MicroMagnetic modelling of naturally occurring magnetic mineral systems

Dr. Chris M. Maynard
EPCC, The University of Edinburgh,
James Clark Maxwell Building,
Kings Buildings, Mayfield Road,
Edinburgh, EH9 3JZ

June 8, 2011

Abstract

This report presents the results of a HECToR dSCE project to a port a serial code modelling the magnetic properties of mineral systems to HECToR. The original project plan was to implement a semi-parallel code, using the Sundials CVODE library to perform the ODE solve in parallel, using a gather/scatter to transform between the serial parts of the code. This turned out to be impractical, and the PETSc library was adopted to develop a parallel version of the code instead. A bug was discovered in the PETSc - Sundials interface which meant a working parallel code could not be delivered. However, significant progress had already been made in developing the parallel code. Most of the remaining tasks required for a parallel code were supported by an Edikt project.

Contents

1 Introduction

There are many problems in the geosciences that rely on the ability to accurately determine the magnetic properties of minerals and the stability of the palaeomagnetic recordings that they contain. For example, the investigation of the behaviour of the geomagnetic field depends largely on the observed field variations on or above the Earths surface. Temporal records of direct observations stretch back less than two hundred years, and so a more detailed analysis is dependent of ancient recordings made in rocks as they are formed. Geological interpretation of the directional recordings of the ancient field led to the discovery, over 50 years ago, that the Earths oceanic and continental plates are continuously moving. Today, a more thorough understanding of magnetic mineralogy enables a detailed analysis of not only of fine scale continental block motions and rotations, but also emplacement temperatures and thermo-temporal history of the rocks to be determined. A further, but my no means only, example is that magnetic mineralogy is frequently used as a proxy for determining palaeoclimate variations.

All these applications depend on an understanding of how the magnetic properties of minerals change with the mineral microstructure, chemistry, grain geometry and inter-grain magnetic interactions. The complexity and diversity of naturally occurring magnetic minerals makes the numerical micromagnetic problem a much more difficult task than that applied to the generally more ideal man-made recording media. The increasing sophistication of environmental-magnetic investigations relies on a high-fidelity magnetic re-coding processes occurring in natural magnetic mineral systems. Yet our understanding of the fundamental processes that enable common magnetic minerals to record the local geomagnetic fields direction and intensity and to retain this information over geological time scales is far from complete. A much better understanding of the recording process in natural materials is required in order to assess the reliability of rock-magnetic recordings.

Equilibrium magnetic domain structures are determined by integration of the Landau-Lifshitz-Gilbert equation of motion. Finite element methods are the preferred technique when dealing with irregular geometries since magnetic domain structures are sensitive to the accuracy with which grain geometries can be modelled. The code, MicroMag, before the dCSE existed as a serial code (written in Fortran90). Some initial work had been undertaken in investigating methods of parallelisation. This was the basis for the work-plan submitted as part of the dCSE proposal.

2 Algorithmic Details

The numerical micromagnetic model represents the magnetisation within a structure by 3D Cartesian vectors (each of unit length) placed at the N vertices of a finite element (FE) mesh. Tetrahedral elements are employed to reduce the discretisation error. The size of the computational cell is governed by the micromagnetic quantity termed the exchange length, which restricts the maximum cell size.

Equilibrium magnetic domain structures are determined by a minimisation of the total free magnetic energy of the system, which is achieved by integration of the Landau-Lifshitz-Gilbert (LLG) equation of motion. Dynamical solutions require the LLG equation of motion to be solved for every time step. The total effective field contributions include a computationally intensive non-local (magnetostatic) field calculation, and also the local exchange, anisotropy and externally applied field calculations. Of the previous four component fields only the magnetostatic field is non-trivial. It is also the most computationally intensive part in the determination of the total effective field. A finite volume approach is employed to calculate the exchange and anisotropy fields and is essentially determined by multiplication of the stiffness matrix by the global magnetisation vector.

The magnetostatic field is solved via a scalar potential approach. This requires the solution of the Poisson equation for the divergence of the magnetisation and solution of the Laplace equation for the boundary vertices. An efficient boundary element method is employed to approximate the long-range interaction. Both equations are discretised using a Galerkin scheme for a linear finite element solution of the equations. The solution is obtained via a standard conjugate gradient solver. The storage required in the solution of the Poisson equation is extremely sparse hence efficient sparse matrix methods are essential. Also, the tetrahedral finite element basis functions provide an efficient way of calculating the remaining three component fields by simple sparse matrix-vector multiplication. Bespoke sparse matrix management is required due to the data layout which arises from the efficient construction of the initial finite element basis functions.

However, the matrix arising from the boundary element approximation to the long-range interaction is of size N^2, where N is the number of vertices on the material boundary. In a parallel code this dense matrix can be distributed efficiently amongst each individual processing unit. Integration of the stiff system of ordinary differential equations (ODEs) arising from the LLG equation of motion requires an implicit solver and thus requires the approximate inverse to a sparse $3N$ system of linear equations. The use of explicit methods would be ineffective due the restrictive size of the

time-step controlled by the stiffness of the system. The implicit method is a variable order solver which is based on a multi-step backward Euler method with variable coefficients.

3 The Work Plan

The original proposal submitted to the dCSE panel requested a combined effort from both NAG and EPCC staff for a full parallelisation of the code with 16 months of effort. The panel approved the proposal but awarded 6 months of effort in order to get a working parallel code. The work plan for the 6 month project is given below in three work package (WP)

WP1 Semi-Parallel Implementation Effort: 3 months

WP1.1 Serial code profiling for two benchmark test cases

- Single non interacting Sphere with 100000 finite element cells
- Non interacting Framboid with 100000 finite element cells

WP1.2 Implement calls to the parallel CVODE solver [1]

- By decomposing the finite elements with a regular ordering
- Study of code performance paying particular attention to load balancing Outputs

D1 WP1 Deliverables

D1.1 Profile of the serial code giving particular attention to detail regarding the performance of CVODE

D1.2 Short technical note to describe the general implementation of CVODE to MicroMag

D1.3 Profile of MicroMag using parallel CVODE

D1.4 Version of MicroMag which uses parallel CVODE

WP2 Fully-Parallel Implementation. Effort: 2 months

WP2.1 Implement calls to the Metis library [2] to give an automated decomposition for a Cubit tetrahedral mesh (as used by MicroMag). This will be done as a preprocessing stage

WP2.2 Implement a local to global mapping for the finite element numbering system within MicroMag which will enable the code to use the decomposition produced in WP2.1

WP2.3 Validate fully-parallel MicroMag with the test cases in WP1.1

D2 WP2 Deliverables

D2.1 Profile of the fully parallel MicroMag code giving particular attention regarding the performance of CVODE

D2.2 Version of MicroMag which uses parallel CVODE and parallel effective field calculations

D2.3 MicroMag code demonstrating parallel scalability up to 256 HECToR processing cores

WP3 Final report and dissemination. Effort: 1 month

WP3.1 Write a technical report detailing work for the project as a whole

D3 WP3 Deliverables

D3.1 Technical report on the work performed during the project

D3.2 A parallel version of the MicroMag code

4 Code Status Before Start of Project

The MicroMag code had been in existence for a number of years and in common with many other scientific codes had been developed by several different people, without a clear development plan resulting in a code that, whilst was rich with scientific functionality, was also difficult to maintain with some legacy features.

Prior to the main software project, core CSE effort had been employed to replace the deprecated DVODEpk ODE solver with a modern library, CVODE. This itself resulted in a modest speed up of the code in serial. However, the CVODE library can also be called in parallel, and this gave an initial starting point for the current dCSE project by looking at how MicroMag could be run in parallel. Prior to the start of the main dCSE project NAG CSE re-factored the original, serial code. This resulted in a code that was much simpler, without obsolete functionality or historical features, but also without the use of the CVODE library.

5 Semi-Parallel Implementation

Starting from the new, refactored code, the serial implementation of CVODE had to be redone. Whilst this wasn't a huge task and the original work could be used as a guide, it still consumed some effort. However, this initial effort resulted in a marked improvement in performance. For a sphere, with 20K vertices and 100K finite elements running in serial on the XT4, took DVODE 3098.86 seconds for a single external field step, whereas the CVODE version took 1890.35 seconds. This represents an impressive speed up factor of 1.64.

For the semi-parallel version, computationally the problem can be represented in two parts, the implicit ODE solver, and the so-called Right hand side (RHS) function which is required to determine the time-step update of the implicit ODE solver. The RHS function determines the energies described in Section 2 and it is the most computational intensive part of the code. The semi-parallel strategy would be to compute the RHS in serial, then scatter the values of the energies computed in the RHS across the parallel elements. The call to the CVODE solver would be done in parallel, and then the results of the time-step gather back together, to compute the next RHS in serial again.

This gather/scatter serial - parallel - serial computation was decided upon as a staged development strategy, the first stage being to implement the parallel CVODE solver call, with the RHS in serial. The parallelisation of the the RHS for a fully parallel code would be implemented in a later package.

When attempting to implement this in the refactored, simplified code, it became clear that this approach simply wouldn't work, due to the way the user supplied RHS function was called by the CVODE library for each time step. When using the CVODE library, the user calls a set up routine in parallel, which sets up the sizes of the arrays to be used by the CVODE library, as shown in the code fragment below.

```
NEQ=3*NMAX
! calculate the local NEQ and the displacements
NEQ_local=NEQ/nprocs
call fnvinitp(comm, 1, NEQ_local, NEQ, IER)
call fcvmalloc(T, Y, METH, ITMETH, IATOL, RTOL, ATOL, $
IOUT, ROUT, IPAR, RPAR, IER)
```

where `NMAX` is the number of finite elements of the mesh and hence this determines the size of the arrays representing the matrices. `NEQ_local` is then the size of the array local to each processor.

In the main program, the call to the CVODE ODE solver is made in parallel. The user supplied RHS function, which is called by the CVODE solver, is also called in parallel. So, all the data structures have local, not global scope.

```
CALL FCVODE(TOUT, T, Y, ITASK, IER)
```

The call to the CVODE solver is shown in the code fragment above, passing the array which holds the total derivatives of the global magnetic energy,

Y. The user supplied RHS function named FCVFUN is shown in the code fragment below,

```
SUBROUTINE FCVFUN (T, Y, YDOT, IPAR, RPAR, IER)
use mpi_global
IMPLICIT NONE
REAL (KIND=DP) :: T,Y(3*NMAX), YDOT(3*NMAX)
```

The array Y is passed to the subroutine FCVFUN by the CVODE library, and is local in scope, so however the array is setup by the fnvinit routine and populated by CVODE. Whilst the local scope array can be declared any size the programmer wishes, the data is spread out across the processors, so there is no array with a complete global copy of the data. In the serial version, it didnt matter because the initial setup was serial, and so the whole array is populated.

The original idea was to compute the RHS in serial on a single (master) process, and then scatter the data to all processors. The array which holds the data in the RHS function, being local in scope only holds `NEQ_local` and not the whole data, so computation cannot be done in serial because the data is already distributed. The serial computation could not be implemented without first have to gather all the data first. Whilst it would be possible to override the CVODE parallel implementation and implement this by hand, and even get it to work correctly, the high unforeseen communication overhead would have seriously undermined any performance gain from working in parallel. The ultimate goal was a fully parallel code, so after consulting with the PI and NAG CSE it was decided to abandon the semi-parallel strategy and move to develop a fully parallel code, where the RHS is implemented in parallel from the beginning.

6 Parallel Implementation with PETSc

After further discussions with NAG CSE, it was suggested that using the PETSc library [3] would have significant benefits both for performance and for future development. After reviewing the functionality contained in PETSc, it was agreed to use the library.

PETSc contains a distributed parallel layer, with support for data types such as vectors and sparse matrices, as required by MicroMag and methods for manipulating them. Critically, the SUNDIALS CVODE library [1] can be called from PETSc. There is significant support for Krylov subspace solvers which are required to determine the RHS. Moreover the Finite Element partitioner Metis, and its parallel variant Parmetis can also be called

from PETSc. Future development work could allow the partitioning of the unstructured grid to be done at run time rather than at a preprocessing phase as originally planned in WP 2.1-2.2.

At this stage, approximately half the development time had been used to perform three tasks.

1. Re-implement the calls to CVODE in serial in the refactored code.
2. Attempt to implement the semi-parallel strategy and reveal its flaws, leading to its abandonment.
3. Evaluate the PETSc library functionality and decide on new strategy.

With less than three months left of development effort to re-write the code to make use of the PETSc data parallel data structures, and calls to the various solvers, there was an uncontrolled risk that there wouldn't be sufficient time. The risk was uncontrolled in the sense there was no time left for any contingency if there were any unforeseen problems with the development. However, it was felt there were significant benefits for future development, including performance, portability and additional functionality, such as using Parmetis, that it was decided to use PETSc despite the small amount of time remaining.

6.1 Parallel Matrix/Vector Assembly

To make efficient use of the PETSc library, the correct use of the PETSc data parallel structures for Matrices and Vectors is crucial for performance. Most of the work necessary to use PETSc was contained in constructing and populating these data structures. The MicroMag code reads in the metis mesh decomposition, and the uses this, along with information about the mineral grain, which is also an input, to create a series of sparse matrices which are then used to update the magnetic field, which in this context is a vector. The MicroMag code constructs the values and stores them in a matrix storage format called SLAP column format.

This is similar to the more generic sparse format compressed sparse column (CSC) format, but order of the stored values changed so that the diagonal element is stored first for each column. The default parallel decomposition for PETSc is decompose the matrix by *row*, rather than *column*, across the number of processing elements. This potentially made the task rather more complicated. However, with two notable exceptions, all the matrices in MicroMag are symmetric. So, the column major order of the micromagnetic data structures can be easily translated into a row major order, and then decomposed. In some sense this is both natural and helpful

translation as MicroMag is a Fortran90 code and multi-dimension arrays are naturally column major order *i.e.* left most index is fastest moving, and PETSc written in C which has the opposite convention. The exceptions to the symmetric matrices can be dealt with as special cases.

The PETSc library has specific data structures for sparse matrices. The CSR format is referred to as AIJ. To use this format in parallel, a method, named `MatMPIAIJSetPreallocation`, is called to create and allocate the data structure. PETSc's default parallel decomposition for matrices is by row. The allocation method makes the distinction between the diagonal block of matrix elements (where the column values are the same as the rows "owned" by the MPI task, and the non-diagonal block of elements (where the column values are different from the rows "owned" by the MPI task). These are passed as arguments of the method as `d_nz` the diagonal number of non-zeroes and `o_nz`, the off-diagonal number of non-zeroes. In particular the documentation states: *By setting these parameters accurately, performance can be increased by more than a factor of 50*. Determining and correctly setting these parameters is key to getting good performance from the PETSc library.

For example in the new MicroMag code which uses PETSc, we now use the SLAP column format to determine the number of non-zeroes which are in the diagonal block.

```
d_nnz=0
o_nnz=0
count=1
do i=(nmax_off),(nmax_off+nmax_l-1)
nze_l=nze_l+(CNR(i+1)-CNR(i))
do j=CNR(i),CNR(i+1)-1
if( (RNR(j).ge.nmax_off) .and. $
$ ( RNR(j).lt.(nmax_off+nmax_l)) )then
! we are in the diagonal block
d_nnz(count)=d_nnz(count)+1
nze_d=nze_d+1
else
o_nnz(count)=o_nnz(count)+1
end if
end do
```

In the code fragment above, each processor determines the whole matrix in SLAP column format, but this involves replicated data. So each process loops over only the number of elements that it owns. For each column of the matrix it owns, it then loops over the column start `(CNR(i))`to end

(CNR(i+1))and counts the number of non-zero elements (nze_l),and how many are diagonal (d_nnz)and off-diagonal (o_nnz).The PETSc matrix data structures are then allocated as shown in the following code fragment.

```
CALL MatCreate(comm,PA,mpierr)
CALL MatSetSizes(PA,nmax_l,nmax_l,NMAX,NMAX,mpierr)
CALL MatSetType(PA,MATMPIAIJ,mpierr)
CALL MatMPIAIJSetPreallocation(PA,0,d_nnz,0,o_nnz,mpierr)
```

and the local PETSc matrix populated from the replicated SLAP column format matrix as shown in the code fragment below.

```
call MatGetOwnershipRange(PA,istart,iend,mpierr)
do i=istart,iend-1
do j=CNR(i+1),CNR(i+2)-1
call MatSetValue(PA,i,(RNR(j)-1),YA(j), $
INSERT_VALUES,mpierr)
end do
end do
write(messageText, $
("Matrix PA B/C/D allocated and assigned"))
CALL writeMessage(nonzerostiff,messageText,0)
CALL MatAssemblyBegin(PA,MAT_FINAL_ASSEMBLY,mpierr)
CALL MatAssemblyend(PA,MAT_FINAL_ASSEMBLY,mpierr)
```

6.2 Parallel CVODE for Time Evolution

Once the data structures have been constructed in parallel, the computational methods could be tackled. There were two main tasks to enable this with the PETSc MicroMag code : the call to the CVODE solver and the RHS function which itself would call the Krylov subspace solver methods in PETSc such as CG. The PETSc library comes with documentation and an extensive set of examples demonstrating how various computational methods can be called. Whilst examining and running the SUNDIALs example it was discovered that there is a bug in the PETSc - SUNDIALs interface for anything other than a trivially small number of processors. The CVODE solver would run, but report that it had run for the maximum allowed number of steps without reaching the final time step. The example would run correctly in serial, and if the number of MPI tasks didn't exceed 6, and if the system size was not increased more that 10%. The maxumim allowed number of steps, and the time taken can also be altered.

Varying all the allowed parameters did not change this behaviour. For example, running the PETSc example 4 (`ts/examples/tutorials/ex4`) for

SUNDIALS time-stepper (ts) solver in version `3.1-p3`, on 128 nodes of HECToR, and increasing the system size to $m = 1024$, where the system size is given by m^2, resulted in the following error.

```
[CVODE ERROR]  CVODE
  At t = 0.182961, mxstep steps taken before reaching tout.
```

This was reported to the PETSc developers who confirmed that there was indeed a bug. They gave a commitment to fixing the bug, but without any timescale for doing so. This presented a significant problem to the successful completion of the project. It was decided that the best course of action was to replace the call to the SUNDIALS solver, with a simple EULER solver which is a native PETSc method. The code could then run correctly in parallel, if with limited scientific use-ability. The solver could be altered to call the SUNDIALS solver once the PETSc developers issued a patch or new release. A further problem was the amount of time consumed discovering and subsequently confirming the bug with the PETSc developers. With time limited to produce a working parallel code a successful conclusion to the project would be difficult.

6.3 Parallel RHS Vector

The RHS function still had to be implemented using PETSc code, and a number of computational tasks needed to be implemented, using the PETSc matrix-vector manipulation methods to calculate the various contributions to the total energy. The MicroMag code holds the magnetic field in two different data structures. The first is a two-dimensional array of size `m(NMAX,3)` where `NMAX` is the number of finite element vertices, and `3` represents the three physical dimensions in Cartesian space. The second data structure is a linearised one dimensional array, `Y(3*NMAX)`

The code switches between these two data structures as the Sundials solver expects a one dimensional array, but it is both more natural to code, to use the other data structure. In the main loop, the magnetic field is packed into the linearised array, and the solver called. The RHS function is called by the solver, and one of its arguments is `Y`. The RHS function then has to unpack this one-dimensional array back into the two dimensional array. The PETSc data structures called Vectors are not arrays in the `FORTRAN` sense, and can only be manipulated by calling the appropriate methods. However, PETSc also has other data structures and methods called *index sets*, and these methods act on PETSc vectors transform the data between the two views as required. These have also been implemented, in particular, a module was developed to transform vectors between the linearised- and three-vector data structures.

A code fragment showing the use of index sets to transform from the linearised vector, to the three vector is shown below.

```
SUBROUTINE xyzToComponents(linearVec,compVec,Nsize)
! converts a linearized vector of size 3*Nsize into an
! array of size 3, vectors of size N
! i.e. convert to a three-vector of x,y and z components
#include "finclude/petscsysdef.h"
#include "finclude/petscvecdef.h"
use petscVec
implicit none
Vec linearVec
Vec ,pointer :: compVec(:)
PetscInt Nsize
integer :: ierr, comm
IS xyz, comp
VecScatter scatter
comm=PETSC_COMM_WORLD
! x compVec first
CALL ISCreateStride(comm,Nsize,0,3,xyz,ierr)
CALL ISCreateStride(comm,Nsize,0,1,comp,ierr)
CALL VecScatterCreate(linearVec,xyz,compVec(1),comp, $
scatter,ierr)
CALL VecScatterBegin(scatter,linearVec,compVec(1), $
INSERT_VALUES,SCATTER_FORWARD,ierr)
CALL VecScatterEnd(scatter,linearVec,compVec(1), $
INSERT_VALUES,SCATTER_FORWARD,ierr)
CALL VecScatterDestroy(scatter,ierr)
CALL ISDestroy(xyz,ierr)
! dont need to destroy comp, as it is the same
! for x,y and z
```

The fragment shows the translation for the X component, and this code is similar for the Y and Z components. Another subroutine has been written to perform the inverse translation. The RHS function performs the following tasks:

- Convert local linearised array `Y` to 3-d array `m` .
- Calculate the total energy `htot` for each element by calling function `totalFieldVec`.
- Calculate the derivative of the total magnetic field `YDOT`, which depend on `htot` and `m`.

The CVODE solver then uses `YDOT` to update `Y`. These three tasks have been implemented.

The MicroMag code makes use of advanced `FORTRAN` 90 features, such as pointers, which are supported by the PETSc library, to create the three-vectors, as shown in the code fragment below.

```
Vec term1,term2
Vec , pointer :: htot(:), mxyz(:), ydot_xyz(:)
CALL VecCreateMPI(comm,nmax_l,NMAX,term1,mpierr)
ithree=3
CALL VecDuplicateVecsF90(term1,ithree,htot,mpierr)
```

The function `totalFieldVec` performs the following tasks to calculate the :

- total force `totfx` for the de-magnetisation by calling function `calcdemagVec`.
- anisotropic contribution to the energy, `hanis`.
- demagnetisation contribution to the energy, `hdemag`.
- exchange contribution to the energy, `hexch`.
- external contribution to the energy, `hext`.

The total energy `htot` is then determined from the sum of the components for `hanis`, `hdemag`, `hexch` and `hext`. These tasks have all been implemented.

6.4 General Matrix Vector Enhancements

The new MicroMag code makes use of the PETSc data manipulation methods, which provide methods for point-wise combinations, which combine the data on a point-by-point basis, for example, the code which will square all the values of a vector is shown below,

```
CALL VecPointwiseMult(asSqr(1),as(1),as(1),mpierr)
```

here both `as(:)` and `asSqr(:)` are three-vectors, 1 being the `X`component. Several variations of `axpy` methods exist for combining vectors and scalars, for example,

```
CALL VecAXPBYPCZ(as(3),Prot(1,3),Prot(2,3),Prot(3,3), $
mxyz(1),mxyz(2),mpierr)
```

where `as(3)`, `mxyz(1)` and `mxyz(2)` are individual components of three-vectors and `Prot(:,:)` is an array, thus individual components are scalars. The routine evaluates the following expression,

$$\vec{z} = \alpha \times \vec{x} + \beta \times \vec{y} + \gamma \times \vec{z} \tag{1}$$

where the order of the arguments of the routine is $\vec{z}$, α, β, γ, $\vec{x}$, $\vec{y}$.

6.5 Expensive Field Calculations within the Code

To compute demagnetisation and exchange energies, which are the most computationally expensive of the MicroMag code, the stiffness matrices are determined at the start of the program, and assigned to PETSc matrices. The Matrix-vector products are shown below.

```
! now compute Hdemag
! Hd_x(i)=totfx(j)*YAB(ij) and similar for y and z
! Petsc matrix is called PAB
! allocate Hdemag
CALL VecDuplicateVecsF90(totfx,ithree,hdemag,mpierr)
CALL MatMult(PAB,totfx,hdemag(1),mpierr)
CALL MatMult(PAC,totfx,hdemag(2),mpierr)
CALL MatMult(PAD,totfx,hdemag(3),mpierr)
! Compute hexch
!hexch_x=m_x*YA [PA in Petsc matrices]
CALL VecDuplicateVecsF90(totfx,ithree,hexch,mpierr)
CALL MatMult(PA,mxyz(1),hexch(1),mpierr)
CALL MatMult(PA,mxyz(2),hexch(2),mpierr)
CALL MatMult(PA,mxyz(3),hexch(3),mpierr)
```

The function `caldemagVec` is the sum of two scalar potentials. The first, ϕ_1, receives contributions to the force vector ($\mathbf{f}_1$) from the divergence of the magnetic field, $\nabla \cdot \mathbf{m}$, and a Neumann boundary condition. A Krylov subspace solver then calculates, ϕ_1, by solving the linear system

$$Y_A \phi_1 = \mathbf{f}_1 \tag{2}$$

where Y_A is the finite element stiffness matrix determined during the set up. The second potential, ϕ_2 is created by an asymmetric contribution to the force vector $\mathbf{f}_2$, and is determined by a second Krylov subspace solution of:

$$Y_{AR} \phi_2 = \mathbf{f}_2 \tag{3}$$

The solution of ϕ_2 depends upon ϕ_1, and the boundary conditions imposed here are to approximate an infinite volume. The resulting stiffness matrix Y_{AR} is then asymmetric. The total magnetostatic potential is the sum of the two components $\phi_1 + \phi_2$ and the resulting demag field is the gradient of this potential, which is found using the spatial components of the finite element basis functions.

Unfortunately, the dCSE project ran out of time before the `calcdemagVec` function could be implemented. Whilst this was ultimately disappointing, significant progress had been made in taking a legacy serial code and developing a fully parallel code which now has the potential for code performance

by using a modern data parallel layer library, PETSc. Several obstacles caused delays to this project which led to it failing to complete on time. The critical issues are listed below:

1. Starting from scratch with refactored code.
2. Flawed parallelisation strategy.
3. Bug in third party library.
4. Short term project lacked contingency time to compensate delays.

The first issue meant that extra work had to be done to before the planned work could be started. This took extra time, but turned out to be a necessary step. The flawed strategy was a failure of the planning stage of the project. However, it was only when attempting the semi-parallel decomposition with the simpler refactored code could the strategy be discovered as awed. Working with the refactored code also meant that using the PETSc library was possible. The bug in the PETSc-Sundials interface was unforeseen and took a significant amount of the remaining time to discover and report. The original proposal requested sixteen months, to parallelise this code. The various delays, bug hunting in libraries and changes to the work plan eventually meant that only two months were available to spend developing parallel code with PETSc.

At the end of dCSE project time, the code ran in parallel, but lacked some of the necessary scientific functionality. At this point, the MicroMag code status can be summarised as that after the pre-processing phase using METIS, the various stiffness matrices for the particular problem, the boundary conditions and the magnetic field are set up in serial. PETSc parallel data structures are created and populated from the serial ones. The time evolution loop is executed and the PETSc native `EULER` solver is called for each time-step. The `RHS` function calculates the energies, without the contribution to the demagnetisation from the total field.

To complete the project, two tasks would require implementation: The PETSc data structure representing the asymmetric stiffness matrix Y_{AR} would need to be populated from the serial MicroMag one, taking account of the *row-column* transformation. The function `calcdemagVec` implemented, with calls to the Krylov subspace solvers in the PETSc library. Once these two tasks have been completed the code would then be scientifically verifiable, if not especially efficient. On August 31 2010, the last day of the project, the PETSc developers issued a patch to fix the bug. This came too late to be included in the project, but the code could quite easily be changed

to use the Sundials solver, instead of the EULER solver. At this stage the code would be both scientifically correct, and algorithmically efficient.

Further development could also include, the PETSc data structures, which would be set up using the wrapper functions, so the same code runs in parallel and serial, and running in serial or parallel becomes a run-time option. Future development work would focus on the set up phase. Firstly, the PETSc data structures for the stiffness matrices would be populated in parallel from the outset, without determining them in serial before hand. Secondly, the ParMetis-PETSc interface would allow ParMetis to be called, thus the Finite element mesh composition to be determined at run-time, eliminating the pre-processing phase.

7 Edikt mini-project

The Edikt project [4] funded two months FTE of effort, in the first quarter of 2011. The PETSc developers issued a patch for the bug in the interface to SUNDIALs, which meant that the micromag code could now call the CVODE library. The remaining functionality required for a parallel code, outlined in section 6.5, was implemented. Testing and benchmarking of up to 16 nodes (384 cores) of the XE6, phase 2b machine revealed a subtle bug in the construction of one of the matrices, and some poor scaling performance. Further work is required to fix the bug, and investigate the cause of the poor performance. The most likely cause of poor performance when using PETSc matrix data objects is the construction and population of the matrices. The bug and poor performance may not be unrelated.

8 Conclusions

The original project was unable to deliver the working parallel code described in the work plan. Several issues resulted in the project suffering multiple delays, and changes to the work plan. The short-term nature of the project meant there was no time for contingency action to recover the project within the timescale and this was a very disappointing outcome. However, significant progress had been made to developing the code and with additional effort from the Edikt mini-project, a working parallel MicroMag code was achieved.

To summarise, the original code has been significantly refactored. This code was greatly simplified, with much obsolete functionality removed from the starting code. The serial version now uses the modern CVODE library from the SUNDIALs software suite. This has resulted in 64% speed up of the code in the test case of a 100K vertex sphere. The parallel MicroMag code

has had extensive modifications made from the serial version. This code now uses the PETSc library and has had its major data structures implemented using the PETSc data parallel layer. Significant progress has been made to developing the parallel code. By using modern coding techniques in Fortran 90, such as modules to abstract and re-use independent functional units, and by employing a modern high-performance scientific library the future development, portability and sustainabilty of the code base has been dramatically increased.

9 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] The SUNDIALs library https://computation.llnl.gov/casc/sundials/main.html

[2] The Metis library http://glaros.dtc.umn.edu/gkhome/views/metis

[3] The PETSc library http://www.mcs.anl.gov/petsc/petsc-as/

[4] The Edikt project http://www.edikt.org.uk/edikt2/