

# The View from the High End Fortran, Parallelism and the HECToR Service

Ian J. Bush  
NAG Ltd.  
Wilkinson House  
Jordan Hill Road  
Oxford OX2 8DR

## Introduction

Since its inception in 1956 Fortran, and indeed FORTRAN, has been the computational language of science and engineering. Through Fortran aeroplanes fly, drugs are designed and nuclear reactors react. But though codes from the original manual still look familiar [1] Fortran itself has changed much over the last half century and more. So how is it currently being used on modern architectures, particularly at the high end? And how has the recent (in Fortran terms!) move from serial architectures to parallel ones affected practice? And indeed how has the evolution of those parallel platforms affected practice?

In this article I shall give a brief and somewhat personal view of current programming practice on HECToR [2], the current UK National Academic Supercomputer service, and how application development is supported. In the next section I shall cover the hardware and software environment on the machine. Following that I shall cover how computational science and engineering (CSE) support is provided, and finally I shall discuss, with examples, how people are exploiting such machines.

## The HECToR Hardware and Software Environment

HECToR, an acronym for High-End Computing Terascale Resource, is the current UK national academic supercomputing resource. It is available for use primarily by UK academics and their collaborators, though there is a small amount of overseas use, for instance via the DEISA project [3]. The service started in 2007, and is funded by the UK research councils [4]. It is provided by a number of partners, EPSRC, Cray Inc, the Numerical Algorithms Group Ltd (NAG), UoE HPCx Ltd STFC Daresbury Laboratory and EPCC, each of which are responsible for different areas of the service; for more details see [5].

Currently the service is being upgraded, and the hardware consists of 2 main components. Both of these are typical modern HPC architectures in the sense that they are based upon commodity processors connected by a high speed communications network. The two components are:

- The phase2a component: This is a Cray XT4 machine, and is based upon quad core nodes. These contain 2.3 GHz AMD opterons. Each node has 8 Gbytes of memory. There are 3072 nodes, making a total of 12,288 cores and 24.5 Tbytes of memory. Associated with each node is a Cray SeaStar2 chip router. This has 6 links which are used to implement a 3D-torus of processors. The point-to-point bandwidth is 2.17 GB/s, and the minimum bi-section bandwidth is 4.1 TB/s. The latency between two nodes is around 6 $\mu$ s.
- The phase 2b component: The newer machine is a Cray XT6. This contains 1856 nodes. Each node consists of two 12 core 2.1 GHz AMD opteron processors. Each node has 32 Gbytes of memory. Thus the total size of the new machine is 44,544 cores and 59.4 Tbytes of memory. The interconnect is currently still based on Cray's SeaStar 2 technology, and so is very similar to that found in the phase2a machine. However this will be upgraded in the fairly near future to Cray's new Gemini interconnect.

Currently the phase2b machine is number 16 in the top500 list, while the phase 2a machine is number 26[6].

The software on both machines is similar. They both run versions of Cray's compute node Linux, version 2.2 on phase2a and 3.0 on phase2b. Both have the full complement of scientific libraries that one would expect on such a national resource, and, unusually for a national service in the UK, users have access to 5 Fortran compilers, namely those from Portland Group (the default), Pathscale, gfortran, Cray, and NAG.

## NAG and CSE Support for the HECToR Service

Though HECToR itself is sited in Edinburgh, the CSE support is provided by NAG Ltd.[7], which has UK offices in Oxford and Manchester. NAG's collaboration in the HECToR project began in 2003 by delivering HPC technology advice and HPC market intelligence to the project. Subsequently, NAG was chosen to design and undertake the procurement benchmarking process, which played a significant part in the supercomputer procurement process. Finally NAG won the competitive tender process for the CSE support portion of the service.

So for those who don't know, what, and who, are NAG? Well NAG is a not for profit company that only does numerical software engineering, either consulting or software products. Around two-thirds of the 90 staff members are active software engineers, with roughly 25 of those active HPC experts. NAG's strength is the breadth and depth of HPC expertise available, with staff possessing experience as end-users in academia, defence and other industries, as national HPC service providers, and in procurement support and strategic advice. That expertise spans highly scalable parallel programming for multi-thousand node supercomputers, robust software engineering for high-volume products, software testing and validation, documentation and training. NAG has a good coverage of common programming languages and platforms, and much of the development is performed in Fortran. Many staff members have international recognition for their contributions in HPC, algorithms, or mathematics. Key members of NAG staff are well linked in the international HPC community, with strong senior business and technical contacts with all major HPC vendors and technology providers, and with many research groups and user organisations.

NAG's best known products are probably its libraries. These include both the NAG library itself, and also NAG's collaboration with AMD in producing the ACML library. However this is not all that NAG provides; it produces and distributes numerical software for the solution of problems in a wide range of applications in such areas as science, engineering, financial analysis and research. Further, as mentioned above, NAG also provides consulting services to provide tailored solutions to meet users particular needs. For more information see [8][9][10].

That's NAG, but how does the CSE support service for HECToR work? Well NAG's HECToR CSE combines traditional helpdesk based support and training with a "distributed CSE" service (dCSE)xi, which dedicates significant amounts of effort onto key user applications.

CSE support (or HPC software engineering) is available to all HECToR users. Short term work is covered by a standard helpdesk approach, while longer term dedicated support is provided by the dCSE mechanism mentioned above. This support will typically be for periods of between six months and a year, and is intended to help users improve the performance, scalability and functionality of their codes [12] on HECToR. Regular calls for proposals are published, and whether a project is funded or not is decided by an independent panel appointed by the research councils. In total, the HECToR CSE service provides around 20 full-time equivalent people per year, and the person performing the work will more often than not be sited at the institution of the research group rather than at one of the NAG offices, so allowing full interaction with the group. Many of the results given later in this article are from dCSE projects.

NAG also provides a range of HPC training courses [13] to help users of HECToR get the most out of the system. The HPC training programme covers topics ranging from getting started on HECToR to improving parallel application performance and scalability on distributed memory machines, and includes a course on Fortran given by the author. Although some of the courses are aimed mainly at HECToR users, many, such as those on MPI, OpenMP and Fortran, are of interest to a wider community. Training courses are held at universities throughout the UK, and attendance is free for HECToR users and for any other researchers sponsored by one of the research councils EPSRC, NERC or BBSRC. The courses have proved to be popular, and in September 2010 alone NAG staff trained over 100 people.

## HECToR in Use

So that's the HECToR service and its CSE support, a fairly standard example of modern high end hardware backed up by a slightly different support service. But how are the users exploiting it? And, of particular relevance here, how are they programming it? Well to date the top 5 codes in terms of usage on the Phase2a machine are VASP [14], CASTEP [15][16], The Unified Model [17], CP2K [18] and HELIUM [19]. While these address 3 different scientific areas, namely condensed matter physics, weather forecasting and climate prediction, and atomic and molecular physics, in terms of programming they have much in common. They are all written in Fortran. Of those that I have access to the source code in all cases they are written in a "modern" post Fortran90 style, and use many of the features introduced in that standard, yet, with the exception of TR15581 (Allocatable Array Extensions [20]), little after Fortran95. And for

high end computing all generally use message passing with MPI to exploit the high degree of parallelism available on modern high end architectures (CP2K also has some mixed OpenMP/MPI capabilities). So apparently little has changed over the last decade or so, high end computing is still about Fortran and message passing. Why is that? And is it really the case that nothing has changed?

Well actually quite a lot has changed, both with regard to how Fortran is used, and how parallelism is exploited. Though the main tools have not changed yet, how they are exploited has become increasingly sophisticated and areas of the Fortran language or the MPI library that were rarely used in large codes in the 1990s are commonplace now on HECToR. And it also true that the tools used have begun to change. Interest in exploiting mixed OpenMP and MPI is becoming much more common, driven by the multicore revolution, but that is not all that is changing. A number of codes are now exploiting the shared memory features offered by POSIX either to reduce their memory footprint, or to exploit rapid intranode communications via shared memory, or even to better exploit the internode communication network by maximising message length. Also, and especially relevant to the Fortran world, are coarrays and their potential use in HPC. I shall briefly discuss all these points, and give illustrations from codes currently used on the HECToR service.

A good example of the increasingly sophisticated use of Fortran is CASTEP, which is a software package that uses density functional theory to provide a good atomic-level description of all manner of materials and molecule. A history of the code can be found at [xxi](#). The important points so far as this article is that in 1999 for various technical reasons, it was felt that the original code was in need of a total re-design and re-write using modern coding styles and Fortran90. The design goals were that the new code should

- Have a well defined specification
- Be portable
- Be designed a priori to be parallel
- Have a clear, modular structure
- Have the same or better feature set as the old F77 code
- Have the same or better performance as the old F77 code

The nett result is a code which is more easily maintained, more portable and more easily extended than the older Fortran 77 based code. This is aided by specifying a hierarchy of modules, named functional, fundamental and utility with the latter being the lowest level. Typically the scientific programmer will use routines mostly from the functional level, for these implement the operations the condensed matter physicist “thinks” in, and so most naturally codes in. All the nasty, fiddly, distracting details are left to the lower level modules, so allowing the scientist to implement whatever he/she needs more naturally, and so more quickly.

Of course this is nothing new, it is simply good code design. What is new over the last decade or so is the realisation that the facilities Fortran 90 and later provide not only help in organising a large scale code, but also help in avoiding the slip back toward spaghetti that can occur once the original code is extended or modified. In CASTEP and other codes this has resulted in an almost object orientated approach; modules define opaque derived types and methods than can act on those types resulting in code that looks very natural (at least to the condensed matter physicist!):

```
! Apply kinetic energy operator to wvfn
call wave_kinetci_energy(wvfn,ek,H_wvfn)
! Calculate the charge density
call density_calculate(wvfn,occ,dens)
! Calculate the local potential
call locpot_calculate(dens,local_pot)
! Apply the local potential to the wave function
call pot_apply(local_pot,wvfn,wvfn_temp)
! Add Vlocpsi to kinetic energy contribution
call wave_add(wvfn_temp,H_wvfn)
! Apply the non-local potential to the wave function
call nlpot_apply(wvfn,...wvfn_temp)
! Add Vnl|psi> to get final H|psi>
call wave_add(wvfn_temp,h_wvfn)
```

Despite this move toward object orientation CASTEP uses only Fortran95 with TR15581, and none of the new features introduced in Fortran 2003. The reason is portability. As shown in xxii the object orientated programming features in Fortran 2003 are far from universally implemented, and there is not even a clear portable subset that could be adopted. CASTEP is not alone in this. More generally, with the exception of TR15581 and C interoperability, the lack of availability of Fortran 2003 has caused most large scale scientific applications to avoid using features in the language that post date Fortran 95. After all it is not the case any more that the national resource is the only machine the code will be run on. Almost all universities, and indeed many departments within universities, now have their own clusters, and the user will run the code as much, if not more, on those machines rather than HECToR. Thus portability is now a key concern of HPC application developers. However given the experience of the last decade or so with Fortran 95 one hopes that as Fortran 2003 becomes more widely available its features will be adopted by the community.

So on HECToR nowadays most applications use Fortran 95, possibly with TR15581. Admittedly CASTEP is in some ways an extreme example when it comes design and use of modern features, but most older codes are, in my experience, moving from a Fortran 77 (or older!) style to use of more modern features of the language. But what of the other traditional tool, message passing? Again the usage has changed, this time driven by the hardware that is available to computational scientists. In 2002 the HPCx service [23] service started. This was the predecessor to the current HECToR service, and was the first UK national service to offer over 1000 cores (it started with 1280), and was also the first UK national service to be based on multicore chips (IBM P690). Just 8 years later HECToR has roughly 35 times as many cores, and multicore is ubiquitous!

To best use such machines as HECToR applications have therefore to both exploit a much greater degree of parallelism than previously, in many cases simply scaling up the problem is impractical, and also the architecture of the machines, where shared memory can be exploited and intra-node communications are expected to be appreciably faster than inter-node. For both these reasons there is much greater use of hierarchical parallelism than previously, both within the algorithm and to exploit the hierarchy of memory accesses in multicore machines.

Simple examples of exploiting hierarchical parallelism in the algorithm are ensembles in weather forecasting and replica-exchange methods in molecular dynamics [24]. In both cases essentially independent calculations run, and then a statistical analysis is performed on the results of each run. So if an individual member of the ensemble only scales to a few tens of cores, using such methods allows the whole code to exploit hundreds or even thousands. However deeper hierarchies and more complex examples are reasonably common. For example the the CASTEP code described above can exploit a hierarchy 5 deep (from top to bottom atomic configuration, spin, k point, band and G vector), and similar in structure is CRYSTAL [25][26][27], another ab initio electronic structure code. The runtime of CRYSTAL, for a given atomic configuration, is dominated by two components: the evaluation of the Hamiltonian and the diagonalization of a number of matrices. The former is all but perfectly parallel. The latter, which is performed using ScaLAPACK [28], is not. However each diagonalization is independent from any of the others. Therefore the code can either

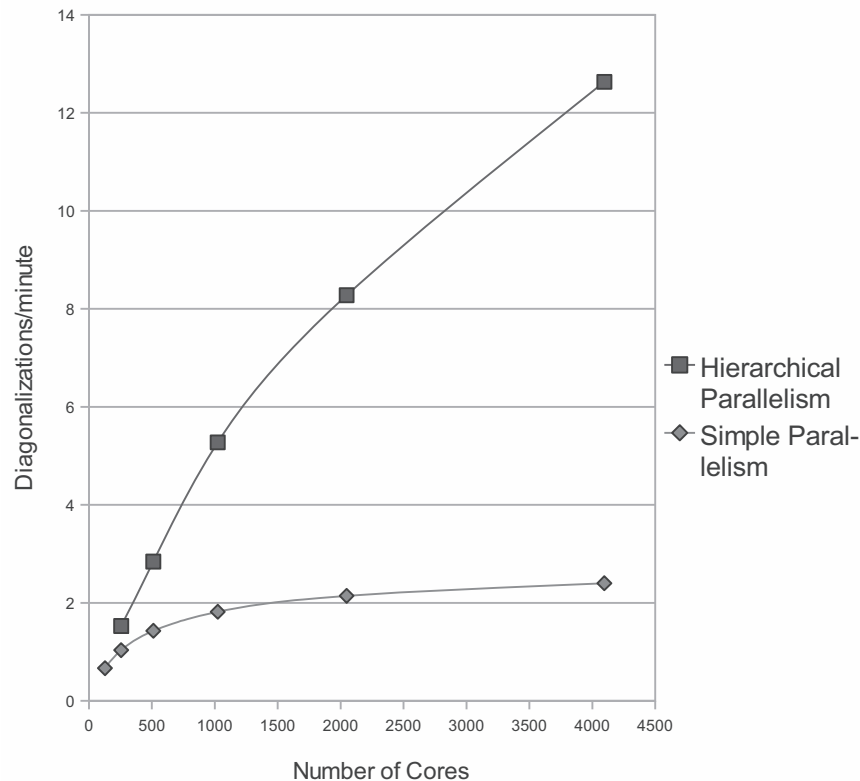
- Perform each diagonalization in turn across all of the processors
- Perform all of the diagonalizations at once, with each of them being performed by a subset of the cores.

Figure 1 compares these possibilities on HECToR [29]. In this case 8 diagonalizations are required, each of order 13608. It can be seen that the second possibility is much more efficient and allows the whole code to scale to thousands of processors. Couple this with a further level in the hierarchy, namely different atomic configurations, and it can be seen how very large parallel machines may be exploited.

This exploitation of hierarchical parallelism has resulted in a much greater use of communicators in MPI codes (and contexts where BLACS is employed), and in particular MPI\_COMM\_SPLIT. For a MPI programmer this provides a very natural way to split the problem up into its component parts and run these parts across subsets of the cores assigned to the job. Thus in the above example each of the matrices to be diagonalized is distributed across a subset of the processors, and this subset all belong to a communicator that is ultimately derived from splitting MPI\_COMM\_WORLD. It should be noted that cores that are not involved with diagonalizing a given matrix hold no data associated with that matrix.

As mentioned above other examples of hierarchical parallelism have been driven by the move to architectures whose nodes are multicore. Possibly the most obvious way to exploit the shared memory features of such machines is to use OpenMP with the multicore nodes, and MPI between the nodes. While in practice efficient “mixed mode” programs

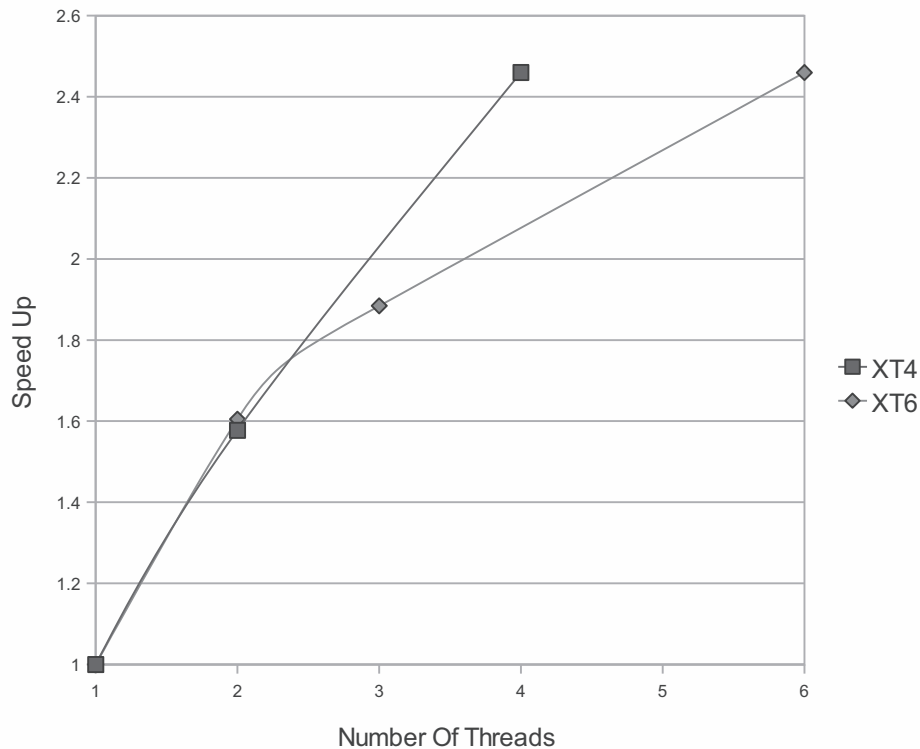
Figure 1 - Diagonalisation



(when compared to simply using MPI throughout) have proved more difficult to write than may have been initially thought, codes that use this method are becoming more common. Of the codes mentioned above CP2K currently can be run as a mixed mode code, currently efforts are under way to introduce mixed mode capabilities in HELIUM, and CASTEP is looking at using multithreaded libraries. Another example is CASINO. This is a quantum Monte Carlo code that for very large problems has been shown to scale to more than 40,000 cores [30] on Jaguar [31][32]. However the number of cores that can be exploited by the pure MPI code is strictly limited by the number of electronic configurations considered by the code. To work around this limit a second, finer level of parallelism has been introduced by Lucian Anton of NAG [33] which uses OpenMP. This allows many more cores to be exploited efficiently on more moderate sized problems, as shown in figure 2, so allowing CASINO to be used on hundreds or even thousands for even day to day problems.

Use of OpenMP within the node is not the only way to try to exploit the multicore architecture. Another approach is to maximise the use of fast intranode communications which may occur via shared memory, and to minimise the comparatively slow internode messages. One approach to this is again to use multiple MPI communicators to ensure the communications occur within the correct “universe.” Another method is the direct use of the shared memory features provided by the POSIX standard. This latter approach is currently, at least on HECToR, most commonly performed by use of the SystemV interface [34], that is shmget and related functions. A shared memory segment is created, and then all the processors within the node can read from or write to it. An example of this approach is Chris Armstrong’s optimisation of the FFT in CASTEP, based in turn on David Tanqueray’s (of Cray) work [35]. Within the FFT a MPI\_ALLTOALLV operation is required. In this all cores send messages to all other cores. This is optimised by exploiting the hierarchical nature of the machine. First all the data that need be communicated from a node is gathered together into a shared memory segment, the parts of that segment required by a given remote node are then communicated to that node in one long message (rather than many short ones) and received in another shared memory

Figure 2 - Multithreaded Casino



segment. The data is then unpacked within the node as required. This maximising of the message length, and hence avoidance of many latencies and potential contentions both at the NIC and on the interconnect itself, can bring very marked benefits. For instance on one run on 144 processors of the XT6 component of the HECToR service using the shared memory method reduces the run time from 2559 seconds when using a pure MPI based solution to 1815 seconds, an improvement of around 30%.

This is not the only reason for interest in shared memory. For instance large replicated objects can be stored in a segment, thus necessitating only one copy of the object to be held per node, rather than one per core. On today's "fat" multicore nodes this can be a very significant saving in memory. An early example of this is at [36]. A more recent example is again Lucian Anton's work with CASINO [37]. In both these cases the saving in memory both allows much bigger simulations to be run, and also more efficient use of the machine.

But the interface to the Unix shared memory routines is in C. Hence because portability is such an important consideration the all but universal implementation of the interoperability with C functionality that is part of Fortran 2003 [38] has resulted in that being of the areas of that standard that is being used by HPC developers today. Again using CASTEP as an example it is intended that the above optimisation will be included in an upcoming release of the code, and it is the "new" functionality within Fortran that is allowing this to be so.

Finally one must mention the most interesting addition to the latest Fortran standard, coarrays [39]. Here I feel it is too early to say much. Implementations are rare, and so much actual practical experience is lacking. However as there is an implementation on HECToR (the Cray compiler) and especially with the coming Gemini interconnect [40] there is definite interest in coarrays in the HPC community, for instance they have been mentioned in dCSE proposals, and one should watch this space!

## Acknowledgments

I would like to thank the CASTEP developers, in particular Keith Refson, Phil Hasnip and Stewart Clark, for allowing me to use CASTEP as an example. I would also like to thank Chris Armstrong and Lucian Anton, both of NAG, and Christine Bailey and Stanko Tomic of STFC Daresbury Laboratory for both sharing their results with me and allowing me to use them in this article.

## References

- [1] <http://www.fortran.com/ibm3.jpg>
- [2] <http://www.hector.ac.uk/>
- [3] <http://www.deisa.eu/>
- [4] <http://www.rcuk.ac.uk/default.htm>
- [5] <http://www.hector.ac.uk/abouthector/partners/>
- [6] [www.top500.org](http://www.top500.org)
- [7] [www.nag.co.uk](http://www.nag.co.uk)
- [8] [http://www.nag.co.uk/products\\_and\\_services.asp](http://www.nag.co.uk/products_and_services.asp)
- [9] <http://www.nag.co.uk/bdu/consultancy.asp>
- [10] <http://www.nag.co.uk/hpc/>
- [11] <http://www.hector.ac.uk/cse/distributedcse/>
- [12] <http://www.hector.ac.uk/cse/distributedcse/reports/>
- [13] <http://www.hector.ac.uk/cse/training/>
- [14] <http://cms.mpi.univie.ac.at/vasp/>
- [15] S. J. Clark, M. D. Segall, C. J. Pickard, P. J. Hasnip, M. J. Probert, K. Refson, M. C. Payne *Z. Kristallog* 220, 567-570 (2005)
- [16] <http://www.castep.org/>
- [17] <http://www.metoffice.gov.uk/science/creating/daysahead/nwp/um.html>
- [18] <http://cp2k.berlios.de/>
- [19] [http://www.am.qub.ac.uk/ctamop/ili\\_1.html](http://www.am.qub.ac.uk/ctamop/ili_1.html)
- [20] Technical report ISO/IEC TR 15581 : 1998(E)
- [21] <http://www.castep.org/>
- [22] ACM SIGPLAN Fortran Forum, 29, 2, (2010), 28-35, Ian D. Chivers and Jane Sleightholme
- [23] <http://www.hpcx.ac.uk/>
- [24] e.g. Y. Sugita and Y. Okamoto (1999). "Replica-exchange molecular dynamics method for protein folding". *Chemical Physics Letters* 314: 141–151.
- [25] <http://www.crystal.unito.it/>
- [26] R. Dovesi, R. Orlando, B. Civalleri, R. Roetti, V. R. Saunders and C. M. Zicovich-Wilson, *Z. Kristallogr.*, 220, 571-573 (2005).
- [27] R. Dovesi, V. R. Saunders, R. Roetti, R. Orlando, C. M. Zicovich-Wilson, F. Pascale, B. Civalleri, K. Doll, N. M. Harrison, I. J. Bush, P. D'Arco and M. Llunell, *CRYSTAL09*, (2009) *CRYSTAL09 User's Manual*. University of Torino, Torino.
- [28] <http://www.netlib.org/scalapack/>
- [29] I. J. Bush, S. Tomic, B. G. Searle, G. Mallia, C. L. Bailey, B. Montanari, L. Bernasconi, and N. M. Harrison, *Proc. Roy. Soc.* (submitted)
- [30] <http://www.nccs.gov/2010/06/04/jaguar-explores-carbon-water-union/>
- [31] <http://www.nccs.gov/computing-resources/jaguar/>
- [32] [http://www.hector.ac.uk/cse/reports/dCSE\\_Report\\_CASIN01.pdf](http://www.hector.ac.uk/cse/reports/dCSE_Report_CASIN01.pdf)

- [33] <http://www.hector.ac.uk/cse/distributedcse/reports/casino/casino/index.html>
- [34] [http://www.opengroup.org/onlinepubs/009695399/functions/xsh\\_chap02\\_07.html](http://www.opengroup.org/onlinepubs/009695399/functions/xsh_chap02_07.html)
- [35] [http://www.hector.ac.uk/cse/reports/castep\\_m.pdf](http://www.hector.ac.uk/cse/reports/castep_m.pdf)
- [36] [http://www.hpcx.ac.uk/research/hpc/technical\\_reports/HPCxTR0701.pdf](http://www.hpcx.ac.uk/research/hpc/technical_reports/HPCxTR0701.pdf)
- [37] <http://www.hector.ac.uk/cse/distributedcse/reports/casino/casino/index.html>
- [38] ACM SIGPLAN Fortran Forum, 29, 2, (2010), 28-35, Ian D. Chivers and Jane Sleightholme
- [39] ACM SIGPLAN Fortran Forum, 29, 2, (2010), 10-27, John Reid
- [40] <http://www.cray.com/Products/XE/Specifications.aspx>