the high-level 'functional' modules, but in principle these new routines may be accessed by various high-level scientific routines. Work is currently underway to introduce further applications of this new underlying structure which is of particular interest for possible terascaling 'capability' use on HPCx.

Acknowledgements

We would like to thank Dr A Wander, Dr M Probert and Dr P Hasnip for useful discussions. Computer time on HPCx was partly funded through the UKCP consortium.

References

- [1]. Please see the websites: <http://www.castep.org/>,
http://www.accelrys.com/mstudio/ms_modeling/castep.html,
<http://www.cse.clrc.ac.uk/cmgnetworks/ukcp/>, <http://www.jiscmail.ac.uk/lists/CASTEP.html>,
<http://www.csar.cfs.ac.uk>
- [2]. *First-principles simulation: ideas, illustrations and the CASTEP code*, M.D. Segall, P.J.D. Lindan, M.J. Probert, C.J. Pickard, P.J. Hasnip, S.J. Clark and M.C. Payne, *J. Phys. Condensed Matter* 14 (2002) 2117 (the authors together with K Refson represent the Castep Developers Group); M C Payne, M P Teter, D C Allan, T A Arias and J D Joannopoulos, *Rev. Mod. Phys.* 64 (1992) 1045 is the basic theory reference for Castep.
- [3]. G Amdahl, *AFIPS Conference Proceedings*, 30 (1967) 483-485. Amdahl's law was originally devised to describe vector processors and has been adapted for parallel processors.
- [4]. <http://www.hpcx.ac.uk/research/hpc/>
- [5]. <http://www.cpmid.org>; J Hutter and A Curioni, *Dual-level Parallelism for ab-initio Molecular Dynamics: reaching Teraflop Performance with the CPMD code*, IBM Research Report RZ 3503 (2003), also submitted to Parallel Computing.
- [6]. <http://www.hpcx.ac.uk/services/hardware/index.html>,
<http://www.top500.org/ORSC/2003/power4.html>, <http://www.top500.org/ORSC/2004/power4.html>,
<http://www.top500.org/ORSC/2004/power5.html>
- [7]. M Plummer and K Refson, HPCx Technical Report HPCxTR0401,
http://www.hpcx.ac.uk/research/hpc/technical_reports/HPCxTR0401.pdf
- [8]. A Jackson and S Booth, HPCx Technical Report HPCxTR0409,
http://www.hpcx.ac.uk/research/hpc/technical_reports/HPCxTR0409.pdf
- [9]. M Ashworth, I J Bush, M F Guest, M Plummer, A G Sunderland and J Hein,
HPCx Technical Report HPCxTR0417,
http://www.hpcx.ac.uk/research/hpc/technical_reports/HPCxTR0417.pdf
- [10]. M Plummer and K Refson, *Comput. Phys. Commun.* *to be submitted*. For a brief overview of the *newton* work please see <http://www.cse.clrc.ac.uk/arc/castep.shtml>
- [11]. Similar L1-cache optimizations of this type were introduced into Castep by Dr B Jesson for the (now defunct) Cray T3E service at CSAR (<http://www.csar.cfs.ac.uk>) and by Dr M Plummer to, for example, the SGI Origin systems at CSAR. Without these optimizations the routine is designed to perform well on vector processors. This optimization is not appropriate for *newton* as there is no L1 floating point data cache. Castep on *newton* employs a library FFT routine.

-
- [12]. <http://www.fft.w.org/>
- [13]. <http://www.hpcx.ac.uk/support/documentation/UserGuide/HPCxuser/Tools.html>
- [14]. G J Pringle, HPCx Technical Report HPCxTR0414,
http://www.hpcx.ac.uk/research/hpc/technical_reports/HPCxTR0414.pdf
- [15]. A Gray, J Hein and S Booth, HPCx Technical Report HPCxTR0505,
http://www.hpcx.ac.uk/research/hpc/technical_reports/HPCxTR0505.pdf
- [16]. G Kyriakou, F J Williams, M S Tikhov, A Wander and R M Lambert, Phys. Rev. B *submitted*
- [17]. M Probert (University of York), *private communication*