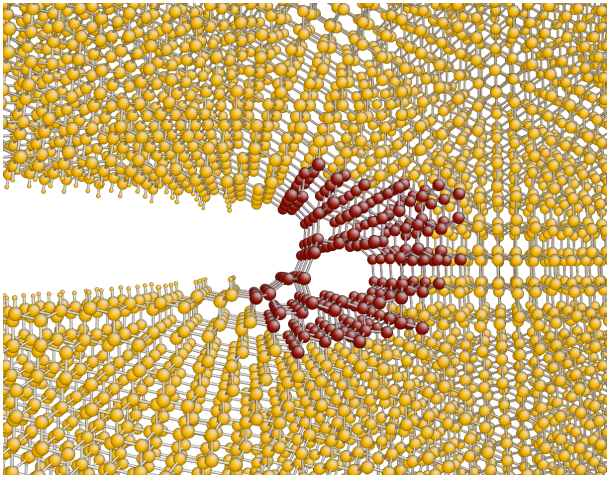
IOP | Institute of Physics
Computational Physics Group

Newsletter



Expressive Programming in Fortran 95+

Spring 2007

MEMBERS OF THE COMMITTEE

Roger Barrett	R.Barrett@surrey.ac.uk
Peter Borcherds (chairman)	p.h.borcherds@birmingham.ac.uk
Alan DuSautoy	alan.dusautoy@npl.co.uk
Hans Fangohr (newsletter)	h.fangohr@soton.ac.uk
Andrew Horsfield	a.horsfield@ucl.ac.uk
Geraint Lewis (vice-chairman,travel bursaries)	dg.lewis@physics.org
Matt Probert (honary secretary,treasurer)	mijp1@york.ac.uk
Michael Sleigh	Michael.Sleigh@awe.co.uk

Our web page can be found here:
http://www.iop.org/activity/groups/subject/comp

Comments about the newsletter and contributions for future editions can be sent to Hans Fangohr.

The picture on the cover of this edition shows the atomic structure at the tip of a crack in silicon is complex, with a nanovoid opening in front of the tip just below the critical loading that results in crack propagation. The red atoms are treated with quantum mechanics within the Learn-on-the-fly scheme, implemented using the libAtoms library. See the article on page 1 for more details.

Newsletter Contents

Expressive programming for computational physics in Fortran 95+

Dedicated to John W. Backus 1924-2007

Gábor Csányi[1], Steven Winfield, James Kermode, Mike Payne
TCM Group, Cavendish Laboratory, University of Cambridge,
J.J. Thomson Avenue, Cambridge, CB3 0HE, United Kingdom

Alessio Comisso
University of Trieste, Dipartimento di Materiali e di Risorse Naturali
Via A. Valerio 6, Trieste, TS Italy

Alessandro De Vita
Department of Physics, King's College London,
Strand, London, United Kingdom, and
DEMOCRITOS National Simulation Center and CENMAT-UTS, Trieste, Italy

Noam Bernstein
Center for Computational Materials Science,
Naval Research Laboratory, Washingon DC, USA

Introduction

Scientific computing has developed alongside Fortran, one of the very first high level computer languages. The Fortran language was designed to be portable and relatively simple for the programmer, yet still produce code that was as fast as lower level and much harder to use languages such as assembly. This combination was a great success, leading to Fortran's ubiquitous use in many computing applications. In the last two decades, other languages such as C and C++ have come to dominate new software development, especially in performance critical applications. Only in a few fields, including scientific computing, is Fortran still in widespread use for new software development today, fifty years after its invention. The persistence of Fortran is motivated by factors including reuse of legacy codes, familiarity through "on the job training" for computational scientists, and a mix of real and perceived speed advantages.

[1]Present address: Department of Engineering, University of Cambridge, Trumpington Street, Cambridge, CB2 1PZ, United Kingdom

However, this legacy has also deprived computational scientists of many modern advances in computer science and programming language development. The biggest such advance in the last two decades is object oriented (OO) design, a trend that is reflected in the widespread use of C++ for software development. The OO approach, where the programmer defines conceptual entities and then implements them in the form of an object that lumps together several associated variables and procedures that operate on the object, might seem to be naturally suited to scientific computing: the main tasks of the computational scientist are to define the essential entities (e.g. an atom), and to implement them in software by defining the variables that describe them (e.g. positions, momenta) and how they can be manipulated (e.g. compute trajectories by propagating positions and momenta forward in time). While the full power and concomitant complexity of OO programming languages may be unnecessary for scientific programming, some aspects of this methodology can be used with great benefit by computational scientists. However, until recently the use of Fortran has prevented the OO approach from being used for scientific software development. The three most recent Fortran standards, Fortran 90, Fortran 95 and especially Fortran 2003, have begun to make some fundamental OO features available within Fortran's historical context of legacy applications and an emphasis on computationally efficient numerical computing.

The structure of large datatypes in computational condensed matter physics tends to be simple: uniform grids of various dimensions and lists of atomic neighbours, where the maximum sizes of the neighbourhood are easy to estimate in advance and have a small variance. This means that once abstract objects are defined, the object hierarchy tends to be rather flat, in contrast for example to the situation in graphical user interface programming, where a strict OO model is the only way to keep order among the thousands of interrelated objects. For many applications, problems that involve such "spatially homogeneous" objects are relatively easy to parallelize, and achieving reasonable load balance is at most moderately difficult. While there are clearly situations where more sophisticated parallelization is necessary and effective [1, 2], for less demanding systems each processing unit simply has to get its equal-sized share of the atoms or the grid or grids to work on. The number of operations that has to be carried out by each unit and the relative time they take are easy to estimate.

The primary objective of good scientific programming, especially in supercomputing applications, is thus increasing the speed of execution which translates into the optimization of relatively simple operations on arrays of floating point numbers. Thus the enduring attraction of Fortran becomes apparent, with its intrinsic multidimensional arrays and good compiler optimization properties for

tight loops. However, even despite these advantages compiled Fortran code can be slower than well optimized libraries, for example Intel's MKL and AMD's ACML. For code segments that contribute significantly to the run time, expressing algorithms in a linear algebraic form and calling an external library to carry out the operations can, at a minimal loss of readability, maintain source code simplicity while achieving optimal performance.

Recent revisions of Fortran have introduced a number of features, some that simply remove legacy restrictions, but many that define higher level programming constructs leading to abstract data types and OO design. The most essential of these, added in the Fortran 90 standard, are dynamic memory allocation (`allocate()` and `deallocate()`), modules, user defined types (`type X ... end type X`), and procedure and operator overloading (`module procedure` and `operator(X)`). These are sufficient for creating object-like entities and procedures that behave somewhat like object methods. Not essential from an OO design point of view, but still very important in our view, are the programming simplicity and computational efficiency of array syntax, and the removal of fixed source format requirements that date back to the use of punched cards.

The Fortran 2003 standard added more explicitly OO design features, such as inheritance, but most are not yet supported by any available compilers. However, a few of these new features, previously defined as extensions to the 95 standard [3], are very useful and are already implemented by some compilers. The most important is the possibility of allocatable components in user defined types. This greatly increases the flexibility of such constructs, making it possible to encapsulate in a derived type an object (e.g. a collection of atoms) whose size (i.e. the number of atoms) is not known at compile time. The Fortran 90/95 equivalent uses pointer type components, which can lead to memory leaks and reduced optimizer efficiency. Another useful extension is allocatable function arguments, which makes it possible to write routines that modify or extend the memory allocation semantics, for example expanding the size of a previously allocated array, or logging every instance of memory allocation to track memory usage.

Expressive programming

We are proponents of the concept of *expressive programming,* a programming style in which the structure and language of the top level code is as close as possible to the abstract algorithm one is trying to implement. For example, in the context of a molecular dynamics simulation, achieving this goal while maintaining reasonable code speed means having objects of type `Atoms` and variables

like `Atoms%positions` and `Atoms%velocities` so that the code which implements a time step integrator can resemble the corresponding mathematical formula. However, one must walk a fine line here, because a full OO approach would create a deeper object hierarchy, e.g. `Container%atom%position` and `Container%atom%velocity`. This implementation can be very efficient for some operations, as we show below, but the tradeoff is that producing an object referring to, for example, the full set of positions would require routines to gather these values, leading to more unreadable code and unnecessary copying of data. This tradeoff between code transparency and efficiency (and even portability in case of more esoteric OO solutions) always exists and our view is that advance knowledge of the critical code fragments is needed to make the correct choice.

Parallelization is another facet of the same tradeoff. With a single module that implements distributed arrays, one gains transparent parallelization for collections of three-vectors and matrices, and the level of the code which implements physics algorithms can be entirely free of parallelization clutter.

A good example where the above approach has been carried through with considerable care and attention to detail is the DFT++ project, which implements a plane wave pseudopotential density functional code in C++ [4]. This project has been the source of inspiration for our own efforts in expressive programming.

Example: libAtoms

By way of an example, we show some code fragments from a freely available molecular dynamics library called LIBATOMS [5]. It is a reusable library for manipulating a large number of atoms in the context of an atomistic simulation, together with a collection of lower level utilities including function minimisation. In the following, we show simplified type definitions, with some auxiliary components omitted for clarity.

The Linear Algebra module extends the already available Fortran intrinsics, mostly by implementing multiplications between arrays considered as vectors and matrices. It also wraps LAPACK routines by autogenerating the work arrays and array sizes.

Other than simple scalars and arrays, the basic container for data is the `type(Table)`. It stores an arbitrary number of fields for an arbitrary number of columns. Here we show only the source relevant for the storage of real fields. The `Table` object is smart in the sense that it extends and shrinks its allocatable data as necessary and figures out from the array sizes whether the caller to `append()` wants to append rows or columns.

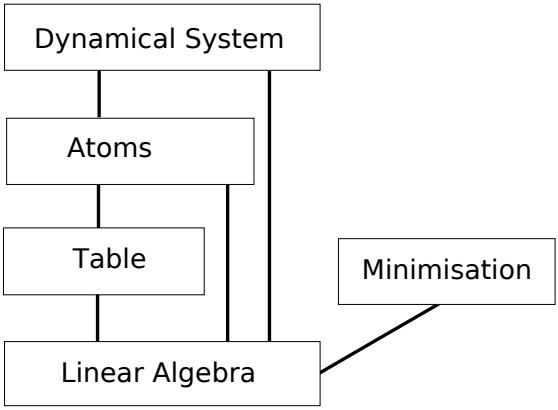


Figure 1: Dependencies of the main modules in libAtoms.

```
module Table_module
type Table
  integer :: Nrows = 0, Ncols = 0
  real(dp), allocatable :: real(:,:)
  ...
end type Table
contains
...
subroutine append(this, array)
  type(Table), intent(inout) :: this
  real(dp) :: array(:,:)
  ...
end subroutine append
```

The `type(Atoms)` uses a `type(Table)` component to keep the data about every atom, where the columns of the table correspond to the different atoms, and rows hold the components of the positions, velocities, accelerations, etc. This arrangement means that data pertaining to any particular atom is localised in memory, and consecutive rows can be passed to functions as arrays with a given stride without creating temporaries. Such sets of rows corresponding to physically meaningful variables are accessed via pointers defined in `type(Atoms)` as shown below. The implications of this choice for code readability and speed is discussed in more detail below.

```
module Atoms_module
use Table_module

type Atoms
  integer  :: N = 0 ! Number of atoms
  real(dp) :: lattice(3,3) ! periodic supercell vectors
  type(Table) :: data ! actual data corresponding to each atom
  integer :: Z(:) ! atomic number
  ! pointers into rows of this%data for frequently used fields
  real(dp), pointer :: pos(:,:), vel(:,:), acc(:,:), mass(:,:)
end type Atoms

contains
...
subroutine Atoms_init(this, N)
  type(Atoms), intent(inout) :: this
  integer :: N ! number of atoms

  call init(this%data, 10, N) ! initialise (10,N) table

  ! assign pointers
  this%pos  => this%data%real(1:3,:)
  this%vel  => this%data%real(4:6,:)
  this%acc  => this%data%real(7:9,:)
  this%mass => this%data%real(10,:)
  ...
end subroutine Atoms_init

end module Atoms_module
```

Dynamics

The module `DynamicalSystem_module` contains the types and procedures necessary for propagating atomic configurations in time. The compact expressions using array syntax make it quite easy to see how the velocity Verlet algorithm is implemented in four lines of source code.

```
module DynamicalSystem_module
use Atoms_module

type DynamicalSystem
  type(Atoms) :: atoms ! atoms in system
end type DynamicalSystem
contains
```

```
...
subroutine AdvanceVerlet(this, dt, force)
  type(DynamicalSystem), intent(inout) :: this
  real(dp), intent(in) :: dt
  real(dp), intent(in) :: force(:,:)

  ! implementation 1
  this%atoms%pos = this%atoms%pos + dt * this%atoms%vel &
                     + 0.5_dp * dt * dt * this%atoms%acc

  this%atoms%vel = this%atoms%vel + 0.5_dp * dt * this%atoms%acc

  forall (i=1:this%atoms%N) &
    this%atoms%acc(:,i) = force(:,i) / ElementMass(this%atoms%Z(i))

  this%atoms%vel = this%atoms%vel + 0.5_dp * dt * this%atoms%acc

end subroutine AdvanceVerlet

end module DynamicalSystem_module
```

The choice of data structures for storing the atomic data influences both source code readability and execution speed. As the `AdvanceVerlet()` example shows, we can use array syntax to write readable code. The speed of this approach (implementation 1) was compared with two other implementations, one using explicitly specified array slices rather than pointers, and the other using an array of `Atom` objects.

```
  type Atom_OO
    real(dp) :: pos(3), vel(3), acc(3), ...
  end type Atom_OO
  type Atoms_OO
    integer N
    type(Atom_OO), allocatable :: at(:)
  end type Atoms_OO

  type(Atoms) at
  type(Atoms_OO) :: at_oo(:)
  .
  .
  allocate(at_oo%at(N))
  .
  .
  ! implementation 2: data table, explicitly specified array slices,
  ! fused loops
  do i=1, at%N
    at%data(1:3,i) = at%data(1:3,i) + dt * at%data(4:6,i) + &
                         0.5_dp * dt * dt * at%data(7:9,i)
```

```
  ...
end do
! implementation 3: Deeper OO hierarchy
do i=1, N
  at_oo%at(i)%pos = at_oo%at(i)%pos + dt * at_oo%at(i)%vel + &
                    0.5_dp * dt * dt * at_oo%at(i)%acc
  ...
end do
```

We compared the timings for the velocity Verlet time propagation algorithm (only fragments are shown here) using the Intel Fortran 90/95 compiler for Linux version 9.1 with the `-fast` flag, on a 3.0 Ghz Pentium D, for $N = 10^7$. Implementations 2 and 3 took about 0.45 s, while implementation 1 took 1.7 s, a ratio of about 4. A factor of two is attributable to replacing the four implicit loops (update positions, update velocities, calculate accelerations, finish updating velocities) in the array syntax with a single loop that fuses all four, and another factor of two is attributable to the use of pointers to array slices. Both could, in principle, be remedied by a more sophisticated optimizer, and indeed these timings are expected to vary with the choice of compiler. This example is in some ways a worst-case scenario, because the operations in the loop are so simple. In practice, the loop fusion can not be accomplished in a more complex molecular dynamics program with features such as constant temperature, constraints, or rigid body dynamics. The overhead due to our implementation in this case is only a factor of two, and even this factor may disappear if the operations in the loop are more complex than the multiply-add of the simple velocity Verlet shown above. Both of the faster implementations suffer from a proliferation of loops, while in addition implementation 2 requires cumbersome constructs to make the `Atoms` structure transparently accomodate run time selection of the fields associated with each atom, while implementation three makes it difficult to extract an object that encompasses all the positions, for example. Ultimately, for most realistic atomistic calculation applications the speed advantages of implementations two and three are too small to compensate for the readability disadvantages: computing energies and forces will dominate the computational expense, certainly for explicitly quantum-mechanical methods or interatomic potential with long-range (Coulumb) terms, but even for physically realistic short range potentials. A factor of four slowdown in a part of the code that takes a negligible fraction of the total time is not significant.

Minimisation

The idea of minimizing some function with respect to some arguments is a very general one in computational physics. Many sophisticated algorithms are available, for example conjugate gradients, damped dynamics, and quasi-Newton just to name a few. Since many of these algorithms can be complex to implement, especially in a robust way, implementations that can be reused, and do not know about the details of the quantity to be minimised, are very useful. A pure OO approach would be to use a class that defines all the methods that the minimisation subroutine would use, for example adding a product of the search direction and some scalar to the current value, and taking the norm of the argument vector. Template functions or inheritance could then be used to provide these methods as an interface that the minimisation subroutine actually calls. By avoiding true OO design, we trade off this complexity for doing a bit more work ourselves, namely manually packing the arguments of the function to be minimised into a vector of real numbers.

The vector, as well as pointers to procedures that compute the function to be minimised and its gradient, are passed to the minimisation procedure. The `transfer()` intrinsic can be used to pack additional arbitrary data to be passed to the function evaluation procedure.

```
module Minimisation_module
contains

function minim(x, func, dfunc, method, convergence_tol, max_steps,&
               extra_data)
  real(dp), intent(inout) :: x(:)
  character(len=*), intent(in) :: method
  real(dp), intent(in) :: convergence_tol
  integer, intent(in) :: max_steps
  integer, intent(in) :: extra_data(:)
  interface
    function func(x, extra_data)
        real(dp) :: x(:)
        integer :: extra_data(:)
        real(dp) :: func
    end function func
  end interface
  interface
    function dfunc(x, extra_data)
        real(dp) :: x(:)
        integer :: extra_data(:)
        real(dp) :: dfunc(size(x)
    end function dfunc
```

```
  end interface

  real(dp) :: E0
  real(dp), allocatable :: F0(:)

  allocate(F0(size(x))

  ! find initial function value and gradient
  E0 = func(x, extra_data)
  F0 = dfunc(x, extra_data)

  ! do minimisation
  ...
end function minim

end module Minimisation_module

module minimise_energy_module

use Atoms_module
use Minimisation_module

contains

subroutine minimise_energy(at)
  type(Atoms), intent(inout) :: at

  integer :: at_packed(AT_PACKED_SIZE)
  real(dp), allocatable :: pos_1d(:)

  allocate(pos_1d(at%N*3)

  pos_1d = reshape(at%pos, (/3*at%N/))

  at_packed = transfer(at, at_packed)

  call minim(pos_1d, calc_energy_1d, calc_forces_1d, "conjgrad", &
             1.0e-6_dp, 100, at_packed)

  at%pos = reshape(pos_1d, (/3,at%N/) )
end subroutine minimise_energy

function calc_energy_1d(pos_1d, at_packed)
  real(dp), intent(in) :: pos_1d(:)
  integer, intent(in) :: at_packed(:)
  real(dp) :: calc_energy_1d

  type(atoms) :: at
```

```
  at = transfer(at_packed, at)
  at%pos = reshape(pos_1d, (/3,at%N/) )

  calc_energy_1d = calc_energy(at)
end function calc_energy_1d
...
end module minimise_energy_module
```

Conclusions

Recent revisions to the Fortran standard have greatly modernised this programming language, enhancing it with very useful features for scientific computing. A judicious use of OO design using array syntax, derived types with allocatable components, and procedure overloading can be used to practice *expressive programming*, where the top level code cleanly reflects the implemented algorithm, without significantly sacrificing performance. We have implemented basic components needed for atomistic simulations using these ideas in the freely available libAtoms library.

References

[1] "Multiresolution algorithms for massively parallel molecular dynamics simulations of nanostructured materials." R. K. Kalia, T. J. Campbell, A. Chatterjee, A. Nakano, P. Vashishta, and S. Ogata, Comp. Phys. Commun. **128**, 245-259 (2000).

[2] "Scalable Algorithms for Molecular Dynamics Simulations on Commodity Clusters," Kevin J. Bowers, Edmond Chow, Huafeng Xu, Ron O. Dror, Michael P. Eastwood, Brent A. Gregerson, John L. Klepeis, Istvan Kolossvary, Mark A. Moraes, Federico D. Sacerdoti, John K. Salmon, Yibing Shan, and David E. Shaw, SC06 (2006).

[3] "ALLOCATABLE attribute," ISO Technical Report TR 15581 (2001).

[4] "New algebraic formulation of density functional calculation," by Sohrab Ismail-Beigi and T.A. Arias, Computer Physics Communications 128:1-2, 1-45 (2000)

[5] `http://www.libatoms.org`

Dr Gabor Csanyi is a lecturer in the Engineering Laboratory at the University of Cambridge and can be contacted at `gabor@csanyi.net`.

Computational Physics Group News

The Computational Physics Thesis Prize 2007

The Committee of the Institute of Physics Computational Group has endowed an annual thesis prize for the author of the PhD thesis that, in the opinion of the Committee, contributes most strongly to the advancement of Computational Physics. A total prize fund of £1000 will be divided between the prize-winner and the runners up. The number of awards is at the discretion of the Committee.

- The deadline for applications is March 1st, 2008.
- The submission format is a 4 page (A4) abstract together with a citation (max. 500 words) from the PhD supervisor and a confidential report from the external thesis examiner. Further details may be requested from short-listed candidates.
- The submission address is:
  Dr M Probert
  Department of Physics
  University of York
  York, YO10 5DD
  email: mijp1@york.ac.uk
- Please enclose full contact details, including an email address.

Applications are encouraged across the entire spectrum of Computational Physics. The competition is open to all students who have carried out their thesis work at a University in the United Kingdom or the Republic of Ireland, and whose PhD examination has taken place in 2007.

The Computational Physics Thesis Prize 2005/2006

The computational physics thesis prize for 2005 and 2006 has been jointly awarded to Alex Robinson and Zhongfu Zhou (£500 each). Congratulations!

International Union of Pure and Applied Physics: Young Scientist Prize in Computational Physics

Applications invited for 2007

The "International Union of Pure and Applied Physics Young Scientist Prize in Computational Physics" (IUPAP Young Scientist Prize) can be awarded to researchers who have a maximum of 8 years research experience following their PhD. See http://c20.iupap.org/prizes.htm for details.

Winner 2006

The Computational Physics group congratulations the winner of the IUPAP Young Scientist Prize 2006: Prof. Stefano Sanvito, of Trinity College Dublin.

IoP Computational Physics Group - Student Travel Award

The Computational Physics Group (CPG) of the Institute of Physics (IoP) is pleased to invite requests for partial financial support towards the cost of attending scientific meetings relevant to the Group's scope of activity, as outlined on our web page: http://groups.iop.org/CP/. The aim of the scheme is to help stimulate the career development of young scientists working in computational physics to become future leaders in the field.

To be eligible the applicant should:

- be a full time PhD student;
- provide evidence of acceptance of a presentation (oral or poster) at the meeting in question;
- give an itemised estimate of cost of attendance;
- provide a letter of support from their project supervisor which:
  - ▷ confirms the applicant's PhD student status;
  - ▷ explains the relevance of the meeting;

- ▷ details the source of the additional funds necessary to attend the meeting.

Applications are invited at any stage in a given year, but will be reviewed by the CPG Committee on a quarterly basis (1st March, 1st June, 1st September, 1st December). Successful applicants will be notified as soon as possible thereafter. Candidates are advised to make their submissions well in advance of the meeting they wish to attend. The maximum support available to any applicant will be £200. The CPG's decision regarding financial support and its level will be final and non-negotiable in all cases.

Successful applicants will be asked to provide a short written report of the meeting suitable for publication in the CPG Newsletter.

For further details, please contact:

Dr D.G.Lewis
Department of Medical Physics
Velindre Hospital
Cardiff CF14 2TL
e-mail: dg.lewis@physics.org
Tel: 029 2019 6192

Reports on meetings

2006 Nuclear Science Symposium and Medical Imaging Conference

San Diego, USA; 1 Nov 4 Nov 2006

Report by: Haval Kadhem (Biomedical Imaging Group, The Centre for Vision, Speech and Signal Processing, University of Surrey)

The Medical Imaging Conference is the longest standing and most respected international scientific meeting on the physics, engineering and mathematical aspects of X-ray and nuclear medicine based imaging. It provides an opportunity to present significant innovations in the field of medical and biomedical imaging.

The event is well attended by scientists and engineers from industry providing unique environment for researchers in the various biomedical imaging to interact and exchange ideas directly with the specialists.

The Medical Imaging Conference, commenced on Wednesday November 1st with two outstanding scientists invited as plenary speakers. The first plenary talk given by Jan Schnitzer, director of the Sidney Kimmel Cancer Center in San Diego. He presented his views on a systems biology approach to cancer therapy and highlighted the importance intravital microscopy and SPECT imaging plays in deciding what proteins to use for the delivery of endogenous molecules and targeted drugs to reach specific tissue and tumour cells.

The second plenary talk given by Ron Nutt, chairman and CEO of Advanced Biomarker Technologies in Tennessee. He spoke of the role PET and SPECT play in molecular medicine and gave a review of the history of molecular imaging emphasising on how successful and widespread PET/CT had become in clinical settings. Similar to the first plenary talk he described how molecular imaging was being applied to the field of drug development. The talk was concluded by focusing on new technologies that were important to the development of molecular imaging and the first published images of a PET insert operating inside a whole body MRI system to perform simultaneous imaging were shown.

The Medical Imaging Conference consisted of twelve sessions of contributed papers, and three substantial poster sessions, with the great emphasis on emission and transmission tomography. This year there was a number of posters presentations on PET/MRI as well as a dedicated workshop to this emerging technology.

The majority of the poster presentations were on a variety of Nuclear Imaging related issues including image registration, attenuation, scatter and motion correction for PET and SPECT. The poster presentations were split into 3 poster presentation sessions, which was really good, as it enabled everyone to present their posters and then get the chance to view other peoples posters in the remaining sessions.

This year, our group was very successful with five quality papers accepted to the conference on various aspects of medical imaging. Two of the papers were nominated for Best Student Paper Awards, one of which was my paper on Ultra Low Dose CT Attenuation Correction Maps for PET/SPECT. The poster presentation was very well received and attracted the attention of a variety of people. I was flattered to have David Gilland, the poster session chair and a distinguished researcher in my field show an interest in my poster and actually spend a large part of the session discussing my research and providing useful suggestions and areas for improvements. I was very conscious of the audience

that had grouped around us and was very excited in having to explain my research and finding to fellow researchers. This proved extremely educational and had highlighted a number of possible extensions to my research that I had not previously thought of.

The Conference Reception which was held on Wednesday evening on the hotel grounds beside the swimming pools were very well attended and it proved an excellent opportunity to meet fellow researchers and making new friendships and contacts. I had thoroughly enjoyed the entertainment provided by a string octet from the Holland-Moritz Ensembles in San Diego at the reception. San Diego had proved to be a great location for a conference and I had thoroughly enjoyed visiting the famous San Diego Zoo, Balboa Park and of course the amazing SeaWorld marine Adventure Park, where we got to enjoy a relaxing time and see the Shamu experience.

I had found the conference extremely rewarding and educational. I was extremely pleased to know that my research work had attracted a good level of interest from distinguished fellow researchers in the same field. I am very grateful to the Institute of Physics Computational Physics Group for partially funding my attendance to such a major conference.

MMM/Intermag conference Jan 2007

Report by Giuliano Bordignon, University of Southampton

I went to the MMM/Intermag conference in Baltimore with other two people from my group at University of Southampton: Matteo Franchin and Thomas Fischbacher. We took the flight from London to Baltimore on Saturday evening and at the destination the approach to the city was very nice, with a fast security check, a mild temperature and a (missionary) taxi driver who told us about his life and his youth in Europe.

The check-in at the Baltimore Marriott Waterfront was nice as well, with a friendly welcome by the staff and a fascinating view on the waterfront from our room. We spent most of the Sunday making the last decisions on the most relevant talks to attend, hard work indeed, as the conference covered all the possible topics and applications concerning the field of magnetism.

As my field of research is the micromagnetic simulations of patterned nanostructures, I decided to attend the sessions on patterned structures and micromagnetics as general topics and I picked up some other talks which looked interesting in the Joint Program booklet.

All the three of us presented either a talk or a poster, and in the spare time between the various sessions we had good fun practicing in the empty rooms of the hotel and making the last corrections to our slides. At the poster session on Tuesday morning, beside presenting our software for micromagnetic simulations, there were a lot of opportunities to get in touch with people doing similar things, and in general these sessions were long enough to have a relaxed look at all the posters showed.

As the conference was sponsored by the AIP and the IEEE, the presentations were a good mix of academic and industrial topics, with the introductory Tutorials on spin torque, the Plenary Session and the various Symposia very interesting from both points of view.

Besides the poster session, the other networking appointments were the Bierstube events: evening informal drinks for all the participants where the main activities were chatting and queueing for a beer. In those occasions it was interesting to see how easy it was for young students to talk with experienced researchers and professors, even the teetotal ones! On Thursday morning I had my 10 minutes of celebrity giving a talk on Analysis of magnetoresistance in arrays of connected nano-rings. Being the final day of the conference the audience was not so numerous, even if my time slot was between two interesting talks by M. J. Donahue and N. Benatmane. All went pretty well (I heard stories of bad communication between laptops and projectors), and if we consider that if was my first conference presentation, I must say that it was a good experience and an excellent practice).

Since our flight back was on Friday evening, I used Friday morning to have a taste and take some photos of the surroundings of the hotel. I particularly liked the inner harbour with its 3d aquarium and enjoyed the local crab cakes, a must for all the tourists visiting Baltimore.

International Workshop on Monte Carlo Codes and 13th UK Monte Carlo User Group Meeting (MCNEG 2007)

Organised by: David Shipley and Alan DuSautoy

These two meetings were held back to back; the International Workshop on Monte Carlo Codes was held on 26 - 27 March 2007 and the 13th UK Monte Carlo User Group Meeting (MCNEG 2007) was held on 28 - 29 March 2007,

both at National Physical Laboratory, Teddington, Middx, UK. Further information is (and presentations will shortly be) available at the meeting website: http://www.npl.co.uk/ionrad/training/montecarlo/

The meetings had a very international flavour with eight invited speakers: Alex Bielajew (University of Michigan, USA) , Iwan Kawrakow (National Research Council, Canada) , Michael James (Los Alamos National Laboratory, USA) , Maria Grazia Pia (Istituto Nazionale di Fisica Nucleare, Italy), Francesc Salvat (University of Barcelona, Spain), Andrey Berlizov (Institute for Nuclear Research, Ukraine), Nick Reynaert (Ghent University, Belgium) and Bruce Faddegon (UCSF Comprehensive Cancer Center, San Francisco). There were 137 attendees from over 20 countries as far a field as Xian in China and Recife in Brazil, with many European countries represented.

These events were kindly supported by: the Institute of Physics (Computational Physics Group and Medical Physics Group), Elekta, Maestro (see later), The Panel on Gamma and Electron Irradiation, and Institute of Physics and Engineering in Medicine; and in cooperation with the International Atomic Energy Agency. Hassan Ali Nedaie (Tehran University of Medical Sciences, Iran) was awarded a travel bursary of £350 kindly provided by the Institute of Physics Computational Physics Group.

The International Workshop on Monte Carlo Codes was devoted to some of the most popular Monte Carlo radiation transport codes in use, and included sessions by key code developers on the following codes: EGSnrc (developed by Stanford University and NRCC), Geant4 (developed by INFN and CERN), MCNPX (developed by Los Alamos), and PENELOPE (developed by University of Barcelona). Interspersed with these, teaching lectures were provided giving an Overview of the Monte Carlo method for radiation transport, as well as topics including Geometry modelling, Variance reduction and Advanced transport algorithms. At the end of the first day, there was also an open-house session with both delegates and developers displaying posters and demonstrating their codes and applications on laptops.

The MCNEG meeting was aimed at users of all radiation transport codes. It provided delegates with the opportunity to present and discuss their applications and recent developments of the Monte Carlo method. We were delighted by the quality and number of the presentations. There were 28 oral presentations and 18 posters including: Radiotherapy treatment planning, Microdosimetry, Radioactivity, Radiotherapy in general, Radiation Protection, Low energy interaction modelling, Brachytherapy, Neutrons and Nuclear safety, and Maestro. The Maestro (Methods and Advanced Equipment for Simulation and Treatment in Radio Oncology) project is a 10MEUR European Framework 6 funded project with 24

collaborators (http://www.maestro-research.org/).

At the end of the first day there were tours around the new state-of the-art ionising radiation facilities at NPL (http://www.npl.co.uk/ionrad/facilities) with the MCNEG annual general meeting held at the end of the meeting.

There have been many positive comments from the delegates such as: "I want to thank You and Your colleagues for the wonderful conference. Everything was perfect and high quality stuff: from the presentations to the food!" "Excellent course organisation, good timekeeping, interesting talks - many thanks to all organisers!" Many thanks go to all those that made the meetings such a success.

The next MCNEG meeting for 2008 will be organised by British Nuclear Fuels (BNFL). Details will be announced on the MCNEG website (http://www.mcneg.org.uk/) in due course.

Upcoming Computational Physics Events

Conference on Computational Physics 2007

The Conference on Computational Physics (CCP) 2007 continues the series of the APS-EPS "Physics Computing". It takes place from September 5 to September 8 2007 in Brussels.

The meeting is organised by the European Physical Society, and IoP members qualify for member rates.

Web page: http://ccp2007.ulb.ac.be/Welcome.html

Non-Adiabatic Molecular Dynamics - A Discussion

This meeting will be held on 10 September 2007, at 76 Portland Place, and will be a discussion on the subject of non-adiabatic molecular dynamics. Experts representing various different methodologies will each give a talk explaining how their particular method works, and how it relates to the other methods being discussed. There will then be time for general discussions. There will also be a poster session so that students and others can present specific results in some detail obtained using the various methods.

Computational Tools

In this part of the newsletter, we provide occasionally information on selected software packages, tips and tricks relating to operating systems and other computational tools. Contributions to the section are very welcome, and should be emailed to the newsletter editor.

"reStructuredText" (rst) – convert ASCII to HTML and TEX

"reStructuredText" is a mark-up language which is intended to be easily readable (you can study the document in the grey box to see whether this is the case). This short introduction of the use of "reStructuredText" (rst) should be self explaining. First, we show the ASCII source code. Then, in figures 1 and 2, we use the "docutils" conversion tools to convert this ASCII source to HTML and LATEX, respectively.

```
=========================================
ReStructuredText (rst): plain text markup
=========================================

What is reStructuredText?
~~~~~~~~~~~~~~~~~~~~~~~~~

An easy-to-read, what-you-see-is-what-you-get plaintext markup
syntax and parser system, abbreviated *rst*. In other words,
using a simple text editor, documents can be created which

- are easy to read in text editor and
- can be *automatically* converted to

  - html and
  - latex (and therefore pdf)

What is it good for?
~~~~~~~~~~~~~~~~~~~~

reStructuredText can be used, for example, to write technical
documentation (so that it can easily be offered as a pdf file or
a web page), create html webpages without knowing html, or to to
document source code

Show me some formatting examples
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
You can highlight text in *italics* or, to provide even more
emphasis in **bold**. Often, when describing computer code, we
like to use a ``fixed space font`` to quote code snippets.

We can also include footnotes [1]_. We could include source code
files (by specifying their name) which is useful when documenting
code. We can also copy source code verbatim (i.e. include it in
the rst document) like this::

  int main ( int argc, char *argv[] ) {
      printf("Hello World\n");
      return 0;
  }

Where can I learn more?
~~~~~~~~~~~~~~~~~~~~~~

reStructuredText is described at
http://docutils.sourceforge.net/rst.html. This example (slightly
extended) is available in source code at
http://www.soton.ac.uk/~fangohr/tools/rst/rstwebpage.html.

-----------------------------------------------------------------

.. [1] although there isn't much point of using a footnote here.
```

After saving the text file shown above under the name of `rst.txt`, we use the following command to convert it into an html document with name `rst.html`:

```
rst2html rst.txt rst.html
```

Figure 1 on page 23 shows a snap shot of the document displayed in a web browser. Note that one can provide a css-style file to the `rst2html` command to customise the formatting of the html output.

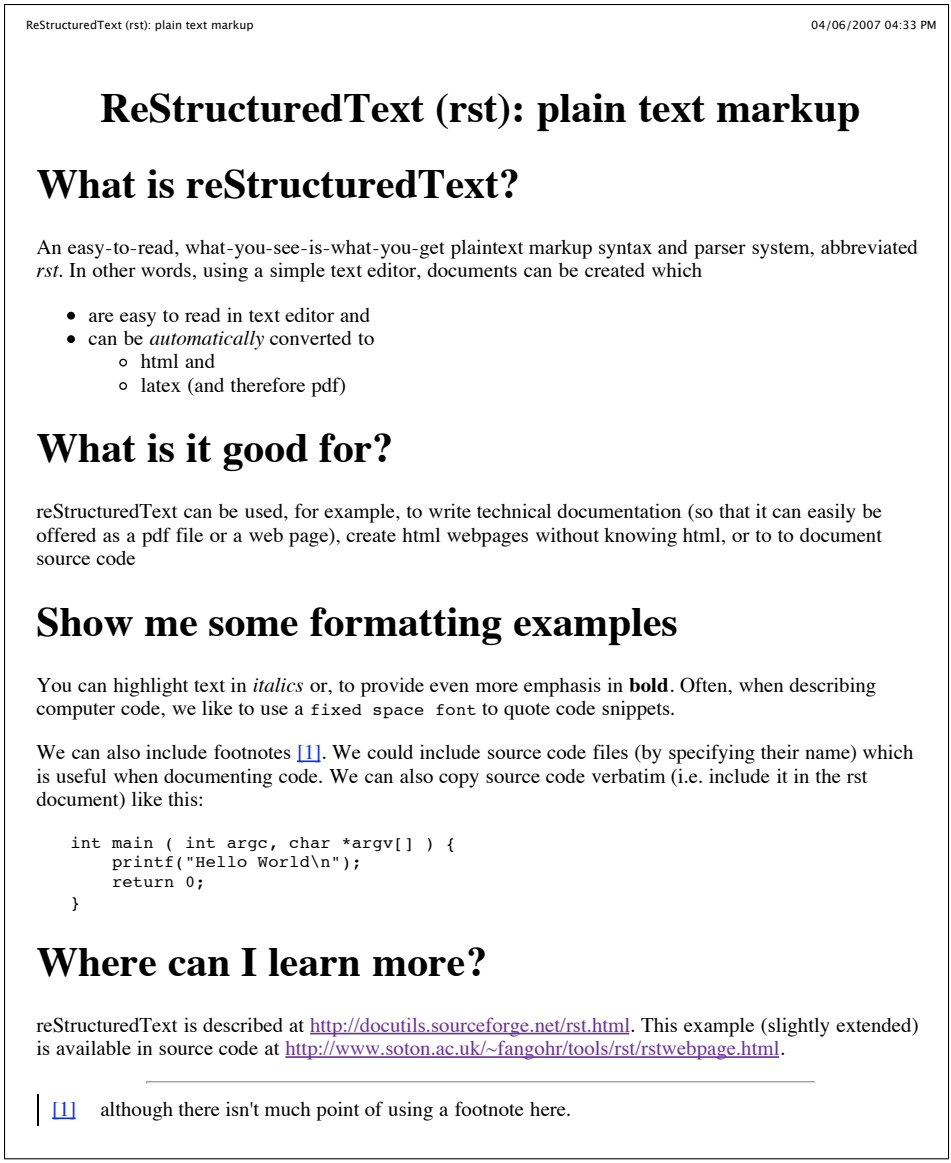
The conversion into latex is done quite similarly:

```
rst2latex rst.txt rst.tex
```

Figure 2 on page 24 shows the pdf file produced from `rst.tex`. The `rst2latex` command takes a number of options to change the LaTeX layout.

The full example can be studied and downloaded from `http://www.soton.ac.uk/~fangohr/tools/rst/rstwebpage.html`. We have shown a small set of the capabilities of reStructuredText that may be useful for computational scientists and which – hopefully – convey the underlying idea quickly.

(Hans Fangohr)



ReStructuredText (rst): plain text markup 04/06/2007 04:33 PM

ReStructuredText (rst): plain text markup

What is reStructuredText?

An easy-to-read, what-you-see-is-what-you-get plaintext markup syntax and parser system, abbreviated *rst*. In other words, using a simple text editor, documents can be created which

- are easy to read in text editor and
- can be *automatically* converted to
  - html and
  - latex (and therefore pdf)

What is it good for?

reStructuredText can be used, for example, to write technical documentation (so that it can easily be offered as a pdf file or a web page), create html webpages without knowing html, or to to document source code

Show me some formatting examples

You can highlight text in *italics* or, to provide even more emphasis in **bold**. Often, when describing computer code, we like to use a `fixed space font` to quote code snippets.

We can also include footnotes [1]. We could include source code files (by specifying their name) which is useful when documenting code. We can also copy source code verbatim (i.e. include it in the rst document) like this:

```
int main ( int argc, char *argv[] ) {
    printf("Hello World\n");
    return 0;
}
```

Where can I learn more?

reStructuredText is described at http://docutils.sourceforge.net/rst.html. This example (slightly extended) is available in source code at http://www.soton.ac.uk/~fangohr/tools/rst/rstwebpage.html.

---

[1] although there isn't much point of using a footnote here.

Figure 1: Conversion result of translating `rst.txt` into html (displayed with Safari).

ReStructuredText (rst): plain text markup

What is reStructuredText?

An easy-to-read, what-you-see-is-what-you-get plaintext markup syntax and parser system, abbreviated *rst*. In other words, using a simple text editor, documents can be created which

- are easy to read in text editor and
- can be *automatically* converted to
  - html and
  - latex (and therefore pdf)

What is it good for?

reStructuredText can be used, for example, to write technical documentation (so that it can easily be offered as a pdf file or a web page), create html webpages without knowing html, or to to document source code

Show me some formatting examples

You can highlight text in *italics* or, to provide even more emphasis in **bold**. Often, when describing computer code, we like to use a `fixed space font` to quote code snippets.

We can also include footnotes[1]. We could include source code files (by specifying their name) which is useful when documenting code. We can also copy source code verbatim (i.e. include it in the rst document) like this:

```
int main ( int argc, char *argv[] ) {
    printf("Hello World\n");
    return 0;
}
```

Where can I learn more?

reStructuredText is described at http://docutils.sourceforge.net/rst.html. This example (slightly extended) is available in source code at http://www.soton.ac.uk/~fangohr/tools/rst/rstwebpage.html.

---

[1] although there isn't much point of using a footnote here.

Figure 2: Conversion result of translating `rst.txt` into LaTeX (shown is the compiled latex document).