Optimisation of VASP Density Functional Theory (DFT)/Hartree-Fock (HF) hybrid functional code using mixed-mode parallelism

Gavin J. Pringle

EPCC, University of Edinburgh, Mayfield Road, Edinburgh, EH9 3JZ

31 October, 2012.

Abstract

VASP, or the Vienna *Ab initio* Simulation Package, is an *ab initio* quantum mechanical software package used to simulate the electronic structure of condensed phase materials. Two new mixed-mode versions have been created by introducing OpenMP to the F90+MPI code. The first employs OpenMP directives within an extant VASP 5.2.12 routine, and we see 50% parallel efficiency, when comparing two and no OpenMP threads per MPI task. This will permit new science as we may now double the number of cores used, thereby increasing the memory available. The second version (currently incomplete) involves a complete restructuring of the DFT/HF code, and is inspired by a GPU-based version. Replacing the internal FFT routines with calls to the FFTW3 library was found to be beneficial: the version of the VASP centrally available to VASP users now employs the FFTW3 library.

Table of Contents

Introduction

In this report, we describe how we undertook to introduce OpenMP [1] parallelisation into a recent version of VASP [2], version 5.2.12, on behalf of the UK's Materials Chemistry Consortium (MCC) [3].

VASP

VASP, or the Vienna *Ab initio* Simulation Package, is an *ab initio* quantum mechanical software package used to simulate the electronic structure of condensed phase materials. VASP is one of the most important and utilised materials science and chemistry codes in the world. Locally, it accounts for over 18% of all AUs used on HECToR between October 2009 and June 2010 – by far the code with the largest usage in this sampled period – equivalent to a notional cost of over £1.5 million. At that time, there were at least 63 registered HECToR users using the package from 14 different HECToR projects.

Hybrid DFT/HF functionals are a new addition to the VASP code (from version 5.2), which offer a means to overcome some of the limitations of traditional density functional approaches. Of particular relevance to the materials science community are the description of localised wave functions (*d* and *f* orbitals); transition states and reaction barriers for catalytic cycles, surface work functions; and electronic and optical band gaps. The parallel performance of the VASP 5.2 code on HECToR is well understood from the previous successful dCSE application [4]. In particular, the hybrid functional part of VASP 5.2 has especially poor scaling performance when compared to the performance of the pure DFT part, limiting the usage of this functionality within the Materials Chemistry Consortium (MCC).

This difference in performance arises both through the relative immaturity of these subroutines – they have not been optimised and developed as thoroughly as the rest of the VASP code; and also through the specific challenges posed by computing the HF exchange component for the hybrid functional with explicit two-electron integrals. The HF scheme does not parallelise in the same way as a pure DFT calculation and so imposes additional constraints on the parallelisation scheme employed by VASP which, in turn, adversely affects the performance of the code when moving to large core numbers.

We aimed to overcome these performance limitations by employing a mixed-mode (MPI + OpenMP) parallelisation scheme for the hybrid-functional strand of VASP 5. This combination of parallel programming models will allow the code to be used for new science on HECToR and future-proof the code for the migration to the massively multi-core machines that are part of the upgrade path for HECToR and HPC facilities in general.

These optimisations and developments constitute a first step on implementing a full, mixed-mode version of the VASP 5 software package. This development of VASP is critical to allowing its effective use for new science on the current- and next-generation of HPC architectures for the reasons outlined above. The current Cray parallel programming roadmap is advocating a directive-based approach (similar to OpenMP directives) to exploiting parallelism within a MPP node – whether that parallelism is based on multi-core CPUs (such as HECToR, a Cray XE6) or on accelerator technology (such as GPUs) which Cray are targeting for the next generation of HPC machines. Mixed-mode

programming is the only way that codes such as VASP will be able to harness the power of these HPC machines. The hybrid-functional strand of the VASP 5 package is an obvious first place to begin these developments which has the potential to deliver real performance, scaling and scientific enhancements on HECToR right now. These developments will benefit a large proportion of scientific projects within the MCC, ranging from crystal structure prediction, heterogeneous catalysis and materials design for solar cell applications.

The Materials Chemistry Consortium

The MCC are concerned with wide-ranging issues in the field of materials science, including electronic, optical, magnetic and nuclear materials and their device applications. The membership of the MCC consists of over 200 users on HECToR based in 36 different UK research groups, where the Principle Investigator and the Project Manager are Richard Catlow (UCL) and Scott Woodley (UCL), respectively.

The VASP 5 licence is owned by several members of the MCC and, more importantly, work on optimising the performance of VASP on HECToR is covered by the current support-licences to HPCx and HECToR.

Baseline profiling

Baseline hybrid DFT parallel performance

All results below are for VASP 5.2.11, using a single point SCF calculation using the 64 atom H defect in Li benchmark calculation supplied by the MCC. All calculations are performed using the best performing Γ-point code on HECToR phase 2a (Cray XT4) with fully-packed nodes.

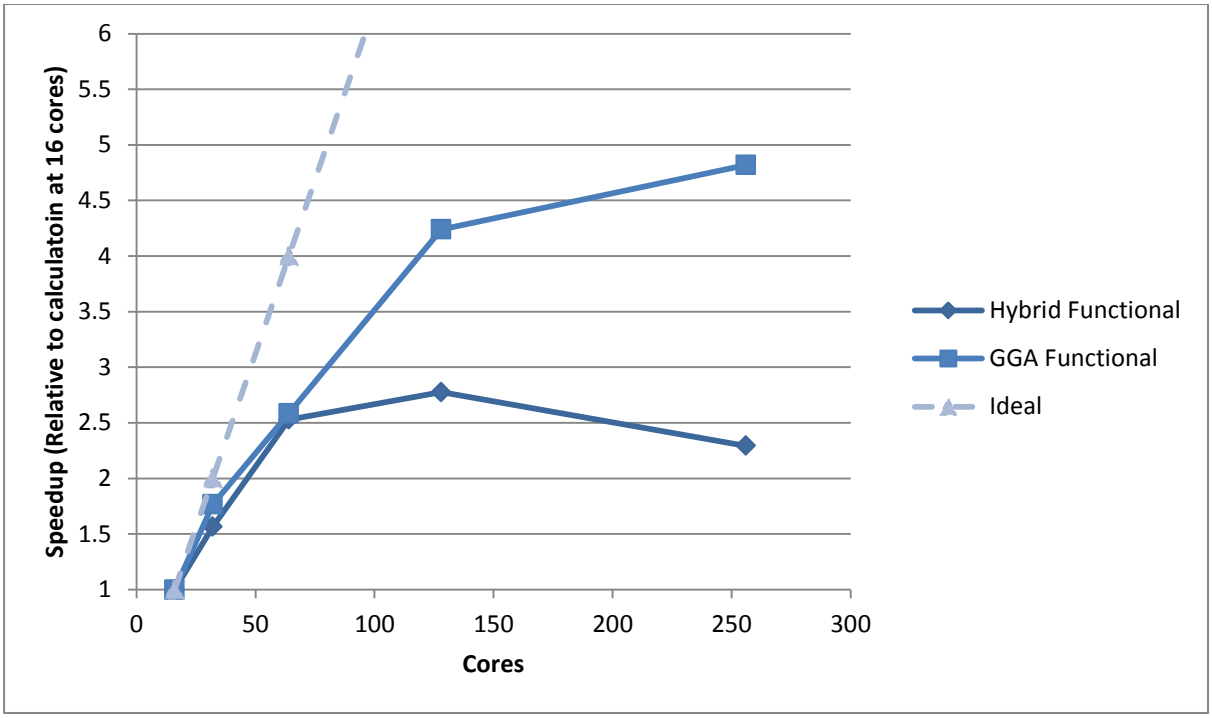
The execution times are presented in Table 1, whilst the scaling of the hybrid benchmark is shown in

Figure 1, compared to the scaling of the same calculation using a pure GGA DFT functional.

Table 1: Runtimes for VASP benchmark at various core counts using GGA and hybrid DFT functionals.

	Runtime (secs)	
Cores	**GGA Functional**	**Hybrid Functional**
16	106	1571
32	60	1004
64	41	621
128	25	566
256	22	685

Figure 1: Scaling of VASP benchmark for pure DFT and hybrid DFT functionals.



We can see that this rather small benchmark (64 atoms) scales out to 128 cores on HECToR phase 2a (Cray XT4) when using the pure DFT functional but only scales to half the number of cores when using the hybrid functional.

One of the reasons for the better scaling in the pure DFT case is that we are able to improve the parallel performance by altering the balance between different parallel parts of the program using the runtime parameter "NPAR". The VASP "NPAR" parameter allows users to alter the number of MPI tasks assigned to both the parallelisation over plane waves and the parallelisation over the electronic bands – this, in turn, allows the parallel performance of VASP to be optimised (at run-time) to the problem being studied and the number of MPI tasks being used. In the hybrid functional calculation we are restricted to selecting a value of "NPAR" equal to the number of MPI tasks that the calculation is running on. This inflexibility is imposed by VASP for good reasons as inter-band communication would be extremely inefficient when Hartree-Fock exchange is included. Even so, the inability to exploit another layer of parallelism within the code is currently limiting the scaling behaviour.

Another limitation in the parallelism of VASP that affects all calculations (pure or hybrid DFT) arises in the PAW pseudopotentials that are used to represent the core electrons in the system. VASP parallelises this part of the calculation over the number of atoms. This strategy effectively caps the scaling that can be achieved by the code relative to the number of atoms in the system. This is especially problematic for hybrid DFT calculations as, due to the more computational expensive nature of the method, smaller systems tend to be studied. Adding another layer of parallelism to this section of the code is therefore a prerequisite of achieving good scaling behaviour.

Baseline hybrid DFT profiling results

Table 2 shows the profile of the MPI routines for the benchmark using the hybrid-DFT functional on 64 cores (when the code is still scaling well) and on 256 cores (where the code does not scale). The major increase is in the MPI_SYNC time for the MPI_Allreduce function indicating an increase in the load-imbalance of the code. The MPI_Allreduce collective operation is acting as a barrier in the code enforcing the synchronisation of all MPI tasks – a large synchronisation time reveals an imbalance in the workload between MPI tasks. This particular call is in the evaluation of the PAW pseudo-potentials and is evidence of the limits imposed by the parallelisation over atoms described above (*i.e.* there are more MPI tasks than atoms). Introducing OpenMP parallelisation would allow for a large improvement in the scaling performance of this portion of the VASP code. Instead of parallelising over atoms, by assigning each atom to an individual MPI task (or *core*), we would be able to parallelise the atoms in the PAW calculation by *nodes* (where a node can correspond to either a physical node on the HECToR system or a subset of the processors – possibly a NUMA region – within a physical node). Each atom would be assigned to a particular node and the OpenMP parallelisation within the node would parallelise the PAW calculation for a single atom.

Table 2: Parallel profile at 64 and 256 cores (fully populated nodes) for the benchmark, MPI_SYNC time relates to wait time on the parallel tasks.

	64 Cores		256 Cores	
Routine	**% Time**	**Time (secs)**	**% Time**	**Time (secs)**
MPI_SYNC	16.0	110.5	33.7	267.5
MPI_Allreduce	10.1	69.6	20.7	164.1
MPI_Bcast (ScaLAPACK)	2.5	16.9	5.6	44.2
MPI_Alltoallv	1.0	6.9	3.3	26.5
MPI_Barrier	1.1	7.9	2.2	17.8
MPI				
MPI_Bcast	7.1	48.8	12.1	95.9
MPI_Allreduce	5.6	38.3	8.7	69.2

Table 3 provides an overview of the time spent in different subroutines within the VASP hybrid-functional code and allows us to identify the routines with which to start the OpenMP parallelisation of the non-PAW parts of the code. The obvious candidate is the “eddav” subroutine which accounts for a large fraction of the runtime at both 64 and 256 cores, followed by “rpro1_hf” and “wpbe_spline” (and the “*_spline” routines corresponding to the other DFT functionals). The FFT routines “fpassm”, “ipassm” and “fft3d” are also key routines for OpenMP parallelisation. Of course, the majority of the hybrid-functional code will have to be OpenMP parallelised in order to maintain performance but the routines identified here are the areas in which the OpenMP optimisation effort should be focussed. Although there is some code shared between the hybrid-functional and pure DFT functional parts of the VASP code, much of the computational work is done in routines specific to the hybrid-functional calculation which allows us to effectively introduce OpenMP parallelisation without having to modify the entire VASP codebase.

The outcome from this work will be a version of the VASP 5.2 code that can be run with MPI+OpenMP when running calculations that employ a hybrid functional and with just MPI for pure DFT functional calculations (this part of the code will be equivalent to the current VASP 5.2 implementation). All of the OpenMP directives will be enclosed in appropriate pre-processor directives so the build type can be selected in the VASP makefile.

Table 3: Profile of VASP subroutines at 64 and 256 cores (fully populated nodes) for the benchmark.

	64 Cores		256 Cores	
Routine	% Time	Time (secs)	% Time	Time (secs)
USER	67.7	467.2	41.2	326.7
eddav_	20.9	144.1	25.8	111.4
nonlr_rpro1_hf_	5.9	10.6	1.9	15.2
wpbe_wpbe_spline_	5.1	35.2	1.3	10.0
fassm_	5.0	34.8	1.6	12.6
ipassm_	4.8	33.4	1.6	12.4
fft3d_	4.6	31.9	1.5	11.9
pawfock_rs_coulomb_green_func_	4.2	29.1	1.5	11.9
racc0_hf_	4.1	28.2	1.3	10.5

Original Work Packages (WPs)

The key areas of improvement to the hybrid-functional part of VASP are in the areas of load-imbalance in the PAW calculation and parallel performance of the hybrid-functional specific routines. The work packages below detail how we planned to produce an optimised version of the VASP hybrid-functional code that includes OpenMP parallelisation.

WP1: OpenMP parallelisation of PAW routines – 4 months

Tasks:

• OpenMP parallelisation of the PAW calculation controlled by the “set_dd_paw” subroutine.

Output:

• Optimised hybrid (MPI + OpenMP) version of PAW calculation.

• Improved load-balancing of VASP code where the number of parallel units exceeds the number of atoms in the system.

• Scaling of PAW routines for benchmark cases (supplied by MCC) to at least four times as many cores as the original pure MPI code.

WP2: OpenMP parallelisation of hybrid DFT routines – 5 months

Tasks:

• OpenMP parallelisation of key VASP hybrid DFT routines: “eddav” “rpro1_hf”, “*_spline”, “fpassm”, “ipassm”, etc.

Output:

• Optimised OpenMP versions of key VASP hybrid-DFT routines

• Improved scaling of OpenMP augmented routines for benchmark cases (supplied by MCC) to at least four times as many cores as the original pure MPI code.

• Profile of new code identifying further routines for OpenMP parallelisation.

WP3: OpenMP parallelisation of remaining bottlenecks – 2 months

Tasks:

• OpenMP parallelisation of remaining VASP routines needed for efficient code operation.

Output:

• Mixed-mode version of the VASP hybrid DFT code that scales for benchmark cases (supplied by MCC) to at least four times as many cores than the original pure MPI code.

WP4: Dissemination – 1 month

Tasks:

• Distribute new code to MCC members and other HECToR VASP users.

• Publicise new code

• Feed developments back to VASP developers

Output:

• Report on performance of mixed-mode VASP code

• Installation of new mixed-mode, hybrid DFT version of VASP as a centrally available package on HECToR.

Key Objectives

In summary, the key objectives were to introduce OpenMP parallelization into the PAW and hybrid-DFT calculation parts of the VASP 5 code.

Changes to the original proposal

New test cases

Half way through the project, the author visited the MCC at UCL. At this meeting, it was decided to replace the test cases described in this Project's Proposal with four new test cases. These new cases require the use of a more recent version of VASP as the initial version employed, namely v5.2.11, did not contain the necessary functionality. A more recent version of VASP, namely 5.2.12, was therefore obtained and installed on HECToR phase3 (Cray XE6).

The change of version required a change of compiler and its associated libraries on HECToR. Specifically, the PGI compiler, 12.3.0, was replaced by the GNU Fortran compiler, 4.6.3. This change of compiler illuminated some minor incompatibilities in VASP which were rectified.

The four simulations were provided by David Scanlon and Stephen Shevlin and are hereafter referred to as Scanlon_1, Scanlon_2, Shevlin_1 and Shevlin_2.

The new test cases represent the simulations that MCC wish to carry out, in that they have the characteristics of the large simulations, and yet run in a short amount of time. This is to aid rapid code development.

Benchmarking the new test cases

The four new test cases were ported to HECToR and benchmarked using the Cray PAT profiling tool to provide a baseline to permit comparison.

For the test cases, we employ VASP 5.2.12 using the GNU compiler on HECToR phase3 (Cray XE6).

The Scanlon Test Cases

The first two simulations, namely Scanlon_1 and Scanlon_2, are two different simulations. They were designed to exercise the PAW and new hybrid DRF routines, respectively. Bot simulations were configured to run using 96 cores whilst under-populating the nodes by 50%. In other words, 192 cores are requested and each MPI task is placed on every second core. Under-populating nodes, in this manner, permit each MPI task to access more memory than tasks running with fully packed nodes.

Gamma version of VASP

When investigating the Scanlon simulations, it was found that both simulations employed only a single k-point at the origin, i.e. they contained a single 'gamma point'. Such cases can employ a special form of the VASP executable which has been compiled employing the pre-processor flag `-DwNGZhalf`. This can significantly reduce the execution time. For Scanlon_1, for instance, running using the standard form of VASP takes 158 seconds, whilst running with the Gamma version, the test takes 129 seconds: an increase of speed of over 18%. For Scanlon_2, the difference is more striking: the standard version runs in 2666 seconds whilst the gamma version runs in 1688 seconds: an increase of speed of over 36%.

Results from Cray PAT

The Cray PAT profiling tool was employed to profile both the Scanlon_1 and Scanlon_2 test cases, using a version of VASP compiled for a single k-point simulation.

The results of the profiling are summarised in the following Table 4.

Table 4 Cray PAT profile of the Scanlon_1 test case, 96 MPI tasks, 50% under-population

Routine		% Time
MPI		44.6
	MPI_Allreduce	31
	MPI_Bcast	7.7
	MPI_Barrier	2.3
	MPI_Alltoall	2.3
USER		40
	paw_fock_rs_coulomb_green_func	5.8
	Fpassm	2.7
	dllmm_kernel	2.5
	dgemv_t	2.4
	Rhosyg	2.2
	Ipassm	2.2
ETC		15.4
	No single routine >1%	-

Table 5: Cray PAT profile of the Scanlon_2 test case, 96 cores in 50% under populated nodes

Routine		% Time
USER	68.5	
	fpassm	19.6
	ipassm	17.1
	dllmm_kernel	6.2
	dgemv_n	2.8
	rholm_kernel	2.7
	fft3d	2.4
	dcopy_k	2.2
	hcomb	1.6
	rcomb	1.4
	dgemm_kernel	1.2
	rhosyg	1.1
	dgemv_t	1.1
	racc0_hf	1.0
MPI	17.6	
	MPI_Barrier	8.1
	MPI_Allreduce	7.0
	MPI_Bcast	1.7
ETC	9.4	
	fock_MOD_fock_acc	1.8
	nonlr_MOD_rpro1_hf	1.3
	augfast_MOD_calc_dlmm_trans	1.1
	remainder_piby2	1.0

When comparing the two tables, it is clear that Scanlon_1 suffers from poor load-balance, due to the time spent in MPI routines. Further, the expensive PAW routine, namely `pawfock_rs_coulumb_green_func` from `pawfock.f`, constituted 5.8% of the total run time for the Scanlon_1 test case, whereas it does not appear in the profile for Scanlon_2 test case (implying it constitutes less than 1% of the total run time).

For the Scanlon_2 test case, it is clear that the majority of execution time is taken up with performing the FFT, as the FFT employs the routines `fpassm`, `ipassm`, `fft3d`, `dcopy_k`, `hcomb`, `rcomb` and `rhosyg`. These FFT routines are since been replaced with calls to the FFT3 library (see the section WP2: OpenMP parallelisation of hybrid DFT routines). The routine `dgemv_n` is a call to the ScaLAPACK routine and is optimised for this platform. Thus, the remaining routines, namely `dllmm_kernel` and `rholm_kernel`, and, to a lesser extent, `dgemm_kernel` and `racc0_hf`, are the routines which may benefit from OpenMP directives in an attempt to optimise the code further.

It is interesting to note that the routines ear-marked for OpenMP directives in the original proposal we also included the `dgemm` and `rholm` kernels.

Due to lack of time, it was decided to focus on the Scanlon_2 test case. The associated INCAR file can be found in the

Appendix.

The Shevlon Test Cases

The Shevlon_1 test case was found to have multiple k-points, thus we employ the standard version of VASP (as opposed to the Gamma version).

The test case Shevlon_1 takes around 55 minutes to run, and uses fully packed nodes employing 64 cores in total.

Results from Cray PAT

The following table contains a summary of the Cray PAT profiling tool.

Table 6 Cray PAT profile of the Shevlon_1 test case, 64 tasks, fully packed nodes

Routine		% Time
MPI		71.6
	MPI_Allreduce	56.2
	MPI_Bcast	10.1
	MPI_Barrier	4.6
USER		24.3
	zlasr	5.2
	msort_with_tmp	1.3
	exp	1.1
ETC		4.1
	fock_MOD_fock_acc	1.3

From this table, it is clear that MPI takes majority of time, suggesting that load-imbalance is this test case's most significant characteristic. The routines which spend most time in MPI_Allreduce are, in order, `elim`, `david_MOD_eddav`, `fock_MOD_fock_acc`, `m_sum_z` and `MPI_Allreduce` itself. The *user* routines which take longer than 1% of the total time, namely zlasr, msort_with_tmp and exp, are not part of the VASP source code but are found in LAPACK, glibc, etc. As such, there is no single VASP routine/loop which presents itself as a candidate for optimisation via OpenMP.

The Shevlon_2 test case was found to be so similar to the Shevlon_1 test case that it was decided to replace this with another test. This new test case provided difficult to get running on HECToR and, due to operational constraints, this final test case was not included in this project's test suite.

Conclusions from baseline profiling

The two Scanlon test cases were found to behave as designed, i.e. the PAW routines and the hybrid DFT routines where exercised; however, due to time restrictions, only Scanlon_2 test case was considered further. The INCAR file associated with Scanlon_2 can be found in the

Appendix.

On the other hand, the two Shevlin simulations proved to be harder to locate candidates for OpenMP directives and, as such, were both dropped from this project's test suite due to operational constraints.

It should be noted that the change of target VASP version, along with the necessary compilers/libraries and compiler options, and the changing of the test cases for benchmarking against added a significant delay to the overall project.

VASP on GPU clusters

During the latter half of this project, a new version of VASP 5.2.12 was kindly made available by Carnegie Mellon University, where the authors of this new code had introduced C and CUDA into VASP to introduce parallelism to the hybrid DFT part of VASP to exploit GPU clusters [5].

This hybrid DFT part of VASP is, of course, exactly the same part of the VASP code that this project was attempting to parallelise using OpenMP statements inserted to the Fortran code to addresses the poor performance of the new hybrid DFT/HF functional.

The GPU version has provided alternative C/CUDA code within the main `fock_acc` and `fock_force` routines within VASP, which can be used for a particular limited set of simulations. If the simulation in question cannot employ the new GPU code, then the code gracefully resorts to the original code. Simulations that can exploit the GPU code provided that the pre-processor flag `gammareal` is not set and the following logical statements are all `TRUE`.

- `(.not. WDESK%GRID%LREAL)`
- `(WDESK%NRSPINORS == 1)`
- `(maxval(abs(KPOINTS_FULL%TRANS(:,:))) <= TINY)`
- `(WHF%WDES%LOVERL)`
- `(FAST_AUG_FOCK%LREAL)`
- `(GRIDHF%MPLWV == GRIDHF%RC%NP)`
- `(GRIDHF%RC%NP == WDESK%GRID%RL%NP)`
- `(LSIF)`
- `(WHF%WDES%COMM_INB%NCPU == 1)`
- `(WHF%WDES%NPRO_TOT == WHF%WDES%NPRO)`
- `(.not. WHF%WDES%LNONCOLLINEAR)`

The GPU code parallelises over the k-point bands: this was achieved by restructuring code within the `fock_acc` and `fock_force` routines thereby changing the algorithm within VASP so that the k-point bands of the new hybrid DFT calculation are no-longer independent.

In the original code, the loop over k-point bands contains loops over each of the ion types and, in turn, these ion type loops contain loops over the number of ions of each type. In the new GPU version, the k-point band loop has been split into multiple k-point loops which remove the k-point band interdependency present in the original code. This major code refactoring (both in terms of algorithmic changes and using C rather than Fortran) permits the introduction of CUDA to exploit GPUs.

It was realised that this version of VASP for GPU clusters could be used as a reference for producing an F90/OpenMP version of the same routines, since the method of introducing CUDA into a code is strongly similar to that of introducing OpenMP, in that both CUDA and OpenMP require a high

degree of small-grain parallelism. In other words, work suitable for GPUs is work suitable for OpenMP threads

In light of the delays in the project arising from the change of benchmark cases; the introduction of a new version of the VASP code; and the associated change in compilers and compiler options it was decided to replace WP3, namely the piecemeal introduction of OpenMP around expensive loops, with the creation of a set of completely new routines written in Fortran/OpenMP that introduced OpenMP parallelism at a higher level in a similar way to the GPU-enabled version of the code. Moreover, these new set of routines would involve restructuring the existing VASP algorithm to that of the structure employed in the GPU-enabled version, i.e. creating independent k-point band calculations and, as a direct consequence, increasing the total memory required. It was also noted that the time assigned to WP3 might not be sufficient, thus WP1, the parallelisation of the PAW routines, would be undertaken if, and only if, the new WP3 is completed.

Work done against Work Packages

WP1: OpenMP parallelisation of PAW routines

Not attempted due to prioritisation of the new WP3 (see below).

WP2: OpenMP parallelisation of hybrid DFT routines

This project's proposal demonstrated that FFTs were very expensive. Indeed, they are employed by *both* pure *and* hybrid DFT and thus any improvement benefits would benefit **all** VASP users.

As per workplan, we attempted to introduce OpenMP into local FFT routines. Unfortunately, this was found to be too complex. We therefore recompiled the code using the FFTW 3 library for the FFT functionality as FFTW 3 includes a threaded version of the library which could potentially be exploited when using threads in other parts of the code. The results below (Table 7) reveal that using the non-threaded version of FFTW in preference to the default VASP internal FFT routines yields a performance improvement of between 14% and 35% depending on the number of cores used.

Table 7: Performance comparison between VASP 5.2.12 using internal FFT routines to using the FFTW 3 routines. All tests on the 'bench2' benchmark on HECToR phase2b with fully-populated nodes.

	Runtime (secs)		
Cores	**Internal FFT**	**FFTW 3**	**Improvement**
48	1719	1509	1.14×
96	1173	866	1.35×
192	1140	879	1.30×

Using more than one thread in the FFTW routines does not improve the performance however (Table 8). In fact, it reduces performance drastically. It is likely that the dimensions of the vectors that are to be transformed are too small for the threaded version of the library to be able to exploit effectively.

Table 8: Performance comparison between VASP 5.2.12 using single-threaded FFTW 3 routines to using multithreaded FFTW 3 routines. All tests on the 'bench2' benchmark on HECToR phase2b with fully-populated nodes. 'Cores' indicates total cores used – threads × MPI tasks.

	Runtime (secs)		
Cores	1 thread	2 threads	Improvement
48	1509	2863	0.53×
96	866	2752	0.31×
192	879	5280	0.17×

The overhead introduced by spawning the threads outweighs the gains by performing the transformation in parallel. Further performance improvements in the FFT calls could possibly be gained by:

1. Saving and reusing the plan for the transform rather than generating and destroying it every time.
2. Performing more than one transform simultaneously by placing the transform inside a threaded loop over the transforms.

Neither of these possibilities was explored due to time constraints on the project.

Threaded Cray LibSci library performance

We also investigated the possibility of using a threaded version of the Cray LibSci library (which provides the BLAS, LAPACK, BLACS and ScaLAPACK linear algebra routines used in VASP) rather than the standard single threaded version. The results (see Table 9), however, were disappointing with the threaded versions actually increasing runtime compared to the non-threaded versions at most core counts. As for the threaded FFTW routines above, the overhead introduced by spawning the threads generally outweighs the gains by performing the transformation in parallel. The exception is at 192 cores where we see a 23% speedup when using the multithreaded library. This result suggests that there may be gains in using multithreaded LibSci at higher core counts. As multithreaded LibSci will be enabled automatically when using more than one thread on HECToR, these gains should be automatically available to the user.

Table 9: Performance comparison between VASP 5.2.12 using single-threaded LibSci routines to using multithreaded LibSci routines. All tests on the 'bench2' benchmark on HECToR phase2b with fully-populated nodes and used the single-threaded FFTW 3 library. 'Cores' indicates total cores used – threads × MPI tasks.

	Runtime (secs)		
Cores	1 thread	2 threads	Improvement
48	1509	1645	0.92×
96	866	1263	0.69×
192	879	712	1.23×

WP3: OpenMP parallelisation of remaining bottlenecks

Using the test cases described in the original proposal, the subroutines DLLMM and RHOLM kernels were found to be expensive, alongside the FFTs and the PAW routines.

OpenMP was introduced into source for the DLLMM_KERNEL subroutine, but initial results demonstrated that this had very poor performance for the "bench2" test case.

Whilst these routines are called many times, the loops within these routines which can be parallelised using OpenMP are simply not large enough to be efficient. One solution would be to raise the occurrence of the OpenMP statement further up the call tree; however, it was felt that, the original workplan would not provide the speed up required by simply introducing OpenMP without algorithmic changes.

Original WP3 with new test cases

The work done in WP3 was copied to the new version, namely v5.2.12, within the aug_fast.F file around the main loops within DLLMM_KERNEL and RHOLM_KERNEL. This new code was timed using the second of the new test cases, namely Scanlon_2.

Performance results of mixed-mode version of VASP

In this section, we present the results of running the Scanlon_2 simulation using either 48 or 96 cores, using 50% under-populated nodes, with none, one or two OpenMP threads per MPI task. Here we employ HECToR phase3 (a Cray XE6), using the GNU gfortran compiler (v4.6.3) and FFTW3 (v3.3.0.0).

Table 10 Performance comparison between VASP 5.2.12 (single-threaded LibSci routines) with the standard VASP and when recompiling to exploit the single gamma-point. All tests on the Scanlon_2 test case on HECToR phase3 with fully-populated nodes and used the single-threaded FFTW 3 library. 'Cores' indicates total cores used – MPI tasks.

	Runtime (secs)		
Cores	**Standard**	**Gamma**	**Improvement**
48	4147	2398	1.7×
96	2526	1726	1.5×
Efficiency	82%	72%	-

From Table 10, we can see that when doubling the number of cores, both the standard and the gamma versions of VASP scale well, but that the gamma version gives at least 1.5 speed up.

Table 11 Performance comparison the standard and gamma versions of mixed-mode VASP 5.2.12 (includes multi-threaded LibSci routines. Executions times (and parallel efficiency)

	Runtime (secs) (parallel efficiency)					
tasks x threads	48 x 1	48 x 2	48 x 4	96 x 1	96 x 2	96 x 4
Standard	4976 (100%)	3429 (73%)	4662 (27%)	3063 (100%)	2189 (70%)	2702 (28%)
Gamma	3308 (100%)	2371 (70%)	2854 (29%)	2077 (100%)	1534 (68%)	1697 (31%)

There are a number of important results that we may deduce from above. Firstly, comparing the times in Table 10 and Table 11, we can see that when running the simulations with just one OpenMP thread, the overall time is increased when compared to running with no OpenMP threads whatsoever: switching on one OpenMP threads makes the Standard code run a factor of 0.8 slower, and makes the Gamma simulation run a factor of 0.7 slower, at most.

To run with one OpenMP thread, we set the environment variable OMP_NUM_THREADS=1. This, not only, sets the number of OpenMP threads per MPI task to 1, but also ensures the Cray LibSci library runs with thread.

If we consider Table 11 again, and calculate the parallel efficiency of the OpenMP parallelism based on running one thread per MPI task, then we can see that the code is not efficient when running on more than 2 threads per MPI task, but does run at around 70% efficiency when running 2 threads per MPI task. Alternatively, if we calculate the parallel efficiency of the OpenMP parallelism based on running no threads per MPI task, then we can see that the code runs at around 50% efficiency when running 2 threads per MPI task.

Cray PAT profiling results for Scanlon_2

The Scanlon_2 simulation was run three times using the Gamma version of VASP 5.2.12, and profiled using the Cray PAT profiler.

The code was run with 96 MPI tasks, using 50% under-populated nodes with no OpenMP threads, one OpenMP thread per MPI task and two OpenMP threads per MPI task. Below we present the percentage times, and the actual time in seconds, for the USER, MPI and ETC routines as given by the PAT profiler.

Table 12 Cray PAT profile of VASP gamma version, no threads, 96 MPI tasks, 50% under-populated nodes

Routine		% Time	Time (secs)
USER		68.5	1183
	fpassm	11.0	190
	ipassm	10.6	183
	dllmm_kernel	9.9	171
	dgemv_t	7.9	136
	rholm_kernel	5.5	95
	dgemv_n	4.2	73
	pw_charge_trace	2.5	43
	vhamil_trace	2.3	40
	dlasr	1.6	28
	fft3d	1.4	24
	rcomb	1.2	21
	dgemm_kernel	1.2	21
	racc0_hf	1.1	19
	hcomb	1.1	19
MPI		22.4	387
	MPI_Allreduce	11.0	190
	MPI_Bcast	9.2	159
ETC		9.1	157
	nonlr_MOD_rpro1_hf	2.0	35
	fock_MOD_fock_acc	1.7	29
	remainder_piby2	1.3	22
	augfast_MOD_calc_dllmm_trans	1.3	22

From Table 12, it can be seen that the profiling results of running with no OpenMP threads differs from the baseline results, presented in Table 5. Despite great care to employ the same GNU Programming Environment on HECToR, and the same FFT3 library, the “perftools” module employed

when producing the baseline results was no longer available. Further to this, it maybe that, despite employing the same Programming Environment, certain runtime libraries, etc, will have changed in the interim and this changing environment explains why the profiles differ.

Specifically, three routines have appeared in the profile, that were not present in the baseline profiles. These are `pw_charge_trace` and `vhamil_trace` from the VASP source file `hamil.F`, and `dlasr` from LAPACK.

Table 13 Cray PAT profile of mixed-mode VASP gamma version, one OpenMP thread per task, 96 MPI tasks, 50% under-populated nodes

Routine		% Time	Time (secs)	
USER		71.1	1477	
	fpassm	10.3		214
	ipassm	10.3		214
	dgemv_t	9.5		197
	dllmm_kernel	9.2		191
	dgemv_n	5.2		108
	rholm_kernel	4.9		102
	pw_charge_trace	3.0		62
	vhamil_trace	2.7		56
	dlasr	2		42
	fft3d	1.7		35
	rcomb	1.4		29
	racc0_hf	1.4		29
	dgemm_kernel	1.1		23
	hcomb	1.0		21
MPI		19.2	399	
	MPI_Allreduce	8.3		172
	MPI_Bcast	8.1		168
	MPI_Barrier	1.1		23
ETC		9.1	189	
	nonlr_MOD_rpro1_hf	2.0		42
	fock_MOD_fock_acc	1.7		35
	remainder_piby2	1.3		27
	augfast_MOD_calc_dllmm_trans	1.3		27

When comparing Table 12 with Table 13, we can see that the overall time has increased. The order of the USER routines, and their percentage of the total time, is roughly the same: the wall clock time has increased, uniformly, for all the USER routines.

The reason for the increase in time for computation is solely due to setting OMP_NUM_THREADS=1 in the batch script. Setting this environment variable to one ensures the Cray LibSci library runs with a single thread. The ETC (which include the LibSci library calls) and MPI routines take slightly longer, but not to a significant degree. As expected, the routines with OpenMP statements, namely dllmm_kernal and rholm_kernel, take longer to run, however, there are USER routines that contain

no OpenMP statements which exhibit a significant increase in execution time, and that this warrants further investigation.

Table 14 Cray PAT profile of mixed-mode VASP gamma version, two OpenMP thread per task, 96 MPI tasks, 50% under-populated nodes

Routine		% Time	Time (secs)	
USER		75.5	1158	
	dgemv_t	11.8		181
	gomp_barrier_wait_end	11.0		169
	Fpassm	7.1		109
	dllmm_kernel	7.1		109
	Ipassm	6.6		101
	dgemv_n	6.5		100
	rholm_kernel	3.6		55
	gomp_team_barrier_wait_end	3.2		49
	pw_charge_trace	2.3		35
	vhamil_trace	2.0		31
	dgemm_kernel	1.2		18
	Dlasr	1.1		17
	racc0_hf	1.0		15
MPI		17.7	272	
	MPI_Bcast	8.3		127
	MPI_Allreduce	7.6		117
ETC		6.9	106	
	nonlr_MOD_rpro1_hf	1.6		25
	remainder_piby2	1.1		17
	fock_MOD_fock_acc	1.1		17

When running with two OpenMP threads, we can see that there are now two OpenMP routines which take significant effort, namly gomp_barrier_wait_end and gomp_team_barrier_wait_end. This suggests that the two loops within dllm_kernal and rholm_kernel are not well balanced. This, in turn, explains why this mixed-mode code does not scale to more than two OpenMP threads per MPI task.

When running with no OpenMP threads, one thread per task and two threads per task, the wall-clock times for the dllmm_kernel is 171s, 191s and 109s, respectively. Thus, when doubling the number of threads per MPI task from one to two, we see a parallel efficiency of 88% for this routine alone.

When running with no OpenMP threads, one thread per task and two threads per task, the wall-clock times for the rholm_kernel is 95s, 102s and 55s, respectively. Thus, when doubling the number of threads per MPI task from one to two, we see a parallel efficiency of 93% for this routine alone.

Memory usage

Lastly, let us now consider the memory use of these three cases. In the following small table, we present the highest 'memory high water mark' for each case over all cores.

Table 15 Memory High Water Mark for mixed-mode VASP running Scanlon_2 test case

Memory High Water Mark			
Number of OpenMP threads per MPI Task	None	One	Two
Memory High Water Mark over all cores (Mbytes)	471.9	471.8	488.0

From Table 15, we can see that setting OMP_NUM_THREADS does not alter the amount of memory required significantly. However, it is interesting to note that, when running two OpenMP threads per MPI task, when running one task on every second core, the high water mark does not increase by a large amount (only 3.4%) as expected.

Summary of profiling results

For the Scanlon_2 test case, the Gamma version of VASP should be used, rather than the default standard version of VASP: the Gamma version if 1.5 times faster when using 96 MPI tasks.

When running this test case in mixed-mode, one should employ as many MPI tasks as VASP permits, given that it scales, and then one may then run two OpenMP threads per task. This ensures that, when running with one MPI task on every 2nd core, the alternate cores are busy running the second OpenMP thread. Further, comparing running no threads per task with two threads per task, we see a parallel efficiency of 50%.

Whilst the resultant mixed-mode code is not efficient for more than two OpenMP threads per MPI task, this implementation does permit the use of a larger number of cores than before. Without OpenMP, users can only employ up to a fixed number of cores for the hybrid simulations, but now the mixed-mode version permits users to employ twice as many cores as before. Thus, new science is now possible, as more memory per task is now available, but at a cost of inefficiency.

For *this* version of the mixed-mode VASP, the next step is to improve efficiency by moving OpenMP parallel regions to a higher level, and to introduce OpenMP statements around declaration statements to ensure that the correct threads are aligned to the matching data structures (so-called "First Touch"). We can also consider introducing OpenMP directives into `pw_charge_trace` and `vhamil_trace`. Finally, OpenMP directives could be introduced to the PAW routines. This will not benefit the Scanlon_2 test case specifically, as the PAW routine's execution time was very small, but will benefit other VASP simulations. Having said this, it should be noted we believe the most fruitful path forward is to employ the mixed-mode version based on the GPU code (see below).

New WP3: description of work undertaken

The process of creating new Fortran/OpenMP code, using the new GPU code as inspiration, was initially considered a large but straight-forward task, but proved to be far more complex and took much longer than expected.

Many of the routines called in the GPU code shared or had similar names to extant Fortran routines within VASP, and it was thought that these extant routines could simply be called instead of the C routines. However, this was found not to be the case: the GPU routines did not match the Fortran

routines but were similar to loops within either the associated Fortran routines or else from very different routines.

Further to this, the data layout of the GPU version is quite different to that of the VASP code, primarily to facilitate an efficient GPU implementation: the GPU code introduces a number of very large data structures. This is not only to permit the efficient use of the GPUs but also due to a direct consequence of the algorithmic change employed by the GPU code, specifically, to permit the parallelisation over k-point bands, the code must retain the wave-function values for all k-point bands, whereas this is not necessary in the original VASP code. In other words, the GPU is able to parallelise over k-point bands at the expense of increased memory use.

The logic behind the large data structures employed by the GPU code was copied directly in the new Fortran routines, using types rather than structures. Whilst this initially increases the memory footprint, it benefits the process of producing this new code as the new code and the GPU code can be compared directly, line by line, when debugging. Thereafter, once the code was running correctly, some of the intermediate large data structures can be removed. Further, some of the other large data structures could be replaced with the extant data structures of VASP, although these extant data structures would themselves have to be extended to retain all the k-point band wave functions.

All routines that were run on the GPU have been rewritten in F90 and can now be parallelised using OpenMP. Indeed, the most basic of OpenMP implementations are now included in the new Fortran routines. Further the routines which pack and unpack the data to and from the new data structures is also complete. The source file fock.F has over 300 new lines, and there is a new file named fock_openmp.F which is 1713 lines long.

The routine `fock_acc_omp` (which mimiks the computation in `fock_acc_cu` of the GPU code, which runs on the host rather than the GPU card) is also now complete, including all its child routines. However, due to lack of time, the larger routine `fock_force_omp` has not been completed. This latter routine employs ten “directions” whilst `fock_acc` employs just one. This multi-directional code section of `fock_force_omp` is complete; however, time did not permit the two most awkward child routines, namely `eccp_nl_fock_sif_k` and `eccp_nl_fock_forhf_k`, to be completed correctly. As such, the resultant VASP only runs the packing and unpacking routines, and the `fock_acc_omp` routine, and the code avoids running incomplete `fock_force_omp` routine by simply running the extant routines.

Testing the new code

The GPU version of VASP can be ported to HECToR GPU and it will be possible to debug the new code by a direct comparison. The GPU version had been ported to the HECToR GPU cluster using the PGI compiler; however, an upgrade to that compiler illuminated a well-known incompatibility in the VASP code and the GNU compiler had to be used in its stead. The installation of the GPU version of VASP using the GNU compiler was not completed as the GNU BLACS routines were not found and locally it was felt that a working version of the new code should take priority.

Despite a lack of an operational GPU version, we were able to debug the new code using the standard VASP code using a simulation which employs a single k-point band, specifically the “bench2” test case.

Current Status of the Mixed-Mode Version of VASP 5.2.12

We now have two version of the mixed-mode version of VASP: firstly, the basic VASP 5.2.12 version with OpenMP introduced in two places, parallelising the main loops within the two subroutines `DLLMM_KERNEL` and `RHOLM_KERNEL` in the file fast_aug.F; and secondly, a new version of VASP.5.2.12, with new code based on the GPU version, where the `force_acc` routines are now complete, but two of the many routines associated with `fock_force` are currently incomplete. The version that was inspired by the GPU version needs further work to complete the two incomplete routines, and to iron out the final bugs but we feel that the majority of code is in place to provide an efficient hybrid implementation.

Conclusions

The key objective of this project was to produce an efficient mixed-mode version of VASP. This has been partially met, in that the mixed-mode version is not particularly efficient, however, this new version can run at 50% efficiency on twice the number of cores than before and, as such, new science can now be undertaken that was previously prohibited by VASP itself.

This new mixed-mode version of VASP 5.2.12 has been made available to the MCC. The code employs FFTW instead of the internal FFT routines, and two loops have been parallelised using OpenMP.

For the Scanlon_2 test case, we gained a factor of 1.5 speed-up by using the Gamma version of VASP. Further, this test case can employ two OpenMP threads per MPI task with a parallel efficiency of around 50%, when compared to running no OpenMP threads per task. (Alternatively, the efficiency is around 60% when running two OpenMP threads per MPI task compared to one OpenMP threads).

This version will also be made available centrally on HECToR. This is especially interesting as the increase in speed due to the introduction of FFTW benefits all users of VASP and not just those interested in the new hybrid DFT/HF routines. All HECToR users will be alerted to this new mixed-mode version once it is made available.

This new version constitutes an investment in the future, in terms of both a saving of cost when running the code using the new FFTW routines rather than the older internal FFT routines, and in terms of new science that can now be run when using the mixed-mode DFT/HF routines.

Perhaps more importantly, significant inroads have been achieved by a major piece of work to implement another new version of VASP 5.2.12, where new F90+OpenMP routines have been written from scratch using a new GPU version of VASP as inspiration. This code is currently incomplete but does represent a significant step towards what appears to be an essential refactoring of the DFT/HF routines.

Lastly, it should be noted that during the closing stages of this work, a significant new version of VASP was released, namely 5.3.

Future Work

Currently, it is recommended that users employ OMP_NUM_THREADS=1 when running VASP and initial investigations confirm that this is indeed the case. However, for a particular VASP simulation employing the internal FFT routines, it was found that introducing the environment variable OMP_NUM_THREADS=1 slowed down USER routines that contained no OpenMP statements: this warrants further investigation.

For the running mixed-mode version, the work to introduce OpenMP into the PAW routines remains outstanding and should be addressed in the future to avoid the inherent load imbalance. The routines `pw_charge_trace` and `vhamil_trace` should also be investigated with a view to introducing OpenMP. Lastly, the OpenMP work can be optimised further but looking at "first touch" parallelisation, to ensure data elements are aligned to the same thread throughout execution.

In the other mixed-mode version, inspired by the GPU version of VASP, the two routines called by `fock_force_omp` should be completed and, once the code is shown to be running correctly, the large intermediate data structures should be either reduced or removed altogether. Special care should be taken to note the increase in required memory, as this may have a negative effect on the size of future simulations

We are confident that this latter version will prove to be more efficient that the former version and, as such, its development work should take priority.

Acknowledgements

Special thanks are due to Andrew Turner at EPCC; and to the MCC group at UCL, in particular Scott Woodley, Stephen Shevlin, David Scanlon and Alekseys Sokols.

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.

Bibliography

[1] "OpenMP specification," [Online]. Available: http://openmp.org.

[2] "VASP website," [Online]. Available: http://www.vasp.at.

[3] "MCC website," [Online]. Available: http://www.ucl.ac.uk/klmc/mcc.

[4] "Optimising the performance of the VASP code on HECToR," [Online]. Available: http://www.hector.ac.uk/cse/distributedcse/reports/vasp01.

[5] M. Hutchinson and M. Widom, “VASP on a GPU: application to exact-exchange calculations of the stability of elemental boron,” *Con. Mat. Mtrl. Sci.,* 2011.

Appendix

Scanlon_2 INCAR file

```
SYSTEM =  MgO rock salt

 Startparameter for this Run:
   NWRITE =      2
   LPETIM=F    write-flag & timer
   ISTART =      1    job   : 0-new  1-cont  2-samecut

 Electronic Relaxation 1
   PREC   = normal
   ENMAX  = 300.00 eV
   NELM   =     10
   INIWAV = 1         random initial wavefunctions
   EDIFF = 1E-04
 Ionic Relaxation
   EDIFFG = -0.01     stopping-criterion for IOM
   NSW    =     0     number of steps for IOM
   NBLOCK =      1;   KBLOCK =     40    inner block; outer block
   IBRION =      1    ionic relax: 0-MD 1-quasi-New 2-CG
   ISIF   =      2    stress and relaxation
   IWAVPR =      1    prediction:  0-non 1-charg 2-wave 3-comb
   ISYM   =      1    0-nonsym 1-usesym
   LCORR  =      T    Harris-correction to forces
   ISMEAR =      0

!!Hybrid-DFT Calculations:
 LHFCALC = .TRUE. (Activate HF)
 PRECFOCK = FAST
 !!ENCUTFOCK = 0    (Reduced HF FFT grid--Cheaper, but loose
accuracy)
 !!NKRED   =   2    (Reduce k-grid: even k only)
 !!ALGO    = DAMPED (Dampen: IALGO=53: Metals/Insulators)
 ALGO    = ALL    (IALGO=58: Insulators--Converges best)
 TIME    = 0.30   (Timestep for IALGO5X)
 HFLMAX  = 4      (Max. quantum number: 4d,6f)
 HFSCREEN= 0.207  (Switch to screened exchange-HSE06)
 AEXX    = 0.25   (X% HF exchange)

 LASPH = .TRUE.
 LREAL = AUTO
 ROPT = 1E-03 1E-03
 ICORELEVEL = 1
 LVHAR = .TRUE.
```