Tight Binding Molecular Dynamics on CPU and GPU clusters

Dimitar Pashov
Department of Physics,
King's College London, WC2R 2LS, UK

August 1, 2013

Abstract

The aim of this dCSE project was to improve the TBE code which is based on the tight binding model with self consistent multipole charge transfer. Given an appropriate parameterisation, the code is general and can be used to simulate a wide variety of systems and phenomena such as bond breaking, charge and magnetic polarisation.

The first goal was to achieve better performance through parallelising all suitable routines with MPI. The next step was to integrate ScaLAPACK's parallel diagonalisation routines transparently and with minimal communication, thus allowing the code to run on multi-node machines as opposed to a single node which was already possible thanks to threaded LAPACK/BLAS libraries. The third and last task was to utilise GPUs as accelerators for the heavy linear algebra calculations and subsequently integrate with the MPI parallelisation.

The goals in the first two work packages were achieved mostly as planned with significant benefit gained from exploiting the sparsity of the tight binding Hamiltonian and reformulating the algorithms for calculations of the density matrix elements and related quantities. The electrostatics routines have also seen a significant reduction of memory usage and parallel speedup. A generic diagonalisation interface for all required diagonalisation routines was developed together with the related transparent communication routines. This is now available for all programs in the LMTO suite of which TBE is part. Nearly all of the code has been updated to Fortran 90 and later standards making it easier and much safer to work with.

The third task, GPU porting of TBE, was the more exciting and riskier part of the project and it did not dissappoint in term of the challenges it provided. The original intention was to minimise risk by avoiding native development as much as possible by using established libraries instead. Unexpectedly the diagonalisation routines were nowhere near as fast as expected and this steered us in slightly uncharted territory, writing CUDA code for a number of matrix operations and researching completely different algorithms for obtaining the density matrix. Eventually, the goals were accomplished even though the acceleration is still far from what it was hoped to be.

Contents

1 Introduction

1.1 General description

The tight binding (TB) model is a quantum mechanical method for electronic structure description within the one-particle and adiabatic approximations[1]. The Hamiltonian (H) of a system and other operators in TB are expanded in a minimal, localised, atom centered basis set resulting in the smallest possible matrix sizes (as in ASA-LMTO). In the semiempirical TB models a further assumption is made for implicit basis functions. The H and overlap (S) matrix elements

$$H_{RLR'L'} = \langle \phi_{RL} | V_{R''L''} | \phi_{R'L'} \rangle \rightarrow V_{RLR'L'}(|R - R'|) Y_{RLR'L'}(\phi_{R-R'}, \theta_{R-R'}) \tag{1}$$

$$S_{RLR'L'} = \langle \phi_{RL} | \phi_{R'L'} \rangle \rightarrow Vs_{RLR'L'}(|R - R'|) Y_{RLR'L'}(\phi_{R-R'}, \theta_{R-R'}) \tag{2}$$

are thus are never explicitly calculated, instead they are approximated by parametrised radial pair functions and tabulated angular factors[1]. Usually only the 1 and 2-centre integrals are given in this manner and the 3-centre integrals are completely ignored for reasons of sanity or because they are deemed negligble, for example in metals. While localised atom based functions are not quite orthogonal a further approximation is often made such that the implicit function are orthogonal, further simplifying the underlying eigenvalue problem. Often only the bonding part of the energy is treated quantum mechanically and most of the repulsive part is combined with the core (ion) repulsion term in a parametrised classical pairwise potential, however there are exceptions, for example adding Hubbard terms will act as an approximation to the correlation energy and in magnetic models the exchange is included through parametrised onsite energy shifts. These are usually included in self consistent models. Another important ingredient, and one unique to our TB model, is the more detailed description of the charge density through multipole expansions[3].

The significant amount of approximations leads to what may be called the most transparent and simplest to implement quantum mechanical model. This however comes with the cost and responsibility of providing an appropriate parameterisation. The empirical parameterisation on the other hand provides freedom to fit important quantities to desired values and thus often gives a significantly better description of the modelled system than more advanced ab initio methods. This combined with the lower computational cost and also the significantly better forces/energy consistency of TB means that system size and time length dependent quantities (e.g. diffusion coefficient) can be calculated quantum mechanically on a small computing cluster, within acceptable statistical errors, given a capable implementation.

The advantage over the much faster classical methods stems from the quantum nature of TB and the ability to properly describe the formation and dissociation of bonds (including H-bonds on the same footing), charge transfers and polarisability, magnetism and also negative Cauchy pressures occuring in certain intermetallics.

1.2 Practical implementation

Starting from a given set of parameters the H and S matrices are built. The objective is then to solve the nonlinear problem of finding a charge distribution consistent with the induced potential. This process consists of iteratively recalculating the charge shifts

$$\delta q_R = \sum_L \rho_{RLRL} - N_R \tag{3}$$

and the resulting update to the Hamiltonian matrix:

$$H = H^0 + V_U + V_{xc}; \qquad V_R^U = U_R \delta q_R; \tag{4}$$

[1]The Slater-Koster tables[2]

There are aditional terms originating from the electrostatics and spin polarisation which shall be expanded upon later.

Generally, either the potential or the density or charges are mixed with certain amounts of previous values to avoid instability and accelerate convergence. The mixing is a very complex problem on its own and shall not be addressed further here. The traditional approach to calculate the charges from H is by solving the secular equations in matrix form:

$$HZ = SZ\Lambda \tag{5}$$

through diagonalisation. Here Z is the matrix containing the eigenvectors of H in its columns and Λ is a diagonal matrix holding the respective eigenvalues. A Fermi level ε_f and occupancy function $f(\varepsilon)$ is then found such that its integral is the total number of electrons in the system. The diagonal matrix f is the matrix representation of the occupancy in the basis of Z. The representation in basis of the implicit local atomic basis functions ρ is calculated, using the overlap condition $ZZ^* = S^{-1}$, by:

$$\rho = SZfZ^{-1} = SZf(SZ)^* = SZfZ^*S \tag{6}$$

or

$$(S^{-1}\rho) = ZfZ^*S \tag{7}$$

The occupancy at 0K is a step function equal to 2 for an occupied state and 0 for unoccupied. If uncoupled spin polarisation is also allowed then the occupancy of each spin orbital, (restricted to up and down only) is 1 and 0 respectively. If there are states very close to, or on the discontinuity, smearing may be applied to avoid problems. This yields an entropic contribution to the energy corresponding to the temperature of the smearing.

The density which is calculated separately for different spins is called spin density, and the difference between the two spin densities gives the magnetic moment m of the system:

$$m = \rho_\uparrow - \rho_\downarrow \tag{8}$$

the magnetic moment induces a potential to be added to the Hamiltonian

$$H^{I\uparrow} = -\frac{1}{2}Im; \qquad H^{I\downarrow} = \frac{1}{2}Im \tag{9}$$

the Stoner I is a model parameter. Since spins are uncoupled there are actually two separate Hamiltonians and subsequent quantities must be treated independently. If there is a coupling through additional parameters then the basis functions double in number simply creating larger matrices. Detailed derivation is provided in [4].

1.2.1 Electrostatics

The electrostatic potential is parametrised with respect to $\Delta_{\ell\ell'\ell''}$ parameters which we shall call polarisabilities.

Firstly a matrix with structure dependent constants and one with their radial derivatives (if the calculation of pressure is required), are built before the start of the self-consistency loop:

$$B_{RLR'L'} = \sum_{L''=1}^{L+L'} \frac{4(-1)^\ell(2\ell''-1)!!}{(2\ell+1)!!(2\ell'+1)!!} C_{LL'L''}K_{L''}(R'-R) \tag{10}$$

The Gaunt coefficients:

$$C_{LL'L''} = \iint Y_L Y_{L'} Y_{L''} d\Omega \tag{11}$$

are precalculated in the beginning and reused. The Hankel functions

$$K_L(r) = r^{-\ell-1}Y_L(r) \tag{12}$$

are the actual structure dependent factors. During the self-consistency the charge multipoles Q are calculated on every step:

$$Q_{RL} = \sum_{L'L''} \rho_{RL'RL''} \Delta_{\ell'\ell''\ell} C_{L'L''L} \tag{13}$$

The components of the potential in a spherical wave expansion are a straightforward product of the structure matrix and the multipole block vector:

$$V^M_{RL} = e^2 \sum_{R'L'} B_{RLR'L'} Q_{R'L'} \tag{14}$$

and the update to the Hamiltonian onsite elements and the contribution to the energy are:

$$H^M_{RLRL'} = \sum_{L''} V^M_{RL} \Delta_{\ell\ell'\ell''} C_{LL'L''} \tag{15}$$

$$E^M_2 = \sum_{RL} Q_{RL} V^M_{RL} \tag{16}$$

$L+1$ in B is required for accurate calculation of the forces and this tends to increase its size very significantly for d elements. The forces due to the Madelung terms are:

$$F^M_R = -\frac{e^2}{2} \sum_{R'L'R''L''} Q_{R'L'} \nabla_R B_{R'L'R''L''} Q_{R''L''} \tag{17}$$

A complication arises however in the nonorthogonal case due to offsite contributions:

$$H^M_{RLR'L'} = \frac{1}{2}(D_R + D_{R'})(S-1); \qquad D_R = V^U_R + \sum_{R'} U_R R' Q_{R'0} \tag{18}$$

And additional forces:

$$F^M_R = -\sum_{R'} (V^M_{R0} + V^M_{R'0}) \partial\rho^S_{RR'} \tag{19}$$

$$F^U_R = -\sum_{R'} (V^U_{R0} + V^U_{R'0}) \partial\rho^S_{RR'} \tag{20}$$

The quantity $\partial\rho^S_{RR'}$ needs to be calculated in every iteration:

$$\partial\rho^S_{RR'} = \sum_{LL'} S'^r_{RLR'L'} \rho_{RLR'L'} \tag{21}$$

1.2.2 Quantities of interest

Once at self-consistency the bandstructure energy can be calculated by occupying the energy levels corresponding to $E^b = \sum_n f_n \varepsilon_n$ or expanded in the implicit basis:

$$E^b = \rho S^{-1} H = Z f \Lambda Z^* \tag{22}$$

where the quantities without indices are matrices to be multiplied as such.
A more smoothly varying quantity is the first order energy:

$$E_1 = Tr(\rho H^0) = \sum_{RLR'L'} \rho_{RLR'L'} H^0_{RLR'L'} = \rho : H^0 \tag{23}$$

Together with the second order energy,

$$E^U_2 = \frac{1}{2} \sum_R \left((U_R - \frac{1}{2} I_R) \delta q^2_R - \frac{1}{2} I_R m^2_R \right) \tag{24}$$

the classical term $E_{class} = \frac{1}{2}\sum_{RR'} V^{class}_{RR'}(R' - R)$ and the electrostatics contribution combine to form the total energy.

The corresponding forces are given by:

$$F_R = -2\sum_{LR'L'} H'_{RLR'L'}\rho_{RLR'L'} + 2\sum_{LR'L'} S'_{RLR'L'}E^b_{RLR'L'} - \sum_{R'} V'^{class}_{RR'}(|R' - R|) \qquad (25)$$

Where F_R, $H'_{RLR'L'} = \partial H_{RLR'L'}/\partial R$ and $S'_{RLR'L'} = \partial S_{RLR'L'}/\partial R$ are 3D vectors. Additionally the pressure may be obtained through the radial derivatives:

$$P = -\sum_{RLR'L'} H'^r_{RLR'L'}\rho_{RLR'L'} + \sum_{RLR'L'} S'^r_{RLR'L'}E^b_{RLR'L'} = -H'^r : \rho + S'^r : E^b \qquad (26)$$

NB There is no magnetic contribution to the forces.

1.2.3 Periodicity

For periodic systems the case of solving one problem for an infinitely large system is transposed to infinitely many problems, each one for a small system. The 'infinitely many' part is of course interpolated by a finite number of sampling points.

The H and S are Bloch transformed[5] for each point k of the Brillouin zone:

$$H^k_{RLR'L'} = \sum_T H_{(R+T)LR'L'}e^{ikT} \qquad (27)$$

where T is the translation vector.

The density matrix and other quantities are integrated over the k index to yield the `real` valued matrices, as described. Each k index is entirely independent from all others in TB and this is a great opportunity for efficient and simple parallelisation. Furthermore the number of k-points can be reduced by a significant factor if symmetry is present. This is employed by providing symmetry weight factors for the eventual summation.

The practical complication is that matrices turn from real symmetric to complex Hermitian and the transposition becomes a conjugate transposition.

To summarise, a general schematic view of the workflow in a typical TB calculation is shown in Fig 1.

2 Project overview

The TBE code originally had k-point MPI parallelisation which worked excellently for systems requiring plenty of k-points. There was also an atom based MPI parallelisation for the second(last) step of the electrostatic potential calculation. Linking with a threaded BLAS and LAPACK libraries did benefit the parts using their routines on multiprocessor/core shared memory machines. Therefore the aim of this project is to modernise and speedup the code for larger systems with few k-points by allowing access to clusters with fast interconnects and modern accelerators such as graphics cards.

The following milestones were set out at the beginning of the project:

- Parallelise serial segments of the TBE code with MPI until diagonalisation is the bottleneck (as it should be in TB).
- Parallelise diagonalisation with available standard routines for all combinations of real symmetric and complex Hermitian type, and orthogonal and generalised problems.
- Accelerate the code by initially exporting the heavy linear operations to MAGMA/CUBLAS running on a GPU chip, then rewrite specific routines to minimise memory transfers and integrate with the MPI paralelisation to enable multinode multiGPU operation.

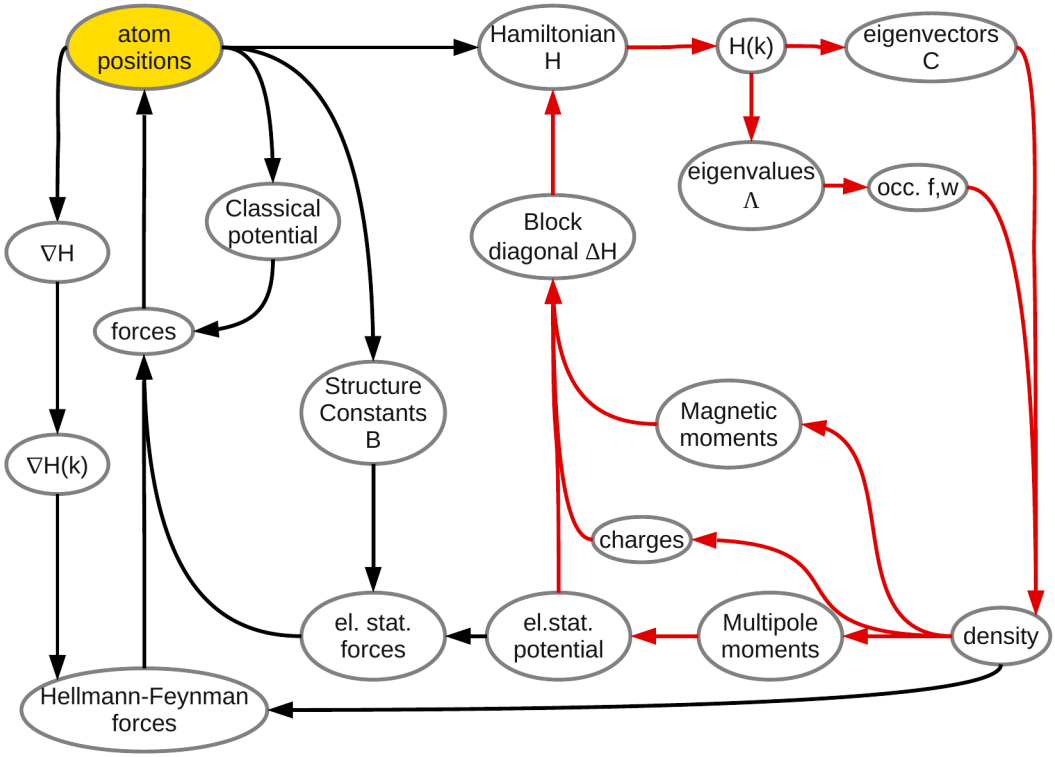

Figure 1: General diagram of the workflow in a typical TB calculation

At the start of the project, the existing independent parallelisation over k-points and atoms in the electrostatic potential were overhauled. Firstly, `mpi_comm_world` was replaced with a communicator with 3D cartesian topology in row-major order. The first dimension now defines how many 2D process arrays will be dealing with the separate k-points and spin. Each array is then allocated a contiguous range of k-points. The dimensioning is performed by an algorithm designed to use the largest possible number of process arrays while keeping the size of these arrays to the minimum necessary and their dimensions as close to square as possible, for efficiency and simplicity. When square 2D arrays cannot be achieved the first dimension is set to be smaller than the second for greater efficiency. The parallel linear algebra routines dealing with H^k,S^k, and ρ^k execute within the 2D process arrays, then a small number of vectors and scalars is reduced across 1D arrays of processes, perpendicular to the 2D arrays, to obtain the integrated quantities. The 2D arrays use a block distribution for the matrix quantities and independent atom distribution density matrix related vector or scalar observables. Outside the k-point/spin loop the electrostatic potential is handled by the 3D communicator containing all active processes.

Since the effort for every k-point is exactly the same in TB the blocks are allocated evenly by number. The atom parallellised regions use more sophisticated but still static allocation because the cost remains the same for diferent iterations and k-points. All communicators: 3D, 2D and 1D and the respective offset, count and blocking related arrays are set once and reused throughout. To enable the reuse of an offsets and counts array in `mpi_gatherv`, block vectors of differing first dimension and type specific MPI contiguous derived types are defined. Although `Gatherv` is generally slower than `gather`, the data consists of comparatively thin vectors, so the effort of packing/repacking outweighs any time difference between the two routines.

2.1 Test systems

The test systems used during development have been a variety of water cells and anatase clusters for symmetric orthogonal cases and FeCr, steel with hydrogen for complex nonorthogonal

cases. Fresh 'stable' versions were successfully deployed by researchers for lengthy simulations of organic compounds immersed in water and other organic solvents and titania/water interfaces.

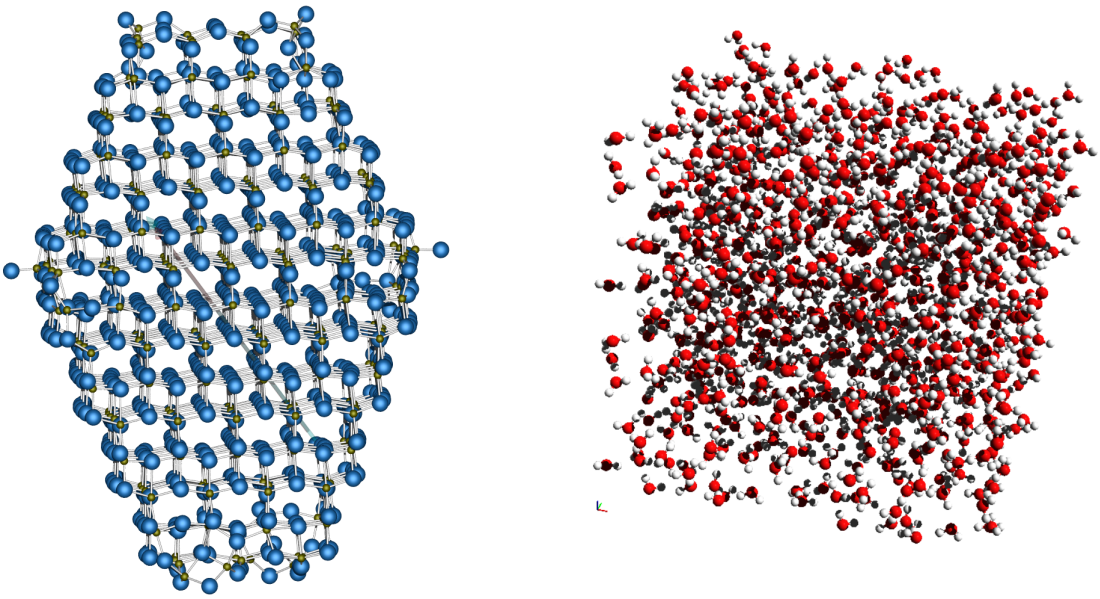

Figure 2: (a) TiO_2 cluster, 1233 atoms, (b) H_2O cell, 3072 atoms

2.2 Hardware and software

The machines used for testing and development during this project include the QUB Linux cluster with over 900 cores in a configuration of dual socket nodes (each with a quadcore Xeon E5530 2.40GHz) and up to 24GB RAM available per node with an Infiniband interconnect; two group owned 16 core dual socket, Xeon E5-2690 2.90GHz systems, each with 64GB of RAM and QDR Infiniband; a 16 core dual socket, Xeon E5-2650 2.00GHz machine with 16 GB of RAM and 2 Tesla K20c GPU cards having 4.8 GB RAM each. It is worthwhile to note that on the E5 range of processors, vectorised FP applications may benefit from the now expanded 256 bit registers and fused multiply add and also the new 3 argument instructions provided by Intel's AVX.

2.2.1 GPU Highlights

The K20c cards implement the latest Kepler GK110 architecture and provide a few useful capabilities over previous generations. A device process/thread is now capable of launching new grids of threads (branding name: 'dynamic parallelism'), up to 32 system processes can now hold contexts on a single device (named 'Hyper-Q'). Another important feature is the direct access to device memory through the kernel driver (conveniently called 'GPUDirect'). The GPU architecture is significantly different from the traditional multi CPU model.

There are 13 multiprocessors (SMX) on one K20c chip, each supplying 192 single precision cores, 64 double precision units, 32 special functions units and 32 load/store units and large shared dynamic register file. Each core executes instructions in-order without preemptive speculative evaluation (similar to the famed but now discontinued Itanium processors) at rather low frequency hence the strength is only in numbers and low utilisation translates directly to low performance.

The software environment is somewhat abstracted away from the lower level grouping details. The execution starts by launching an, up to 3D, 'grid' of again, up to 3D blocks of threads. A thread block can contain up to 1024 threads. Each thread has access to private static memory on one level then to a shared block memory (in configurations of 16, 32 or 48 KB) and on the 3^{rd} level to the global grid memory. The latency and transfer slowdown increase in the same

order. A grid can contain up to $2^{31} - 1$ blocks and the management is handled by the system. How many blocks/threads run simultaneously will depend greatly on the static thread memory and static shared memory allocation besides the limitations on the thread and block numbers / multiprocessor. It is worth noting that every 32 threads share a scheduler and grave inefficiently will be incured if there is any divergence in the instructions they execute. Such a bundle is called a 'warp' and it is a good idea to allocate block sizes in multiples of these and generally keep in mind. Useful tips and best practices may be found in the official documentation and the numerous examples, guides, and tutorials published by NVIDIA.

2.2.2 Software

The clusters previously described use ScientificLinux 5.x and 6.x, and OpenSUSE 12.2 respectively. The Intel compilers and libraries are kept up to date and the OpenSUSE machine benefits from a recent version of gcc/gfortran. MKL's optimised BLAS and LAPACK routines are also available along with precompiled PBLAS/ScaLAPACK/BLACS libraries. The MPI implementation available is OpenMPI 1.6.x with 1.7.1 (testing) only used for the GPU code since it is the only implementation known so far to accept pointers to divide memory and utilise the driver interface for direct memory access. This feature reduces the latency of transfers and is also significant for one part of the code.

3 Parallelise serial segments of the TBE code

The major points in need of improvement are the routines for charges, band energy energies, and force calculations and the structure constant generation. We will start with the structure constants since it is outside the self-consistency loop and is largely independent of the rest, as can be seen from the diagram in Fig 1.

In the early stages of the development, routines for conversions between the global actual and virtual (represented through local pieces) arrays were written avoiding `p{d,z}elset{,2}` and thus saving on the immense latency of millions of connections. Initially for a `scatter` style routine a single block buffer was formed from the global value and set to the appropriate position in the local array, for a gather, the buffer was sent to a root process which then broadcasts the global array after the complete assembly. A more elegant approach was implemented later by employing the MPI `darray` type and `mpi_alltoallw`. There is now a trifold speedup compared to the buffer implementation which is likely to be due to the blocking blacks routines. Since alltoallw is very general, an attempt was made to configure it so that all processes would receive the result instead of just the root but this showed significantly worse performance than the broadcast approach. This is now integrated in the basic library (slatsm), which is used by all programs in the LMTO suite together with a wrapper around the unified diagonalising routine which scatters the input matrices, calls the diagonaliser then gathers and broadcasts them. The overhead is small for the gather operation and negligble for the scatter on a QDR Infiniband network. The `gather` approach is likely to be faster than the `reduce` one which is used in the PLATO code, due to a difference in the basic operations but it was not tested.

3.1 Parallel structure constants matrix with MPI

The structure constants were stored in a 4 index fortran array (L, L', R, R'). This way the atom pair with the largest (L, L') block defines the memory requirements which may be of substantial size since the maximal $L = (\ell + 1)^2$, in this case may reach 2ℓ, as used in the basis functions for the operators and $2\ell + 1$, for when forces or pressure also needs to be calculated in the end of the last self-consistency iteration. For example, a d element like Ti is given up to a d orbital $\ell = 2$ resulting in structure block size for the pair Ti- Ti of up to $((2 \times 2 + 1) + 1)^2 \times (2 \times 2 + 1)^2 = 36 \times 25$ compared with $(2 + 1)^2 \times (2 + 1)^2 = 9 \times 9$ for a

Hamiltonian block. To reduce the memory requirements the original implementation exploited two symmetry properties: $B_{RLR'L'} = B_{R'L'RL}$ and $B_{RLR'L'} = (-1)^{\ell+\ell'} B_{R'LRL'}$.

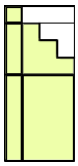

Figure 3: Example B block with the first symmetry rule applied.

The second rule means that the extra $L+1$ part may be calculated only once then obtained from this for the reverse pair. The calculation of a single block consists of first evaluating the Hankel functions for $R - R', L'' = L + L'$ then calculating a dot product over L'' with the Gaunt coefficients stored in packed format. The Hankel function is directly computed for a non periodic case while Ewald summation is performed in the periodic case. This is the slowest part of the calculation.

The electrostatic potential calculation computation consists of a matrix by block vector product between the structure constants and the charge multipoles along the $R'L'$ indices. A convenient optimisation is to swap the indices, resulting in (L', L, R', R) and then spread the R atoms over all available processes as evenly as possible without splitting a block to avoid doubling of the effort for the Hankels. In this way the storage is distributed over the processes and only a small gather operation for the final potential block vector is needed. This strategy was implemented and showed superlinear performance which is most likely due to the decreased memory requirements per process, see Fig 4.

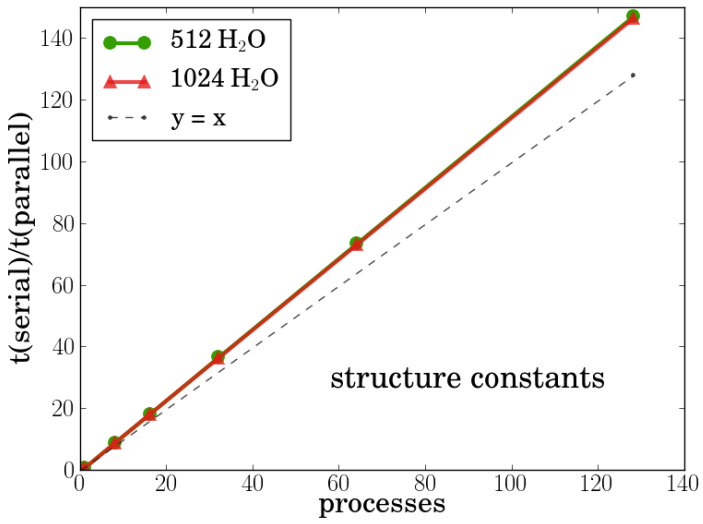

Figure 4: Parallel structure constants generation times.

Since often one needs to study systems containing atoms of different type e.g. H_2O, Ti_2O, etc., the pair with lower L requirements waste a significant portion of the allocated block size. For example in a H_2O molecule we would allocate $(3 \times 9 \times 3 * 16) * 8/2^{10} = 10.125$KB and only use $((9 + 1 + 1) \times (16 + 4 + 4)) \times 8/2^{10} = 2.0625$KB. The difference is striking and for this reason the fixed block format was abandoned. Having the block structure however is more convenient and cache friendly than dealing with the normal array so the solution was to create an index of pointers and preserve the storage format while achieving a fivefold decrease the total memory requirements. This was actually a second revision due to difficulty interfacing the first implementation, which was based on a 2D Fortran pointer array with C/CUDA.

3.2 Density matrix from eigenvectors and occupancies

The density matrix has a fundamental place in the derivation of the nearly all other properties of interest as shown in section 1.2.2. In the original implementation however, the band/state index in the outermost summation results in a repeated recalculation of quantities which require regular updates to wide areas of memory for the charges array, the site energies and forces. Issues were severely complicated by the presence of a non unit overlap matrix.

There is an elegant and efficient way to obtain the full ρ for a positive occupancy function e.g. as in Methfessell-Paxton[6] with N=0, or a plain step function. Each occupied state's eigenvector can be scaled efficiently by the square root of the occupancy function then the array containing all occupied eigenvectors may be transposed. Then an equivalent of `dgemm('t','n', .., .., nstates, 1.0d0,` $\tilde{Z}$`, .., `$\tilde{Z}$`,...)` will access elements sequentially and will be efficient in obtaining ρ. The positive occupancy requirement can be relaxed when the eigenvectors are complex (i.e. not at the Γ point).

$$\tilde{Z} = \left(Z\sqrt{f}\right)^*; \qquad \rho = \left(\tilde{Z}^*\right)\tilde{Z} \tag{28}$$

A second conclusion from section 1.2.2 is that the full density matrix is not actually needed. For all listed properties of interest it will suffice to know of only the blocks matching a corresponding nonzero Hamiltonian block. Fortunately the Hamiltonian (and overlap) matrix in TB is fairly sparse, often no more than 10% filling[2] which decreases with system size. Furthermore the density matrix blocks required need not be saved, except for the diagonal ones necessary for the electrostatics. They can be used on the spot to accumulate a local contribution to the quantities of interest which are only diagonal, and then overwritten on the loop for the next atom. This leads to a walk over the atoms and their neighbours, performing a dot product between block columns of $\tilde{Z}$, the effort for which scales linearly with the number of atoms since the hoppings are short ranged. Additionally if the site energies and forces are not strictly required during self-consistency, the diagonal density matrix blocks will suffice further and reduce the effort by a factor equal to the average number of neighbours per atom. Since a block may be dimensioned anywhere from 1 to 9 by 1 to 9, the dot product is performed by either the `_dot/c`, `_gemv` or `_gemm` routine, depending on the case.

Separate versions for real and complex handling were originally written. Since this was mostly a duplication, one approach would be to use the preprocessor to produce two object files from one source (as done in the ScaLAPACK diagonalisation tester) but a more convenient solution was to use the `real(8)` type and then add an extra index of passed size on the left side of the arrays, which is either 1 for the real case or 2 for the complex. Then procedure pointers `dot`, `gemv` and `gemm` are associated with the appropriate 'd' or 'z' version. Since dot is a function which returns the result in a register, a generic subroutine was defined to work around Fortran's strong typing and type compatibility rules. The overhead of dealing with the extra index and function pointers is not significant and in the author's opinion this solution provides better maintainability. However, since an assumption is made about the storage of the complex type there may be problems with platforms which do not follow this.

3.3 Direct space Hamiltonian

The Hamiltonian building routine has seen a significant rewrite which has resulted in a tenfold speedup, due to memory access optimisations. The cost of the routine was already linear and the real space Hamiltonian storage format sparse. Parallelisation over the atoms did show a speedup but the necessary gather negated any gains. This was not because the gather was slow but rather due to the routine being particularly fast. The time for this routine is now negligble and it is executed only once before the self-consistency takes place.

[2]This may not be considered sparse at all for finite difference style matrices but the total sizes are also of vastly different scale.

The Bloch transform applied at each k-point produces k-dependent matrices which use the real space ones. This routine has also been modified, such that only the necessary local pieces of the global 2D block cyclic distributed arrays are built, thus avoiding the usage of `darray_scatter`.

atoms/edge	k-pts/edge	atoms	1 k-pt	all k-pts	B	B'
1	16	1	0.00	0.06	0.00	0.00
2	8	8	0.00	0.51	0.00	0.00
3	6	27	0.01	2.47	0.00	0.00
4	4	64	0.06	4.11	0.03	0.02
5	4	125	0.25	15.69	0.10	0.07
6	4	216	0.73	46.85	0.31	0.22
7	2	343	1.85	14.77	0.79	0.55
8	2	512	4.11	32.91	1.76	1.22
9	1	729	8.34	8.34	3.56	2.47
10	1	1000	15.69	15.69	6.71	4.66
11	1	1331	27.80	27.80	11.88	8.25
12	1	1728	46.85	46.85	20.02	13.90

Table 1: Projected minimal memory requirements in GB if all H^k and S^k are stored instead of recalculated. Structure constants sizes are also included. Atoms are assumed to be d-elements,

It is possible to avoid the repeated Bloch transforms during self-consistency by saving the matrices for all k-points then only transform the diagonal blocks. This is because the off diagonal updates, when needed, are applied directly to H^k. Table 1 lists the estimated requirements for typical use scenarios. There are cases with few atoms, 100-300, which should be possible to execute on small computers but the memory usage will be prohibitively high.

3.4 Parallelisation

The most efficient parallelisation in view of the above developments was to let each process, of the passed 2D communicator, handle a contiguous sequence of atoms, and not communicate anything globally until the end of the k-point/spin loop. Afterwards a `reduce` operation may then be performed across the 1D communicators followed by a `gather` operation for the vector quantities. For the scalars the `reduce` is then performed on the 3D communicator. The 2D array quantity $\partial\rho^S_{RR'}$ is only required in rare cases and is treated as a rather thick block vector. Inside the 2D communicators the scheduling is similar to the one used for the structure constants, for the generally smaller 2D communicator. This is static by the external routine with a view on spreading the effort evenly, rather than the number of atoms.

4 Diagonalisation with ScaLAPACK and ELPA

Arrays with a block cyclic distribution were mentioned earlier, and in particular with a 1D block cyclic distribution, a vector (or block vector) may be split into blocks of equal size, given by a blocking factor and the blocks are then asigned to a vector of processes in cyclic manner. The concept is simple to extend to an N-dimensional array distributed over a process array of the same dimensionality.

ScaLAPACK works with such 2D arrays and the spread of a square matrix on a square process array is shown in Fig 5. Operations are performed on the blocks with a single process, single or multithreaded, routines from BLAS and LAPACK. The distributed storage format reduces both communication and data duplication.

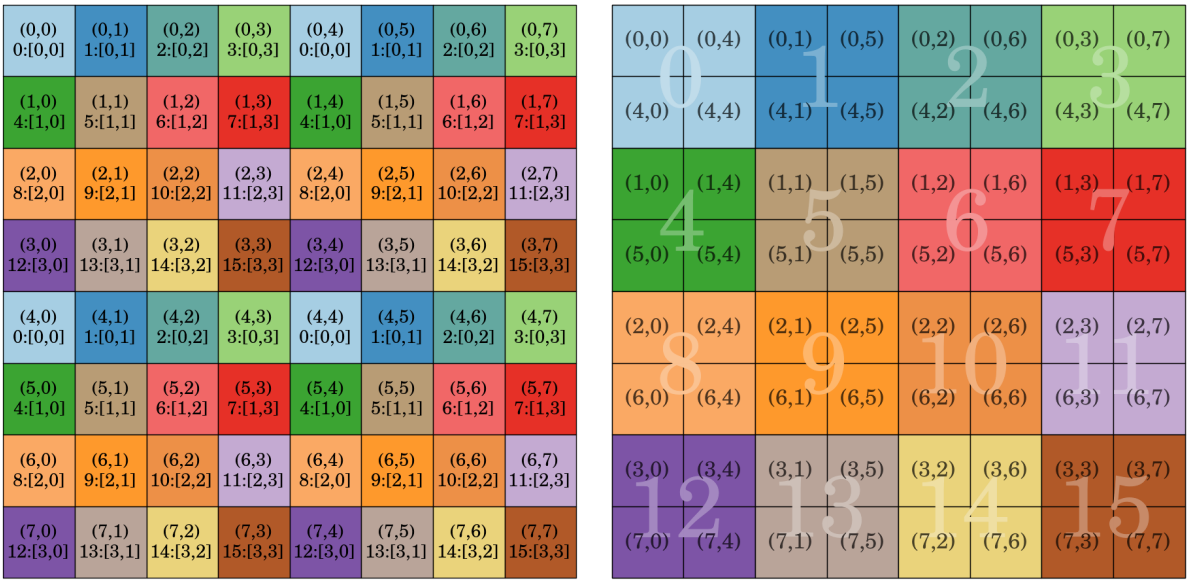

Figure 5: Square virtual matrix with a block cyclic distribution on a square process array. Left: global view of the array split in blocks, right: local parts as stored on each process in the p-array. Block coordinates in parentheses, process coordinate in square brackets and its corresponding linear 'rank' in the watermarks.

A generic wrapper routine was written which mimics closely the 'advanced' type diagonalisation drivers from ScaLAPACK with a few practical deviations. The routine manages all work arrays necessary internally and handles all combinations of { ,p}{dsy,zhe}{e,g}v{z,d,r} covering all cases of interest. It is also an interface to the diagonalisation routines in ELPA[7] since these use the same 2D block cyclic distribution as ScaLAPACK.

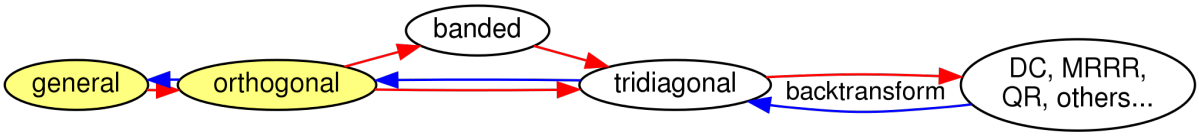

Figure 6: Two stage tridiagonalisation diagram in the context of diagonalisation of symmetric/Hermitian matrices

The ELPA library attempts to speedup the tridiagonalisation step (the bottleneck in diagonalisation) by replacing the BLAS2 `*mv` routines with BLAS3 `*mm` 6 at the cost of slower backsubstitution and the hope that not all eigenvectors will be required, which is the case in TB, especially for insulators. The optimal blocking factor was found to be 32.

The wrapper routine accepts 2D block cyclic distributed local arrays with respective descriptors and in the serial case calls the appropriate LAPACK routine instead of the parallel ones. This allows one to use exact wrappers or null libraries and thus produce single object file which can be packaged into a purely serial or parallel executable.

Since the Bloch transform is fast, local only and relatively simple even when one atom was spread across 2 or 4 processes and since the cubic scaling cost of diagonalisation with respect to the matrix dimensions a decision was made to not apply any padding and in this way minimise the size. The time for diagonalising genuine sparse Hamiltonian atoms or a randomly generated one is the same and any diagonal padding entries will increase the effort unnecessarily.

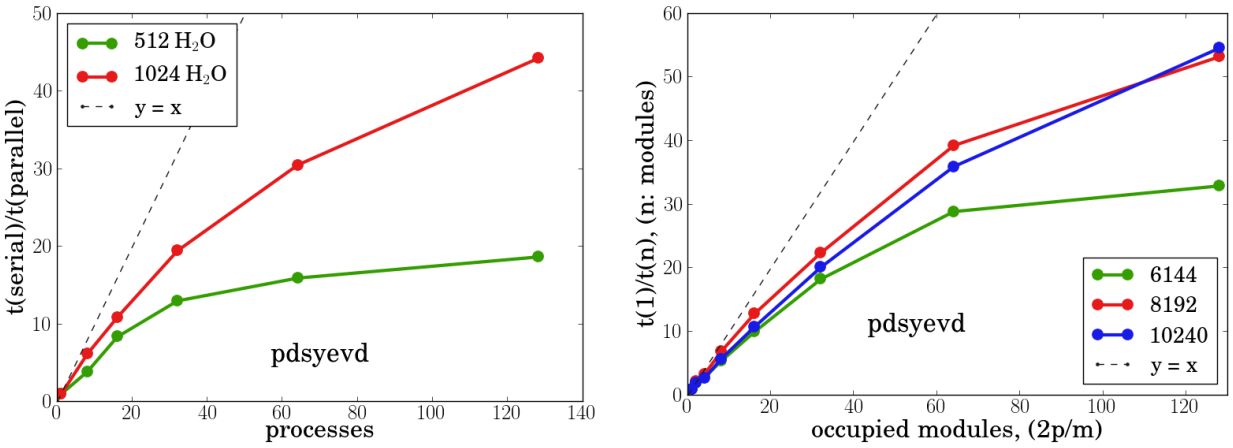

Figure 7: `dpsyevd` speedup on: (a) QUB's cluster and (b) HECToR. 1024 H_2O molecules corresponds to matrix size 6144 × 6144.

4.1 Modifications to the density matrix procedure

It is more appropriate to apply any padding during the calculation of the density matrix from the eigenvectors. This is because some communication will be necessary either from there, or from the libraries, or from both. It is not straightforward to determine the best approach, after the developments in `tbfrce` from section 3.2, specifically this is due to the linear scaling atom parallellisation. The `tbfrce` routine has been modified to accept virtual global array for the eigenvectors and the transposition is now done with a single `mpi_sendrecv_replace`/process on the local block for the square process array case and blockwise with an asynchonous ready mode `send` and `recv` for the general case of the rectangular process array. `p{d,z}tran` was not used because it relies on blocking communication. A further optimisation is also possible when the process array dimensions have a common multiple. The blocks to communicate will them be aligned in columns or rows but this was not pursued since the general case is not slow by any means.

The neighbour table walk for the off-diagonal density matrix derived quantities was parallelised over the atoms in WP1. This is such that there is now a choice of keeping this parallelisation and then communicating the necessary blocks of eigenvectors spanning all neighbours of an atom, or alternatively use PBLAS's pdgemm to multiply the necessary block columns. However, the second option will result in plenty of small implicit communications:

$$(number\ of\ neighbour\ pairs) \times ((nstates/blocking\ factor)\ \texttt{modulo}\ column\ processes)$$

while the first will require more networking in the beginning it will then be able to use local BLAS gemm for better load balancing. The first idea was chosen for both its simplicity and better efficiency. On a side note, testing pdgemm and multithreaded dgemm revealed that surprisingly dgemm is up to 2× faster (on a single, 16 core E5 machine) than pdgemm, while a similar benchmark of the diagonalisation routines showed either close timing or an advantage for the ScaLAPACK routine. It is thought that the threaded MKL that `dgemm` uses employs an algorithm which is faster than $O(N^3)$ and can therefore gain an advantage on larger sizes, while the process management and communication slows down the ScaLAPACK version at smaller sizes.

Overall the set target of a twenty fold speedup of the whole code over the serial version on a 24 core machine was not achieved. At least 32 cores were required to reach anything near this performance.

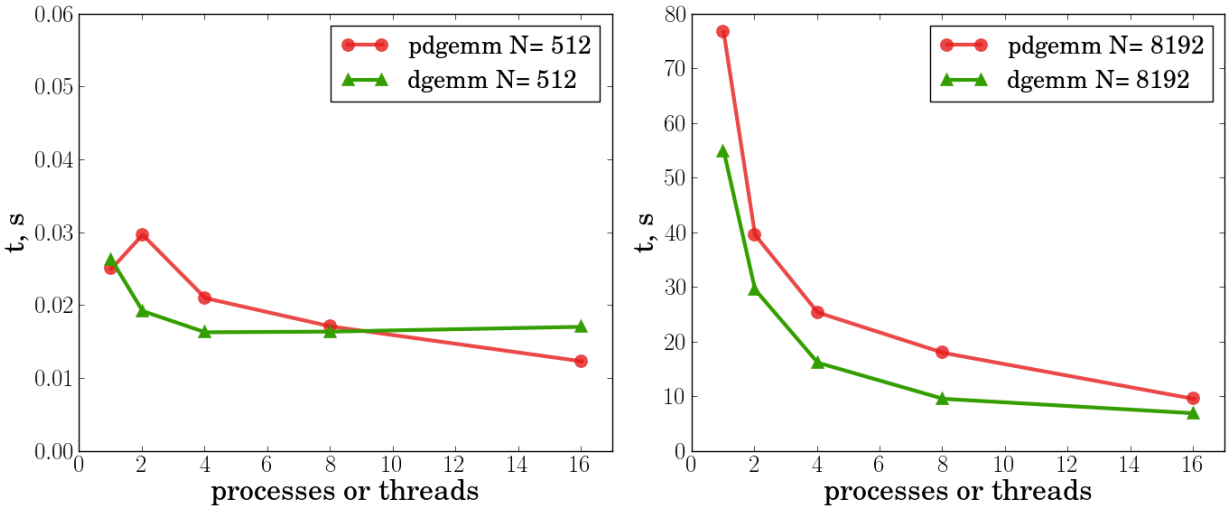

Figure 8: `pdgemm` vs threaded MKL `dgemm` times for small and larger matrices. `pdgemm` blocking factor=32

5 Porting to GPU(s)

The initial aim of this task was to speedup the slowest part of the TB calculations, the diagonalisation, by offloading it to a single GPU. Once achieved, the plan would then be to utilise more GPUs which could possibly be connected to the nodes of a cluster. This would enable the combining of ScaLAPACK's MPI parallelisation with accelerated CUBLAS executing locally on the GPUs. Later, the other two heavy parts of the code: the structure constants evaluation and the density matrix were also to be ported. The Hamiltonian assembly involves rather divergent code paths. It is also very quick on a CPU and it was therefore deemed unnecessary to port it.

Initial testing of the new GPU revealed disappointing results for our greatest hope: the diagonalisation routines. While one card was nearly twice as fast, for sizes of ≈10000, when compared with performance on the 16 cores of the Xeon E5-2650 2.0GHz machine. But it was also slower than on the Xeon E5-2690 2.90Hz machine. It was originally thought that this may have been due to a problem with either the test or setup. However, this was dissolved after the results were confirmed by the MAGMA team. The 2.0GHz cores are not twice slower than the 2.9GHz cores of the same model but the memory availability, latency and frequency do differ. Memory access affects `dgemv` severely and this is the major cause of the difference in the performance of the diagonalisation drivers.

N	E5 2.9GHz	K20c	E5 2.0 GHz
6144	11.58	12.23	27.27
10240	47.62	50.45	121.7

Table 2: Time for diagonalisation with `pdsyevd` and MAGMA's `dsyevd` in s.

For matrix multiplication however the situation was more satisfactory with one card being from 2 to 4 times faster, for sizes ranging in the mid thousands. For smaller sizes a CPU may still outperform a K20 card even with the highly optimised `cublasDgemm`. This is likely to be the reason why simply plugging CUBLAS/MAGMA in TBE to replace the BLAS/LAPACK in ScaLAPACK showed a significant slowdown. In this case, it is feeding matrices of block size equal to the single process BLAS/LAPACK and block sizes of 2048-3072 are impractical for TBE just to break even, on the slower Xeon E5-2650 2.0GHz CPU.

N	E5 2.9GHz	K20c	E5 2.0 GHz
6144	3.70	0.45	4.16
10240	9.82	2.05	16.93

Table 3: Time for `dgemm` from MKL and CUBLAS in s.

5.1 Direct density matrix solution

Therefore, the solution was to use the cards for what they were good at, namely large matrix multiplication (dgemm)[3], with an algorithm for obtaining the density matrix only through multiplication and no diagonalisation.

The basic idea behind the method is as follows. The Hamiltonian matrix and the eigenvalue diagonal matrix are representations of the same operator but in different bases. Exactly the same transformation links the occupancy function and the density matrix. The occupancy is a function of the eigenvalues and for the simple case of insulators it is a Heaviside step function positioned anywhere within the band gap. If a fast converging polynomial approximation is found for the occupancy function and its position (Fermi level) is known a priori, then evaluating the polynomial directly on the Hamiltonian by only using matrix multiplications and scaling will result in the full density matrix. A suitable polynomial may be generated by recursively applying $f(X) = X^2(3I - 2X)$[8]. Each iteration brings the resultant polynomial closer to the step function. For water and titania approximately 7 iterations are enough to converge ρ to within machine precision, from the one computed through diagonalisation. There are two matrix multiplications per iteration and a serial diagonalisation is roughly 5-6 times slower than a matrix multiplication in the same conditions. In the end a full density matrix obtained from a single GPU card was about 1.5-2 times faster than one obtained from the fast Xeon E5-2690 2.90Hz machine. A number of polynomials were found or developed and tested and the most efficient and convenient is the sign function approximation: $X_{n+1} = \frac{1}{2}X_n\left(3I - X_n^2\right)$ with a subsequent flip and shift of the resulting array. It is desirable to pick the chemical potential closer to the centre because the order of the polynomial will be lower and the number of operations will be reduced.

5.1.1 Multi GPU

An attempt was made to cautiously use source code from pdgemm and its associated routines to develop an MPI parallel matrix multiplication routine using `cublasDgemm`, but this did not perform well. Eventually, a custom block column distributed routine did show an acceptable speedup over a single card. This was then developed to duplicate the full matrix on all devices by using OpenMPI 1.7b's ability to work with GPU memory addresses directly.

5.2 Structure constants on GPUs and MPI

With the density matrix obtained, albeit in a way not envisioned in the beginning, the next and final challenge was the structure constants matrix. All code necessary for the structure constants was rewritten in `C` and tested for bugs and performance on the CPU. An initial naive implementation on a single GPU using a single thread for a block corresponding to one atom pair was developed first. However, the performance was an order of magnitude slower than what could be achieved with 16 CPUs. After a number of incremental improvements, the time to build the full matrix on the GPU was brought lower than the one from the CPUs. The final implementation check was to determine the maximal block size available in the matrix and then launch an appropriate template function instance which is known to give reasonable performance. It is worthwhile to note that a thread block size set to the maximal B block size is fairly wasteful, but on the other hand a thread block too small (i.e. less than 32) may be

[3]surprisingly dsymm was slower than dgemm

divergent and will also overload the threads with the local memory requirements, thus resulting in not very many being launched, so there is a balance to be struck. Therefore, inside each thread block, one thread initialises all shared variables to the appropriate values and all threads cooperate in the evaluation of the Hankel function. Afterwards each thread computes one or more elements of the structure constants. This routine is integrated within the overarching MPI parallelisation giving each GPU a proportionally lower amount of data to be set. This setup gives a significant performance increase over the 16 core system.

6 Conclusion and future work

The parallel scaling of the diagonalisation routines is not particularly impressive at about 40%, however on a machine with a lower CPU/network performance ratio than an Intel Xeon E5 with a QuickPath Interconnect, it would be expected to be better. But, what is most important to researchers is the wall clock time and this will not improve. The greatest speedup in this project was obtained from using the sparsity of the Hamiltonian, H, to enable a selective density matrix calculation with an efficient parallelisation, turning the $O(N^3)$ problem into a linear one. The other significant benefit was from the changes in storage and parallelisation of the structure constants generation. The size of the matrix of structure dependent constants, B, was previously a significant barrier to the amount of atoms simulated. This is now fully distributed and is therefore significantly smaller, particularly if there are only few heavy atoms in a system of predominantly light ones.

Even though the GPU platform results were far off the set target it is likely, as the phenomenal performance of the `cublasDgemm` shows, that in future versions the libraries could be improved. The exploration of the iterative methods for obtaining the density matrix led to plenty of experimentation and ideas for extentions to metals which are worth exploring in the future. During this project, contact was also made with two international groups, who have each developed their own $H \rightarrow \rho$ sparse libraries, one of them with a novel algorithm different than the classical approach implemented here. In view of the intrinsic limits of dense diagonalisation and the inefficiencies of its parallel implementations it may turn out that the direct iterative procedures are a better solution for simulating larger numbers of atoms in the tight binding framework.

In closing, the CUDA development was an interesting experience and one definitely worth having.

Acknowledgements

This project was funded under the HECToR Distributed Computational Science and Engineering (CSE) Service operated by NAG Ltd. HECToR – A Research Councils UK High End Computing Service – is the UK's national supercomputing service, managed by EPSRC on behalf of the participating Research Councils. Its mission is to support capability science and engineering in UK academia. The HECToR supercomputers are managed by UoE HPCx Ltd and the CSE Support Service is provided by NAG Ltd. `http://www.hector.ac.uk`. I would like to thank Dr. P. Ridley for the patience and help with the proposal and report, and Prof. A. T. Paxton, Dr. A. Lozovoi, T. Sheppard and Dr. A. Elena for the help and discussions.

References

[1] M. Finnis. *Interatomic Forces in Condensed Matter*. Oxford University Press, USA, 2003. 3

[2] J. C. Slater and G. F. Koster. Simplified LCAO Method for the Periodic Potential Problem. *Physical Review*, 94(6):1498–1524, 1954. 3

[3] M. W. Finnis, A. T. Paxton, M. Methfessel, and M. van Schilfgaarde. Self-consistent tight-binding approximation including polarisable ions. 491:265–274, 1997. 3

[4] A. T. Paxton and M. W. Finnis. *Phys. Rev. B*, 77, 2008. 4

[5] F. Bloch. ber die quantenmechanik der elektronen in kristallgittern. *Zeitschrift fr Physik A Hadrons and Nuclei*, 52(7-8):555–600, 1928. 6

[6] M. Methfessel and A. T. Paxton. High-precision sampling for brillouin-zone integration in metals. *Phys. Rev. B*, 40:3616–3621, Aug 1989. 11

[7] T. Auckenthaler, V. Blum, H. J. Bungartz, T. Huckle, R. Johanni, L. Krämer, B. Lang, H. Lederer, and P. R. Willems. Parallel solution of partial symmetric eigenvalue problems from electronic structure calculations. *Parallel Computing*, 37:783–794, 2011. 13

[8] X. P. Li, R. W. Nunes, and D. Vanderbilt. Density-matrix electronic-structure method with linear system-size scaling. *Phys. Rev. B*, 47:10891–10894, Apr 1993. 16