Introducing k-Point Parallelism into VASP

Asimina Maniopoulou
HECToR CSE Team, NAG Ltd., UK
support@nag.co.uk

August 24, 2011

Contents

1 Introduction

This document is a report for the dCSE-funded project that introduces k-point parallelism into VASP v5.2.2 on HECToR. In what follows I will briefly show why the performance of VASP is important on HECToR, how it is parallelized, and then I shall discuss the introduction of k-point parallelism into VASP and how I implemented what I will call k-point parallelized version of VASP. I shall then present the results of the k-point parallelized code and finally discuss the improvements in VASP that my work delivers.

1.1 VASP

For many years *ab inito* electronic structure calculations have been one of the main stays of high performance computing (HPC). Whilst the methods used for those calculations have changed, for many years now the method introduced by Car and Parinello in 1985 [1] has been one of the most common to be employed. This is based upon density functional theory [2]; the Kohn-Sham [3] equations are solved within a plane wave basis set by minimisation of the total energy functional, with the use of pseudopotentials [4]-[5] to obviate the representation of core states. A review of the method can be found in [6]. Such is the importance of these methods that over 30% of all the cycles used on the phase2b component of HECToR [7], the UK's high-end computing resource, in the period from December 2010 − August 2011 were for packages performing total energy pseudopotential calculations.
One of the best known and widely used packages for performing this type of calculation is VASP [8], [9], [10], [11], the Vienna *Ab initio* Simulation Package. Indeed on HECToR it is the most extensively used package of all, and thus maximising its performance is vital for researchers using this, and related, machines. In this report I will describe my recent work on improving the parallel scalability of the code for certain classes of common problems. I have achieved this by introducing a new level of parallelism based upon the use of k-point sampling within VASP. Whilst this is common in similar codes, the latest release of VASP when I started off the project, version 5.2.2, does not support it, and I will show that through its use the scalability of calculations on small to mid-sized systems can be markedly improved. This is a particularly important class of problems as often the total energy calculation is not the only operation to be performed in the calculation. An important example is geometry optimisation. Here very many total energy calculations may need to be performed one after another. Thus the total size of the system under study is limited by time constraints, and so parallel scaling of the calculation on such moderate sized systems must be good if many cores are to be exploited efficiently.

1.2 The Parallelisation of VASP

Details of how VASP is parallelised is covered in some detail elsewhere [12], and I shall only cover the details which are relevant to the work here.
VASP 5.2.2 offers parallelisation (and data distribution) over bands and over plane wave coefficients, and both may be used together. How this division occurs is controlled by the NPAR tag in the INCAR file. In particular if there are a total of NPROC cores in the job, each band will be distributed over NPROC/NPAR cores. Thus, if NPAR=1 all the bands will be distributed over all processors, while if NPAR=NPROC the coefficients for a given band are all associated with one core.
NPAR reflects a tensioning between conflicting requirements. If bands are distributed across all processors, the communication costs for the parallel three dimensional Fast Fourier Transforms (FFTs) required by the plane wave pseudopotential method are high, but the cost for linear algebra operations required by the method, such as orthogonalisation and diagonalisation, are relatively low. On the other hand if NPAR=1 the communication cost for the FFTs is non-existent, but it is high for the linear algebra operations. Thus NPAR must be chosen carefully to obtain the best performance possible, and it will depend upon the chemical system being studied, the hardware upon which the run is being performed and the number of cores being used (amongst other possibilities).
Thus the use of NPAR allows a run to either stress the parallel FFT or parallel diagonalisation. Unfortunately neither of these operations scale well on distributed memory parallel architectures [13]-[14], especially for the moderate size grids and matrices used in many VASP runs.

1.3 Introducing k-point Parallelism

Parallelisation over bands and over plane waves are not the only possible ways that *ab initio* electronic structure codes can exploit modern HPC resources. Many such codes, but not VASP, also exploit parallelisation over k-points, examples being CASTEP [15],[16] and CRYSTAL [17],[18]. k-points are ultimately due to the translational symmetry of the systems being studied [19]. This symmetry also results in many, but not all, operations at a given k-point being independent from those at another k-point. This naturally allows another level of parallelism, and it has been shown that exploitation of k-point parallelism can greatly increase the scalability, a recent example being [20]. More generally this use of hierarchical parallelism is one of the more common methods of scaling to very large numbers of cores [21].
The standard release of VASP does not, however, exploit this possibility. Therefore we have modified the code from VASP 5.2.2 to add this extra level of parallelisation. The code is organised so that the cores may be split into a number of groups, and each of these groups performs calculations on a subset of the k-points. The number of such groups is specified by the new KPAR tag, which is set in the INCAR input file. Thus if the run uses 10 k-points and KPAR is set to 2 there will be 2 k-point groups each performing calculations on 5 k-points. Similarly if KPAR is set to 5 there will be 5 groups each with 2 k-points. It can therefore be seen that KPAR has an analogous role to NPAR mentioned above, except that it applies to k-point parallelism. Currently the value of KPAR is limited to values that divide exactly both the total number of k-points and the total number of cores used by the job. It should be noted that NPAR is also subject to the latter restriction.
This introduction of another level of parallelism through use of the k-points does potentially greatly increase the scalability of the code. However it should not be viewed as a panacea. Not all operations involve k-points, and thus Amdahl's law [22] effects will place a limit on the scalability that can be achieved. Further some quantities are not perfectly parallel across k-points, evaluation of the Fermi level being an obvious example. And it should be noted that introduction of k-point parallelism does introduce some extra communication and synchronization.
However probably the biggest limitation on the use of k-point parallelism is that due to system size. As the size of the unit cell that the calculation uses is increased fewer k-points are required to converge the calculation to a given precision, and in the limit only a single k-point may be sufficient to accurately represent the system. This limits what can be achieved by k-point parallelism, but for many practising computational scientists the calculations they require are not so large that they can be converged with 1 k-point, some recent examples being [23]-[26]. Therefore for many cases parallelisation over k-points is a useful technique. This is especially true when the total energy calculation is only one part of a larger calculation, for instance in a geometry optimisation.

1.4 Implementation of k-point parallelism

In the original code the tag NPAR is used to create intra-band and inter-band communicators. MPI processes are bound to an intra-band communicator to work on specific bands and when information is required about the other bands, MPI processes communicate through the inter-band communicators. To exploit k-point parallelism, I constructed an extra layer of communication using the value of KPAR. KPAR intra-k-point communicators (named *COMM*) and MPI processes/KPAR inter-k-point communicators (named *COMM_CHAIN_K*) are created. Each *COMM* communicator has all the information needed for the k-points it is assigned. Subsequently, as before the MPI processes in the *COMM* communicator are divided into MPI processes bounded to inter- and intra-band communicators. When a loop over k-points is encountered, the processes in each *COMM* communicator iterate through all the k-points they are responsible for and as soon the loop exits the *COMM_CHAIN_K* communicators are used to gather

together the results. This is the general design followed, although the main difficulties at the development process arose at places where information from all k-points was required e.g., for implementing the linear tetrahedron method in the evaluation of the Fermi level etc.

Also extra care was needed when a full, non-symmetrized k-grid was to be employed, as some particular quantities have to be stored in all communicators, even when they are referring to k-points the communicator is not responsible for. The cases where the Hartee-Fock exchange is used were further complicated as they involve nested k-loops. Extra communication is needed there (compared to the non-Hartee Fock) cases, but as seen in the test cases (2.4) and (2.5) this does not hinder the scaling of the code, at least for the cases studied.

In regards with I/O, depending on the case I have chosen either to communicate all relevant information on the master node, or have a 'leader' in each *COMM* communicator to do the writing (using a loop so that information was written in orderly form).

2 Benchmarking

We have examined several test cases with the new code. Here we present five:

- Test 1: A hydrogen defect in 32 atoms of palladium. 10 k-points are used.
- Test 2: A unit cell of Litharge ($\boldsymbol{\alpha}$ -PbO), a total of 4 atoms. 108 k-points are used.
- Test 3: A cell of PbO using 126 k-points.
- Test 4: Again $\boldsymbol{\alpha}$ -PbO using 24 k-points.
- Test 5: A phonon calculation with 20 k-points.

In Tests 1-3 the PBE exchange correlation functional is used and the k-mesh is generated by the Monkhorst-Pack method. Tests 4-5 involve Hartee-Fock calculations and the k-mesh is generated with the Gamma centered method. All runs except for the phonon calculation are a single point energy calculation.

All runs have been performed on the phase 2b component of the HECToR system, the UK's national supercomputing service. This is a large Cray XE6 system. The nodes are based upon AMD Magny-Cours processors, and contain 24 cores each clocking at 2.1GHz. There is 32 Gbytes of memory associated with each node, and inter-node communication is via Cray's Gemini network. More details may be found at the HECToR web site ([7]).

In Tables (1), (3), (5), (7) and (8) we compare the performance of VASP 5.2.2 with the new k-point parallelized code and study the scaling of the new code. In each case the original code is compared with the k-point code with increasing numbers of k-point groups. All times reported are total run times, i.e. not just the time for the energy minimisation.
In Tables (2), (4), (6) and (9) we compare the performance of VASP 5.2.2 using the optimal NPAR value with the performance of the k-point parallelized code, when the same number of cores is utilized. We demonstrate that efficient use of large number of cores is now possible for cases with more than one k-point.
It should be noted that the use of an appropriate NPAR value is imperative for the efficient running of

VASP. The optimal value of NPAR in the original code depends on the total number of cores employed. For the k-point parallelized code, the optimal value of NPAR depends on the number of cores in one k-group. Hence the value of NPAR that was optimal for the original code on x cores, will be also the most efficient choice for n k-groups on $n \times x$ cores, when using the k-points parallelized code.

2.1 Test Case 1

The system simulated is a hydrogen defect in 32 atoms of palladium. It uses 10 k-points and the PBE exchange correlation functional ([27]-[28]).

Test Case 1	Cores	Time (secs)	Speedup
VASP 5.2.2	64	298.187	1
KPAR=2	128	159.982	1.863
KPAR=5	320	75.357	3.956
KPAR=10	640	47.795	6.239

Table 1: Scaling of Test Case 1.

Table (1) shows that the k-point parallelized code scales satisfactorily to 320 cores, where it is twice as fast as the original code at the same number of cores (see Table (2)).

Test Case 1	Cores	Time (secs)	Speedup
VASP 5.2.2	128	206.517	1
KPAR=2	128	158.686	1.301
VASP 5.2.2	320	146.197	1
KPAR=5	320	75.201	1.944
VASP 5.2.2	640	147.606	1
KPAR=10	640	47.795	3.088

Table 2: The optimal NPAR for the original code is 4, 8, 32, and 32 for 64, 128, 320 and 640 cores respectively. For the k-point parallelized code the optimal NPAR is 4 for any number of cores, provided one k-group consists of 64 cores.

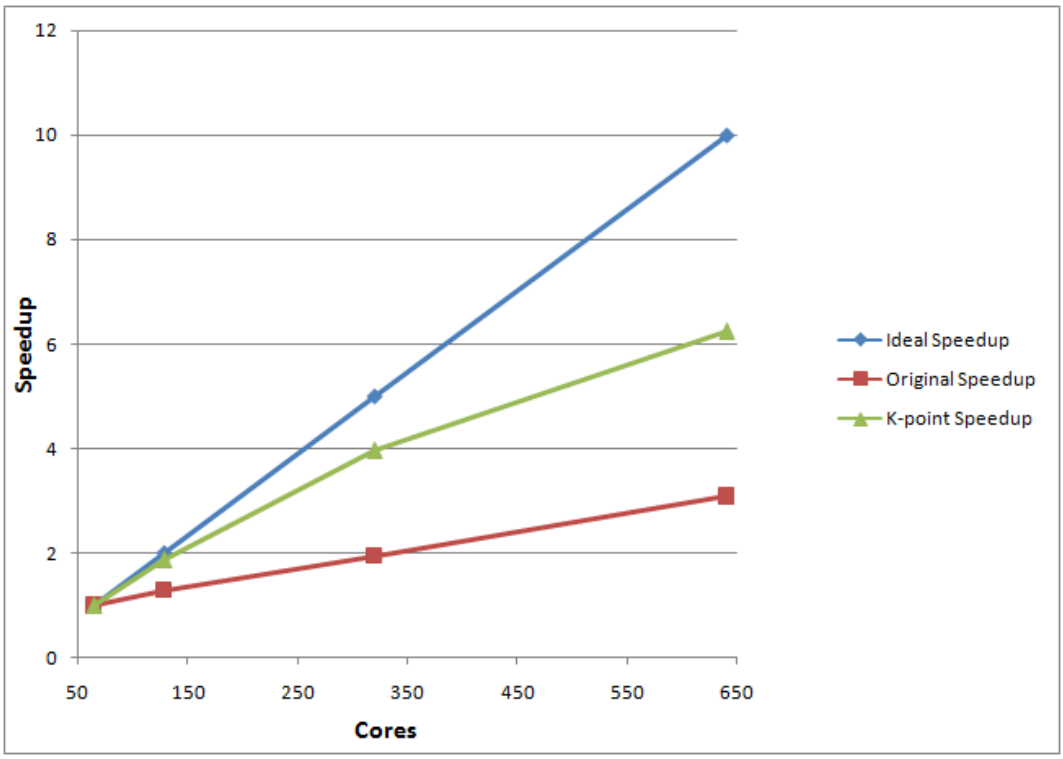

Figure 1: Speedup for Test Case 1 (where Speedup is taken to be 1 for 64 cores).

2.2 Test Case 2

The system simulated is a unit cell of Litharge (a-PbO), a total of 4 atoms. 108 k-points are used.

Test Case 2	Cores	Time (secs)	Speedup
VASP 5.2.2	144	2141.206	1
KPAR=2	288	1097.981	1.950
KPAR=3	432	766.196	2.794
KPAR=4	576	607.69	3.524
KPAR=6	864	410.018	5.222
KPAR=9	1296	278.585	7.686
KPAR=12	1728	215.757	9.924
KPAR=18	2592	150.177	14.258
KPAR=27	3888	105.563	20.284
KPAR=36	5184	87.569	24.452
KPAR=54	7776	65.332	32.774
KPAR=108	15552	41.055	52.154

Table 3: Scaling of Test Case 2.

Table (3) shows that the k-point parallelized code scales rather satisfactorily to 3888 cores, and at 1738 cores it is 7 times faster than the original code at the same number of cores (see Table (4)).

Test Case 2	Cores	Time (secs)	Speedup
VASP 5.2.2	288	1530.244	1
KPAR=2	288	1097.981	1.394
VASP 5.2.2	432	1348.436	1
KPAR=3	432	766.196	1.76
VASP 5.2.2	576	1473.4	1
KPAR=4	576	607.69	2.425
VASP 5.2.2	864	1562.76	1
KPAR=6	864	410.018	3.811
VASP 5.2.2	1296	1899.499	1
KPAR=9	1296	278.585	6.818
VASP 5.2.2	1728	1532.32	1
KPAR=12	1728	215.757	7.1021

Table 4: The optimal NPAR for the original code is 18, 18, 18, 36, 36, 36 and 36 for 144, 288, 432, 576, 864, 1296 and 1728 cores respectively. For the k-point parallelized code the optimal NPAR is 18 for any number of cores, provided one k-group consists of 144 cores.

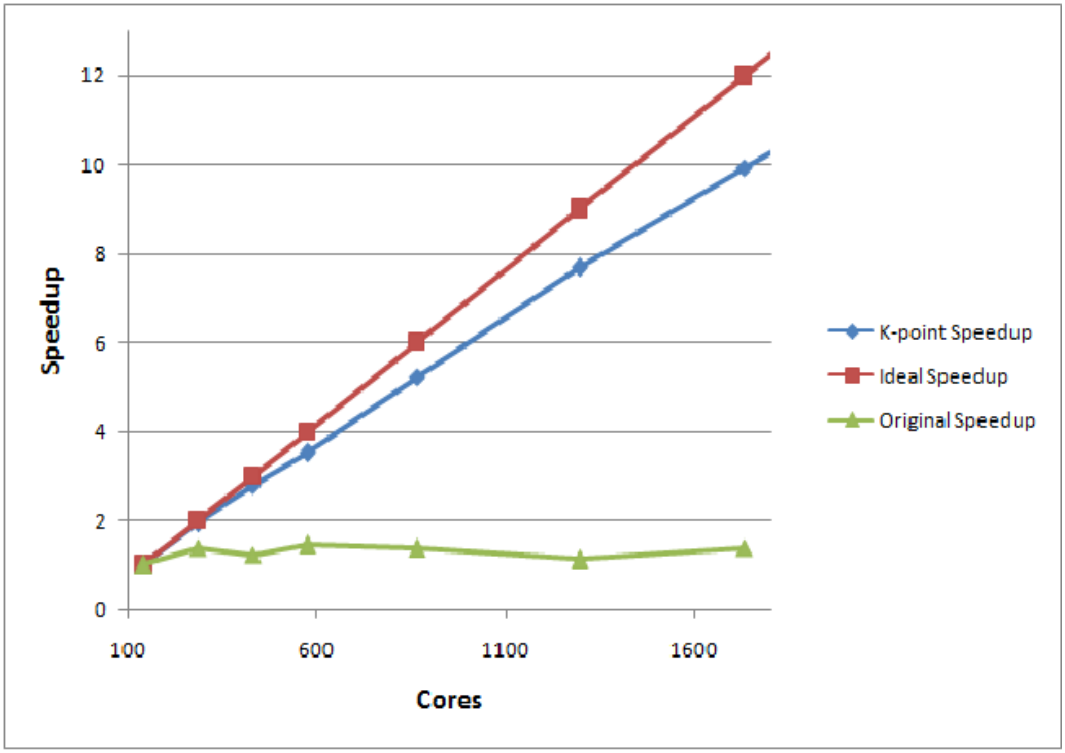

Figure 2: Speedup for Test Case 2 (where Speedup is taken to be 1 for 144 cores).

2.3 Test Case 3

The simulated system is PbO using 126 k-points. The original code exhibits slowdown for over 144 cores and the only way to employ more cores is to use the k-point parallelized code. The latter scales quite satisfactorily up to 1008 cores.

Test Case 3	Cores	Time (secs)	Speedup
VASP 5.2.2	144	120.388	1
KPAR=2	288	69.208	1.740
KPAR=3	432	48.315	2.492
KPAR=6	864	30.234	3.982
KPAR=7	1008	27.402	4.393
KPAR=9	1296	40.443	2.976

Table 5: Scaling of Test Case 3.

Test Case 3	Cores	Time (secs)	Speedup
VASP 5.2.2	288	130.688	1
KPAR=2	288	69.208	1.888
VASP 5.2.2	432	122.236	1
KPAR=3	432	48.315	2.530
VASP 5.2.2	864	320.196	1
KPAR=3	864	30.234	10.591
VASP 5.2.2	1008	318.516	1
KPAR=7	1008	27.402	11.624

Table 6: Here the optimal NPAR for the original code is 18, 12,18, 36 for 144, 288, 432, 1008 cores respectively. For the k-point parallelized the optimal NPAR for any number of cores is 18, provided a k-group consists of 144 cores.

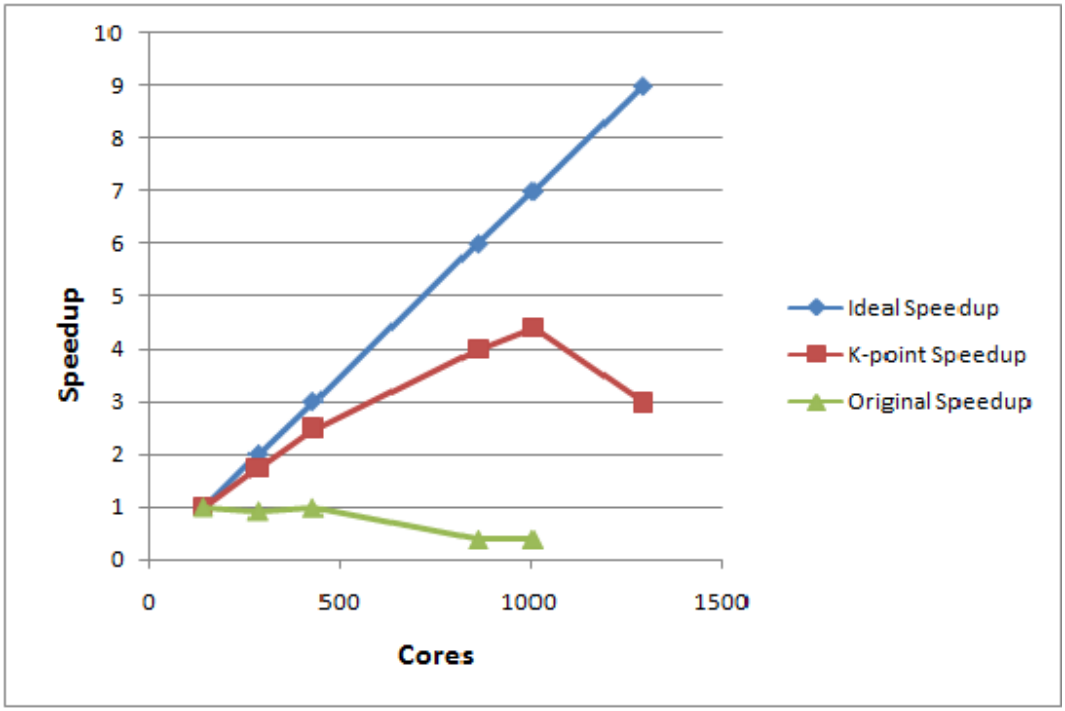

Figure 3: Speedup for Test Case 3 (where Speedup is taken to be 1 for 144 cores.)

2.4 Test Case 4

The system simulated is PbO with 24 k-points. This simulation involves Hartee-Fock exchange calculations. Optical properties are also examined.

Test Case 6	Cores	Time (secs)	Speedup
VASP 5.2.2	256	14674.49	1
KPAR=2	512	7578.614	1.936
KPAR=3	768	4979.435	2.947
KPAR=4	1024	3790.061	3.871
KPAR=6	1536	2552.615	5.749
KPAR=8	2048	1944.401	7.548
KPAR=12	3072	1399.796	10.953
KPAR=24	6144	1053.134	13.93411

Table 7: Scaling of Test Case 4.

The original code failed to run on 512 cores or more. Hence the only way to run efficiently this problem is to employ the k-point parallelized version. This runs efficiently for this test case on up to 3072 cores.

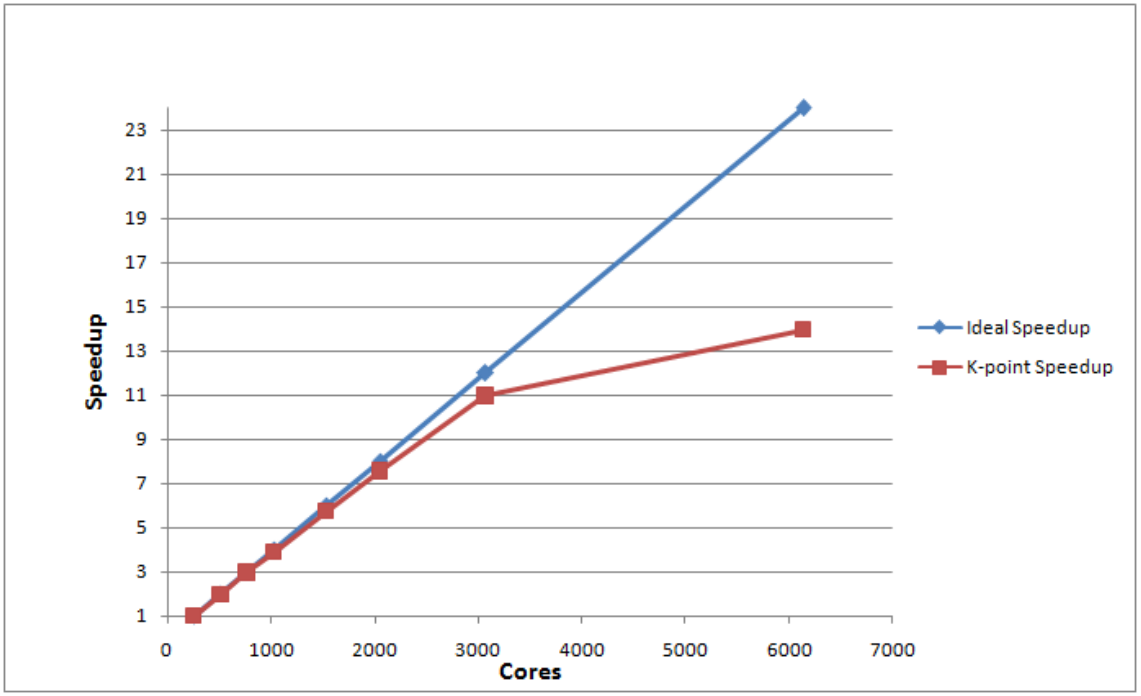

Figure 4: Speedup of Test Case 4 (where Speedup is taken to be 1 for 256 cores.)

2.5 Test Case 5

The last test case is a phonon system with 20 k-points. This is a test case where NPAR has to be equal to the total number of cores in the original code. This, according to the discussion in 1.2 means that the linear algerbra operations cannot be parallelized and only the FFT calculations are performed in parallel. For the k-parallelized code accordingly, NPAR should be equal to the number of cores in one k-group.

Test Case 5	Cores	Time (secs)	Speedup
VASP 5.2.2	32	399.929	1
KPAR=2	64	221.94	1.802
KPAR=4	128	112.407	3.558

Table 8: Scaling of Test Case 5 (where Speedup is taken to be 1 for 32 cores).

Firstly, Table (9) shows that the original code does not scale at all over 32 cores. The FFT communications cost becomes the bottleneck and it is not possible to perform the computation in less than 400 secs with the original code.
Table (8) on the other hand shows that with the k-points parallelized code we can employ 4 times more cores and we complete the simulation in 122 secs (3.6 speedup). The problem though is that we cannot use more than 128 cores in this case, where potentially we could use 640 (number of k-points (20) $\times$ 32) for this case. This is because during the specific calculation new k-point meshes are generated. When KPAR is an exact divisor of the number of the k-points in the new mesh our k-points parallelized code performs the calculation efficiently. When not, it exits. In this case the original k-mesh had 20 k-point, the second k-mesh 52 and the third 68. Only the numbers 2 and 4 are common divisors of the aforementioned 3 numbers. Hence the biggest value that can be used for KPAR is 4.

Test Case 5	Cores	Time (secs)	Speedup
VASP 5.2.2	64	603.294	1
KPAR=2	64	221.94	2.718
VASP 5.2.2	128	2477.339	1
KPAR=4	128	112.407	22.039
VASP 5.2.2	160	4651.663	1
KPAR=5	160	64.96	71.608

Table 9: As seen above the original code does not scale at all.

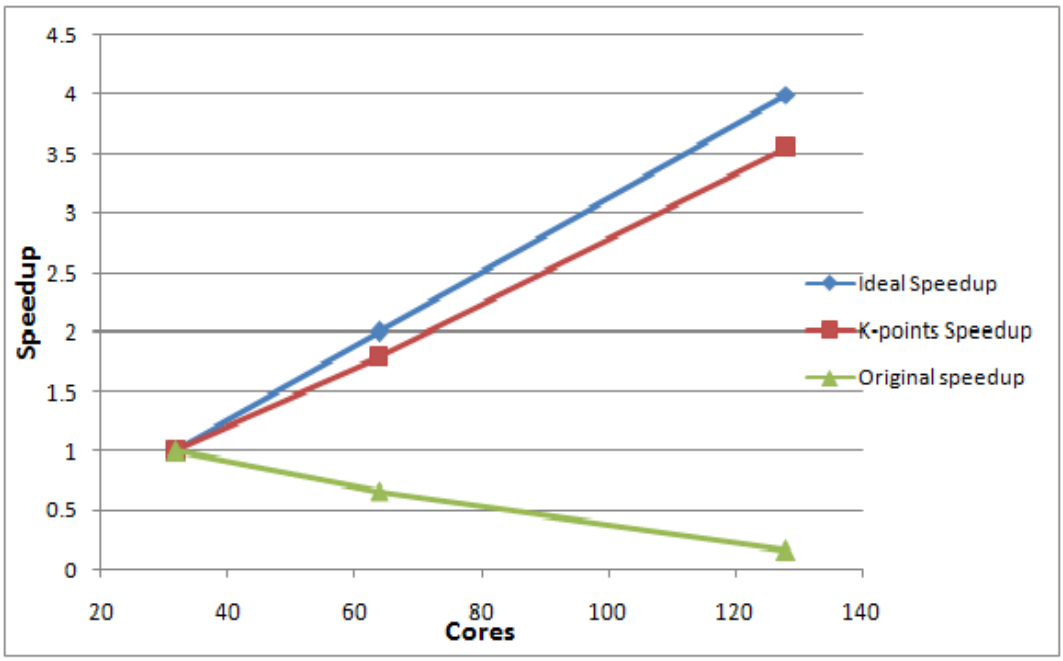

Figure 5: Speedup of Test Case 5 (where Speedup is taken to be 1 for 32 cores.)

3 Limitations/Constraints

At this point we should note some limitations that our k-point parallel code has.

- The number of k-groups (i.e the value of the KPAR tag) should be an exact divisor of the total number of cores *and* an exact divisor of the number of k-points. If a new k-mesh is generated,then the new number of k-points should be divided exactly by the value of KPAR (See 2.5).
- No *vasprun.xml* output file is produced.
- Non-collinear spin calculations are not available with the k-point parallelized code.

4 Applications

The code has been used by researchers in UCL to study more efficiently some systems. A relevant paper by Maniopoulou A., Grau-Crespo R., Davidson E., Walsh A., Bush I.J., Woodley S.M., is in preparation. Here we very briefly describe what was achieved using the k-point paralleized code for two of the three systems studied provided by Dr Aron Walsh and Dr Ricardo Grau-Crespo respectively.[1]

4.1 Dielectric Function of Epitaxially Strained Indium Oxide.

Calculation of optical properties of semiconducting materials is a very slowly converging process with respect to k-point sampling in the the Brillouin zone. Such a calculation of particular importance is the

[1]No results are available for the third system at the time of writing.

change in optical response for ultra-thin epitaxially strained Indium Oxide, which is used as transparent conducting oxide in optoelectronic devices. The convergence of the dielectric function is assessed through the change in the high-frequency dielectric constant (ε_∞) with respect to k-point sampling.
On the HECToR Phase 2b system, using a standard compilation of VASP 5.2, the maximum k-point density possible within the queue limit of twelve hours is $2 \times 2 \times 2$ using 96 cores, while using k-point parallelism this can be increased to $6 \times 6 \times 6$ and scale up to 1,536 cores. For the latter k-mesh the convergence required is achieved, a result attainable on HECToR only with the k-point parallelized code.

4.2 Solution energies of tetravalent dopants in metallic VO_2

The stability of three dopants has been examined for VO_2. The room temperature phase is metallic and hence a dense k-mesh is required for the calculations of pure and doped VO_2.
Using k-point parallelism in VASP, Dr Grau-Crespo tested $4 \times 4 \times 4$ (18 irreducible k-points) and $8 \times 8 \times 8$ (75 irreducible k-points) meshes for pure VO_2. The latter calculation was performed over 360 processors in the HECToR Phase2b supercomputer, with a speedup factor of 13 compared to the standard version running on 24 processors (ideal scaling would correspond to a speedup factor of 15). The calculations of the doped cell were performed on 168 processors, with KPAR=7 groups (of 3 k-points each) running in parallel, on 24 processors each. All geometries were optimised until the forces on the ions were all less than 0.01 eV/Å.

5 Conclusions

The k-point parallelized codes can certainly provide more efficient use of computation cycles as it can be demonstrated in tests cases (2.1)-(2.2) and application 4.1. There are also other cases, where either the original code does not scale at all to larger number of cores -see test cases (2.3),(2.5) , or the original code does not even work on large number of cores (see test case (2.4)) or exterior limitations such as the 12 hour queue on HECToR in application (4.2) do not allow large simulations. In all these cases we demonstrated the calculations could be achieved with the k-point parallelized code.

6 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. The author would very much like to thank the following people for their contribution to this work:

- Dr Ian J. Bush for the constant support and the fruitful discussions and corrections during the code development process, the proof-reading and corrections on this report and all the presentations relevant to this work. He should be thanked also for the coordination of the successful collaboration with the HPC Materials Chemistry Consortium.

- Dr Scott Woodley, Dr Ricardo Grau-Crespo, PhD candidate Erland Davidson and Dr Aron Walsh for discussions, for providing test cases and benchmarks, confirming the accuracy of the results of the k-parallelized code and also using the code for the applications in Section 4.

References

[1] Car,R., and Parrinello, M., 1985, *Phys. Rev. Lett.* 55,2471

[2] Hohenberg, P., and Kohn, W., 1964, *Phys. Rev.* 136, 864B

[3] Kohn, W., and Sham, L.J., 1965, *Phys. Rev.* 140, 1133A

[4] Phillips, J.C., 1958, *Phys. Rev.* 112, 685

[5] Cohen, M.L, and Heine, V., 1970, *Solid State Physics* Vol 24,37

[6] Payne, M.C., Teter, M.P., Allan, D.C., Arias, T.A., and Joannopoulos, J.D., 1992, *Rev. Mod. Phys.,* 64, 1045

[7] `http://www.hector.ac.uk/service/hardware/`

[8] Kresse, G., and Hafner, J., 1993, *Phys. Rev. B.* 47, 558

[9] Kresse, G., and Hafner, J., 1994, *Phys. Rev. B.,* 49, 14251

[10] Kresse, G., and Furthmller, J., 1996, *Comput. Mat. Sci.*, 6, 15

[11] Kresse G., and Furthmller, J., 1996, *Phys. Rev. B,* 54, 11169

[12] `http://cms.mpi.univie.ac.at/vasp/vasp/vasp.html`

[13] `http://www.hpcx.ac.uk/research/hpc/technical_reports/HPCxTR0303.pdf`

[14] `http://www.hpcx.ac.uk/research/hpc/technical_reports/HPCxTR0510.pdf`

[15] Segall M.D., PLindan P.L.D., Probert M.J, Pickard C.J., Hasnip P.J., Clark S.J., Payne M.C *J. Phys.: Cond.Matt.* 14(11) pp.2717-2743 (2002)

[16] Clark S.J., Segall M.D., Pickard C.J., Hasnip P.J., Probert M.J., Refson K., Payne M.C., *Zeitschrift fr Kristallographie* 220(5-6) pp.567-570 (2005)

[17] Dovesi R., Orlando R., Civalleri B., Roetti R., Saunders V.R., and Zicovich-Wilson C.M., *Z. Kristallogr.* 220, 571 (2005)

[18] Dovesi R., Saunders V.R., Roetti R., Orlando R., Zicovich-Wilson C.M, Pascale F., Civalleri B., Doll K., Harrison N.M, Bush I.J., DArco p, and Llunell M., *CRYSTAL09 (CRYSTAL09 User's Manual. University of Torino, Torino, 2009).*

[19] Ashcroft, N.W., and Mermin, N.D., 1976, *Solid State Physics* (Holt Saunders, Philadelphia)

[20] Bush, I.J, Tomic, S., Searle, B.G., Mallia, G., Bailey, C.L., Montanari, B., Bernasconi, L., Carr, J.M., Harrison, N.M., *Proc. R. Soc. A* 8 July 2011 vol. 467 no. 2131 2112-2126

[21] Bush, I.J., "The view from the high end: Fortran, parallelism and the HECToR service", *ACM SIGPLAN Fortran Forum*, Volume 29 (3) Association for Computing Machinery - Nov 12, 2010

[22] Amdahl, G., 1967, *AFIPS Conference Proceedings (30):* 483485

[23] Smith, C., Fisher T.S., Waghmare U.V., and Grau-Crespo. R. *Physical Review B* 82, 134109 (2010).

[24] Chauke H.R., Murovhi P., Ngoepe P.E., de Leeuwand N.H., Grau-Crespo. R *Journal of Physical Chemistry C* 114, 15403-15409(2010).

[25] Grau-Crespo R., Smith K.C., Fisher T.S., de Leeuw N.H., and Waghmare. U.W *Physical Review B* 80, 174117 (2009)

[26] Grau-Crespo R., Cruz-Hernndez N., Sanz J.F and de Leeuw. N.H. *Journal of Materials Chemistry* 19, 710-717 (2009).

[27] Perdew J.P, Burke K., and Ernzerhof. M. *Phys. Rev. Lett.*, 77:3865, 1996.

[28] Perdew J.P., Burke K., and Ernzerhof. M. *Phys. Rev. Lett.*, 77:3865, 1996.