HPCx CAPABILITY COMPUTING

Optimisation of VASP on HPCx

Gavin J. Pringle

EPCC

The University of Edinburgh

Edinburgh EH9 3JZ

Scotland, UK

December 20, 2004

Contents

1 Abstract **2**

2 Introduction **2**

2.1 General description of VASP . . . 2

3 How is VASP parallelised **3**

3.1 Pane-waves . . . 3

3.2 Bands . . . 3

3.3 Images . . . 3

3.4 Parallelising over *k*-points . . . 3

4 Running VASP on HPCx **3**

4.1 Versions of VASP . . . 3

4.2 Different simulations . . . 3

4.3 Different software stacks on HPCx . . . 4

4.4 Compiling VASP on HPCx . . . 4

4.4.1 Gotchas . . . 4

4.5 Running VASP on HPCx . . . 4

4.6 Run time parameters . . . 4

4.6.1 NPAR . . . 5

5 Profiling VASP **5**

5.1 Optimising NPAR . . . 5

5.1.1 Discussion of optNPAR . . . 6

5.2 Timing Results . . . 6

5.2.1 Profiling using xprofiler . . . 7

5.2.2 Profiling using MPI_Trace . . . 7

6 **Conclusions** 8
6.1 NPAR 9
6.2 Future Work 9
6.3 Numerical Libraries 9
6.4 Further profiling 9
6.5 Capability Incentives 9
6.6 General code improvement 10

7 **Appendix** 10

1 Abstract

An investigation into the performance of VASP on HPCx has been carried out. Changing the value of NPAR to be proportional to the number of processors employed at runtime can reduce the execution time, specifically, on 128 processors, we say a speed up of 8.4% on the 288-atom Fe simulation. The code has a flat execution profile, however, SGI has recently optimised the code for the Altix, and this code runs around twice as fast on HPCx. The simulation studied for the Capability incentives did not reach the qualifying scaling values, however, we are confident that another type of simulation would.

2 Introduction

This report describes an initial investigation into the performance of VASP [1] on HPCx. Three different problem sizes were provided [2] and tested, using two different versions of VASP running under two different HPCx software configurations, between August and November, 2004.

VASP, which stands for Vienna Ab-initio Simulation Package, is a large parallel code, written in FORTRAN 90 with MPI, and has over 90,000 lines of code. It is not a public domain code. Access to the code may be given by a request via the VASP website [1]

From a code optimising point of view, the code steps through time, and performs a Fast Fourier Transform, or FFT, and performs a matrix diagonalisation every timestep. The diagonalisation is done using ScaLAPACK's expert driver routine PDSYEVX, which solves the standard Eigenvalue problem for positive definite symmetric matrices.

2.1 General description of VASP

The following paragraph is taken from the VASP website [1] and gives a general description of what VASP is used for.

VAMP/VASP is a package for performing ab-initio quantum-mechanical molecular dynamics (MD) using pseudopotentials and a plane wave basis set. The approach implemented in VAMP/VASP is based on a finite-temperature local-density approximation (with the free energy as variational quantity) and an exact evaluation of the instantaneous electronic ground state at each MD-step using efficient matrix diagonalization schemes and an efficient Pulay mixing. These techniques avoid all problems occurring in the original Car-Parrinello method which is based on the simultaneous integration of electronic and ionic equations of motion. The interaction between ions and electrons is described using ultrasoft Vanderbilt pseudopotentials (US-PP) or the projector augmented wave method (PAW). Both techniques allow a considerable reduction of the necessary number of plane-waves per atom for transition metals and first row elements. Forces and stress can be easily calculated with VAMP/VASP and used to relax atoms into their instantaneous groundstate.

3 How is VASP parallelised

VASP is parallelised in 3 ways, and the method of parallelisation is controlled by altering parameters in the INCAR file (see Section 4.6.1 on NPAR). The code can be parallelised over 'plane-waves', 'bands' and 'images'.

3.1 Pane-waves

Plane-waves contain FFTs and a parallelisation over plane-waves can incur heavy communications.

3.2 Bands

Computations within bands are somewhat independent, and a parallelisation over bands can be more efficient, especially for HPC systems with a relatively slow interconnect.

3.3 Images

Computations over images are the most independent and can have minimal intercommunications between images. There are two forms of 'image' runs with VASP. The first involves computing a string of images and computing their interaction between every time step. The second involves computing a set of almost completely independent images, where the communications between images is very small. This last case is practically an embarrassingly parallel case.

The method of parallelisation is chosen *after* compilation via input files read at runtime.

3.4 Parallelising over k-points

It is interesting to note that VASP, unlike many computational chemistry codes, does *not* parallelise over so-called k-points. Indeed, in most cases, parallelising over k-points can introduce a very poor load-balance, unless the number of individual k-points far exceeds the number of processors, and a task-farm can be employed to even out the load balance.

If VASP was parallelised over k-points, then, for the VASP simulations currently of interest to VASP users on HPCx [2], then this load-balance would not be an issue. However, the value of k is usually very small. Indeed, for capability computing, where the number of atoms employed should be high, the number of k-points is set to one.

4 Running VASP on HPCx

4.1 Versions of VASP

Three versions of VASP were employed during this investigation: namely vasp.4.6, vasp.4.6.13.g and vasp.4.6.13.k. The code vasp.4.6.13.g was recently optimised by SGI for the Altix [3] This code runs around twice as fast as vasp.4.6 on HPCx on 128 processors (see Section 5.2 below). Version 4.6.13.k was also considered but found to contain bugs which caused it to crash on HPCx.

4.2 Different simulations

The problems given to the author were a 288-atom simulation of iron (denoted *Fe* below) and a 1000-atom simulation of aluminium (denoted *Al* below).

4.3 Different software stacks on HPCx

Over the course of profiling VASP, the underlying software stack on HPCx was updated from SP7 to SP9 (where SP stands for Service Pack). This update provides more advanced compilers and communication libraries from IBM.

4.4 Compiling VASP on HPCx

The gzipped tarball unpacks into two separate directories: the VASP code itself: `vasp`; plus all the routines from ScaLAPACK and LAPACK that are necessary for the code to run: `vasp.lib`. This feature adds portability, however, may miss updates of new releases of these two public domain libraries. However, it is straightforward to exchange the routines given within VASP for the routines installed on HPCx. Further, it is also straightforward to employ IBM's own (P)ESSL libraries, to replace the ScaLAPACK routines. (See Section 6.2 on Future Work)

4.4.1 Gotchas

- VASP fails at runtime if compiled with the re-entrant compiler, i.e. one must use mpxlf rather than mpxlf_r, as the
- VASP fails at run time if compiled in 64-bit mode and, therefore, must be compiled in 32-bit mode

It is felt that both code stability, portability and performance may be improved if the code is debugged to compile in 64-bit mode, using the re-entrant compiler.

4.5 Running VASP on HPCx

The standard LoadLeveler script was employed, with the following executable statements:

```
export MP_EAGER_LIMIT=65536
export MP_SHARED_MEMORY=yes
export MP_USE_BULK_XFER=yes
export MEMORY_AFFINITY=MCM
export TASK_AFFINITY=MCM
export MP_EUILIB=us
export TMPDIR=/hpcx/work/z001/z001/gavin/tmp
time poe /hpcx/home/z001/z001/gavin/VASP/rundir/vasp
```

There are 4 files which must be present in the directory where the job will run. These are INCAR, POSCAR, POTCAR and KPOINTS, and are described in the following Section.

4.6 Run time parameters

The behaviour of the code is governed by the following input files.

- INCAR: includes parameters such as NPAR
- POSCAR: gives the initial positions of the atoms
- POTCAR: gives the potential energy of the atoms
- KPOINTS: gives the *k*-point stencil

All the parameters contained in the INCAR file have defaults within the VASP code itself. To override the default, the parameter name and it's associate value will appear in INCAR. (Technically, INCAR could be empty).

4.6.1 NPAR

The parameter NPAR, set in INCAR, determines the method of parallelisation.

If NPAR=1, then the code is parallelised over plane-waves, which means the matrix diagonalisation is serial and the FFT work is parallelised. If NPAR=NCPU, where NCPU is the total number of processors employed at runtime, then the code is parallelised over bands, i.e. the matrix diagonalisation is parallelised and all the FFTs are serial. If NPAR equals some constant which lies between 1 and NCPU, then the code is parallelised over both plane-waves and bands. For example, if NPAR=2, then the matrix diagonalisation is parallelised over two processors while the FFT computation is parallelised over half of the processors.

5 Profiling VASP

5.1 Optimising NPAR

In this Section, we investigate how varying the value of NPAR affects the execution time. A range of different values of NPAR were tests for each of the two different software stacks (SP7 and SP9), the two different simulations (Al and Fe) and 2 codes (v.4.6 and v.4.6.13.g) for a range of fixed processor numbers.

Firstly, it was found that NPAR must be a factor of NCPU.

Secondly, we found that if NPAR<NCPU/32, then the ScaLAPACK diagonalisation routine, PDPOTRF, fails[1] at the Choleski Decomposition state. The routine PDPOTRF is called from PDTRTRI which, in turn, is called from PDSYEVX, the expert driver for solving the standard Eigenvalue problem for positive definite symmetric matrices. VASP allows the user to not use the ScaLAPACK routine, but use a diagonaliser internal to VASP. However, a brief investigation showed that, when NPAR=8 and NCPU=128, the code ran with and without ScaLAPACK routines and executed in 680 and 4638 seconds, respectively. Thus, when the internal diagonalisation routine was employed, the code slowed down by almost a factor of 7.

The execution times for various values of NPAR for SP9, Fe and v.4.6 is displayed in figure 1.

We define optNPAR as the value of NPAR which gives the minimum execution time for a given number of processors. The number of processors, NCPUs, the associated value of optNPAR and the resultant values of NCPUs/optNPAR are presented in table 1.

NCPUs	optNPAR	NCPUs/optNPAR
16	2	8
32	2	16
64	4	16
128	8	16
256	16	16
512	32	16

Table 1: Values of NCPUs, optNPAR and NCPUs/optNPAR, where NCPUs is the total number of processors and optNPAR is the associated value of NPAR which gives the fastest execution time

For all the tests done,

$$\text{optNPAR} = \frac{\text{NCPU}}{16} \qquad [1]$$

[1]The routine fails with INFO=2, the leading minor of order 2, A(IA:IA+1, JA:JA+1) is not positive definite, and the factorisation could not be completed.

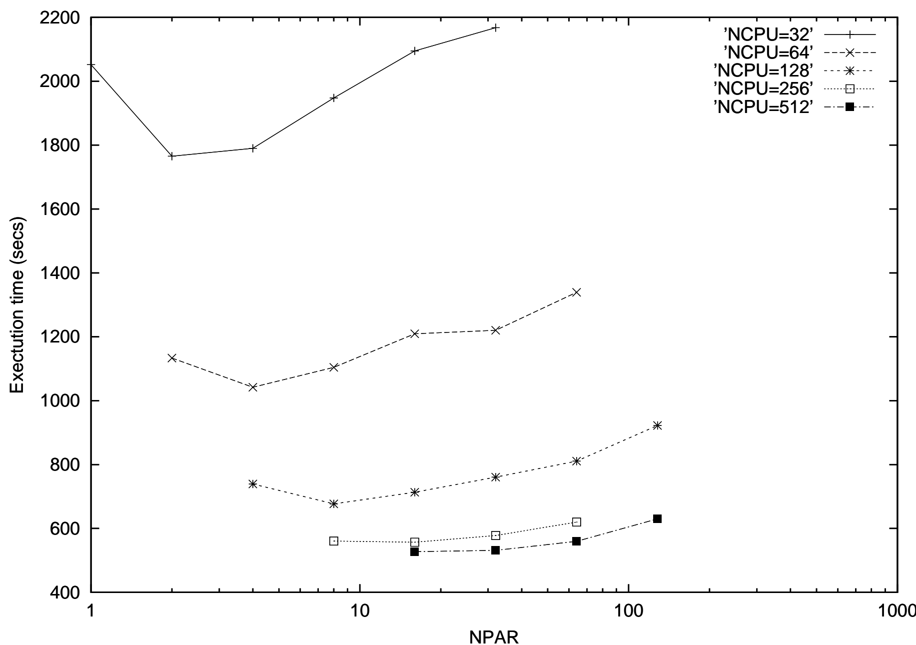

Figure 1: Execution times for various values of NPAR, for a varying numbers of processors, running SP7, version 4.6, for the smaller Fe simulation (288 atoms)

holds true, except for when using SP9 and v.4.6 with Al, where the execution times for various values of NPAR are shown in figure 2. Here, the value of optNPAR did not have a linear relationship with NCPUs. However, for the same problem, using v.4.6.13.g, the relationship (1) is once more applicable.

5.1.1 Discussion of optNPAR

It would appear that, for HPCx, the optimum value of NPAR is such that NPAR=NCPUs/16, provided that NCPUs> 16. Using the terminology of VASP, this means there are 16 *nodes* and NCPUs/16 *groups*.

One would reasonably expect that the value of optNPAR would be such to force each FFT to keep within a single frame of HPCx, where each frame as 32 processors, thus NCPUs/optNPAR=32. However, we have found that optNPAR=NCPUs/16, which means there are two groups of FFTs per node.

The default value of NPAR was 4. For NCPUs=16, 32, 64 and 128, all the FFT groups will communicate within an HPCx node, however, when NCPUs=256 or 512, then the FFTs would communicate between nodes and the code would be expected slow down. However, when NCPUs=256 or 512, and NPAR=4, the code fails at run-time due to a restriction in the ScaLAPACK routine, as stated above.

5.2 Timing Results

Using optNPAR, i.e. NPAR=NCPU/16, we ran both v4.6 and v4.6.13.g codes, using both SP7 and SP9, for both the Fe and Al simulations. The times are presented in figure 3.

Basically, the VASP code scales (gets faster as the number of processors increases) to around 256 processors and, thereafter, the code's execution time starts to get slower.

It is particularly interesting to note that the larger simulation, Al (1000-atoms) doesn't scale better than the smaller simulation, Fe (288-atoms). It is expected that if a larger simulation is employed, then the ratio of computation to communicating increases and, thus, the code will

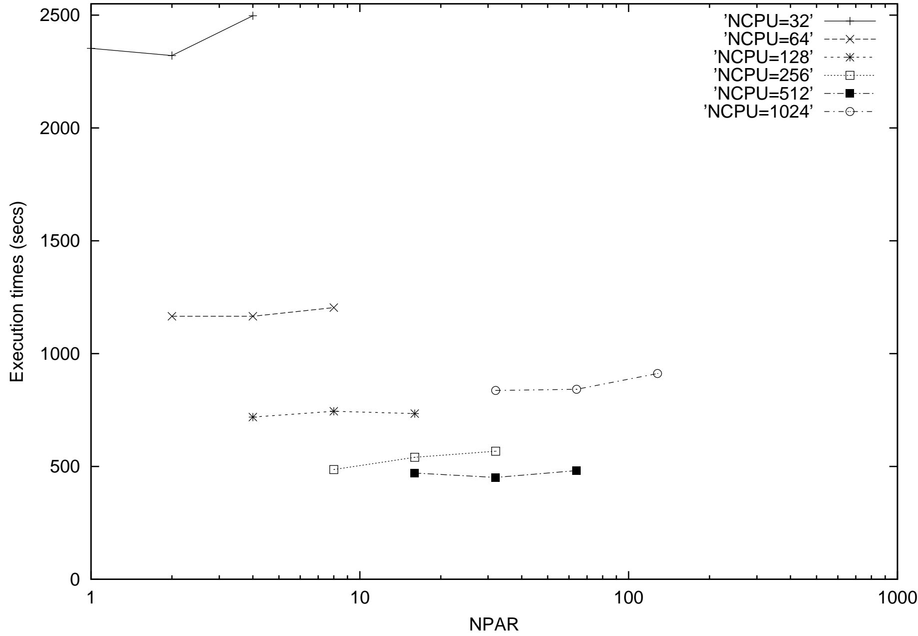

Figure 2: Execution times for various values of NPAR, for a varying numbers of processors, running SP9, version 4.6.13.g, for the larger Al simulation (1000 atoms)

scale to a higher number of processors. However, this was not the case for VASP and the Fe and Al simulations.

Unfortunately, the code does not achieve the quality of scaling to get a Capability Incentive Star, [4], however, one possible solution to this is suggested in Section 6.5.

From figure 3, we can see that the upgrade from SP7 to SP9 has improved the performance for a large number of processors, however, the code is slower for smaller numbers of processors, with the cross over point around 64 processors. This would imply that the communications between frames of HPCx has been improved, at the expense of single-frame performance.

It is clear that v.4.6.13.g is much faster than v.4.6. For example, for 128 processors on SP9 for both Al and Fe, v.4.6.13.g is more than twice as fast. Due to lack of time, no attempt was made to investigate why this version performs so well. (See Section 6.2 on Future Work).

It is interesting to note that for Fe and 1024 processors on SP9, v.4.6.13.g crashes near the beginning of the run. The crash occurs in routine `sort_redis_asc` of `wave.F`, called from routine `mapset` of `fftmpi_map.F`, called from `main.F`. This requires further investigation.

5.2.1 Profiling using xprofiler

For one particular run, xprofiler was used to profile the code. The profile is was flat, i.e. no one routine took a significant amount of time to compute. See Section 6.2 regarding future work.

5.2.2 Profiling using MPI_Trace

MPI_TRACE [6] was employed to study the performance of VASP on 128 processors. For process 0, the following is reported

```
-----------------------------------------------------------------
MPI Routine                  #calls     avg. bytes      time(sec)
-----------------------------------------------------------------
MPI_Comm_size                     6            0.0          0.000
```

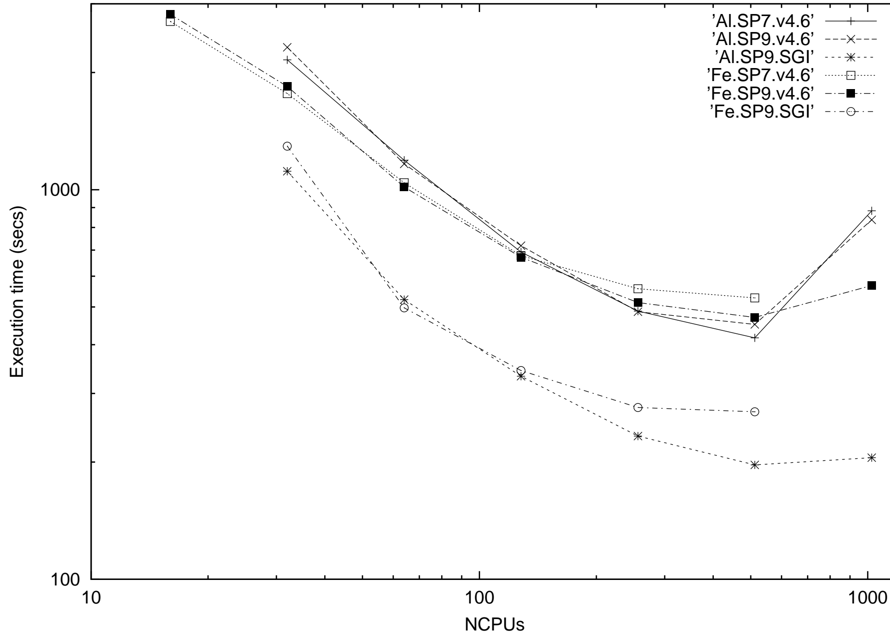

Figure 3: Execution times vs numbers of processors, using optNPAR

```
MPI_Comm_rank                47          0.0        0.000
MPI_Send                  75678      16361.8        2.077
MPI_Rsend                  9360         16.0        0.039
MPI_Isend                 69431       1807.1        0.338
MPI_Recv                 312436       4028.4       13.878
MPI_Waitall              143364          0.0        0.181
MPI_Bcast                204607       7822.6       49.168
MPI_Barrier                 424          0.0        0.560
MPI_Reduce                 7137      47538.2        1.770
MPI_Allreduce            221014         46.7       19.773
MPI_Alltoall             354848       2310.9      230.515
MPI_Alltoallv            136865       1251.5       41.217
-----------------------------------------------------------------
total communication time = 359.516 seconds.
total elapsed time       = 753.493 seconds.
user cpu time            = 749.770 seconds.
system time              = 2.180 seconds.
```

Further mpi_profiler output is included in the Appendix below.

From mpi_profiler, we can see that the most expensive routine of VASP is MPI_Alltoall. The author tried using VASP's own MPI_Alltoall and a local version of MPI_Alltoall optimised specifically for HPCx, however, the routine used, namely the default IBM routine, proved to be much faster.

6 Conclusions

This section briefly outlines the work achieved during this investigation. The following subsection contains suggestions for work that remains to be done with VASP to achieve better performance on HPCx and possibly achieve Capability Incentive Stars [4].

Examining the performance of VASP's individual routines showed that there is not one routine that takes a relatively large amount of time to execute, i.e. VASP version 4.6 has a flat profile, however, further research is possible (see Section 6.2 below).

6.1 NPAR

After investigation effect of varying NPAR, presented in Section 4.6.1, we found

- NPAR must be a factor of NCPU;
- NPAR $\geq$ NCPU/64, otherwise ScaLAPACK fails;
- Set NPAR=NCPU/16 for optimal performance.

By employing the optimum value of NPAR, on 128 processors, the Fe simulation ran in 677 seconds, compared to 739 seconds using the given, default settings. This represents a speed up of 8.4% for the Fe simulation.

6.2 Future Work

6.3 Numerical Libraries

A simple procedure for optimising the code was not investigated due to lack of time: reordering the load flags in the makefile to employ vendor libraries rather than public domain libraries, may decrease the overall execution time.

- Switch from the LAPACK and ScaLAPACK routines given in the VASP suite in vasp.4.lib to the LAPACK and ScaLAPACK routines installed on HPCx.
- Use IBM's (P)ESSL libraries when ever possible. Note that the ESSL 'feature' wherein the order of the routines arguments don't match the LAPACK order, can be avoided by employing a library available via [5]. This solution is currently not available for PESSL/ScaLAPACK.

6.4 Further profiling

The initial investigation of the code, using xprofiler, gave a flat profile. Further tests may include

- Running VASP on 1 processor to see if the profile remains flat;
- Checking the load imbalance using the mpi_trace data on, say, 128 processors;
- Compare the profiles of v.4.6 and v.4.6.13.g and remove the Altix-specific optimisations of v.4.6.13.g which may slow the code down on HPCx.

It should be noted that, as described in Section 5.2, v.4.6.13.g crashed on 1024 CPUs near the start of the simulation in the routine `sort_redis_asc` of `wave.F`, called from routine `mapset` of `fftmpi_map.F`, called from `main.F`. This requires further investigation, especially when running capability jobs that will require 1024 CPUs.

6.5 Capability Incentives

As previously stated, the code failed to achieve the scaling required to qualify for Capability Incentives [4] for both the Fe (small) and Al (large) simulations.

The author is confident that another use of the VASP code, specifically, the method of parallelising over images (Section 3.3), could be utilised to gain VASP Capability Incentive stars.

6.6 General code improvement

As described in Section 4.4.1, performance many be improved if the code is fixed to compiler in 64-bit using the re-entrant compiler.

Further, the compiler contains a number of minor non-standard FORTRAN which should be re-factored for reasons of portability. These include incorrect use of the FORMAT descriptor, including the TAB character in a source file and using null literal strings.

Lastly, there are two runtime errors which could be corrected, namely opening and closing unit 0.

7 Appendix

```
sh-2.05a$ cat mpi_profile.0
----------------------------------------------------------------
MPI Routine                 #calls     avg. bytes      time(sec)
----------------------------------------------------------------
MPI_Comm_size                    6            0.0          0.000
MPI_Comm_rank                   47            0.0          0.000
MPI_Send                     75678        16361.8          2.077
MPI_Rsend                     9360           16.0          0.039
MPI_Isend                    69431         1807.1          0.338
MPI_Recv                    312436         4028.4         13.878
MPI_Waitall                 143364            0.0          0.181
MPI_Bcast                   204607         7822.6         49.168
MPI_Barrier                    424            0.0          0.560
MPI_Reduce                    7137        47538.2          1.770
MPI_Allreduce               221014           46.7         19.773
MPI_Alltoall                354848         2310.9        230.515
MPI_Alltoallv               136865         1251.5         41.217
----------------------------------------------------------------
total communication time = 359.516 seconds.
total elapsed time       = 753.493 seconds.
user cpu time            = 749.770 seconds.
system time              = 2.180 seconds.
maximum memory size      = 122512 KBytes.

----------------------------------------------------------------
Message size distributions:

MPI_Send                 #calls     avg. bytes      time(sec)
                            800            4.0          0.005
                             39           32.0          0.000
                             78           56.0          0.001
                            156          104.0          0.002
                            312          200.0          0.003
                            624          392.0          0.006
                           1248          776.0          0.012
                          24336         1307.1          0.104
                          40289         3100.5          0.191
                             39        13120.0          0.001
                             80        24576.0          0.002
                            821        50779.8          0.057
                           4120       102400.0          0.862
```

	#calls	avg. bytes	time(sec)
	2151	201991.9	0.700
	585	307114.7	0.132

MPI_Rsend	#calls	avg. bytes	time(sec)
	9360	16.0	0.039

MPI_Isend	#calls	avg. bytes	time(sec)
	4680	0.0	0.025
	39	4.0	0.000
	39	8.0	0.000
	46800	1280.0	0.207
	17200	2560.0	0.089
	80	16384.0	0.002
	422	27788.4	0.010
	131	33600.0	0.003
	40	102400.0	0.000

MPI_Recv	#calls	avg. bytes	time(sec)
	4526	0.0	0.234
	800	4.0	1.261
	18954	16.0	0.287
	234	32.0	0.002
	468	56.0	0.005
	936	104.0	0.010
	1872	200.0	0.019
	3744	392.0	0.038
	7488	776.0	0.079
	170898	1303.1	3.204
	95144	2770.4	2.487
	201	5132.7	0.116
	109	13392.4	0.014
	438	26378.5	0.067
	2445	39830.2	1.473
	2299	93046.4	2.198
	1295	200135.7	1.187
	585	307200.0	1.197

MPI_Bcast	#calls	avg. bytes	time(sec)
	488	0.0	0.009
	11352	4.0	0.428
	78	8.0	0.002
	6396	16.0	0.076
	273	28.0	0.006
	10454	56.2	0.252
	5989	111.8	0.168
	16539	251.4	0.292
	4040	404.6	0.087
	5811	761.1	0.093
	75192	1295.5	12.574
	62297	2935.1	28.494
	202	5137.7	0.006
	1760	34094.5	0.266
	667	71019.9	0.215
	819	204800.0	0.470

```
                         2250      458896.0          5.731

MPI_Reduce             #calls     avg. bytes      time(sec)
                           39            8.0          0.040
                         3321         2560.0          0.101
                          520         6104.6          0.035
                          759        34167.9          0.146
                         1679        79773.2          0.759
                          819       204800.0          0.689

MPI_Allreduce          #calls     avg. bytes      time(sec)
                          761            4.0          0.345
                        89266            8.0         12.352
                       100527           16.0          5.286
                           38           24.8          0.003
                           16           52.0          0.001
                           67           85.4          0.071
                        30209          255.9          1.540
                           46          512.0          0.009
                           79         2406.7          0.005
                            5         6912.0          0.160

MPI_Alltoall           #calls     avg. bytes      time(sec)
                           24            4.0          0.002
                          228           96.0          0.064
                          114          379.6          0.058
                       106650          544.0          5.477
                       230960         2719.3         70.500
                        16872         7935.8        154.413

MPI_Alltoallv          #calls     avg. bytes      time(sec)
                            1           58.0          0.003
                            1          120.0          0.003
                           28          232.0          0.065
                           61          468.6          0.152
                           29          960.0          0.092
                       136745         1252.1         40.902

sh-2.05a$ cat mpi_profile.1
----------------------------------------------------------------
MPI Routine                  #calls     avg. bytes      time(sec)
----------------------------------------------------------------
MPI_Comm_size                     6            0.0          0.000
MPI_Comm_rank                    47            0.0          0.000
MPI_Send                      84803        14691.3          1.749
MPI_Rsend                     18720           16.0          0.082
MPI_Isend                     72826         1924.7          0.356
MPI_Recv                     402156         3917.0         11.941
MPI_Waitall                  152568            0.0          0.192
MPI_Bcast                    223093         7164.1         47.967
MPI_Barrier                     424            0.0          0.563
MPI_Reduce                     7176        47554.6          2.027
MPI_Allreduce                221014           46.7         19.549
MPI_Alltoall                 354848         2307.0        228.164
```

```
MPI_Alltoallv              136865        1251.5         39.150
----------------------------------------------------------------
total communication time = 351.739 seconds.
total elapsed time       = 753.493 seconds.
user cpu time            = 747.930 seconds.
system time              = 2.350 seconds.
maximum memory size      = 156080 KBytes.

----------------------------------------------------------------
Message size distributions:

MPI_Send                 #calls     avg. bytes      time(sec)
                            800            4.0          0.005
                            156           16.0          0.001
                            195           32.0          0.001
                            390           56.0          0.003
                            780          104.0          0.006
                           1560          200.0          0.012
                           3120          392.0          0.025
                           6240          776.0          0.047
                          23517         1295.3          0.110
                          40249         3049.3          0.192
                             39        13120.0          0.001
                             40        24576.0          0.001
                            781        51282.8          0.053
                           4274       102400.0          0.599
                           1999       201778.3          0.526
                            663       307124.7          0.169

MPI_Rsend                #calls     avg. bytes      time(sec)
                          18720           16.0          0.082

MPI_Isend                #calls     avg. bytes      time(sec)
                           4641            0.0          0.026
                             39            4.0          0.000
                             39            8.0          0.000
                             39           16.0          0.000
                             39           32.0          0.000
                             78           56.0          0.000
                            156          104.0          0.001
                            312          200.0          0.001
                            624          392.0          0.002
                           1248          776.0          0.005
                          49413         1293.3          0.222
                          15289         2539.8          0.080
                             40         5120.0          0.000
                             40        16358.4          0.000
                            381        25555.1          0.009
                            248        37722.6          0.005
                            200        80896.0          0.002

MPI_Recv                 #calls     avg. bytes      time(sec)
                           4487            0.0          0.234
                            839            4.0          1.034
```

```
                         39          8.0         0.004
                      37596         16.0         0.218
                        195         32.0         0.002
                        390         56.0         0.004
                        780        104.0         0.008
                       1560        200.0         0.015
                       3120        392.0         0.028
                       6240        776.0         0.051
                     228930       1291.2         1.982
                     108516       2807.9         2.135
                        160       5120.0         0.077
                        121      13072.4         0.015
                        658      26144.7         0.013
                       2747      39166.3         1.428
                       3626      97655.6         2.214
                       1645     201128.1         1.481
                        507     307101.5         0.999

MPI_Bcast            #calls   avg. bytes     time(sec)
                        488          0.0         0.009
                      11352          4.0         0.491
                         78          8.0         0.027
                      12831         16.0         0.154
                        468         29.7         0.019
                      10844         56.2         0.277
                       6769        110.9         0.221
                      18099        247.0         0.457
                       7160        399.1         0.314
                      12090        768.4         0.582
                      78078       1290.2        11.126
                      59178       2954.0        27.510
                        162       5142.1         0.006
                       1880      34042.6         0.343
                        547      71998.2         0.173
                        819     204800.0         0.467
                       2250     458896.0         5.793

MPI_Reduce           #calls   avg. bytes     time(sec)
                         39          8.0         0.001
                       3321       2560.0         0.102
                        520       6104.6         0.021
                        798      34969.0         0.231
                       1679      79773.2         0.974
                        819     204800.0         0.699

MPI_Allreduce        #calls   avg. bytes     time(sec)
                        761          4.0         0.678
                      89266          8.0        11.754
                     100527         16.0         5.332
                         38         24.8         0.003
                         16         52.0         0.001
                         67         85.4         0.070
                      30209        255.9         1.520
                         46        512.0         0.009
```

```
                         79         2406.7          0.019
                          5         6912.0          0.162

MPI_Alltoall         #calls     avg. bytes      time(sec)
                         24            4.0          0.002
                        228           96.0          0.065
                        114          379.6          0.059
                     106650          544.0          5.428
                     230960         2713.2         70.363
                      16872         7935.8        152.246

MPI_Alltoallv        #calls     avg. bytes      time(sec)
                          1           58.0          0.003
                          1          120.0          0.002
                         28          232.0          0.067
                         61          468.6          0.152
                         29          960.0          0.089
                     136745         1252.1         38.836

sh-2.05a$ cat mpi_profile.2
----------------------------------------------------------------
MPI Routine                 #calls     avg. bytes      time(sec)
----------------------------------------------------------------
MPI_Comm_size                    6            0.0          0.000
MPI_Comm_rank                   47            0.0          0.000
MPI_Send                     85044        14484.6          1.696
MPI_Rsend                    18720           16.0          0.081
MPI_Isend                    69392         1821.1          0.464
MPI_Recv                    399002         3838.5         12.792
MPI_Waitall                 149526            0.0          0.186
MPI_Bcast                   220051         7208.6         47.706
MPI_Barrier                    424            0.0          0.566
MPI_Reduce                    7176        47554.6          2.459
MPI_Allreduce               221014           46.7         19.988
MPI_Alltoall                354848         2307.0        233.306
MPI_Alltoallv               136865         1251.5         40.994
----------------------------------------------------------------
total communication time = 360.239 seconds.
total elapsed time       = 753.493 seconds.
user cpu time            = 747.420 seconds.
system time              = 2.630 seconds.
maximum memory size      = 156024 KBytes.

----------------------------------------------------------------
Message size distributions:

MPI_Send             #calls     avg. bytes      time(sec)
                        800            4.0          0.004
                        156           16.0          0.001
                        195           32.0          0.001
                        390           56.0          0.002
                        780          104.0          0.005
                       1560          200.0          0.013
                       3120          392.0          0.026
```

	#calls	avg. bytes	time(sec)
	6240	776.0	0.052
	23556	1295.3	0.111
	40329	3051.4	0.191
	39	13120.0	0.001
	120	24576.0	0.001
	861	50323.6	0.058
	4390	102400.0	0.600
	1845	201526.1	0.448
	663	307124.7	0.182

MPI_Rsend	#calls	avg. bytes	time(sec)
	18720	16.0	0.081

MPI_Isend	#calls	avg. bytes	time(sec)
	4649	0.0	0.023
	39	4.0	0.000
	39	8.0	0.000
	39	16.0	0.000
	39	32.0	0.000
	78	56.0	0.000
	156	104.0	0.001
	312	200.0	0.002
	624	392.0	0.004
	1248	776.0	0.009
	49296	1293.4	0.297
	12169	2534.6	0.111
	40	5120.0	0.000
	228	26000.0	0.006
	236	34569.5	0.007
	200	80896.0	0.003

MPI_Recv	#calls	avg. bytes	time(sec)
	4478	0.0	0.221
	839	4.0	1.067
	39	8.0	0.001
	37752	16.0	0.260
	351	32.0	0.003
	702	56.0	0.007
	1404	104.0	0.014
	2808	200.0	0.026
	5616	392.0	0.049
	11232	776.0	0.099
	230148	1293.0	2.848
	94165	2820.9	2.078
	120	5120.0	0.047
	156	12219.5	0.025
	547	28131.9	0.020
	3103	39074.7	1.278
	3428	98636.2	2.192
	1607	201041.2	1.649
	507	307101.5	0.909

MPI_Bcast	#calls	avg. bytes	time(sec)
	488	0.0	0.009

```
                           11352          4.0         0.430
                              78          8.0         0.027
                           12831         16.0         0.167
                             468         29.7         0.017
                           10844         56.2         0.273
                            6769        110.9         0.218
                           18099        247.0         0.438
                            7160        399.1         0.270
                           12090        768.4         0.490
                           78156       1290.2        11.438
                           56098       2975.7        27.131
                             122       5149.4         0.004
                            2000      33996.8         0.314
                             427      73526.6         0.178
                             819     204800.0         0.486
                            2250     458896.0         5.815

MPI_Reduce                #calls    avg. bytes    time(sec)
                              39          8.0         0.000
                            3321       2560.0         0.097
                             520       6104.6         0.099
                             798      34969.0         0.174
                            1679      79773.2         1.408
                             819     204800.0         0.682

MPI_Allreduce             #calls    avg. bytes    time(sec)
                             761          4.0         0.750
                           89266          8.0        12.015
                          100527         16.0         5.430
                              38         24.8         0.003
                              16         52.0         0.001
                              67         85.4         0.071
                           30209        255.9         1.530
                              46        512.0         0.008
                              79       2406.7         0.018
                               5       6912.0         0.161

MPI_Alltoall              #calls    avg. bytes    time(sec)
                              24          4.0         0.002
                             228         96.0         0.069
                             114        379.6         0.057
                          106650        544.0         7.411
                          230960       2713.2        70.597
                           16872       7935.8       155.171

MPI_Alltoallv             #calls    avg. bytes    time(sec)
                               1         58.0         0.003
                               1        120.0         0.003
                              28        232.0         0.067
                              61        468.6         0.151
                              29        960.0         0.092
                          136745       1252.1        40.678

sh-2.05a$ cat mpi_profile.100
```

```
----------------------------------------------------------------
MPI Routine                 #calls     avg. bytes      time(sec)
----------------------------------------------------------------
MPI_Comm_size                    6            0.0          0.000
MPI_Comm_rank                   47            0.0          0.000
MPI_Send                    133601         6113.5          1.612
MPI_Rsend                     6240           16.0          0.025
MPI_Isend                    11671         2130.5          0.065
MPI_Recv                    151232         5549.4          8.198
MPI_Irecv                     6240           16.0          0.013
MPI_Waitall                 133926            0.0          0.458
MPI_Bcast                   197914         7501.1         63.084
MPI_Barrier                    424            0.0          0.536
MPI_Reduce                    6536        42497.6          5.228
MPI_Allreduce               221129           46.7         25.511
MPI_Alltoall                354940         2310.9        230.377
MPI_Alltoallv               136911         1251.1         40.195
----------------------------------------------------------------
total communication time = 375.304 seconds.
total elapsed time       = 753.485 seconds.
user cpu time            = 748.980 seconds.
system time              = 2.570 seconds.
maximum memory size      = 121544 KBytes.

----------------------------------------------------------------
Message size distributions:

MPI_Send                 #calls     avg. bytes      time(sec)
                            156            0.0          0.001
                            800            4.0          0.004
                           9555           16.0          0.040
                            234           32.0          0.001
                            468           56.0          0.002
                            936          104.0          0.005
                           1872          200.0          0.011
                           3744          392.0          0.022
                           7488          776.0          0.045
                          75972         1293.4          0.418
                          27458         2559.7          0.208
                             80        24576.0          0.001
                            400        49152.0          0.016
                           2863       102722.6          0.468
                           1575       206068.6          0.371

MPI_Rsend                #calls     avg. bytes      time(sec)
                           6240           16.0          0.025

MPI_Isend                #calls     avg. bytes      time(sec)
                           4641            0.0          0.022
                             39            4.0          0.000
                             39            8.0          0.000
                           6240           16.0          0.026
                             80         2560.0          0.001
                            312        26861.5          0.008
```

	240	33280.0	0.008
	80	102400.0	0.001

MPI_Recv	#calls	avg. bytes	time(sec)
	4641	0.0	0.375
	839	4.0	1.071
	39	8.0	0.001
	15795	16.0	0.128
	234	32.0	0.001
	468	56.0	0.002
	936	104.0	0.005
	1872	200.0	0.011
	3744	392.0	0.020
	7488	776.0	0.042
	79092	1292.9	0.947
	30578	2559.7	0.649
	39	8000.0	0.000
	77	15937.7	0.011
	432	24099.3	0.011
	360	48924.4	0.069
	3059	102701.9	2.862
	1539	199444.6	1.991

MPI_Irecv	#calls	avg. bytes	time(sec)
	6240	16.0	0.013

MPI_Bcast	#calls	avg. bytes	time(sec)
	488	0.0	0.009
	11352	4.0	0.495
	78	8.0	0.056
	12870	16.0	0.730
	507	29.8	0.050
	10922	56.2	0.350
	6925	110.7	0.348
	18434	246.2	0.695
	7784	398.5	0.802
	13338	769.1	1.478
	78780	1289.1	27.737
	31138	2555.8	20.372
	42	5205.3	0.050
	2000	33996.8	3.093
	187	82467.6	0.263
	819	204800.0	1.612
	2250	458896.0	4.945

MPI_Reduce	#calls	avg. bytes	time(sec)
	39	8.0	0.000
	3401	2560.0	0.257
	280	5120.0	0.046
	1118	35767.9	0.432
	879	68150.2	0.782
	819	204800.0	3.710

MPI_Allreduce	#calls	avg. bytes	time(sec)

```
                         761          4.0          0.835
                       89289          8.0         17.013
                      100573         16.0          5.450
                          38         24.8          0.003
                          16         52.0          0.001
                          67         85.4          0.473
                       30255        255.9          1.374
                          46        512.0          0.010
                          79       2406.7          0.019
                           5       6912.0          0.333

MPI_Alltoall          #calls   avg. bytes    time(sec)
                          24          4.0        0.007
                         228         96.0        0.058
                         114        379.6        0.058
                      106650        544.0        6.346
                      231052       2719.1       68.254
                       16872       7935.8      155.656

MPI_Alltoallv         #calls   avg. bytes    time(sec)
                          85          0.0        0.251
                           1         58.0        0.004
                          28        232.0        0.075
                           5        339.4        0.008
                           1        960.0        0.000
                      136791       1252.1       39.856
```

References

[1] VASP webpage, http://cms.mpi.univie.ac.at/vasp/

[2] Dario Alfé, pers. comms., Dario Alfé's homepage: http://chianti.geol.ucl.ac.uk/ dario

[3] The Altix at CSAR webpage http://csar.web.mcc.ac.uk/newton

[4] HPCx Capability Incentive Scheme http://www.hpcx.ac.uk/services/policies/capability.html

[5] LAPACK/ESSL workaround from NETLIB http://www.netlib.org/lapack/essl/

[6] MPI Trace on HPCx details http://www.hpcx.ac.uk/support/documentation/IBMdocuments/mpitrace