Efficient massively-parallel tools for the study of catalytic chemistry
A dCSE Project

Thomas W. Keal
STFC Daresbury Laboratory

29th June 2010

Abstract

A task-farm parallel framework has been added to the ChemShell computational chemistry environment to provide a facility for parallelising common chemical calculations, including Hessian evaluation and a number of geometry optimisation algorithms. As ChemShell can already exploit the parallelisation of external programs for energy and gradient evaluations, this results in a two-layer parallel approach that gives efficient scaling up to at least 1000 cores. The task-farmed methods will be used to run large-scale heterogeneous catalysis simulations on HECToR.

Contents

1 Background

The ChemShell computational chemistry environment [1,2] provides a means of integrating quantum mechanical (QM) and molecular mechanical (MM) software packages to perform combined QM/MM calculations. The external packages are used solely for energy and gradient evaluations while ChemShell routines are used to form the combined QM/MM gradient and to handle higher-level tasks such as geometry optimisation or molecular dynamics. ChemShell calculations are controlled using scripts written in the Tcl language [3], with generic commands that hide as far as possible the details of the external programs from the user.

The QM/MM approach is particularly useful in the study of heterogeneous catalysis. ChemShell supports embedded cluster calculations [2, 4] in which the bulk material or surface is represented by a finite MM cluster model, optionally with additional point charges to mimic the effect of bulk electrostatics. The active site is modelled by a high-level QM calculation with electrostatic embedding, where the cluster environment is represented by point charges in the QM Hamiltonian. This setup is intended to give an optimal balance between accuracy at the active site and computational expense.

ChemShell may be built as a serial or parallel program, although the current release (v3.3) does not contain any parallel algorithms of its own. The parallel version instead is intended to exploit any parallelism contained in the packages used to evaluate the energy and gradient. Usually the evaluation step of the calculation is by far the most computationally expensive, and so parallelisation of this step should scale well. However, on large supercomputers such as HECToR, where calculations involving thousands of cores are routine, the parallel algorithms in the external programs often do not continue to scale efficiently. In this domain we expect there to be significant gains from adding a second layer of parallelism at the ChemShell level.

2 The dCSE project

The aim of this distributed Computational Science and Engineering (dCSE) project is to add a task-farm parallel framework to ChemShell in order to parallelise tasks that are commonly employed in materials cluster simulations. The dCSE proposal was written by C. Richard A. Catlow, Alexey Sokol and Fedor Goumans (University College London) and Paul Sherwood, Huub van Dam and Johannes Kästner (STFC Daresbury Laboratory). The project work was carried out by Thomas Keal (STFC Daresbury Laboratory).

9 months of development effort were deployed over an 18 month period

from January 2009 to June 2010. The code development objectives for the project were:

- ChemShell (and associated Tcl libraries) adapted to support task farmed parallelism using split communicators. Commands can be executed on specific workgroups by using conditional tests within the Tcl interpreter.
- Finite difference Hessian code within ChemShell re-written to allow task-farmed execution. Speed-up factor of 2 on a test calculation (as approved by PI) when comparing existing implementation (a single parallel task running across a large HECToR partition of over 1000 processors) with the task farmed version.
- Interface with the parallel implementation of DL-FIND such that the separate tasks within ChemShell are given independent work to perform. Demonstration of parallel NEB for transition-state determination in a surface chemistry problem, speed-up (existing relative to running the points sequentially on a large parallel platform) of 2.
- DL-FIND parallel optimisation capabilities interfaced to task farmed version of ChemShell. Demonstration of global minimum search using stochastic search and genetic algorithms. The stochastic search can also be run using the baseline code (the non-task farmed version of ChemShell) code and will show speed-up of 3.

All the objectives were achieved, and the benchmark targets surpassed, as detailed below.

The benchmark calculations in this report were performed on the XT4 component of HECToR in its Phase 2a configuration. This is a Cray machine with 2.3 GHz quad-core AMD opteron processors and 2 GB of memory per core. Due to the simple nature of the parallelisation method employed, we expect the new framework to run equally efficiently on the recently-introduced Phase 2b configuration of HECToR and on other parallel platforms. The test cases have been chosen to represent calculations that might realistically be run by users on HECToR and so should give a good indication of the performance gains expected from the new approach.

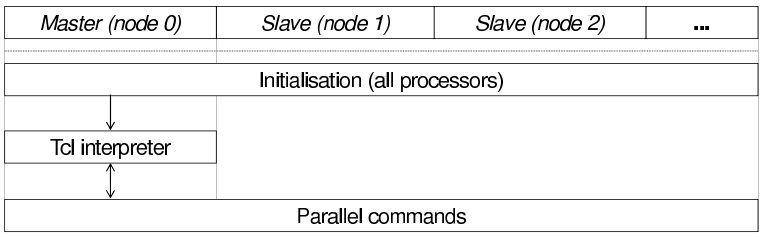
ChemShell, GAMESS-UK and GULP were compiled on HECToR using the Cray compiler wrapper `ftn` with the PGI C and Fortran compilers (v9.0-4).

3 Task-farm parallelisation

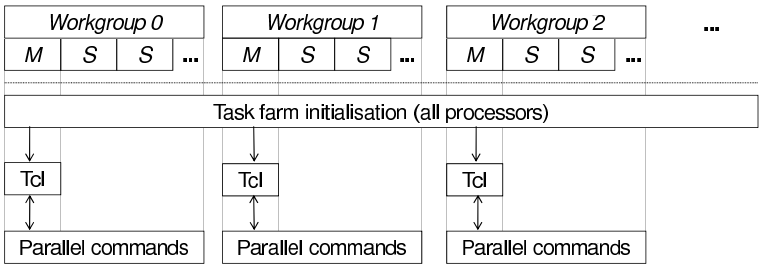
Task-farming parallelism is useful in situations where a set of independent calculations have to be performed. The 'task farm' consists of all available

Figure 1: The original and task-farming parallel frameworks in ChemShell. In the original case (a) the Tcl input script is parsed by a single interpreter, with parallel ChemShell commands executed across all processors. In the task-farming case (b) the input script is parsed by one interpreter in each workgroup, with parallel commands executed on only the processors within the workgroup (M = master, S = slave).

(a)



(b)



processors, which are then divided into subsets of processors called workgroups. The tasks are split between the workgroups which work on them in parallel. As the calculations are independent, no information needs to be exchanged between workgroups during this time, and sharing of results can be postponed until all the tasks have completed.

To implement task-farming in ChemShell a number of modifications to the code and external programs were necessary.

3.1 ChemShell

ChemShell is implemented in parallel using MPI (message passing interface). The parallel framework used before task-farming was added is illustrated in Figure 1a. One node acts as a 'master', on which a Tcl interpreter runs and executes the input script. The other nodes are 'slaves' that remain idle until a parallel input command is executed. This would typically be a request for a gradient evaluation using an external program. To achieve maximum

efficiency the external program is linked into the ChemShell executable so it can be called directly without spawning an additional process. The external program therefore shares the same MPI environment as ChemShell and can make use of all the nodes for its own parallel calculations. When the command has completed control returns to the master node.

Task-farming parallelism adds an extra layer of organisation to the processors, which are grouped into independent workgroups using MPI communicators. In the original parallel framework the nodes were grouped into the default `MPI_COMM_WORLD` communicator. To create independent workgroups `MPI_COMM_WORLD` is split into smaller sets of processors, each with their own Workgroup communicator (named `MPI_COMM_WORKGROUP`). The user specifies the number of workgroups to be created using the command-line argument `-nworkgroups` when ChemShell is executed.

In each workgroup one node acts as a master and the rest are slaves (Figure 1b). They operate in the same way as in the original parallel framework, except that there are now multiple master nodes (if the number of workgroups is set to one, the new framework reduces to the original). Each master runs a copy of the Tcl interpreter and independently executes the same ChemShell input script. A number of Tcl commands have been implemented to report workgroup information, such as how many workgroups exist (`nworkgroups`) and which workgroup the script is running in (`workgroupid`). These commands may be used to make parts of the script conditional on the workgroup ID and this provides a mechanism to distribute tasks between workgroups, as in this simple input script:

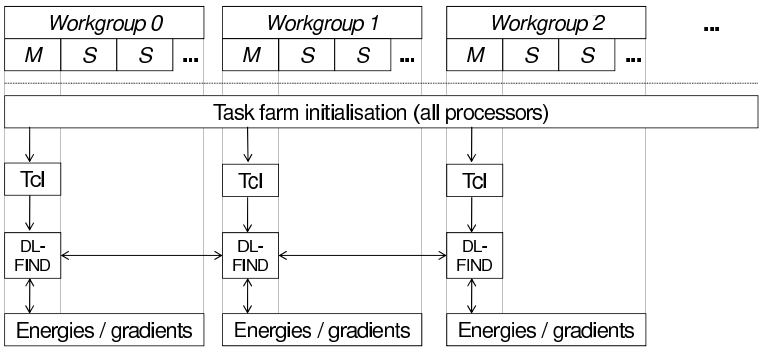
```
set wid [ workgroupid ]
if { $wid == 2 } {
   do_task
}
```

Although the input is parsed by all workgroups, the procedure `do_task` would only be run on workgroup 2.

To prevent file conflicts, a scratch working directory is created for each workgroup (`workgroup[n]/`). This is important for ChemShell data objects, which may be saved to disk. By default ChemShell objects will be loaded from the working directory, but if not present the common parent directory will also be searched. This makes it possible to create globally-accessible objects. Local objects may be 'globalised' using a Tcl procedure (`taskfarm_globalise_objects`). This command first synchronises the workgroups using a call to `MPI_Barrier` to prevent data corruption.

The ChemShell standard output and standard error are also separated out into the working directories, under the names `workgroup[n].out` and `workgroup[n].err`, with the exception of the output of workgroup 0 which is treated as the main output and not redirected.

Figure 2: A DL-FIND calculation running under the task-farm parallel framework in ChemShell. The DL-FIND command is executed on the master node of each workgroup while energy and gradient evaluations may be run across all workgroup nodes. The multiple DL-FIND instances share gradient data with each other directly using MPI calls, usually at the end of each optimisation cycle.



3.2 DL-FIND

DL-FIND is a geometry optimisation library [5, 6] that is included in the ChemShell distribution as the standard optimisation driver. ChemShell and DL-FIND communicate via a well-defined interface which is used for passing options, geometries, energies and gradients between the two codes.

DL-FIND has its own implementation of task-farming parallelism [6]. The parallel facilities consist of a collection of wrapper functions around MPI library calls to share data between processors and a parallel interface to pass details of the task farm to or from the calling program. Two parallel strategies have been implemented: in the first, the calling program sets up the task farm and passes the setup information and MPI communicators to DL-FIND, while in the second DL-FIND sets up the task farm and passes the setup information in the opposite direction.

When linked to ChemShell the first strategy is the most appropriate as ChemShell sets up a task-farmed environment when it is initialised and DL-FIND must operate within this. This setup is illustrated in Figure 2. Note that DL-FIND communicates directly between workgroups using MPI calls. This usually occurs at the end of each optimisation cycle to share gradient data. A communicator provided by ChemShell called `MPI_COMM_COUNTERPARTS` is used for this purpose, which groups together the master nodes.

To make the ChemShell environment accessible to DL-FIND, interface subroutines were added to ChemShell to provide the required information about the task farm setup:

- `dlf_get_params`: additional parameters `glob%ntasks` (number of workgroups) and `tldf_farm` (=0 to indicate that ChemShell sets up the

task farm).

- `dlf_get_procinfo`: supplies the number of processors, node ID and the `MPI_COMM_WORLD` communicator.
- `dlf_get_taskfarm`: supplies the number of workgroups, number of processors per workgroup, the rank of the current processor in the workgroup, the rank of the workgroup and two MPI communicators: `MPI_COMM_WORKGROUP` and `MPI_COMM_COUNTERPARTS`.

Two other interface routines (`dlf_put_procinfo` and `dlf_put_taskfarm`) are only used when DL-FIND sets up the taskfarm and are therefore not required for the ChemShell interface. A number of ChemShell routines were written to expose the above information to the interface.

There is a subtle difference between the ChemShell and DL-FIND taskfarming environments which has to be reconciled by the interface. On the ChemShell side only the master nodes (one per workgroup) execute ChemShell commands such as DL-FIND calls. However, the DL-FIND parallel routines assume that they are running on all processors. To work around this ChemShell only provides information on the master nodes to DL-FIND so that DL-FIND sees only one node per workgroup. DL-FIND's parallel algorithms can then run unchanged in the ChemShell environment. In practice this means that the 'world' communicator passed to DL-FIND is actually `MPI_COMM_COUNTERPARTS`, the total number of processors is set equal to the number of workgroups and the number of processors per workgroup is set equal to 1. A dummy value can then be sent for `MPI_COMM_WORKGROUP`.

3.3 External programs

All parallel work below the level of the ChemShell interpreter must be carried out within a workgroup. Therefore if an external program is called for an energy and gradient evaluation it must use `MPI_COMM_WORKGROUP`, not `MPI_COMM_WORLD`. An interface function has been added to ChemShell to pass the workgroup communicator to external programs.

We have modified the GAMESS-UK [7, 8] QM package and GULP [9] MM package to accept the `MPI_COMM_WORKGROUP` communicator from ChemShell. These programs can be compiled as libraries into the ChemShell executable so a simple function call is used to pass the information.

In the case of GAMESS-UK, a significant amount of computational time is spent evaluating matrix eigenvectors and eigenvalues and it is therefore important to use a parallel diagonaliser such as PeIGS [10] in order to make efficient use of a massively-parallel platform. As the PeIGS library also contains MPI calls, these were modified so that PeIGS could work in the taskfarmed environment. This involved replacing instances of `MPI_COMM_WORLD` with the workgroup communicator passed from ChemShell.

The changes made to GULP have been incorporated into the released version and are available as of v3.5.3 or later. The changes to GAMESS-UK are in the development trunk code and will be incorporated into the next release.

In principle any parallel external program that has been interfaced to ChemShell can be used in a task-farmed calculation providing it can be modified to accept `MPI_COMM_WORKGROUP` and use it instead of `MPI_COMM_WORLD`. Alternatively the parallelism at the ChemShell level could be exploited while only using a serial program for energy and gradient evaluations. In this case the task farm would be set up so that there is only one processor per workgroup. No modifications would be required for the external program but the potential for scaling up the calculation would obviously be more limited.

4 Performance of parallel methods

Three types of chemical calculation in ChemShell were modified to take advantage of the task-farming implementation. They each involve multiple independent energy or gradient evaluations and should therefore benefit from high-level parallelism.

4.1 Finite difference Hessian

An obvious choice for parallelisation is the calculation of a finite difference Hessian. Each entry in the Hessian matrix is calculated using the difference of two gradients. Using a forward difference algorithm an N-atom system requires $3N + 1$ independent gradient evaluations. With a central difference algorithm this rises to $6N$ evaluations.

In the original ChemShell implementation the gradient calculations and Hessian evaluation are performed using a single Tcl command (`force`). In the task-farmed version this command has been split up into three stages to facilitate parallelisation. In the first stage (`force_precalc`), the required set of gradients is calculated and stored on disk as ChemShell objects. This work can be divided up among the workgroups to be carried out in parallel using the option `task_atoms` with a list of atoms. In the second stage the ChemShell gradient objects are made available to all workgroups using the command `taskfarm_globalise_forcegradients`. Finally, the Hessian matrix is evaluated using the pre-calculated gradients (using `force` with the option `precalc=yes`). The Hessian calculation can be restricted to a single workgroup if desired by a conditional test on the workgroup ID.

The 57-atom silicate-VO_3 cluster shown in Figure 3 was used to assess the performance of the task-farmed implementation. Energies and gradients were calculated using GAMESS-UK with the B3LYP functional [11]. Two basis sets were used: the LANL2 effective core potential basis [12] (giving 413 basis functions) and the TZVP [13, 14] all-electron basis (1032 basis

Figure 3: The silicate-VO_3 cluster used for the Hessian benchmark calculations.

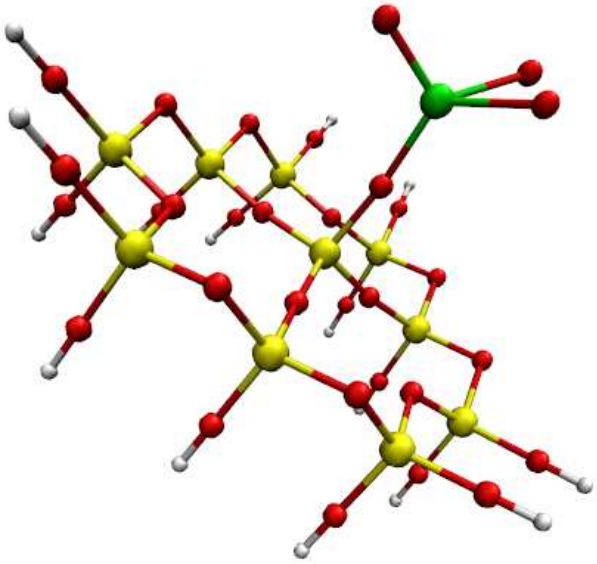
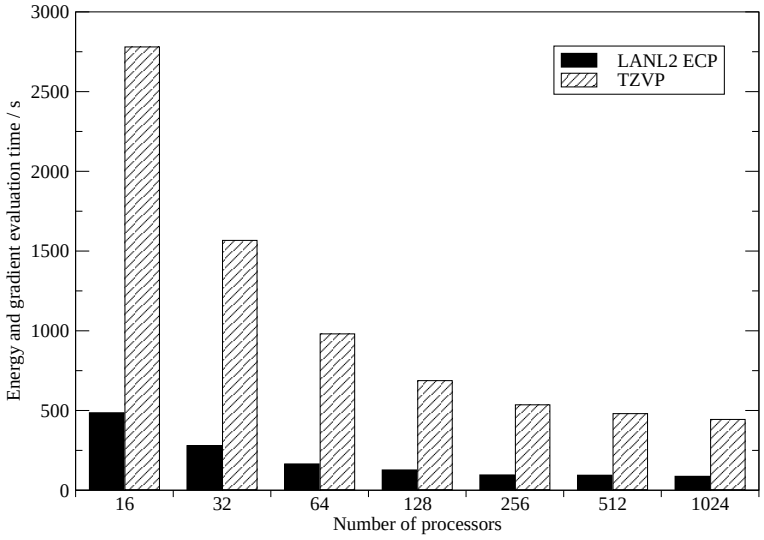


Figure 4: Calculation time in wall clock seconds for a single point energy and gradient evaluation of a 57-atom silicate-VO_3 system using the LANL2 ECP and TZVP basis sets.



functions). The larger basis test is present to fully assess the PeIGS build of GAMESS-UK, as PeIGS is only used to diagonalize matrices larger than the total number of processors.

To give an indication of the time required for the full Hessian calculation, single-point calculations were carried out with differing numbers of processors. The results are shown in Figure 4. If perfect scaling were achieved the calculation time would halve with each doubling of the number of processors (and a second level of parallelism would be unnecessary). For both basis sets reasonable scaling is achieved up to approximately 128 processors, but for larger processor counts the gains are very small. This suggests that large efficiency gains should be possible using task-farmed calculations.

The full forward difference Hessian was evaluated using a set of 1024-processor calculations with differing numbers of workgroups. The tasks were parallelised using a simple static load-balancing scheme where as far as possible an equal number of gradient calculations were assigned to each workgroup. As each gradient calculation should take approximately the same amount of time (apart from the first where no wavefunction guess is provided), no major gains would be expected from a more sophisticated load-balancing mechanism.

The results are shown as wall clock times in Table 1. To correctly interpret the results it is important to keep in mind that all calculations run with the same number of processors and only the division into workgroups is changed. As the number of workgroups increases, the number of processors

Table 1: Calculation time in wall clock seconds for a forward difference Hessian matrix evaluation using 1024 processors divided into workgroups. Speed-up factor is compared to the single workgroup calculation.

LANL2 ECP basis			
Workgroups	Procs/workgroup	Time / s	Speed-up
1	1024	7896	
2	512	4354	1.8
4	256	2444	3.2
8	128	1665	4.7
16	64	1290	6.1
32	32	1176	6.7
64	16	1151	6.9
128	8	2165	3.7
TZVP basis			
Workgroups	Procs/workgroup	Time / s	Speed-up
1	1024	52762	
64	16	7812	6.8

in each workgroup falls proportionally. The calculation with the highest speed-up factor therefore gives the best balance between parallelisation of individual gradient evaluations and parallelisation of the Hessian as a whole. This is different to the benchmarking of single-level parallelism where scaling of calculation time with number of processors is used to measure efficiency. There is no scaling in this sense in Table 1, as the change in the the speed-up with the number of workgroups approaches zero at peak efficiency. If the number of workgroups is too high the speed-up will begin to fall again.

Speed-up factors are calculated by comparison with the single workgroup calculation as it is the slowest. For the LANL2 ECP basis set substantial speed-ups are seen up to a maximum of 64 workgroups (with 16 processors per workgroup), where a speed-up factor of almost 7 is achieved. The task-farmed approach is therefore considerably more efficient than using the parallel routines in GAMESS-UK alone. Further gains were not achieved by going beyond 64 workgroups. This is firstly because a larger number of workgroups means that a larger proportion of the calculations do not benefit from an initial wavefunction guess (although for a non-benchmark calculation this could be provided using a preliminary single point evaluation step). Secondly, only 172 gradient evaluations in total are required and therefore the load is not efficiently balanced in the 128 workgroup calculation. For larger systems it may be advantageous to use 128 or more workgroups.

For the TZVP basis set calculations were performed using a single workgroup and 64 workgroups. Similar results are seen, with the 64 workgroup calculation again achieving a speed-up of approximately 7. This indicates that the efficiency gains remain even when large matrix diagonalisations are involved.

4.2 Nudged elastic band optimisation

The nudged elastic band (NEB) algorithm is a method for finding a minimum energy path between two structures. Typically it is used to characterise a reaction path including an energy barrier. The improved-tangent variant of NEB [15] is available in DL-FIND [6].

Each NEB optimisation cycle consists of energy and gradient evaluations for a sequence of structures (images) with geometries that sit along a path between the two endpoints. The final NEB gradient is constructed using spring forces that connect the images. However, the gradient calculations for the images are independent and therefore can be evaluated in parallel.

The NEB algorithm in DL-FIND has been parallelised using static load-balancing. An image is assigned to a workgroup if the image number modulo the number of workgroups is equal to the workgroup ID. This method ensures that a particular image is always assigned to the same workgroup, which is important if the external program uses restart files (as is the case for a GAMESS-UK calculation). At the end of each cycle the energies and gradients are shared between all workgroups by an MPI call within DL-FIND.

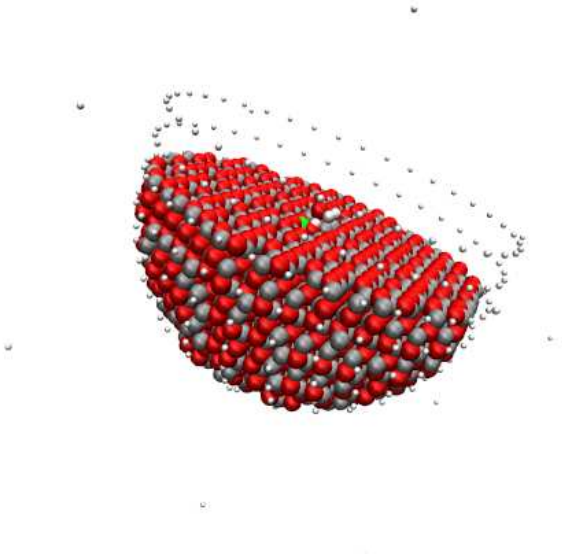
The first NEB cycle is different from the others in that the workgroups each calculate the gradients in serial along the whole path. This is to help convergence of the external QM program, as the wavefunction of the previous image can be used as a guess for the next. In subsequent cycles the corresponding image from the previous iteration can be used as the guess.

The benchmark system for the NEB method is shown in Figure 5. It is a 3207-atom QM/MM cluster consisting of CO_2 and $2H_2$ adsorbed onto an Al-doped zinc oxide surface. The NEB method was used to find the barrier for the interchange of H in H-CO_2 between two different sites. The QM region consisted of 32 atoms with a PVDZ basis (except for Zn, where a Stuttgart ECP was used [16]). This gives a total of 194 basis functions. Following Ref. [4], this region is partitioned into an inner 19 atoms treated fully by quantum mechanics and a surrounding QM/MM interface region with pseudopotentials placed on 13 Zn atoms. The interface region provides a localised embedding potential that prevents the electrons from the inner region spilling out onto the positively charged centres in the MM region.

GAMESS-UK was used for the QM calculations with the B97-1 functional. GULP was used to provide MM energies and gradients using the shell model interatomic potential of Ref. [17]. 10 images are used to describe the path, with the two endpoints frozen, giving 8 gradient evaluations in total per cycle.

A single-point energy and gradient evaluation for the test system is actually an iterative cycle of QM and MM calculations. This is because the QM region is polarised by the MM atoms as point charges and the shells of the MM system are polarised in turn by the QM region. The QM/MM gradient

Figure 5: The ZnO cluster used for the nudged elastic band benchmark calculations. The small spheres on the outside of the cluster are the point charges used for approximating the electrostatic effect of the bulk crystal.



must therefore be iterated until converged each time it is calculated.

Timings for the single point calculation are shown in Figure 6. For large processor counts the scaling is very poor, with a 1024-processor calculation actually taking a longer time to complete than a 256-processor calculation. This means that the overhead of message passing between so many processors outweighs any advantage from the extra computing power. Although the full iterated single-point calculation takes a significant amount of time, the individual QM and MM calculations are quite modest and do not benefit from such large processor counts. Increasing the size of the QM region would make larger processor counts useful, but this would have made running a full benchmark too computationally expensive. The results for this system indicate that the number of workgroups should be set as high as possible (i.e. to 8).

The NEB benchmark calculations were performed over 50 cycles. This results in an effectively converged path. Continuing on to full convergence was not desired as small numerical differences can lead to variation in the total number of cycles for the optimisation and this would not reflect the intrinsic performance of the parallelisation. The most basic form of the NEB method was used, with no climbing image [18] and no freezing of intermediate images during the optimisation.

The results are shown in Table 2. Speed-up factors are calculated rel-

Figure 6: Calculation time in wall clock seconds for a single point energy and gradient evaluation of an embedded cluster model of a ZnO surface reaction.

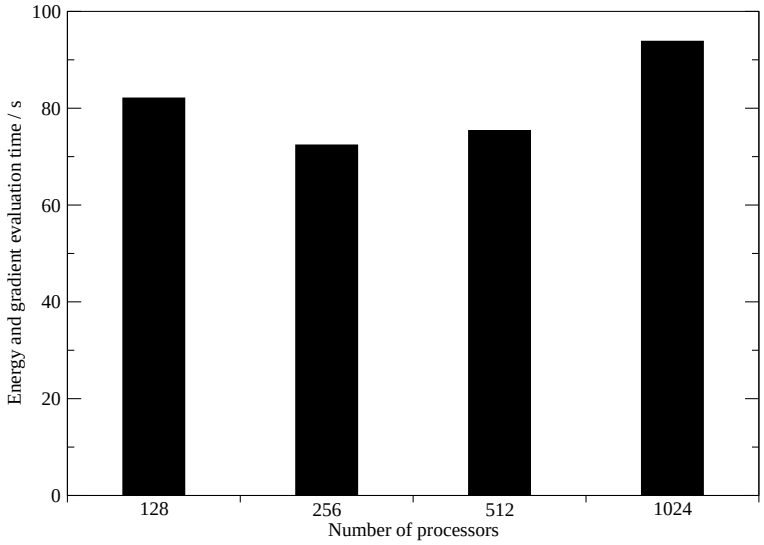


Table 2: Calculation time in wall clock seconds for 50 cycles of a nudged elastic band optimisation. Speed-up factors are compared to single workgroup calculations.

Procs	Workgroups	Procs/ workgroup	Time / s	Speed-up vs 1024	Speed-up vs 256
1024	1	1024	26404		
256	1	256	23536		
1024	2	512	14673	1.8	1.6
1024	4	256	7089	3.7	3.3
1024	8	128	3110	8.5	7.6

ative to the full single workgroup run of 1024 processors and also a run of 256 processors. The 256 processor run represents the best performance achievable using a single workgroup.

The improvement offered by the task-forming approach is significant, and as expected using the maximum number of workgroups gives the highest performance. When compared to a single workgroup 1024-processor run a speed-up of over 8 is found, which is only possible because the single-point 128-processor calculation is faster than the 1024-processor equivalent. When compared to the 256-processor NEB run, the speed-up is lower than 8 but still very substantial.

4.3 Global optimisation methods

Two intrinsically parallel optimisation methods are available in DL-FIND [6], namely a genetic algorithm and stochastic search. These methods are typically used for finding the global minimum on a potential energy surface. Both methods involve calculations on a population of structures during each cycle, which can be run in parallel.

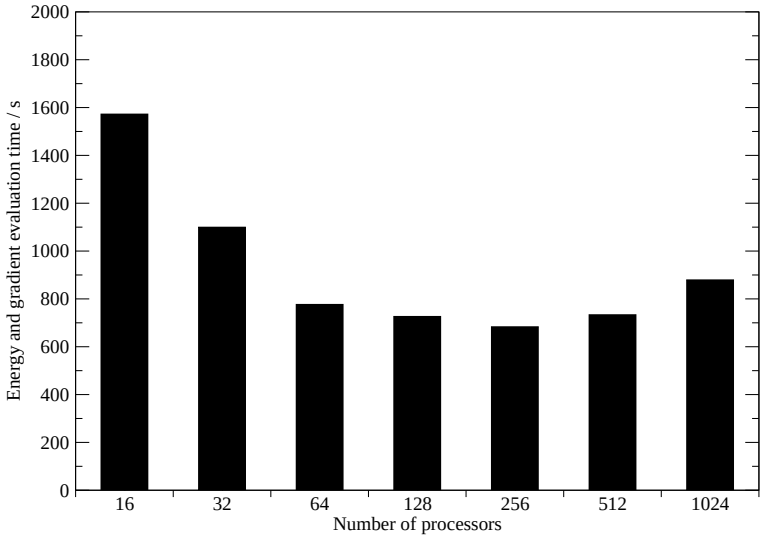
The genetic algorithm and stochastic search methods use the same parallel DL-FIND interface that was used as the basis for parallelising the NEB method. To support the new methods the interface was modified on the ChemShell side to pass the relevant input options to DL-FIND.

A change to the handling of shell model systems in ChemShell was also required to ensure that the parallel optimisations would work with these systems if desired. The DL-FIND optimiser only uses atom positions and ignores shells, which are a feature specific to ChemShell. Previously the shell positions were stored in ChemShell as absolute coordinates and relaxed starting from the old positions when a new geometry was created. This relaxation can be slow or difficult to converge if the geometry changes radically, which is often the case for the global optimisation routines, where a wide variety of geometries is considered at every step. The shell handling in ChemShell was improved by storing shell positions as relative coordinates, so they would stay near to their parent atoms even under a large change of geometry. This change benefits all the other optimisation methods in DL-FIND as well, but is most important for the global optimisers.

Following Ref. [19], benchmark calculations were performed on ZnO nanoclusters. In Ref. [19] rigid ion MM calculations were used but for the purpose of benchmarking a much more demanding QM calculation was set up using GAMESS-UK. Timing calculations were performed on a $(ZnO)_{28}$ cluster. The B97-1 functional was again used with a PVDZ basis (560 basis functions) and the Stuttgart ECP for the Zn atoms. A population of 32 structures was used, which is a typical size for these methods and is an efficient number for task-farm parallelisation.

The scaling properties of the genetic algorithm and stochastic search

Figure 7: Calculation time in wall clock seconds for a single stochastic search cycle (32 energy and gradient evaluations) of a $(ZnO)_{28}$ cluster.



methods are expected to be identical. For the benchmark tests stochastic search was used. A single energy and gradient evaluation is very quick and so to obtain reliable preliminary timings a full cycle of 32 evaluations was performed. The results are shown in Figure 7. Again, the best performance is given by 256 processors with a slowdown observed for higher processor counts. This again suggests that substantial gains can be made by splitting the processors into workgroups.

Normally a genetic algorithm or stochastic search optimisation would be run for hundreds or thousands of cycles in order to have a good chance of finding the global minimum. However, due to computational expense it was not feasible to run a baseline calculation of this length using a single workgroup. To obtain a benchmark the stochastic search algorithm was run for 20 cycles instead. This is expected to give a good representation of the scaling behaviour as each cycle contains the same number of evaluations and should therefore take approximately the same amount of time.

The results are shown in Table 3. Surprisingly, of the two single workgroup runs, the 1024-processor run is faster than 256 processors. This could have been due to natural variation in SCF convergence due to the random element to the geometries created. To test this the benchmark was run twice, but the same trend was found in both runs. This implies that there is some overhead to the large processor count in the first cycle that is not present in subsequent cycles.

Speed-up factors for the task-farmed calculations are therefore given

Table 3: Calculation time in wall clock seconds for 20 cycles of a stochastic search optimisation. Speed-up factors are compared to single workgroup calculations.

Procs	Workgroups	Procs/ workgroup	Time / s	Speed-up vs 1024
1024	1	1024	23535	
256	1	256	26560	
1024	2	512	14881	1.6
1024	4	256	8930	2.6
1024	8	128	5819	4.0
1024	16	64	4270	5.5
1024	32	32	4197	5.6

compared to the 1024-processor run. Gains in performance are again substantial, with the best performance as expected given by maximising the number of workgroups, although the performance of the 16 workgroup calculation is very nearly as good, with speed-ups of over 5 achieved in both cases.

4.4 Overall performance

Substantial gains in performance were observed on implementing task-farming for all three types of calculation. The highest gains were obtained by maximising the number of workgroups (up to the limit where the load becomes unbalanced). This result is expected because the ChemShell level of parallelism involves very little overhead compared to parallel evaluation of the energy and gradient. Single point calculations were good predictors of the final performance and this initial step is recommended as a method for determining the optimal number of workgroups in advance.

5 Conclusion and outlook

The task-farming framework introduced into ChemShell has resulted in a substantial increase in the scalability of the code for the types of calculation considered. All three benchmarked cases (finite-difference Hessian, parallel nudged elastic band, and stochastic search) showed performance gains in excess of the targets set in the dCSE work program. All the programming objectives have therefore been achieved.

The main aim of future work will be to use the task-farming approach on HECToR for scientific applications. An ongoing collaboration with Prof. Catlow's group at UCL will use task-farmed calculations for the study of heterogeneous catalysis, with systems similar to the QM/MM test cluster that was used to benchmark the parallel NEB method. The task-farmed

approach will also be useful in other areas where large-scale calculations are required, such as biomolecular modelling.

There are also still opportunities for further technical developments. First, the task-farming method could be extended to other types of chemical calculation, such as multiple trajectory calculations in molecular dynamics. We also plan to parallelise the finite-difference Hessian code in DL-FIND, which is used by several optimisation algorithms (this is equivalent to the `force` command in ChemShell and will benefit in the same way). Second, further modifications to GAMESS-UK could be made to allow alternative parallelisation methods (ScaLAPACK, global arrays) to work in the task-farmed environment. This is expected to be more technically challenging than the minor modifications required for PeIGS. Third, other codes could be modified to accept the `MPI_COMM_WORKGROUP` communicator and so work in task-farmed mode. The internal DL_POLY code in ChemShell is an obvious first candidate for this work, but collaborations with the developers of other external software packages is also a possibility.

The task-farming implementation will be made generally available as part of ChemShell version 3.4, which is scheduled for release in July 2010. Until then the pre-release code is available on request.

The work detailed in this report will be submitted for publication to *Proceedings of the Royal Society A* as part of a special issue on the activities of the Materials Chemistry Consortium.

6 Acknowledgements

I would like to thank Paul Sherwood for his support during this work and Alexey Sokol and Gargi Dutta (UCL) for their help with the QM/MM cluster example. Huub van Dam (PNNL, USA) provided the silicate structure used for the Hessian benchmark calculations.

References

[1] ChemShell, a computational chemistry shell. See www.chemshell.org

[2] Sherwood P, de Vries AH, Guest MF, Schreckenbach G, Catlow CRA, French SA, Sokol AA, Bromley ST, Thiel W, Turner AJ, Billeter S, Terstegen F, Thiel S, Kendrick J, Rogers SC, Casci J, Watson M, King F, Karlsen E, Sjøvoll M, Fahmi A, Schäfer A, Lennartz C (2003) J Mol Struct (THEOCHEM) 632:1–28

[3] Tcl, the tool command language. See www.tcl.tk

[4] Sokol AA, Bromley ST, French SA, Catlow CRA, Sherwood P (2004) Int J Quantum Chem 99:695–712

[5] DL-FIND, a geometry optimiser for quantum chemical and QM/MM calculations. See ccpforge.cse.rl.ac.uk/projects/dl-find/

[6] Kästner J, Carr JM, Keal TW, Thiel W, Wander A, Sherwood P (2009) J Phys Chem A 113:11856–11865

[7] GAMESS-UK, a package of ab initio programs. See http://www.cfs.dl.ac.uk/gamess-uk/

[8] Guest MF, Bush IJ, van Dam HJJ, Sherwood P, Thomas JMH, van Lenthe JH, Havenith RWA, Kendrick J (2005) Mol Phys 103:719–747

[9] Gale JD, Rohl AL (2003) Mol Simul 29:291–341

[10] The PeIGS parallel eigensolver library. See http://www.emsl.pnl.gov/docs/global/peigs.html

[11] Stephens PJ, Devlin FJ, Chabalowski CF, Frisch MJ (1994) J Phys Chem 98:11623–11627

[12] Hay PJ, Wadt WR (1985) J Chem Phys 82:270–283

[13] Dunning TH (1971) J Chem Phys 55:716–723

[14] McLean AD, Chandler GS (1980) J Chem Phys 72:5639–5648

[15] Henkelman G, Jónsson H (2000) J Chem Phys 113:9978–9985

[16] Igel-Mann G ECP28SDF. See http://www.theochem.uni-stuttgart.de/pseudopotentials/clickpse.en.html

[17] Whitmore L, Sokol AA, Catlow CRA (2002) Surf Sci 498:135–146

[18] Henkelman G, Uberuaga BP, Jónsson H (2000) J Chem Phys 113:9901–9904

[19] Al-Sunaidi AA, Sokol AA, Catlow CRA, Woodley SM (2008) J Phys Chem C 112:18860–18875