Parallelisation and porting of UKRMol-in, the electron-molecule scattering inner region R-matrix codes

Paul Roberts
Numerical Algorithms Group Ltd,
Wilkinson House, Jordan Hill Road, Oxford, OX2 8DR, UK,
email: support@nag.co.uk
Michael Lysaght
Department of Physical Sciences, The Open University.

1st June, 2012

Abstract

This dCSE project will concern the parallelization of the SCATCI program within the UKRmol-in suite of codes. For large Hamiltonian systems, most of the compute time within SCATCI was spent on diagonalizing the Hamiltonian matrix and so efforts have focused on parallelizing this part of the SCATCI program. The original project plan was to implement a parallel code, using mixed mode OpenMP/MPI and the PARPACK library. SCATCI has been interfaced with the Scalable Library for Eigenvalue Computations (SLEPc) which is built on top of the well-known PETSc library and the Message Passing Interface (MPI) library. The UKRMol-in suite contains a series of programs, which perform various tasks such as: the calculation of integrals over target and continuum orbitals (and corrections due to the finite size of the R-matrix) and orthogonalisation of the set of orbitals; the construction and diagonalization of the Inner Region Hamiltonian; and in the target run mode the calculation of density matrices and from them target properties. The suite also contains modules to generate Hartree Fock-SCF or pseudonatural orbitals and the basis sets for the description of the continuum. These tasks are fundamental to astrophysics, plasma physics and damage process in biological environments. UKRMol is currently being used to study mechanisms of DNA strand breaking caused by low-energy electron collisions.

This project was approved for 12 months effort in April 2011 and was completed May 2012. The work was supervised by Dr Jimena Gorfinkiel of the Department of Physical Sciences at The Open University. The original candidate was Dr Michael Lysaght who worked on the project from April 2011 until November 2011. Dr Paul Roberts then took over until the conclusion in May 2012.

Contents

1 Introduction

1.1 Application Overview

The ability to model and understand electron-molecule scattering processes is of fundamental relevance in a variety of research and technology areas: astrophysics, plasma physics, the understanding of the damage process initiated by ionising radiation in biological environments (the cell), etc.. The methodology to treat these processes at low projectile kinetic energies (below the ionisation threshold) is fairly well developed. In particular, the UK has been at the forefront of the field, with use of the R-matrix method to treat the problem in an ab initio manner. Specifically, the UK R-matrix polyatomic suite [1], UKRMol, is one of the most accurate codes in the world to describe the electronic part of the problem. These codes are also used to study positron scattering. Radiation uses in medicine (for treatment and diagnosis) have been developed for many decades mostly as empirical macroscopic techniques. However, recent important biomedical advances involving radiation are demanding an increasingly detailed level of description of the nanoscale, molecular interaction processes involved. Experiments confirmed almost a decade ago that secondary electrons with energies up to 20 eV can damage DNA [2]. Detailed experimental studies with DNA strands and DNA/RNA constituents have confirmed that electron collisional mechanisms (such as dissociative electron attachment) are highly efficient in producing structural changes leading to biological and physiological alterations. This intense experimental activity [3] has not been matched by theoretical studies, with such work tackling only the completely elastic process. Exceptions are the studies of uracil [4] and the sugar molecule tetrahydrofuran using the R-matrix codes [5]. Application of UKRMol to other new fields like near threshold ionisation [6] have also recently been pioneered.

The atomic versions of the R-matrix codes (PRMAT) have been ported and optimized on HECToR (by Dr M Plummer and Dr A G Sunderland) in a previous dCSE project and there is further dCSE funded work to interface the PFARM part of PRMAT with the UKRMOL-in suite of codes and to parallelize the construction of the atomic Hamiltonian (Dr M Plummer).

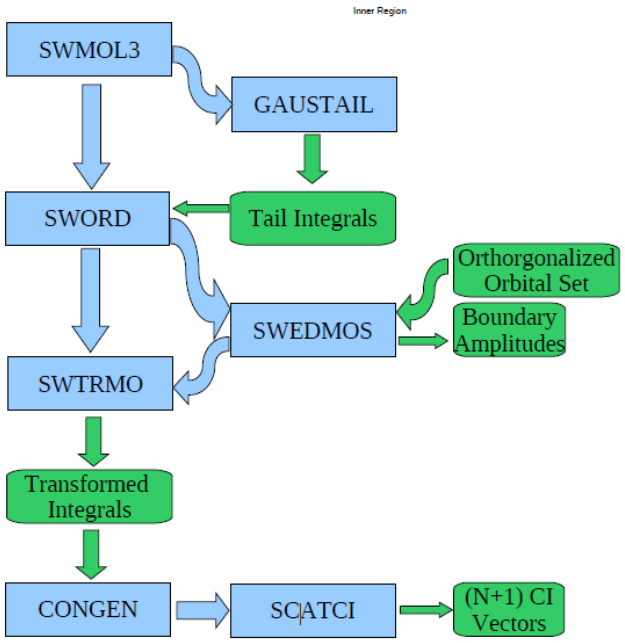

Figure 1: A flowchart of the main programs of UKRMol-in.

The UKRMol-in suite contains a series of programs, see Figure 1, which perform the following tasks:

1. The calculation of integrals over target and continuum orbitals (and corrections due to the finite size of the R-matrix) and orthogonalisation of the set of orbitals (the SW- and GAUSTAIL programs in Figure 1);
2. The construction and diagonalization of the Inner Region Hamiltonian (the CONGEN and SCATCI programs);
3. In the "target run" mode, the calculation of density matrices and from them target properties (GAUSPROP and DENPROP programs, not included in Figure 1).

The suite also contains modules to generate Hartree Fock-SCF or pseudonatural orbitals and the basis sets for the description of the continuum. The former is now being superseded by the use of standard quantum chemistry codes to generate more sophisticated orbitals (in particular MOLPRO). The latter are run infrequently as normally these basis sets only depend on the radius of the R-matrix sphere and the charge of the target. The computational requirements to run these programs are very modest. The UKRMol codes are available to UK academics and to non-UK scientists through the CCPForge website [8] The code is currently used by groups in India, the US, Canada, France and Japan. A user community for the codes is being developed using the tools available in the site (mailing lists, forums, bug reporting and tracking, etc.). In addition, UKRMol-in will form the basis for the time-dependent suite to treat the dynamics of molecules in ultra-short light pulses.

1.1.1 Code Status Before Start of Project

The UKRMol-in suite had been run on a Linux cluster at the OU, on the Dell Legion cluster at UCL, on a SGI machine and on Linux workstations (the most powerful with 64Gb memory and 8 processors). Most "production" runs were serial and could take anything from a few hours to up to a couple of months (where more than 99% of the time is taken by the Hamiltonian diagonalization step; a checkpoint scheme being in place so that this step can be re-started if necessary). Memory and disk requirements vary but for large runs these could be of tens of Gb each. For example, a medium-sized calculation for thymine (C5H6N2O2) using a cc-pVDZ basis set and a (14,10) active space will generate a Hamiltonian of size around 185000×185000. SCATCI requires 422 minutes of CPU time just for the Hamiltonian construction. Using the serial ARPACK diagonalizer, $\sim$ 33Gb of memory are used when 5000 eigenvectors are requested and the job takes 2606 minutes. (CONGEN takes around 76s CPU time to generate the configurations; all times are for Intel Xeon CPU E5540 @ 2.53GHz CPU and executable compiled with Intel v11.1). An OpenMP version of the program has been produced by Dr Pavlos Galiatsatos which uses the ARPACK diagonalisation routine. This current implementation performs as follows on a 64Gb Linux workstation (Xeon processors) for a 524,000 matrix dimension when 3000 roots are requested (this OpenMP version has also been run on an SGI machine): using four cores the speed-up is 2.5; using 8 cores the speed-up is 4.0. The current serial version of SCATCI interfaces with the well known ARPACK library which is based upon an algorithmic variant of the Arnoldi/Lanczos process called the Implicitly Restarted Arnoldi/Lanczos Method (IRAM). The ARPACK library adopts a reverse communication interface which, in practice, means a spMV multiplication mechanism needs to be provided when interfacing to the library. In the current serial version of SCATCI, spMV multiplication is implemented within the subroutine MKARP. Within MKARP, the lower triangular part of the symmetric H^{N+1} matrix is read into RAM from a sequential access file (where it is stored in an unordered coordinate (COO) symmetric storage format). In order to obtain the eigenvalues and eigenvectors, a loop until convergence is performed with successive calls to ARPACK's DSAUPD subroutine, which requires the reverse communication interface and where spMV multiplication mechanism is provided in house.

Most of the compute time within IRAM is spent in the spMV multiplication stage and therefore initial efforts have focused on parallelizing this stage of the algorithm. To this end, simple modifications to the existing serial spMV multiplication mechanism have been introduced using OpenMP DO directives. Since many of the subroutines within ARPACK interface with

LAPACK and BLAS subroutines, multi-threaded parallelism was also introduced to these stages within the IRAM algorithm by linking to optimized threaded LAPACK and BLAS libraries such as Intel's Math Kernel Library (MKL) with the inclusion of relevant flags at compile time.

2 Project Background

The original proposal submitted to the dCSE panel requested 12 months effort. The original objectives for this project were as follows:

Work package 1: Familiarization and porting to HECToR. (Effort: 1 month, Full time)
By mid-May Dr Michael Lysaght will be familiar with running a standard, non-parallel run on HECToR (the water test currently available in the CCPForge site will be appropriate for this)

Work package 2: Parallelisation of the Hamiltonian construction. (Effort: 3.5 months, Full time)

Work package 3: Optimization of a sparse-matrix vector multiplication mixed OpenMP-MPI approach. (Effort: 3 months, Full time)

Work package 4: Implementation and optimization of PARPACK for the diagonalization step. (Effort: 4 months, Full time)

Work package 5: Final Report. (Effort: 0.5 months, Full time)
To be produced in html and .pdf format ready for publication on the HECToR website. Status update on code being released to users.

For the milestones in Work Packages 2, 3 and 4, the code developments will be demonstrated on three different tests that will cover several different characteristics of the calculations we hope to run in the future:

1. positron – HCCH scattering: D2h symmetry , use of pseudostates and partial waves up to l=6;
2. electron – adenine scattering: Cs symmetry, 15 atoms;
3. electron – water dimer: C2v symmetry, 6 atoms, standard close-coupling model.

All these systems have been extensively studied at the OU and UCL, so a variety of different models (larger and smaller Hamiltonians) can be provided if needed. Each test case will include a Hamiltonian with 0.5 million configurations (a 500,000 x 500,000 matrix). Preliminary tests on HECToR show that the sparse matrix vector multiplication for a Hamiltonian of this size can be performed within a single Phase 2b XE 6, 24 core node. However, it will be fundamental for milestone 3 that the tests are run on two Phase 2b equivalent nodes. It is also expected that the Hamiltonian construction (milestone 2) will need to run on more than one node, so efficiency will be tested when running on two nodes. The test for milestone 4 is expected to run on 24 cores (single node).

2.1 Changes to the Workplan

At the time Michael Lysaght started work on the project he, along with Jimena Gorfinkel, attended a meeting sponsored by CCP2 on Methods and Codes for Atoms and Molecules in Strong Laser Fields in April 2011 [9]. Whilst there they discussed their work with several colleagues who suggested they considered using PETSc/SLEPc for the Hamiltonian construction and diagonalization. They were particularly impressed by reports from a group in Madrid (who run on Mare Nostrum) on its efficiency and the size of the matrices they were diagonalizing.

SLEPc is the Scalable Library for Eigenvalue Problem computations that is built on top of PETSc (Portable, Extensible Toolkit for Scientific Computation which is available on HECToR) and is considered an extension of PETSc. It enforces the same programming paradigm as PETSc.

Michael Lysaght had a look at PETSc/SLEPc in more detail and concluded that they were a better option than PARPACK for the following reasons:

1. it uses a data-structure neutral implementation - problems can be solved with matrices stored in parallel and serial, sparse and dense formats;
2. it also has run-time flexibility, giving control over the solution process;
3. it is usable from code written in C, C++, F77 and F90 and offers portability to a wide range of parallel platforms;
4. the flexibility of PETSc/SLEPc in terms of eigenvalue solvers (there are several implemented);
5. there is extensive documentation users manual, example programs, online manual for subroutines;
6. and there is also seamless integration with well-established packages such as ARPACK.

This led to the goals in WP3 and WP4 of a hybrid OpenMP/MPI and PARPACK approach being replaced by using PETSc/SLEPc (which could also be used for the parallelisation of the Hamiltonian construction in WP2.)

The SLEPc library is based on PETSc data structures and it employs the MPI standard for message-passing communication. The three basic abstract PETSc data objects are index sets, vectors and matrices. Built on top of this foundation are various classes of solver objects, which encapsulate virtually all information regarding the solution procedure for a particular class of problems, including various options such as convergence tolerances, etc. The Eigenvalue Problem Solver (EPS) is the main object provided by SLEPc. It is used to specify an eigenvalue problem, either in standard or generalized form, and provides uniform and efficient access to all of the eigensolvers included in the package. There are several methods available for solving eigenvalue problems within SLEPc including the Arnoldi/Lanczos method, the Jacobi-Davidson method and the Krylov-Schur method, the latter being the default method (and the one currently used) due to its very effective restarting technique.

3 Implementation

3.1 An overview of the modifications made to SCATCI

The construction and diagonalization of both the N- and (N+1)-electron Hamiltonian in the R-matrix inner region occurs within the SCATCI program. Upgrade work on SCATCI has focused on parallelizing subroutines involved in both the construction and diagonalization of the (N+1)-electron Hamiltonian, H^{N+1}. For problems in which the order of H^{N+1} is large ($>$ 100,000), over 90% of UKRmol-in compute time is spent within subroutines associated with the diagonalization of H^{N+1}. In choosing numerical methods to diagonalize H^{N+1}, it is noted that the partitioned R-matrix method requires only $\sim 5\%$ of eigenpairs from the diagonalization of H^{N+1} and also that, in the majority of cases, H^{N+1} is extremely sparse ($\sim 99\%$ sparsity). For these reasons, iterative Krylov-subspace-based eigensolvers have been chosen to compute eigenpairs, due to their efficiency over direct methods for such problem types. Since the kernel method of such eigensolvers is sparse Matrix-Vector (spMV) multiplication, these eigensolvers also lend themselves to parallelization on both shared memory and distributed memory architectures. The serial version of SCATCI that has been modified as part of this dCSE project is hereafter referred to as scatci_serial.
Hereafter, the current development version of SCATCI will be referred to as scatci_slepc. The

following sections describe in detail the modifications that have been made to integrate MPI and PETCs/SLEPc into UKRMol-in and specifically SCATCI.

Significant effort has been spent on making this parallel version of SCATCI backwards compatible with scatci_serial. To this end a 'slepc switch' has been implemented, where if `slepc_switch == .true.`, a parallel build of the Hamiltonian matrix is invoked, along with subsequent calls to the SLEPc Library during the diagonalization stage. Correspondingly, if `slepc_switch == .false.`, scatci_slepc will "emulate" scatci_serial. This was achived through a combination of **if** statements and having all calls to MPI and SLEPc/PETSc be made via dummy subroutine calls which can be switched for serial versions at compile time via Make.

For example, rather than calling **SlepcInitialize()** directly we use a call to a dummy subroutine **Slepc_Initialize()** which looks like this when compiled for use in scatci_serial:

```
subroutine Slepc_initialize(slepc_switch,srank,ssize,scomm)
   implicit none
   logical, intent(out)    :: slepc_switch
   integer, intent(out)    :: srank,ssize,scomm

   slepc_switch = .false.
   srank = 0
   ssize = 1
   scomm = 0
   print*, 'SCATCI run based on serial workflows'

 end subroutine Slepc_initialize
```

and like this when compiled for use in scatci_slepc:

```
 subroutine Slepc_initialize(slepc_switch,srank,ssize,scomm)
   implicit none
   logical, intent(out)    :: slepc_switch
   integer, intent(out)    :: srank,ssize,scomm
   integer, parameter      :: master=0
   integer (kind=shortint) :: ierr,srank_32,ssize_32

   slepc_switch = .true.
   call SlepcInitialize(PETSC_NULL_CHARACTER,ierr)
   call MPI_COMM_RANK(PETSC_COMM_WORLD,srank_32,ierr)
   call MPI_COMM_SIZE(PETSC_COMM_WORLD,ssize_32,ierr)
   scomm = int(PETSC_COMM_WORLD)
   srank = int(srank_32)
   ssize = int(ssize_32)
   if (srank == master) then
      print*, 'SLEPc has been initialized'
   end if


 end subroutine Slepc_initialize
```

Rather than writing the Hamiltonian to file, as is the case in the serial version of SCATCI, in this MPI-based version of SCATCI the Hamiltonian is built in parallel on separate compute cores with each core subsequently inserting its local sparse Hamiltonian matrix elements directly into a PETSc matrix (Mat) object which is subsequently passed to EPS as the operator that

defines the eigenvalue problem. Various options are then set for a customized solution, the problem is then solved and finally the the solution is retrieved.

3.2 Parallelisation of the Hamiltonian Construction.

Since the (N+1)-electron Hamiltonian matrix (hereafter referred to simply as the Hamiltonian matrix) is real and symmetric the construction phase sees the construction of only the lower-triangular part of the matrix (along with the main diagonal) which is then written to disk in the serial version. It sweeps through several deeply nested loops associated with each of these blocks and writes the column and row indices along with the matrix values to a fortran binary sequential access file (lower triangular part only). The matrix is stored on disk in what amounts to a coordinate (COO) sparse storage format. The storage format is not a genuinely COO format in that the matrix is unordered and only the lower part is stored.

The Hamiltonian is made up of three separate blocks that are hereafer named the Continuum-Continuum (CC) block, the Bound-Continuum (BC) block and the Bound-Bound (BB) block.

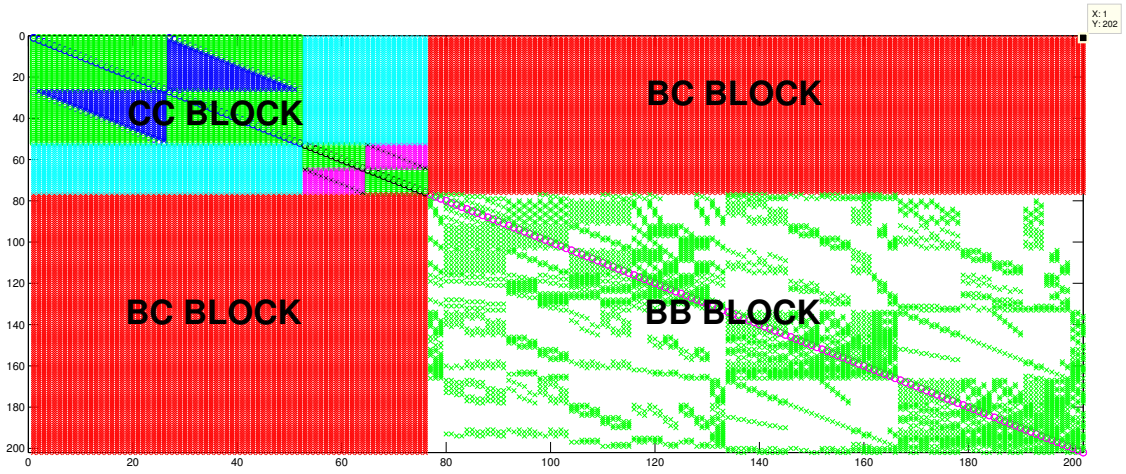

Figure 2: The structure of the Hamiltonian associated with a small test-case H_2O calculation. Note the internal "class" structure of the CC block which is always dense. The BC block is also always dense. Note also that the BB block is the only sparse part of the matrix and that it is unstructured and changes from molecule to molecule.

Figure 2 shows the structure of the Hamiltonian associated with a water molecule and it can be seen that the only genuinely sparse part of the matrix is found within the BB block. The sparsity of the BB block is unstructured and changes from molecule to molecule. The CC and BB blocks are always dense blocks. It should be noted straight away that the H_2O test case represents a very small Hamiltonian and for the case of larger molecular systems, such as phosphate and guanine, the dimension of the BB block is several orders of magnitude larger than the CC block (e.g in phosphate CC order $\sim$ 700 and BB order $\sim$ 120,000), so that for molecules of future interest, the BB block dominates the Hamiltonian matrix. A schematic of such a Hamiltonian (a phosphate test-case that has made the BC block smaller than preferred) can be seen in figure 3

Rather than write the sparse matrix elements (i.e. the rows, columns and values) to disk we insert said elements into the PETSc Mat object (PETSc Matrix) as the matrix is being constructed. Unfortunately, this is not entirely straight forward due to the fact that PETSc only allows for a ordered row-wise insertion of the upper-triangular part of a sparse symmetric matrix. The fact that in SCATCI the Hamiltonian is generated as a lower-triangular matrix means that, for both the CC block and the BC blocks, elements first need to be inserted into arrays so that the blocks can be sorted before they are inserted into the PETSc Mat object.

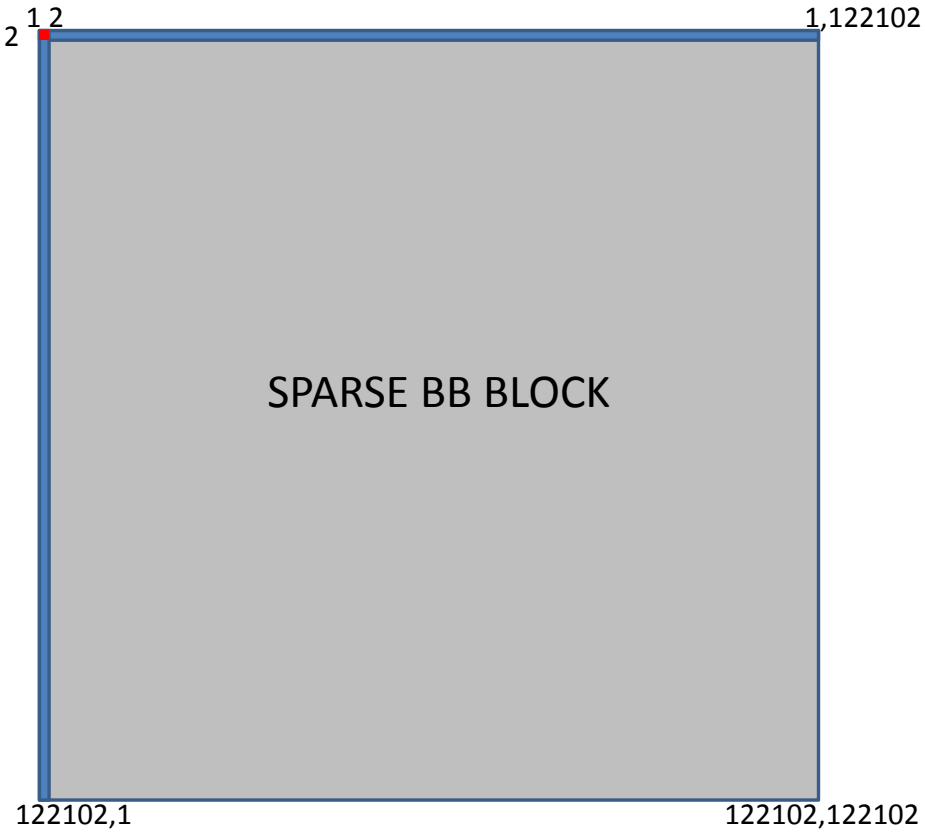

Figure 3: Schematic of the Hamiltonian associated with a test-case phosphate calculation. Typically, the CC (red) and BC blocks (blue) would be larger, but still much smaller than the BB block.

A description of how each of the blocks is constructed and inserted into the PETSc matrix now follows.

3.2.1 The CC Block

During the PETSc assembly stage the CC elements may be passed about so as to assign the CC elements to the correct MPI process for the diagonalization stage. This message passing is carried out implicitly by PETSc. Because the CC block is small and also because the master process usually ends up being assigned the whole CC block for the diagonalization, this is not a communication intensive stage.

From an algorithmic viewpoint the construction of the CC block is the most complicated phase, but, due to its small dimension, is the least computationally demanding. The reason that the construction of the CC block is complicated is due to the fact that it is built up in terms of so-called "classes", with what essentially amounts to a separate algorithm for each class. This makes the task of parallelizing the construction of the CC block non-trivial. However, because the serial compute time for the construction of the CC block is minimal, when compared to the construction of the BB block, the approach taken here has been for only a single MPI process to insert the CC elements into the PETSc Mat object. While only one process inserts the CC elements into the Mat object each of the processes constructs the CC block in its entirety. The reason for this is that there are arrays filled during the CC block construction that are needed by all processes involved in the construction of the BC block, i.e., each MPI process involved in the contruction of the BC block needs to have these arrays. In summary, the construction of

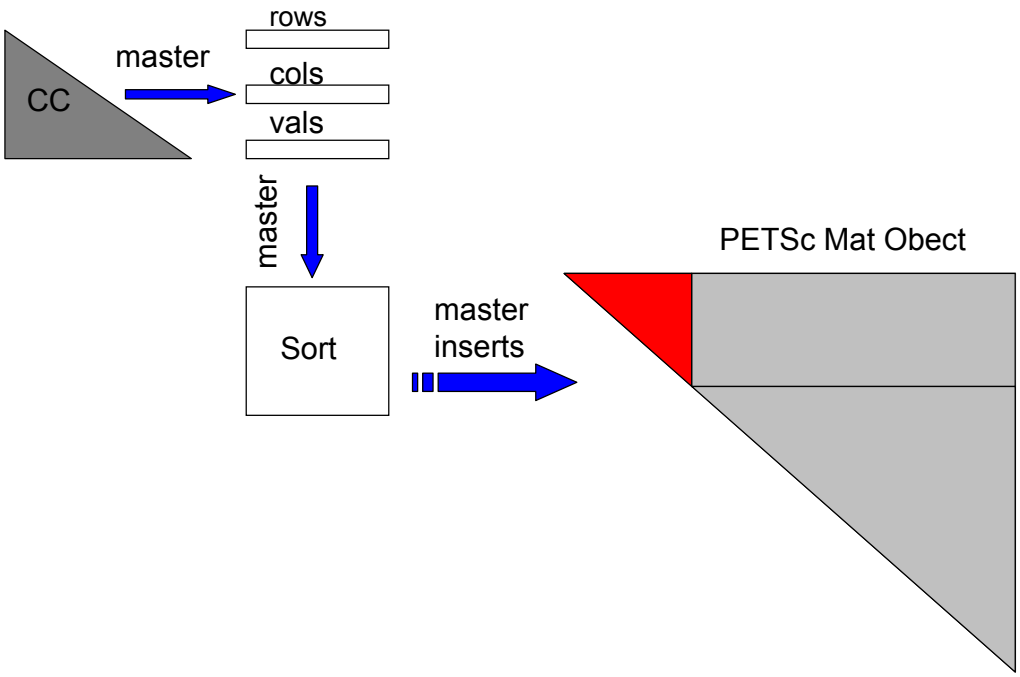

Figure 4: A schematic of how the CC block is inserted into the PETSc Mat object. Note that only the master process inserts the CC block elements into the PETSc Mat object. Note also that PETSc only accepts the upper-triangular part of the symmetric matrix and so row and column indices are interchanged upon insertion.

the CC block is not partitioned among MPI processes, but is rather constructed (in full) on all processes with only a single process subsequently invoking the sorting of the CC arrays along with insertion of sorted elements into the PETSc Mat object. Because PETSc accepts only the upper-triangular part of a symmetric matrix, the indices associated with column number in scatci_serial now point to row number and the indices associated with row number in scatci_serial now point to column number. This amounts to a simple transpose of the sparse matrix. A schematic of how the CC elements are inserted into the PETSc Mat object can be seen in figure 4

3.2.2 The BB Block

A communicator is set up with the name `bb_comm`. Within this communicator each of the MPI processes constructs each of the rows that it has been assigned. Since each process constructs the rows that it will subsequently own during the diagonalization phase, little communication is required during the PETSc assembly stage. Note also that due to the underlying algorithm implemented during the construction of the BB block, the BB elements constructed in an ordered fashion and so no sorting is required. Process numbers in figure 5 are process IDs within `bb_comm`

Although the BB block is the largest block in the matrix and is also the genuinely sparse

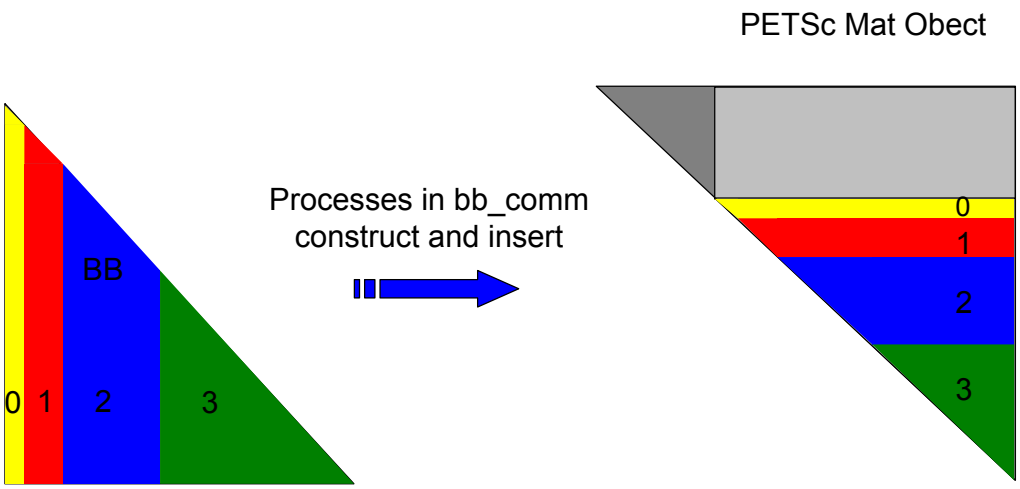

Figure 5: A schematic of how the BB block is inserted into the PETSc Mat object.

(and unstructured) part of the matrix, from an algorithmic viewpoint its construction is quite straightforward and relies on conditional statements that are associated with symmetry principles occuring within nested loops. Fortunately, the BB block is constructed in an ordered fashion in that it is built column by column. Again, because PETSc accepts only the upper-triangular part of a symmetric matrix, the indices associated with column number in scatci_serial now point to row number and the indices associated with row number in scatci_serial now point to column number. In order to parallelize the construction of the BB block, the outer-most do loop of all the do loops associated with the BB block construction has been modified so that each MPI process runs over its local range of the outer do loop.

While the structure of the CC and BC blocks are known before the matrix is constructed, the unstructured nature of the BB block is not known and so there is little possibility of load-balancing the construction of the matrix. Likewise, there is little possibility of load-balancing the diagonalization stage of scatci_slepc unless information on the structure of the BB block is known before inserting the Hamiltonian matrix elements into the PETSc Mat object. A crude approach taken within this version of scatci_slepc has been to first sweep through the nested loops associated with construction of the BB block. During this initial sweep, the BB block is not constructed, but rather an array is filled that holds the number of non-zeros associated with each row of the BB block. This initial sweep is parallelized so that each process is assigned an equal number of rows to sweep through and is implemented by all MPI processes in PETSC_COMM_WORLD (MPI_COMM_WORLD). Upon each process filling its array containing the number of non-zero elements within each of the rows assigned to it, each of these arrays is gathered onto the master process. Once this is done, a load-balancing mechanism is invoked that returns the number of rows that should be assigned to each MPI process so as to ensure that each MPI process holds approximately the same number of non-zero values for the diagonalization stage. Then a second sweep through the loops associated with the construction of the BB block occurs, this time with each process constructing its assigned number of rows and subsequently inserting these elements directly into the PETSc Mat object. Only processes within a newly created communicator called `bb_comm` invoke this second sweep. Fortunately, the BB elements do not need to be sorted prior to being inserted into the PETSc Mat object. A schematic of how the BB elements are inserted can be seen in figure 5.

3.2.3 The BC Block

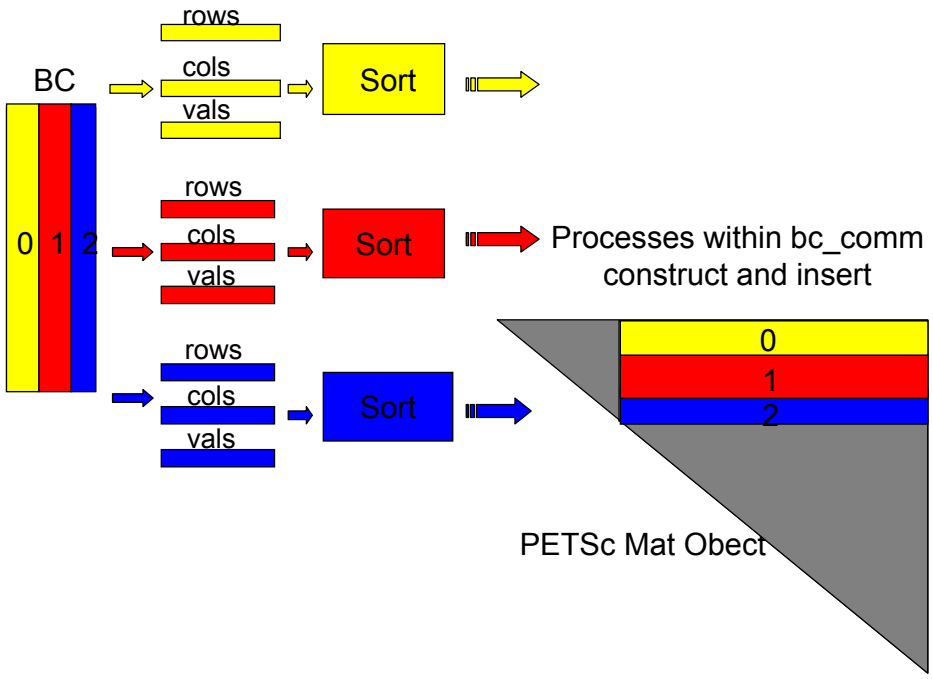

Figure 6: A schematic of how the BC block is inserted into the PETSc Mat object.

A communicator is set up with the name `bc_comm` and within this communicator each of the MPI processes inserts each of the rows that it has been assigned into arrays to be sorted. Since each process sorts the rows that it will subsequently own during the diagonalization phase, little communication is required during the PETSc assembly stage. Process numbers in figure 6 are process IDs within `bc_comm`. Note that although the schematic might imply parallelism during the construction of the BC block each process in fact runs through the entire nested loops involved during construction, i.e, there is no partitioning of work during the construction of the BC block. Rather, during the construction, each process only inserts the rows that it has been assigned into arrays to be sorted.

While the BB block is difficult to handle, the BC block has caused more issues during this project. Initially, the BC block was parallelized in a straight-forward manner. The BC block is built row-by-row but the columns within each row are unordered. The construction of the BC block was initially parallelized by partitioning equal blocks of rows across all MPI processes. This was implemented by a straight-forward parallelization over the outer-most loop of the all the loops associated with the BC block construction. As each process constructs its BC rows, it subsequently inserts the sparse matrix elements into "BC arrays" to be sorted. Upon sorting the arrays, the elements are inserted into the PETSc matrix. The problem with this mechanism is that the parallelization over the outer-most loop lends itself to the insertion of the elements into a lower-triangular matrix. However, as noted above, PETSc only accepts an upper-triangular matrix. While the matrix elements can be easily transposed and subsequently inserted into the PETSc Mat object, the subsequent assembly stage of the PETSc Mat object is extremely costly as too much data needs to be communicated between processes. In essence, if the construction

of the BC block is parallelized over the outer-most loop, each process is not constructing the elements that it will later own during the diagonalization stage. It is important that each process should construct (as much as possible) the elements that it will own during the diagonalization stage so as to cut down on communication during the PETSc assembly stage. Re-engineering the construction of the BC block so that this can be the case for a parallelized build appears to be a non-trivial task.

The current approach taken is for each MPI process within `bc_comm` to sweep through the loops associated with the construction of the BC block in full, but for each process to insert only the rows that have been assigned to it. This clearly amounts to a serial construction of the BC block, but cuts down significantly on the amount of communication during the PETSc matrix assembly stage due to the fact that each process is inserting only the elements that it will later own during the diagonalization stage.

3.2.4 The PETSc assembly stage

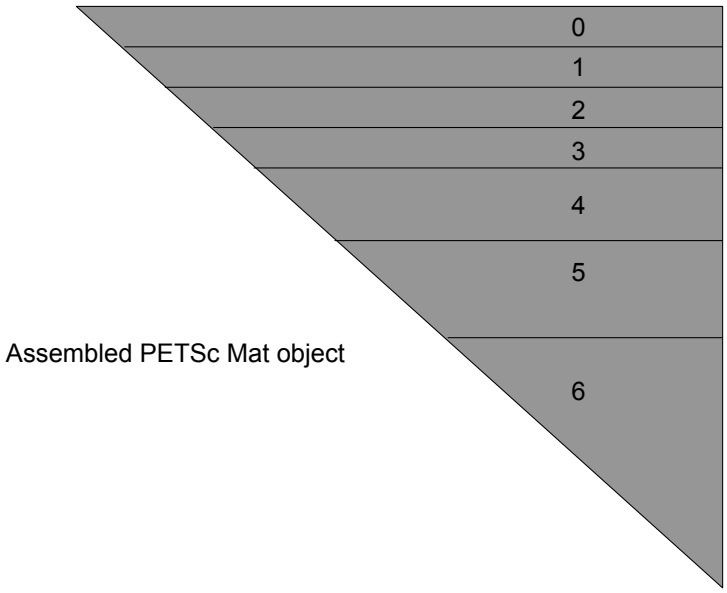

Figure 7: A schematic of the assembled PETSc matrix. Note that, for this example, the CC block has now been partitioned among processes 0, 1 and 2 automatically during the assembly process. Processes are numbered according to IDs in PETSC_COMM_WORLD.

Once all the elements have been inserted into the PETSc mat object as described above, PETSc assembly routines are called (`MatAssemblyBegin()` and `MatAssemblyEnd()`). Little communication between MPI processes should occur during this phase as most of the elements have been inserted by the processes that subsequently own them during the diagonalization phase. It can be seen from figure 6 that, for the example given above, the CC block elements have been passed between some processes, as only the master process has inserted the CC elements and processes 1 and 2 have been assigned elements within the CC block for the diagonalization phase.

3.2.5 The SLEPc solution stage

With the Hamiltonian stored as a PETSc mat object performing the diagonalization with SLEPc is a simple matter of performing calls to various SLEPc routines:

```
!Create eigensolver context
    call EPSCreate(scomm,eps,ierr)
!Set operators. In this case, it is a standard eigenvalue problem
    call EPSSetOperators(eps,ham,PETSC_NULL_OBJECT,ierr)
    call EPSSetProblemType(eps,EPS_HEP,ierr)
    call EPSSetTolerances(eps,tol,PETSC_DECIDE,ierr)
    call EPSSetDimensions(eps,nstat_32,PETSC_DECIDE,PETSC_DECIDE,ierr)
!Solve the eigensystem
    call EPSSolve(eps,ierr)
    call EPSGetIterationNumber(eps,its,ierr)

!Get some information from the solver
    call EPSGetType(eps,tname,ierr)
    call EPSGetDimensions(eps,nev,sncv,smpd,ierr)
    call EPSGetTolerances(eps,tol,maxit,ierr)

!Get the number of converged eigenpairs
    call EPSGetConverged(eps,nconv,ierr)


    do i = 0_shortint,nconv-1_shortint
!Get the converged eigenpairs: i-th eigenvalue is stored in kr
       call EPSGetEigenPair(eps,i,kr,ki,Vr,PETSC_NULL_OBJECT,ierr)
       call VecScatterCreateToZero(Vr,vscat,vout,ierr)
       call VecScatterBegin(vscat,Vr,vout,INSERT_VALUES,SCATTER_FORWARD,ierr)
       call VecScatterEnd(vscat,Vr,vout,INSERT_VALUES,SCATTER_FORWARD,ierr)
       call VecScatterDestroy(vscat,ierr)
       if (srank == master) then
          print*, kr + dtnuc(1)
          if (i == 0_shortint) allocate(hmt(nocsf*nconv),eig(nocsf),dgem(nocsf))
          eig(i+1) = kr
          call VecGetArrayF90(vout,xx,ierr)
          do j = 1, nkeep
             hmt(j+(i*nkeep)) = xx(j)
          end do
          call VecRestoreArrayF90(vout,xx,ierr)
       end if
    end do
```

The results are in excellent agreement with serial code - one of the requirements of the code maintainers was that they have one version of the code to look after. This meant that any modifications to the code had to be done in a way that the same code could still be used on a desktop machine with no access to PETSc/SLEPc and that the output was written to disk as fortran binary files since SCATCI is just one step in the suite of UKRMol and the output would need to be used in UKRMol-out. The serial code writes unformatted/binary sequential fortran files whilst PETSc is c based and so uses a different approach to binary which is not directly readable from Fortran. Unfortunately this approach does have a negative impact on the parallel performance of the code but there are plans in place to feed the output of SCATCI directly into PFARM in UKRMol-out which should bypass this issue.

Tests have been carried out for water and Phosphate (the DNA backbone is sugar molecules linked by phosphate groups). Phosphate has a Hamiltonian matrix order = 122102, with 1.610^8 non-zero elements. Figure 8 shows the performance of the diagonalization stage compared to the serial and OpenMP versions of SCATCI.

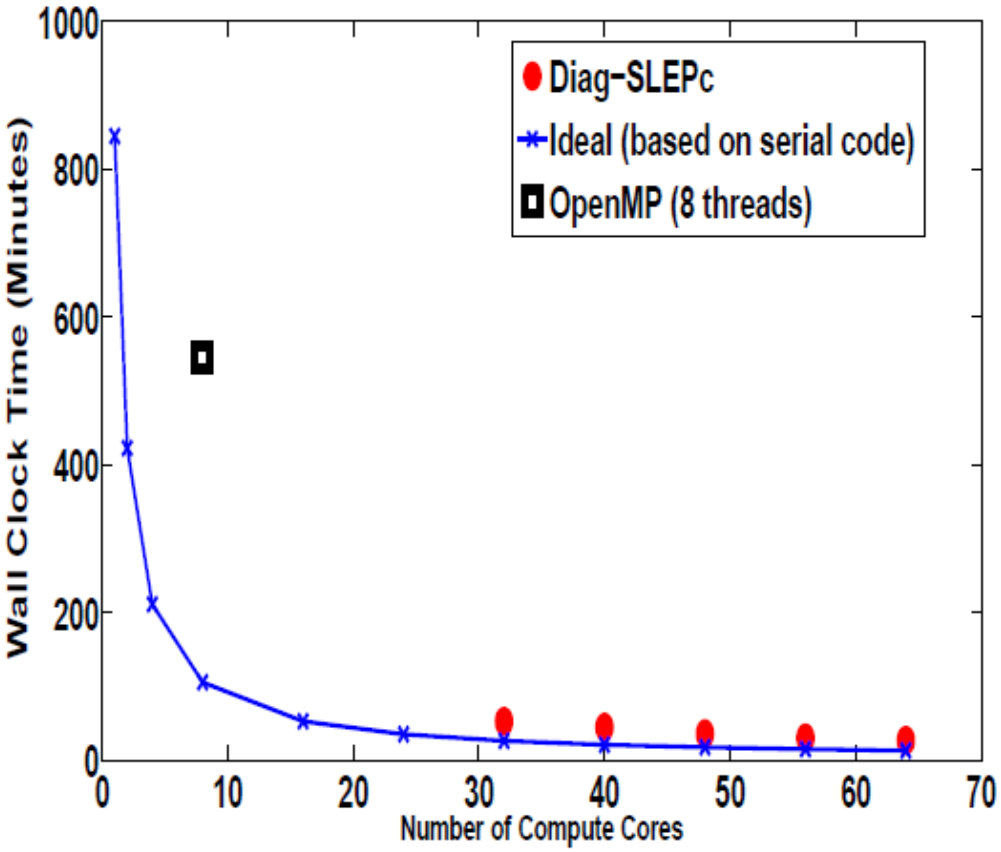

Figure 8: A comparison of the performance of the diagonalization stage

Finally to be able to solve the largest target jobs the UKRMol suite is required to us 64 bit integers for array indexing. This is problematic since we only have 32 bit PETSc/SLEPc on HECToR. There is a 64 bit MPI available with PGI but that is then incompatible with PETSC_COMM_WORLD. Fortunately the parallel partitioning of the data meant that it was possible to cast all integers to 32 bit when interacting with PETSc/SLEPc/MPI.

4 Conclusion

The key aims of this project were: to complete the parallelisation SCATCI and to port it to HECToR. This has been achieved and a scalable parallel version of UKRMol-in has been developed for use on HECToR and other large HPC resources. The Hamiltonian build and diagonalisation steps were implemented by calls to SLEPc and PETSc. Further improvements to the initial load balancing and performance of the parallel UKRMol-in code were made such that scalability to several nodes on HECToR is now achievable and Zdenek Masin, a PhD student at the Open University has performed runs using 640 processes on HECToR. Much larger problem sizes may now be investigated with UKRMol-in. This work has been made available to HECToR users of UKRMol-in and has been submitted to CCPForge.

5 Acknowledgments

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

References

[1] J. Tennyson, *Phys. Rep.*,**491**, 29-76 (2010)

[2] Boudaiffa B, Cloutier P, Hunting D, Huels M A and Sanche L, 2000 *Science* **287** 5458

[3] Shukla MK et al and Leszczynski J Eds 2008 *Radiation Induced Molecular Phenomena in Nucleic Acids* Springer

[4] A Dora, J Tennyson, L Bryjko, and T van Mourik 2009 *J. Chem. Phys.* **130** 164307

[5] Bouchiha D, Gorfinkiel JD, Caron LG and Sanche L 2006 *J Phys B* **39** 975

[6] Gorfinkiel JD and Tennyson J 2004 *J Phys B* **37** L343; 2005 *J Phys B* **38** 1607

[7] Gorfinkiel J D, Morgan L A and Tennyson J, *J. Phys. B* **35**, (2002) 543.

[8] `http://ccpforge.cse.rl.ac.uk/gf/project/ukrmol-in/`

[9] `http://www.physics.dcu.ie/ damot/`

[10] `http://www.hector.ac.uk/cse/distributedcse/reports/cabaret/`