DARESBURY LABORATORY

INFORMATION QUARTERLY for COMPUTER SIMULATION OF CONDENSED PHASES

An informal Newsletter associated with Collaborative Computational Project No. 5 on Molecular Dynamics, Monte Carlo and Lattice Simulations of Condensed Phases.

Number 23 December 1986

Contents

Editor: Dr. William Smith
Deputy Editor: Dr. Maurice Leslie

Science & Engineering Research Council
Daresbury Laboratory, Daresbury,
Warrington WA4 4AD, England.

Editorial.

Exciting times are ahead for the U.K. simulation community. The much-heralded Cray XMP/48 supercomputer has at last been delivered to our shores and is safely housed at the S.E.R.C.s Rutherford and Appleton Laboratory. This community was particularly active when the U.K. took delivery of the first Cray 1s (for academic use) in 1980 and a good deal of new and important work was done. It is sincerely hoped that the enthusiasm that was so evident then will come through again despite the rather gloomy outlook that persists on U.K. science in general. It is expected that CCP5 will be an important catalyst in the exploitation of this new and powerful resource.

Our thanks (and the compliments of the season!) go to this month's contributors to our newsletter.

Contributors:

M.P. Allen — H.H. Wills Physics Laboratory, Royal Fort, Tyndall Avenue, Bristol BS8 1TS.

D. Frenkel — Fysisch Laboratorium, Rijksuniveriteit, Princetonplein 5, 3584 CC Utrecht, The Netherlands. Future correspondence - FOM-Institute for atomic & Molecular Physics, Kruislaan 407, 1098 SJ Amsterdam, The Netherlands.

M. Mezei — Department of Chemistry, Hunter College of the City University of New York, New York, NY 10021, USA.

M. Leslie, W. Smith — Theory and Computational Science Division, S.E.R.C. Daresbury Laboratory, Daresbury, Warrington WA4 4AD.

New Horizons
in The Computer Simulation of Condensed Phases

A CCP5 Meeting at UMIST, Manchester 8-9 January 1987

Program

THURSDAY 8 JANUARY

12.00 Registration - outside lecture theatre F14, Renold Building
Mounting of Posters

12.30 Buffet Lunch (Harwood Room, Barnes Wallis Building)

Session 1

13.45 W.L. Jorgensen (Purdue University)
"Computer Simulations of Organic and Biomolecular Systems"

14.35 M.P. Allen (Bristol) and D. Frenkel (Utrecht)
"MD Simulations of Molecular Liquids and Nematic Liquid Crystals"

15.00 S. Toxvaerd (Copenhagen)
"Generalised Langevin Equation for a Polymer in a Solvent"

15.25 TEA

Session 2

16.00 D. Rapaport (Bar-Ilan University, Israel)
"Microscale Hydrodynamics - what von Karman didn't know about billiard balls"

16.35 D. Heyes (Royal Holloway and Bedford New College)
"Simulation Studies of the Viscosity of Simple Liquids"

17.00 D. Tildesley, M.R. Stapleton (Southampton) and N. Quirke (BP)
"Computer Simulation of Polydisperse Liquids"

19.15 CONFERENCE DINNER (Harwood Room)

FRIDAY 9 JANUARY

Session 3

09.00 D. Price (UCL)
"Computer Simulations of Mineral Behaviour and Properties"

09.45 R.M. Lynden-Bell (Cambridge)
"Fluctuations in Orientationally Disordered Crystals and Phase Transitions"

10.10 C.R.A. Catlow (Keele)
"Molecular Dynamics Simulation of Beta-Alumina"

10.35 COFFEE

Session 4

11.00 M. Gillan (Harwell)
"Simulations of Quantum Systems"

11.40 B.H. Wells (Oxford)
"Green's Function Monte Carlo - New Developments"

12.05 D. Fincham (York)
"Parallel Computers - Developments and Prospects"

13.00 LUNCH (Harwood Room)

13.45 Steering Group meeting

POSTERS

D. Brown and J.H.R. Clarke
"Dynamical Simulation of a Model Reverse Micelle"

L.T. Wille
"Simulated Annealing of Atomic Clusters"

A.P. Sutton and A.T. Paxton
"Computer Simulation of Grain Boundaries in Metallic and Covalent Solids"

P. Clancy and D.K. Chokappa
"Laser Processing of Materials: A Non-Equilibrium MD Study"

M. Heggie, R. Jones and Y. Zhang
"Computer Simulation of Plasticity in Minerals"

R. Vetrivel, C.R.A. Catlow and E.A. Colbourn
"Simulation Studies on Adsorption of Methanol over Silicalite"

R.A. Jackson
"Computer Simulation of Zeolite Stability and Structure"

C. Bruin
"Generalised Hydrodynamics and Eigenmodes for a Lennard-Jones Fluid"

J. Talbot, P. Ballone, Ph. de Smedt and J.L. Lebowitz
"Computer Simulation of a Classical Fluid with Internal Quantum States

General News.

a) CCP5 is organising a meeting entitled "New Horizons in the Computer Simulation of Condensed Phases". The meeting will take place in Manchester 8-9 January. The purpose of the meeting is to discuss recent developments and future possibilities in the following fields:

Molecular and macromolecular systems
Polar solids
Fluid Transport Properties
Quantum Mechanical Simulations

A copy of the programme for the meeting appears in the preceeding two pages. A CCP5 Steering Committee meeting will take place on the Friday, after lunch.

Those interested in participating should contact Dr. J.H.R. Clarke, Department of Chemistry, U.M.I.S.T., Sackville Street, Manchester M60 1QD.

b) The Rutherford and Appleton Computer Centre have just taken delivery of the Cray XMP/48 supercomputer on 3 December 1986. It is hoped that the first user jobs will be run by the end of January 1987 and the formal acceptance will be completed by April. For the uninitiated, the Cray XMP/48 is a 4 processor machine with a clock cycle time of 8.5 ns and 8 Mwords of memory. It comes with a 32 Mwords solid state device (which operates logically as a disc storage device) that is connected to the main memory via a 1000 Mbyte/sec channel and 14.4 Gbytes of disc storage. It has a hardware scatter/gather facility to permit vectorisation of indirect addressing within Do loops and a hardware performance monitor, which can be used to assess performance of user codes. The Cray XMP/48 will be connected to the IBM 3081, which will serve as the front end and provide access to JANET users. Scientific support for this new machine has not yet been finalised, but it is clear that the CCPs will be very important in the proving of the machine.

The Atlas Computing Centre at RAL has released a document entitled "Advanced Research Computing Newsletter" outlining the state of play in this new development.

c) The University of Manchester Regional Computing Centre has announced that it is to replace the current Amdahl 470/V8 front end computer with an Amdahl 5890 processor in March/April 1987. This is part of a joint procurement with ULCC which is intended to provide a Common User Interface for National Centre users. The new Amdahl will be three times more powerful than the 470/V8 with 64 Mbytes of memory, 32 channels and 12 Gbytes disc storage. Further details are in the November issue of the UMRCC newsletter. UMRCC also intend to phase out the CDC 7600's and the CYBER 176 (services MFY, MFZ and MFX) in the course of 1987. The replacements for these have not yet

been decided upon. (MFZ, it should be noted has been given an extended lease of life of several months.)

A new Job Transfer and Manipulation Protocol (JTMP) service is now operating from Manchester.

d) The University of London Computing Centre is also about to upgrade its front end computer to an Amdahl 5890. At the time of writing, the delivery date was set for late November and production use is anticipated for December. Meanwhile, the second Cray service is now available as described in the November issue of the ULCC newletter. A new Cray compiler CFT 1.14 (bugfix 5) is available on a trail basis.

e) U.K. CCP5 participants are reminded that at Daresbury Laboratory the S.E.R.C. has available an FPS 164 attached processor, currently with 3 MAX (matrix accelerator) boards, which is available for grant supported computing within the Science Board Community. In the first year of its operation U.K. research groups are invited to apply informally to the Director of Daresbury Laboratory (Professor L.L. Green) for time on the FPS for benchmarking purposes. If the FPS proves viable for a given project, a formal application (using the familiar forms RG2 and AL54) may be made. Advice on using the FPS at Daresbury may be obtained from the User Interface Group (in the person of Dr. D. Taylor) or from the CCP5 representative Dr. W. Smith. A document entitled 'Using the FPS Attached Processor at the Daresbury Laboratory' by Dr. M.F. Guest is available from the User Interface Group.

f) In the last issue of this newsletter we reported that the simulation group at Groningen in the Netherlands are setting up a distribution service for their program package GROMOS86. CCP5 participants with access to the Daresbury Laboratory computing system may be interested to know that we are arranging to install a version of this package on the Daresbury FPS 164. Users of the FPS will then be able to assess the package themselves prior to purchasing it from Groningen or employing the package in a grant supported project on the Daresbury FPS. We should be able to provide more news on this subject in the March issue of the newsletter.

Readers interested in obtaining the package directly should write to "BIOMOS", The Laboratory of Physical Chemistry, The University of Groningen, Nijenborgh 16, 9747 AG Groningen, The Netherlands.

g) We have received a report from our colleagues in Bristol that an error is present in the CCP5 program MDMANY. It appears in the subroutine FURIER, which treats the reciprocal space part of the Ewald sum, in the part which defines the trigonometric terms. The offending line is:

```
EM(K,1)=COS(YC)+(0.,1.)*SIN(ZC)
```

in which the variable ZC should be replaced by YC. The impact of this error is unclear, it is present in an important part of the program but does not obviously manifest itself in the test case considered. This may well be due to the symmetry of the molecules considered or reflect the fact that the terms contributing are generally small. We urge users of this program to check their own versions and extend our apologies for any inconvenience.

We note that the sister program MDMPOL does NOT contain the corresponding error.

h) Our sister project CCP4, which is concerned with Protein Crystallography, is organising a study weekend on "Protein Crystal Data Analysis" at Daresbury Laboratory, 23-24 January 1987. For those members of CCP5 who also have an interest in this area, we include an information sheet at the end of this newsletter.

i) Readers may be interested to know of an international conference on 'The Impact of Supercomputers on Chemistry' is being organised and will take place in the University of London from 13 - 16 April 1987 The conference will cover all aspects of the use of supercomputers; applications and methodology. Some of the areas to be covered include: the simulation of condensed matter, molecular mechanics, simulation of biological systems, protein crystallography and molecular collision processes. A session on hardware developments and a Computer Exhibition is also planned. Plenary lectures will be given by M.J. Gillan, H.J.C. Berendsen, D. Ceperley, E. Clemmenti and T.L. Blundell among others. Enquiries regarding the conference should be addressed to Dr. J. Altmann, ISOC 87 Secretariat, Room 209, University of London Computer Centre, 20 Guildford Street, London WC1N 1DZ. (Telex 8953011).

j) It has been suggested to the CCP5 Editors that it would be useful for CCP5 participants to have a list of the EARN and BITNET addresses of other members of the project. For this purpose we would be happy to receive from our readers their addresses, which we shall collect here at Daresbury for publication in the newsletter at a later date. So that we don't fall foul of the 'Data Protection Act', it should be understood by all participants that the contributed addresses will be available to everyone.

k) Anyone wishing to make use of the CCP5 Program Library is invited to do so. Documents and programs are available free of charge to academic centres upon application to Dr. M. Leslie (*) at Daresbury Laboratory. Listings of programs are available if required but it is recommended that magnetic tapes (to be supplied by the applicant) be used. It may also be possible to transfer a small number of programs over the JANET network to other computer centres in the U.K.. Users wishing to send magnetic tapes are

instructed to write to Dr. Leslie for information before sending the tape. PLEASE DO NOT SEND TAPES WITHOUT CONTACTING DR. LESLIE FIRST. Delays are caused by applicants sending new tapes which have to be initialised at Daresbury (i.e. tape marks have to be written on them). Also tapes sent in padded bags have to be cleaned before use. Please do not use this form of packing. (A list of programs available follows in the next few pages.)

We should also like to remind our readers that we would welcome contributions to the Program Library. The Library exists to provide support for the research efforts of everyone active in computer simulation and to this end we are always pleased to extend the range of software available. If any of our readers have any programs they would like to make available, please would they contact Dr. Leslie.

* (Full address: S.E.R.C. Daresbury Laboratory, Daresbury, Warrington WA4 4AD, U.K.)

LIST OF PROGRAMS IN THE CCP5 PROGRAM LIBRARY.

MDATOM by S. M. Thompson.

M.D. simulation of atomic fluids. Uses 12/6 Lennard - Jones potential function and fifth order Gear integration algorithm. Calculates system average configuration energy, kinetic energy, virial, mean square force and the associated R.M.S. deviations and also system pressure, temperature, constant volume specific heat, mean square displacement, quantum corrections and radial distribution function.

HMDIAT by S. M. Thompson.

M.D. simulation of diatomic molecule fluids. Uses 12/6 Lennard - Jones site - site potential functions and a fifth order Gear algorithm for centre - of - mass motion. Angular motion is calculated by fourth order Gear algorithm with quaternion orientation parameters. Calculates system average configuration energy, kinetic energy, virial, mean square force, mean square torque and the associated R.M.S. deviations and also system pressure, temperature, constant volume specific heat, mean square displacement and quantum corrections.

MDLIN by S. M. Thompson.

M.D. simulation of linear molecule fluids. Uses 12/6 Lennard - Jones site - site potential functions and a fifth order Gear algorithm for centre - of - mass motion. Angular motion is calculated by fourth order Gear algorithm with quaternion orientation parameters. List of calculated properties is the same as HMDIAT.

MDLINQ by S. M. Thompson.

M.D. simulation of linear molecule fluids. Uses 12/6 Lennard - Jones site - site potential functions plus a point electrostatic quadrupole. Uses a fifth order Gear algorithm for centre - of - mass motion. Angular motion is calculated by fourth order Gear algorithm with quaternion orientation parameters. List of calculated properties is the same as HMDIAT.

MDTETRA by S. M. Thompson.

M.D. simulation of tetrahedral molecule fluids. Uses 12/6 Lennard - Jones site - site potential functions and a fifth order Gear algorithm for centre - of - mass motion. Angular motion is calculated by fourth order Gear algorithm with quaternion orientation parameters. List of calculated properties is the same as HMDIAT.

MDPOLY by S. M. Thompson.

M.D. simulation of polyatomic molecule fluids. Uses 12/6 Lennard - Jones site - site potential functions and a fifth order Gear algorithm for centre - of - mass motion. Angular motion is calculated by fourth order Gear algorithm with quaternion orientation parameters. List of calculated properties is the same as HMDIAT.

ADMIXT by W. Smith.

M.D. simulation of monatomic molecule mixtures. Uses 12/6 Lennard - Jones atom - atom potential functions and a Verlet leapfrog algorithm for centre - of - mass motion. Calculates system average configuration energy, kinetic energy and virial and associated R.M.S. deviations and also pressure, temperature, mean square displacements and radial distribution functions.

MDMIXT by W. Smith.

M.D. simulation of polyatomic molecule mixtures. Uses 12/6 Lennard - Jones site - site potential functions and a Verlet leapfrog algorithm for centre - of - mass motion. Angular motion is calculated by the Fincham leapfrog algorithm using quaternion orientation parameters. Calculates system average configuration energy, kinetic energy and virial and associated R.M.S. deviations and also pressure and temperature.

MDMULP by W. Smith.

M.D. simulation of polyatomic molecule mixtures. Uses 12/6 Lennard - Jones site - site potential functions and point electrostatic multipoles (charge, dipole and quadrupole). Long range electrostatic effects are calculated using the Ewald summation method. Uses a Verlet leapfrog algorithm for centre - of - mass motion. Angular motion is calculated by the Fincham leapfrog algorithm using quaternion orientation parameters. Calculates system average configuration energy, kinetic energy and virial and associated R.M.S. deviations and also pressure and temperature.

MDMPOL by W. Smith & D. Fincham.

M.D. simulation of polyatomic molecule mixtures. Uses 12/6 Lennard - Jones site - site potential functions and fractional charges to represent electrostatic multipoles. Long range electrostatic effects are calculated using the Ewald summation method. Uses a Verlet leapfrog algorithm for centre - of - mass motion. Angular motion is calculated by the Fincham leapfrog algorithm using quaternion orientation parameters. Calculates system average configuration energy, kinetic energy and virial and associated R.M.S. deviations and also pressure and temperature.

MDZOID by W. Smith & K. Singer.

M.D. simulation of ellipsoidal gaussian molecules. Uses ellipsoidal gaussian probability distribution to represent molecules and a sum of gaussian functions to represent the potential, giving a model of great flexibility. Uses the Verlet leapfrog algorithm for centre - of - mass motion. Angular motion is calculated by the Fincham leapfrog algorithm using quaternion orientation parameters. Calculates system average configuration energy, kinetic energy, virial and associated R.M.S. deviations and pressure and temperature. Also calculates centre-centre RDF, orientational order parameters and translational and rotational velocity autocorrelation functions.

DENCOR by W. Smith.

Calculation of density correlation functions. Processes atomic M.D. data to produce the Fourier transform of the particle density, the intermediate scattering functions and the dynamic structure factors.

CURDEN by W. Smith.

Calculation of current density correlation functions. Processes atomic M.D. data to produce the Fourier transform of the current density, the current density correlation functions and their temporal Fourier transforms.

HLJ1 by D. M. Heyes.

M.D. simulation of atomic fluids. Uses 12/6 Lennard - Jones site - site potential function and a Verlet leapfrog algorithm for centre - of - mass motion. Calculates system average configuration energy and kinetic energy and associated R.M.S. deviations and also pressure, temperature, mean square displacements and radial distribution function.

HLJ2 by D. M. Heyes.

M.D. simulation of atomic fluids. Uses 12/6 Lennard - Jones site - site potential function and a Verlet leapfrog algorithm for centre - of - mass motion. Calculates system average configuration energy and kinetic energy and associated R.M.S. deviations and also pressure, temperature, mean square displacements, radial distribution function and velocity autocorrelation function.

HLJ3 by D. M. Heyes.

M.D. simulation of atomic fluids. Uses 12/6 Lennard - Jones site - site potential function and a Verlet leapfrog algorithm for centre

- of - mass motion. The link - cell method is employed to enable large simulations. Calculates system average configuration energy and kinetic energy and associated R.M.S. deviations and also pressure, temperature, mean square displacements and radial distribution function.

HLJ4 by D. M. Heyes.

M.D. simulation of atomic fluids. Uses 12/6 Lennard - Jones site - site potential function and a Verlet leapfrog algorithm for centre - of - mass motion. The algorithm allows either the temperature or the pressure to be constrained. Calculates system average configuration energy and kinetic energy and associated R.M.S. deviations and also pressure, temperature, mean square displacements and radial distribution function.

HLJ5 by D. M. Heyes.

M.D. simulation of atomic fluids. Uses 12/6 Lennard - Jones site - site shifted potential function and a Verlet leapfrog algorithm for centre - of - mass motion. This method removes the discontinuities at the potential cutoff radius. Calculates system average configuration energy and kinetic energy and associated R.M.S. deviations and also pressure, temperature, mean square displacements and radial distribution function.

HLJ6 by D. M. Heyes.

M.D. simulation of atomic fluids. Uses 12/6 Lennard - Jones site - site shifted potential function and the Toxvaerd algorithm for centre - of - mass motion. This algorithm is more accurate than the Verlet algorithm. Calculates system average configuration energy and kinetic energy and associated R.M.S. deviations and also pressure, temperature, mean square displacements and radial distribution function.

MCRPM by D. M. Heyes.

M.C. simulation of electrolytes. Monte Carlo program using restricted primitive model of an electrolyte. The potential is regarded as infinite for r<d and Coulombic for r>d. The properties calculated are the average configuration energy and its R.M.S. deviation, the pair radial distribution function and the melting factor.

SURF by D. M. Heyes.

M.D. simulation of model alkalai halide lamina. Molecular dynamics simulation for ionic laminae using the Tosi-Fumi / Born-Mayer-Huggins potential and the Evjen method for evaluating

the lattice sums. The integration algorithm used is the Verlet method. The program calculates the system potential and kinetic energies, the pressure and the final averages and R.M.S. fluctuations. The program also calculates density profiles such as number density, temperature, energy and pressure.

HSTOCH by W. F. van Gunsteren & D. M. Heyes.

S.D. or M.D. simulation of molecules in vacuo or in a rectangular cell with solvent or lattice atoms (i.e. Langevin or Brownian dynamics of large molecules).

MDATOM by D. Fincham.

M.D. simulation of atomic fluids. Uses 12/6 Lennard - Jones potential function and Verlet leapfrog integration algorithm. Calculates system average configuration energy, kinetic energy, virial and the associated R.M.S. deviations and also system pressure, temperature, mean square displacement and radial distribution function.

MDDIAT by D. Fincham.

M.D. simulation of diatomic molecule fluids. Uses 12/6 Lennard - Jones site - site potential functions and the Verlet leapfrog algorithm for centre - of - mass motion. Angular motion is is calculated using the constraint algorithm. Calculates system average configuration energy, kinetic energy, virial and the associated R.M.S. deviations and also system pressure, temperature and mean square displacement.

MDDIATQ by D. Fincham.

M.D. simulation of diatomic fluids. Uses 12/6 Lennard - Jones site - site potential functions and a point quadrupole electrostatic term. Employs the Verlet leapfrog algorithm for centre - of - mass motion. Angular motion is calculated using the constraint algorithm. Calculates system average configuration energy, kinetic energy, virial and the associated R.M.S. deviations and also system pressure, temperature and mean square displacement.

MDIONS by D. Fincham & N. Anastasiou.

M.D. simulation of electrolytes. Uses exp/6/8 potential function and the Coulomb electrostatic potential. Long range interactions are calculated using the Ewald summation method. Uses the Verlet leapfrog algorithm for particle motion. Calculates system average configuration energy, kinetic energy, virial and the associated R.M.S. deviations and also system pressure, temperature, radial distribution functions, static structure factors and mean square

displacements.

MDMANY by D. Fincham & W. Smith.

M.D. simulation of polyatomic molecules. Uses 12/6 Lennard - Jones site - site potential functions and fractional charges to represent electrostatic multipoles. Long range electrostatic effects are calculated using the Ewald summation method. Uses a Verlet leapfrog algorithm for centre - of - mass motion. Angular motion is calculated by the Fincham leapfrog algorithm using quaternion orientation parameters. Calculates system average configuration energy, kinetic energy and virial and associated R.M.S. deviations and also pressure and temperature. FORTRAN 77 standard program.

CARLOS by B. Jonsson & S. Romano.

M.C. simulation of a polyatomic solute molecule in an aqueous cluster. (i.e. a molecule surrounded by water molecules). The water - water potential is calculated using an analytical fit to an ab initio potential energy surface due to Matsuoka et al. The solute-solvent potential is optional. The program provides an energy and coordinate 'history' of the M.C. simulation. An analysis program CARLAN for processing the data produced by CARLOS is also available.

MCN by N. Corbin.

M.C. simulation of atomic fluids. Standard (Metropolis) Monte Carlo program for atomic fluids.

SCN by N. Corbin.

M.C. simulation of atomic fluids. Standard (Rossky,Friedman and Doll) Monte Carlo program for atomic fluids.

SMF by N. Corbin.

M.C. simulation of atomic fluids. Standard (path integral method) Monte Carlo program for atomic fluids.

STATIC SIMULATION CODES

CASCADE by M. Leslie and W. Smith.

Calculates the structure and energy of a defect in an ionic crystal for a given potential model. Both two- and three-body potentials may be used. The properties of the perfect lattice are calculated

as well by default. Use is made of symmetry only in the defect calculation and only for two-body potentials. A second derivative method using a Hessian update algorithm is used to minimise the defect energy. The program runs on the Cray, AS7000 and FPS 164.

THBREL

Determines the minimum energy configuration of a perfect lattice for a given potential. Both two- and three-body potentials may be used. Relaxation to constant volume or zero bulk strain is possible. No use is made of symmetry to speed up the calculation.

THBFIT

Empirically fits a potential to experimentally observed properties of a perfect lattice. (Structure, elastic constants, dielectric constants.) Both two- and three-body potentials may be fitted.

SYMLAT

Determines the minimum energy configuration of a perfect lattice for a given potential. Only two-body potentials may be used. The program makes full use of symmetry to reduce time and memory requirements. The program runs on the Cray, AS7000 and FPS 164.

THBPHON

Calculates phonon dispersion curves for ionic crystals with three-body terms in the potential.

IMPLEMENTATION OF THE STATIC SIMULATION PROGRAMS ON THE FPS-164

This report is not intended to be a complete guide to the use of the FPS. It describes some of the work done on the static simulation programs to implement them on the FPS. Firstly, an introduction to the FPS hardware and software is given for those readers unfamiliar with it. Next, preliminary work on the CRAY FORTRAN is given. This includes some examples of the use of MATHLIB routines in order to speed up the execution of certain DO loops. There are then sections on matrix inversion, calculation of square roots, use of multi-dimensional arrays and the use of gather/scatter techniques.

INTRODUCTION TO THE FPS HARDWARE AND SOFTWARE

The FPS 164 is a parallel pipelined computer. The parallel architecture is used by the instruction word, which is 64 bits long (cf. 16 bits on the CRAY) and is divided into 10 subsections or parcels which can in principle drive up to 10 independent function units on each instruction. The adder and multiplier units are both hardware pipelines. The adder is a two stage pipeline, that is two numbers need to be pushed through the adder for one instruction and the result will be available the second instruction after the add instruction. The multiplier unit is a three stage pipeline. Fetches and writes to main memory also take three clock cycles, although the fetches and writes are pushed automatically by the CPU clock rather than needing an instruction to do this. Like the CRAY, the memory is divided into banks with consecutive memory addresses in different memory banks. Attempting to reference the same memory bank on consecutive instructions will cause all operations of the FPS to be suspended for one cycle until the first memory fetch is completed. The multi-operation instruction word exploits local parallelism within a loop to ensure that several processes are occuring simultaneously. In addition, global parallelism between computations on different data sets can be exploited in a software pipeline. In this the computation is divided into a number of stages, say for example 2. The first stage will contain the first half of the instructions of the loop and the second stage the second half. Provided the operations (parcels in the instruction word) for an instruction of the first stage of the pipeline are different from the corresponding instructions of the second stage, a single instruction can be written to carry out both parcels simultaneously (but on different data sets) exploiting the parallel architecture of the computer. Thus it may be seen that a pipelinable loop will differ in a number of ways from a vectorisable loop on the CRAY. Non-linear addressing may be pipelined without difficulty. Also recursion may be pipelined provided the memory reference which may be recursive is written to main memory and read from main memory in the same stage of the pipeline.

In practice the FPS has a large library of subroutines which have been pipelined. These include simple FORTRAN external functions such as SQRT, SIN, COS, EXP. Consideration of the use of these is the easiest way to speed up the execution of a program on the FPS. The FORTRAN compiler will also attempt to pipeline loops at the compiler optimisation level OPT(3).

General experience has been that a large number of loops which satisfy all the criteria for being pipelinable in fact are not pipelined by the FORTRAN compiler. This does not mean that the loop is failing to fully exploit the the potential of the FPS. Usually such loops have a large amount of local parallelism which the compiler is exploiting to generate overlapped code at OPT(2) so there is little or no further gain in pipelining the loop.

A few further comments are in order concerning FORTRAN pipelined DO loops. The compiler will generate startup stages to progressively fill the initial pipeline stages before the loop. However, no compiler generated shutdown stages are produced, where just the final stages of the pipeline are run. In the above example of a 2 stage pipeline, this means that the first stage is executed N+1 times, where N is the loop count. This may result in floating point or other arithmetic exceptions. Hence the arithmetic error detection must be switched off when the compiler optimisation option OPT(3) is used. Also, all writes to main memory will be placed by the compiler in the final stage of the pipeline so that main memory does not become corrupt by executing the first stage an extra time. Sometimes reconsideration of where main memory writes occur in a DO loop can result in improvement in pipelining. (For example, switch to using scalar temporaries instead of elements in COMMON blocks for intermediate results.)

INITIAL STAGES OF PROGRAM DEVELOPMENT

The following static simulation programs have so far been implemented on the FPS-164.

CASCADE Calculates ionic crystal defect energies.

THBFIT Preliminary program for fitting empirical potentials to known crystal properties.

THBREL Program for simulating perfect lattices including three-body forces.

SYMLAT Another perfect lattice simulation program with no three-body forces but with full symmetry adaption.

All of the CRAY versions were initially FORTRAN-4. Although not essential, it was felt desirable to update the programs to be fully FORTRAN 77 compatible. This involved the following:

(i) Variable names used to store Hollerith constants were defined to be type CHARACTER.

(ii) Hollerith constants were converted to CHARACTER constants.

(iii) Common blocks containing mixed CHARACTER/REAL data were split into two.

(iv) SAVE statements were introduced where appropriate.

All of the programs are compiled at OPT(3). There is one subroutine in SYMLAT which contains a DO loop for which the compiler generates incorrect code at OPT(3), this subroutine was compiled at OPT(2). It proved very easy to track down the subroutine causing the problem by comparing OPT(3) with OPT(2) results. Progressively smaller sections of the program were compiled at OPT(2), linking in the rest of the OPT(3) subroutines using the librarian. For those who may be interested, the DO LOOP causing the

problem was very short and is given below.

```
      DO 86 JINEB=IINEB,INEB
      JSYMT=NSYMT(JINEB)
      ISUM=ISUM+NNUMA(ISYMT,JSYMT)*6
   86 CONTINUE
```

The compiler was generating one pipeline push too many on the *6 instruction resulting in a zero result from this product.

All of the further optimisation work was done exclusively on CASCADE, since the other programs were essentially peripheral to it. The next step was to replace CRAY library calls with FPS library calls. Then certain key DO loops were examined and MATHLIB calls used if appropriate. Two examples are given below

Loop as run on the CRAY-1S

```
      DO 82 I=1,ICC
      RA(I,1)=XC(I,1)-XC(I,4)
      RA(I,2)=XC(I,2)-XC(I,5)
      RA(I,3)=XC(I,3)-XC(I,6)
      RA(I,4)=XC(I,10)-XC(I,1)
      RA(I,5)=XC(I,11)-XC(I,2)
      RA(I,6)=XC(I,12)-XC(I,3)
      RA(I,7)=XC(I,7)-XC(I,4)
      RA(I,8)=XC(I,8)-XC(I,5)
      RA(I,9)=XC(I,9)-XC(I,6)
      RA(I,11)=RA(I,1)*RA(I,1)+RA(I,2)*RA(I,2)+RA(I,3)*RA(I,3)
      RA(I,13)=RA(I,4)*RA(I,4)+RA(I,5)*RA(I,5)+RA(I,6)*RA(I,6)
      RA(I,12)=RA(I,7)*RA(I,7)+RA(I,8)*RA(I,8)+RA(I,9)*RA(I,9)
      RA(I,15)=SQRT(RA(I,11))
      RA(I,16)=SQRT(RA(I,12))
      RA(I,17)=SQRT(RA(I,13))
   82 CONTINUE
```

Loop as rewritten to use MATHLIB calls for the FPS

```
      DO 82 I=1,ICC
      RA(I,1)=XC(I,1)-XC(I,4)
      RA(I,2)=XC(I,2)-XC(I,5)
      RA(I,3)=XC(I,3)-XC(I,6)
      RA(I,4)=XC(I,10)-XC(I,1)
      RA(I,5)=XC(I,11)-XC(I,2)
      RA(I,6)=XC(I,12)-XC(I,3)
      RA(I,7)=XC(I,7)-XC(I,4)
      RA(I,8)=XC(I,8)-XC(I,5)
      RA(I,9)=XC(I,9)-XC(I,6)
      RA(I,11)=RA(I,1)*RA(I,1)+RA(I,2)*RA(I,2)+RA(I,3)*RA(I,3)
      RA(I,13)=RA(I,4)*RA(I,4)+RA(I,5)*RA(I,5)+RA(I,6)*RA(I,6)
      RA(I,12)=RA(I,7)*RA(I,7)+RA(I,8)*RA(I,8)+RA(I,9)*RA(I,9)
   82 CONTINUE
      CALL VSQRT(RA(1,11),1,RA(1,15),1,ICC)
      CALL VSQRT(RA(1,12),1,RA(1,16),1,ICC)
      CALL VSQRT(RA(1,13),1,RA(1,17),1,ICC)
```

The rewritten loop was a factor of 2 faster than the loop containing SQRT calls. The second example uses the MATHLIB calls VPOLY and VEXP to evaluate a vector of polynomials and exponentials respectively.

Loop as run on the CRAY-1S.

```
      DO 350 I=1,IERF
      SCT1=RA(I,5)*HFCT0
      SCT2=SCT1*SCT1
      SCT3=EXP(-SCT2)
      SCT4=       (((((((P(1)*SCT1+P(2))*SCT1+P(3))*SCT1+P(4))*SC
     1T1+P(5))*SCT1+P(6))*SCT1+P(7))*SCT1+P(8))/(((((((Q(1)*SCT1+Q(2))*
     2SCT1+Q(3))*SCT1+Q(4))*SCT1+Q(5))*SCT1+Q(6))*SCT1+Q(7))*SCT1+Q(8))*
     3SCT1+Q(9))*SCT3)*CHGPRD*RA(I,1)/SCT1
      SCT3=SCT3*FACTOR*CHGPRD*RA(I,1)
      SCT1=(-SCT4-SCT3)*HFCT1/SCT2
      RA(I,1)=SCT4*HFCT0
      RA(I,2)=RA(I,2)*SCT1
      RA(I,3)=RA(I,3)*SCT1
      RA(I,4)=RA(I,4)*SCT1
  350 CONTINUE
```

Loop as rewritten for the FPS

```
      DO 351 I=1,IERF
      VTMP(I,1)=RA(I,5)*HFCT0
      VTMP(I,2)=-VTMP(I,1)*VTMP(I,1)
  351 CONTINUE
      CALL VEXP(VTMP(1,2),1,VTMP(1,3),1,IERF)
      CALL VPOLY(P,1,VTMP(1,1),1,VTMP(1,4),1,IERF,7)
      CALL VPOLY(Q,1,VTMP(1,1),1,VTMP(1,5),1,IERF,8)
      DO 350 I=1,IERF
      SCT4=(VTMP(I,4)/VTMP(I,5))*VTMP(I,3)*CHGPRD*RA(I,1)
     1 /VTMP(I,1)
      SCT3=VTMP(I,3)*FACTOR*CHGPRD*RA(I,1)
      SCT5=(SCT3+SCT4)*HFCT1/VTMP(I,2)
      RA(I,1)=SCT4*HFCT0
      RA(I,2)=RA(I,2)*SCT5
      RA(I,3)=RA(I,3)*SCT5
      RA(I,4)=RA(I,4)*SCT5
  350 CONTINUE
```

In this case the gain in speed was a factor of 2.2. These two examples show how simple changes to DO loops can significantly enhance the execution speed of the program.

MATRIX INVERSION

CASCADE uses a Hessian updating algorithm and therefore only needs to invert a matrix once for each defect considered. This means that the matrix inversion step is never the step requiring the most cpu time, despite the fact that the matrix inversion time increases as N**3 while the

matrix setup time and gradient vector calculation time increase as N**2. It is therefore more important for CASCADE to ensure that the matrix setup is well optimised than to have a very fast matrix inverter. However, a number of different matrix inverters have been used and compared with the CRAY library routine MINV. The timings given in the table below are for a 213 * 213 matrix.

COMPUTER	INVERSION ROUTINE	TIME S	FPS/CRAY RATIO
CRAY	MINV	0.3344	
FPS	FORTRAN	18.86	56.4
	MATINV	12.74	38.1
	PFINV	4.864	14.5

MATINV and PFINV are both FPS MATHLIB routines. They both require 2*N*N words of memory, whereas MINV and the FORTRAN used both require N*(N+2). At present the program selects either PFINV or the FORTRAN according to whether there is sufficient main memory for PFINV. A number of cases have been encountered for very large matrices where MINV on the CRAY fails with a FLOATING POINT ERROR, but PFINV successfully inverts the matrix. The FPS also has a second library, FMSLIB, which is primarily intended for use with very large matrices as it stores the matrices on disc and handles all of the disc I/O without any intervention by the user. The timings given in the table below are for a 500*500 matrix. This shows that for such small matrices it is not worth implementing into the program.

ROUTINE	CPU TIME	I/O TIME
FMSLIB	41.67	112.21
PFINV	62.17	0.0

THE CALCULATION OF SQUARE ROOTS

A square root may be calculated by two approaches using the Newton-Raphson method. The first is to obtain an initial estimate y(1) = x**0.5 and then to obtain successively better approximations by means of

```
y(i+1) = 0.5 * y(i) + 0.5 * x / y(i)
```

This method was suggested by K. Singer (CCP5 Info. Quart. March 1983 P 47) and revised by J. Powles (CCP5 Info. Quart. Jan 1984 P 39) using a third order polynomial to estimate x ** 0.5 in the range 0.1 to 1.0 and two iterations of the Newton-Raphson method. The disadvantage of this method is that divide operations needed for the Newton-Raphson iteration will be expensive. The alternative approach is to estimate z(1) = 1.0 / x**0.5. The Newton-Raphson iterations then become

```
z(i+1) = 0.5 * z(i) * (3.0 - z(i)**2 * x)
```

Finally y = x**0.5 is obtained by a multiplication y = z * x This method does not need any divide operations to carry out the Newton-Raphson iteration. This method is used by the FPS-164 with the initial estimates stored in read only memory and three iterations needed for full machine precision. This requires 10 multiplication operations as the rate-limiting resource, hence the MATHLIB library routine VSQRT will calculate the square root of a vector at a rate of 1 result per 10 clock cycles. (Clock cycle time is 182 * 10 ** -9 s on FPS 164). The table below gives comparison timings for the two methods on the FPS, times are per result for a large vector. (Times are in microseconds)

METHOD	TIME
FORTRAN DO LOOP CALLING SQRT	7.30
FPS MATHLIB CALL OF VSQRT	1.82
SINGER METHOD CODED IN FORTRAN	14.24
SINGER METHOD CODED USING MATHLIB CALLS	5.07

For the static defect programs, also for a number of MD programs, it would be more useful to have the reciprocal square root in any case. This would avoid taking a reciprocal. On the FPS, the MATHLIB routine VRECIP, which calculates the reciprocal of a vector, takes 6 clock cycles per result. This gives a total of 16 per result for calculating 1.0 / x**0.5. Using FPS assembler it is possible to use the normal square root method without the final multiplication to work out the reciprocal square root directly. The number of multiplications needed is still the rate-limiting resource, so the time will be 9 clock cycles per result. An assembler routine to do this has been written and implemented into CASCADE.

USE OF MULTI-DIMENSIONAL ARRAYS

While in general the use of multi-dimension arrays does not significantly slow down the execution of a program on the FPS, in one case it was found that using single dimension arrays is significantly faster than multi-dimension arrays. This occurs when elements are being summed to different colomns of an array in the same DO loop, and the first dimension of the array is adjustable dimension. The example below should make this clearer. Loop 1 is pipelinable but is not pipelined by the compiler at optimisation level OPT(3). The loop is 59 instructions long, but may require slightly more than this to execute if there are any memory bank conflicts. Loop 2 is apparently very similar, but this loop is pipelined by the compiler and generates a loop 24 instructions long, a gain in speed of a factor of over 2.

LOOP 1

```
      SUBROUTINE DIS(GXX,NX)
      COMMON/CDIST/RA(512,17),INC(512,2),WGT(512),CHGPRD,V0,ICC
      DIMENSION GXX(NX,*)
      DO 1243 I=1,ICC
      SCT1=RA(I,15)
```

```
      V0TT=CHGPRD*WGT(I)*SCT1
      SCT2=-V0TT*SCT1*SCT1
      V1TT1=SCT2*RA(I,1)
      V1TT2=SCT2*RA(I,2)
      V1TT3=SCT2*RA(I,3)
      V0=V0+V0TT
      GXX(INC(I,1),1)=GXX(INC(I,1),1)+V1TT1
      GXX(INC(I,1),2)=GXX(INC(I,1),2)+V1TT2
      GXX(INC(I,1),3)=GXX(INC(I,1),3)+V1TT3
      GXX(INC(I,2),1)=GXX(INC(I,2),1)-V1TT1
      GXX(INC(I,2),2)=GXX(INC(I,2),2)-V1TT2
      GXX(INC(I,2),3)=GXX(INC(I,2),3)-V1TT3
 1243 CONTINUE
      RETURN
      END
```

LOOP 2

```
      SUBROUTINE DIS(GXX,GXY,GXZ)
      COMMON/CDIST/RA(512,17),INC(512,2),WGT(512),CHGPRD,V0,ICC
      DIMENSION GXX(*),GXY(*),GXZ(*)
      DO 1243 I=1,ICC
      SCT1=RA(I,15)
      V0TT=CHGPRD*WGT(I)*SCT1
      SCT2=-V0TT*SCT1*SCT1
      V1TT1=SCT2*RA(I,1)
      V1TT2=SCT2*RA(I,2)
      V1TT3=SCT2*RA(I,3)
      V0=V0+V0TT
      GXX(INC(I,1))=GXX(INC(I,1))+V1TT1
      GXY(INC(I,1))=GXY(INC(I,1))+V1TT2
      GXZ(INC(I,1))=GXZ(INC(I,1))+V1TT3
      GXX(INC(I,2))=GXX(INC(I,2))-V1TT1
      GXY(INC(I,2))=GXY(INC(I,2))-V1TT2
      GXZ(INC(I,2))=GXZ(INC(I,2))-V1TT3
 1243 CONTINUE
      RETURN
      END
```

If instead the array GXX in example 1 is dimensioned the other way round, the loop is 29 instructions long. On the CRAY, there is no significant difference between the three loops.

USE OF GATHER/SCATTER LOOPS

This is again best illustrated by an example. Loop A shows FORTRAN for a sequential computer. This is representative of the sort of calculation that CASCADE carries out, where a calculation is only performed if a symmetry test is TRUE. However, the actual calculation performed is far more complex than this simple example. There are two approaches to vectorise this. First, a vector merge may be used. Loop B gives the CRAY FORTRAN for the vector merge. There is no equivalent to this on the FPS

but the vector merge may be simulated in FORTRAN. (Replacing SQRT and 1.0/ by MATHLIB vector equivalents VSQRT and VRECIP). The table gives the timings for execution for M=100 and for the IF condition TRUE for 33% of the time. The vector merge is only marginally faster on the CRAY in this case. As the loop structure becomes more complex, vector merge will become less and less favourable. On the FPS the FORTRAN equivalent of vector merge is slower than the sequential FORTRAN. Loops C and D give two ways to carry out the gather/scatter. Loop C gathers 6 elements on which the calculation will be performed and then has a DO loop with no non-linear addressing in. Loop D stores two indices and then has a DO loop with non-linear addressing in, which is still pipelinable on the FPS. The CRAY FORTRAN has again been given, on the FPS the MATHLIB calls VSQRT and VRECIP were used instead. As the table shows there is a considerable advantage to gather the indices (Loop D) on the FPS. Sections of FORTRAN which just carry out memory references, as a GATHER is doing, will not exploit the full potential of the FPS parallel architecture. It is better if other functional units of the FPS can be active at the same time that the GATHER is being performed. On the CRAY-1S, both C and D are vectorisable loops but as the timings show D is significantly slower in this case. Although the gain in timing in this example is not significant, further gains would be expected if the loop structure were more complex. On the CRAY, implementation of gather/scatter increased the speed of the matrix setup by a factor of 5.

LOOP A

```
      DO 81 MAMC=1,M-1
      DO 80 MAMD=MAMC+1,M
      INDEX=IEQ(ICLS(MAMC),ICLS(MAMD))
      N=NUM(INDEX)
      IF(N.GT.0)THEN
      RA(1)=XCRD(1,MAMC)-XCRD(1,MAMD)
      RA(2)=XCRD(2,MAMC)-XCRD(2,MAMD)
      RA(3)=XCRD(3,MAMC)-XCRD(3,MAMD)
      RA(11)=RA(1)*RA(1)+RA(2)*RA(2)+RA(3)*RA(3)
      RA(15)=SQRT(RA(11))
      E=E+1.0E+0/RA(15)
      ENDIF
   80 CONTINUE
   81 CONTINUE
```

LOOP B

```
      DO 81 MAMC=1,M-1
      DO 83 MAMD=MAMC+1,M
      INC(MAMD,1)=NUM(IEQ(ICLS(MAMC),ICLS(MAMD)))
   83 CONTINUE
      DO 80 MAMD=MAMC+1,M
      RA(MAMD,1)=XCRD(1,MAMC)-XCRD(1,MAMD)
      RA(MAMD,2)=XCRD(2,MAMC)-XCRD(2,MAMD)
      RA(MAMD,3)=XCRD(3,MAMC)-XCRD(3,MAMD)
      RA(MAMD,11)=RA(MAMD,1)*RA(MAMD,1)+RA(MAMD,2)*RA(MAMD,2)+RA(MAMD,3)
     1 *RA(MAMD,3)
```

```
      RA(MAMD,15)=SQRT(RA(MAMD,11))
      RA(MAMD,17)=1.0E+0/RA(MAMD,15)
      RA(MAMD,17)=CVMGT(RA(MAMD,17),0.0E+0,INC(MAMD,1).GT.0)
      E=E+RA(MAMD,17)
   80 CONTINUE
   81 CONTINUE

LOOP C

      DO 81 MAMC=1,M-1
      DO 80 MAMD=MAMC+1,M
      INDEX=IEQ(ICLS(MAMC),ICLS(MAMD))
      N=NUM(INDEX)
      IF(N.GT.0)THEN
      ICC=ICC+1
      XC(ICC,1)=XCRD(1,MAMC)
      XC(ICC,2)=XCRD(2,MAMC)
      XC(ICC,3)=XCRD(3,MAMC)
      XC(ICC,4)=XCRD(1,MAMD)
      XC(ICC,5)=XCRD(2,MAMD)
      XC(ICC,6)=XCRD(3,MAMD)
      IF(ICC.EQ.64)THEN
      DO 82 I=1,ICC
      RA(I,1)=XC(I,1)-XC(I,4)
      RA(I,2)=XC(I,2)-XC(I,5)
      RA(I,3)=XC(I,3)-XC(I,6)
      RA(I,11)=RA(I,1)*RA(I,1)+RA(I,2)*RA(I,2)+RA(I,3)*RA(I,3)
      RA(I,15)=SQRT(RA(I,11))
      RA(I,17)=1.0E+0/RA(I,15)
      E=E+RA(I,17)
   82 CONTINUE
      ICC=0
      ENDIF
      ENDIF
   80 CONTINUE
   81 CONTINUE
      IF(ICC.NE.0)THEN
      DO 83 I=1,ICC
      RA(I,1)=XC(I,1)-XC(I,4)
      RA(I,2)=XC(I,2)-XC(I,5)
      RA(I,3)=XC(I,3)-XC(I,6)
      RA(I,11)=RA(I,1)*RA(I,1)+RA(I,2)*RA(I,2)+RA(I,3)*RA(I,3)
      RA(I,15)=SQRT(RA(I,11))
      RA(I,17)=1.0E+0/RA(I,15)
      E=E+RA(I,17)
   83 CONTINUE
      ICC=0
      ENDIF

LOOP D

      DO 81 MAMC=1,M-1
      DO 80 MAMD=MAMC+1,M
```

```
      INDEX=IEQ(ICLS(MAMC),ICLS(MAMD))
      N=NUM(INDEX)
      IF(N.GT.0)THEN
      ICC=ICC+1
      INC(ICC,1)=MAMC
      INC(ICC,2)=MAMD
      IF(ICC.EQ.64)THEN
      DO 82 I=1,ICC
      NAMC=INC(I,1)
      NAMD=INC(I,2)
      RA(I,1)=XCRD(1,NAMC)-XCRD(1,NAMD)
      RA(I,2)=XCRD(2,NAMC)-XCRD(2,NAMD)
      RA(I,3)=XCRD(3,NAMC)-XCRD(3,NAMD)
      RA(I,11)=RA(I,1)*RA(I,1)+RA(I,2)*RA(I,2)+RA(I,3)*RA(I,3)
      RA(I,15)=SQRT(RA(I,11))
      RA(I,17)=1.0E+0/RA(I,15)
      E=E+RA(I,17)
   82 CONTINUE
      ICC=0
      ENDIF
      ENDIF
   80 CONTINUE
   81 CONTINUE
      IF(ICC.NE.0)THEN
      DO 83 I=1,ICC
      NAMC=INC(I,1)
      NAMD=INC(I,2)
      RA(I,1)=XCRD(1,NAMC)-XCRD(1,NAMD)
      RA(I,2)=XCRD(2,NAMC)-XCRD(2,NAMD)
      RA(I,3)=XCRD(3,NAMC)-XCRD(3,NAMD)
      RA(I,11)=RA(I,1)*RA(I,1)+RA(I,2)*RA(I,2)+RA(I,3)*RA(I,3)
      RA(I,15)=SQRT(RA(I,11))
      RA(I,17)=1.0E+0/RA(I,15)
      E=E+RA(I,17)
   83 CONTINUE
      ICC=0
      ENDIF
```

TIMINGS IN SECONDS FOR THE EXAMPLES

	FPS	CRAY
LOOP A	0.0533	0.00927
LOOP B	0.0635	0.00607
LOOP C	0.0494	0.00735
LOOP D	0.0403	0.00957

THE FPS 164, A USER'S TALE

W. Smith

Most of our readers will be aware of the existence of the FPS 164 Attached Processor at Daresbury and that it is available to the U.K. academic community on a trail basis, prior to a grant application, for the purposes of testing its suitability for computational projects. For this reason I thought it would be informative to recount my own experiences of using it, to give prospective users some idea of what is entailed. (I should perhaps apologise to American readers if the content of this note is "old hat", or worse, boring, but the use of FPS computers in the U.K. is not at all as commonplace as in the U.S.A.).

The processor has at first sight some significant attractions for a would-be simulator. Firstly, it is a single-user dedicated processor with a memory of 1.5 Mword. When a given program is executing, all the user accessible memory is available to it. This means that the user may contemplate simulating very large systems, of the order of ten thousand particles using a link cells program. Such freedom is an unlikely occurrence on a multi-user mainframe. Secondly, its capacity to process vector operations at optimal efficiency (through the facility known as pipelining) suggests that MD simulations would be particularly suited. Thirdly, the FPS 164 is provided with an extensive range of software (MATHLIB) which is likely to be of great assistance.

To evaluate the FPS 164 for MD work I wrote a simple atomic Lennard Jones MD program MDTEST, using the simplest computational strategy possible; that is, a cubic simulation cell with the maximum permissible potential cut-off; pair interactions outside this cut-off were simply neglected and no list procedures were used to render the operation more efficient. I chose this strategy because my exprience with the Cray 1s led me to think that this would be the easiest to pipeline and give the greatest efficiency for least effort. My intention was to run the program on the FPS 164, the NAS AS7000 (at Daresbury) and the Cray 1s (at London) to get some idea as to the relative power of the FPS. The benchmark simulation consisted of a 108 particle system run for 5000 time steps. The first 1000 steps constituted the equilibration period (during which no data were accumulated). The production period lasted for 4000 timesteps, during which an RDF was accumulated at 20 time step intervals.

On the Cray 1s the program MDTEST was fully vectorised, and the test simulation took 45.78 seconds. On the NAS AS7000 the same test took 487.18 seconds. On the FPS 164 an un-pipelined MDTEST (i.e one containing a branching IF statement within the forces DO loop) took 423.7 seconds. Thus it appeared that in this test the FPS was slightly more powerful than the NAS (which, incidentally has roughly the power of an IBM 370). This however was only the first step in producing an optimised code.

I then attempted to pipeline the forces DO loop using logical shift function designed to mimic the Cray vector merge routine CVMGP. The function of this routine is to set the interactions that are outside the range of the cut-off to zero without employing a branching IF statement (which would disrupt the vectorisation). The price paid for this vectorisation is the additional work that is performed in the DO loop, which in this application amounts to an approximate doubling of the computational effort. The corresponding changes in the FPS version of MDTEST were however to avail; the test simulation ran in a time of 488.9 seconds!

The inference drawn from this result is that the FPS pipelining strategy does not carry the relative power that vectorisation does on the Cray 1s, and the additional computation that results from its use in this case is not offset by the increased processing power. But was there an increase in computational power resulting from the pipelining? I attempted to check this in the following simple way. Taking the un-pipelined version of MDTEST I removed the cut-off from the forces calculation (i.e. removing the branching IF statement) and ran a simulation without the pipelining option in force. The simulation, which (theoretically) has about the same computational labour as the pipelined case, took 695.5 seconds. I concluded from this that pipelining did provide a significant improvement in the power of the FPS. The problem then was to find out how to exploit it.

A number of different strategies came to mind fairly quickly. To begin with I though of using a modification of the link-cells [1] or neighbour list [2] methods, which though not immediately thought of as "vectorisable" can be made so with some effort. These are essentially FORTRAN solutions of the problem, another option (and one which I shrank away from!) was to delve into the use of FPS assembler coding (APAL). By talking with other FPS users and reading around I came to suspect that a FORTRAN solution to my problem was not going to be easy. The link-cells and neighbour-list methods require the use of SCATTER and GATHER [3] facilities to offer realitic possibilities. The FPS did not seem to offer much in this direction and an APAL solution looked like being the best bet. At this stage however fate took a hand.

Some years ago I had made friends with a quantum chemist by the name of Aatto Laaksonen (currently at the Arrhenius Laboratory, Stockholm), who was a short-term visitor to Daresbury. While he was here he told me of his interest in molecular dynamics and was about to join Professor Clemmenti's group in New York to learn about simulation methods. Being the custodian of the CCP5 Program Library at the time I naturally let him depart with a copy of the CCP5 Program Library. Years later, recalling the favour done to him by CCP5 he wrote to me offering to provide some MD programs of his own, and being familiar with the architecture of the FPS would be pleased to show me how to get the best out of it! As a result of this he visited Daresbury in October and in a couple of days showed me how his programs worked.

The essential feature of the method described to me by Aatto is that the forces are not calculated, as they usually are, in a double Do loop over the particles in the system. Instead the method loops over the pairs of interacting sites in a single large loop. This, at least, is the principle, because a little thought reveals that it is necessary to store a very large number of terms if the full (N*(N-1))/2 pairs are to be considered. In practice one attempts to calculate as many of the pairs as the memory available will allow, and repeat the procedure until all the pairs have been considered. (One begins to see how the large memory feature of the FPS is being expoited to the full here). Of course this method requires some careful book-keeping if the detailed interactions of the whole system are to be correctly accounted for. This is not the whole story however, because having set up the problem in this way one still needs to ensure that the new method will pipeline. Here (as I suspected) it was necessary to revert to APAL coding and Aatto supplied a number of auxiliary APAL routines that perform the pipelining and book-keeping. The author of these routines is Roland Sonnenschein, formerly of the Max Planck Institute, Mainz and IBM Kingston, New York, and former colleague of Aatto. (These APAL routines, which are applicable to polyatomic molecules as well as to the simple Lennard-Jones atoms used here, will shortly be available as an FPS utility package through the CCP5 Program Library. The program MDTEST is also available, under the name of MDAFPS, in the CCP5 Program Library).

To try out this method Aatto and I made the necessary modifications to MDTEST and performed the standard test run. The job took 257.5 seconds. Clearly, the new approach was considerably better than my own!

So what can be learned from this tale? It would appear that the way ahead in the use of the FPS 164 is to be prepared to adopt new strategies and if necessary to delve into the use of APAL coding. As yet, I see no simple route to the full processing power of the FPS using FORTRAN alone (but perhaps our American colleagues have better ideas ?). Past experience using the Cray 1s did not really offer any useful clues in this regard and one is reminded of the re-thinking that is necessary in using other attached processors such as the ICL DAP, though in that case one expects to have to learn new methods because of its unique architecture. Finally, I am reminded once again that a few moments in the company of a wise man is worth a lifetime of searching. How helpful it is to discuss matters with someone knowledgable! Is this not why CCP5 exists?

Acknowledgements

As I hope this article makes clear I am extremely indebted to Aatto Laaksonen for his generous help and to Roland Sonnenschein for his APAL software.

References

[1] W. Smith, CCP5 Info. Quart. No.20 p.52 (1986).

[2] D.M. Heyes, CCP5 Info. Quart. No.2 p.11 (1981), also D. Adams, CCP5 Info. Quart. No.3 p.32 (1981), also S.M. Thompson, CCP5 Info. Quart. No.8 p.20 (1983) and S.F. O'Shea, CCP5 Info. Quart. No.9 p.41 (1983).

[3] D. Fincham and B. Ralston Comp. Phys. Comm. V23 p127 (1981).

MOLECULAR DYNAMICS SIMULATION OF HARD MOLECULES

M. P. Allen and D. Frenkel

In an earlier issue of this newsletter, David Heyes described the basic techniques used in computing the molecular dynamics of hard spheres [1]. In this article, we shall set out the program structure and give technical details for the simulation of hard non-spherical molecules. We have been using this approach successfully now for more than a year in simulations of the hard-ellipsoid and hard-spherocylinder systems, both of which we regard as benchmarks for hard-core molecular liquids in general, and which in addition form liquid crystalline phases. The program runs efficiently on a vector-processing supercomputer with scatter-gather capability , and there is every indication that more complex hard-particle systems can be simulated in the same way.

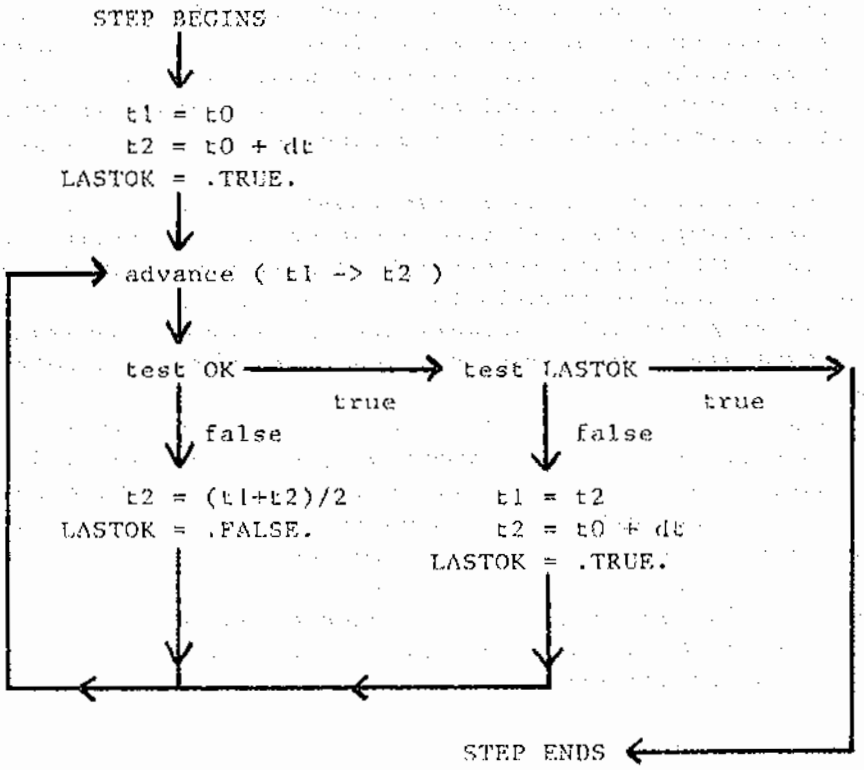
The usual procedure for hard sphere simulations is to solve the collision equations for each possible colliding pair of atoms, locating the earliest collision time, and then advancing the system using free-flight Newtonian dynamics to the point of collision. The collision dynamics are then implemented for the colliding pair, and a search started for the next collision. The process is quite efficient, since translational motion is rectilinear and the collision equation is quadratic in the time and hence easily solved. List structures, similar to those used for continuous potentials, can be used to speed the program up. Although collision-by-collision dynamics does not look as though it would vectorize easily, for large systems most of the time is spent searching pairs for solutions of the collision equation, and so it is still worthwhile to use a supercomputer for this type of work.

The situation for hard non-spherical particles, though, is more forbidding. The collision equation becomes highly nonlinear in the time, both because free-flight dynamics now involves rotation about, as well as translation of, the centre of mass, and because the colliding particle surfaces are in general non-spherical. To our knowledge, only one or two simulations involving direct solution of the collision equations have been carried out. Bellemans and coworkers [2] conducted some preliminary simulations of hard dumb-bells in two dimensions, while Frenkel and Maguire [3] simulated the hard line segment model of rod-like molecules. Both these pieces of work have been extended recently [4]. However, the programming effort involved, and the expense of the simulations, are significant deterrents. A simpler approach has been adopted by Rapaport [5] and developed by Chapela and coworkers [6] (see also [2]). The molecules in this approach are built out of hard-sphere units, and are not rigid.

However, a fairly efficient alternative approach to rigid body dynamics exists, and has been described by Rebertus and Sando [7] and by McNeill and Madden [8]. Here, the molecular configuration is advanced on a regular step-by-step basis just as in standard

molecular dynamics, using free-flight without collision. However, at the end of each step, the configuration is examined for overlapping pairs of molecules. For each overlapping pair, the collision equation is solved retrospectively to find the time at which collision should have occurred. The system is rewound to the point of collision, collision dynamics implemented, and the system run forward again to the end of the step. The expensive part of this, the search for overlaps, is readily vectorized or parallelized. The complicated (although not very time-consuming) part consists of handling multiple overlaps and resolving sequences of (possibly) inter-dependent collisions occurring within a very short time of each other.

The following flow diagram shows the way in which our main program handles this problem. This section of the program is enclosed in a loop over timesteps, and it advances the configuration from time t0 to time t0+dt incorporating all collisions correctly (apart from a very small number of grazing collisions which are not detected). The advance routine attempts to advance the configuration from an initial time t1 to a final time t2 incorporating all collisions correctly. If this is accomplished without difficulty, it returns as .TRUE. the logical flag OK; otherwise OK is set .FALSE. The main routine keeps track of OK and the flag LASTOK, which records whether or not the last attempt at advance was successful.



By coding the main program in this way, we can allow the advance routine to be quite simple, catering for the overwhelming majority of cases (for a suitably-chosen interval dt) in which a single collision or a small number of non-interfering collisions occurs during a time step. Whenever advance is unable to resolve the collisions unambiguously, it sets OK to be .FALSE. and returns with the configuration unmoved at time t1. Following this, the program simply halves the timestep, records the fact that a failure occurred, and tries again. When an advance is successful, the program tries to cover the remainder of the time step in a single go, and the process continues. A time step is completed when two successive advances produce an OK result, or when the very first advance covered the whole step in one.

Within the advance routine the procedure is as follows. The initial configuration at time t1 is stored, and all molecules are moved forward to the new time t2. A search for overlapping pairs is made, and a list of these constructed. If there are any overlaps, the collision equation is solved for each such pair. Specifically, we look for roots of an equation f(i,j;t)=0 where the overlap function f is positive if i and j do not overlap and negative if they do. In the case of ellipsoids, suitable prescriptions have been provided by Vieillard-Baron [9] and by Perram and Wertheim [10]. We do this by a standard Newton-Raphson procedure, which requires the time derivative of f as well as the function itself. The pair collisions are sorted into chronological order. Any secondary collisions (collisions involving atoms i and j, either of which have been involved in a collision appearing earlier in the list) are removed. The collisions are then examined in order of occurrence. For each collision, the colliding pair i and j are moved back to the point of collision, allowed to collide, and moved forward again to time t2. They are then examined to see if, after this, they overlap with any other molecules. If such an overlap is detected then the step is immediately abandoned: the original configuration (at time t1) is restored, the flag OK set to .FALSE. and the routine returns. The reason for this quick rejection is explained below. If no overlap is detected, the routine goes on to consider the next colliding pair, and so on through the whole list. The collision list is typically quite short; almost always less than ten collisions per step in our simulations, and often just two or three. If all collisions on the list are tackled successfully, the OK flag is set and the system moved to time t2. In this way, we may be confident in the results of the routine given an OK return; if there is any complexity at all in the sequence of collisions then OK is set to .FALSE. The reason for this is that such events are infrequent. They could probably be resolved by deft programming, but it is not worth the effort. They are handled (following a failed return) when the time step is reduced in the leftmost branch of the flow diagram. Thus, we may take two or more sub-steps, each involving a call to the advance routine, to cover a single time step dt. As measured by the average length of sub-steps covered successfully by the advance routine, relative to the input timestep dt, we run the program at an efficiency greater than 90%.

One or two additional technical points should be mentioned. In the search for colliding pairs, we look at minimum-image neighbours in the usual periodic system. It is not absolutely certain that the next collision between i and j will be between molecules that are minimum images at the time the search is undertaken. Consequently, the first action undertaken by the advance routine is to examine molecular velocities and check that no pair could move relative to one another by half a box length minus their own diameter, in any coordinate direction, during the interval t2-t1. If the configuration fails this test, OK is set .FALSE. and the routine returns. Secondly, it is well-known that the Newton-Raphson routine occasionally fails, by getting into a cycle bracketting the correct solution. With runs extending over millions of collisions, this is bound to happen every so often, and so the program includes an iteration count and an "emergency" binary search to locate the root whenever too many Newton-Raphson iterations occur. Finally, the most expensive part of the program remains the complete search for pair overlaps. This may be speeded up by the use of neighbour lists in the usual way, and made faster if this vectorizes. We find it most convenient to use a Verlet-type list [11] rather than a cell-structure, since the former may be made to reflect the molecular shape, which in our case is quite anisometric. We are already using a function f which signals the overlap of two ellipsoids. By changing a parameter in this function, we turn it into a new function F, which indicates when two rather larger ellipsoids, containing the original molecules, overlap. We construct our Verlet-type lists using F to decide whether or not two molecules are neighbours, rather than employing the usual criterion of distance between the centres. This approach works well in the dense liquid state, when neither translation nor rotation occur particularly quickly. The list is updated at regular intervals, but the program includes a "safety skin depth" check, based on yet a third version of the f function, to warn when molecules not on the neighbour list are penetrating too far into the neighbourhood region between updates. This acts as a trigger for automatic reduction of the update interval. Construction and use of the list are vectorizable on the CYBER-205, involving "gather" and "compress" operations.

For a system of 144 prolate ellipsoids, with axial ratios of 2-3, at liquid densities around 0.7-0.8 of the close-packed solid density, the program generates around one million collisions per hour of CYBER cpu time. We have used the program to investigate the slowing-down of molecular rotation in the isotropic liquid on approaching the nematic-isotropic transition, and the way in which this is associated with the onset of nematic long-range order. These results will be reported elsewhere [12]. We also intend to investigate transport coefficients in the nematic liquid crystal itself, where, of course, the slow dynamics of the system may present a significant challenge.

[1] D. M. Heyes, CCP5 Newsletter, 10, 21 (1983).

[2] A. Bellemans, J. Orban, and D. van Belle, Mol. Phys. 39 781

(1980).

[3] D. Frenkel and J. F. Maguire, Mol. Phys. 49 503 (1983).

[4] M. P. Allen and I. C. H. Cunningham, Mol. Phys. 58 615 (1986); M. P. Allen and A. A. Imbierski, Mol. Phys. (to appear).

[5] D. C. Rapaport, J. Chem. Phys. 71 3299 (1979).

[6] G. A. Chapela, S. E. Martinez-Casas, and J. Alejandre, Mol. Phys. 53 139 (1984).

[7] D. W. Rebertus and K. M. Sando, J. Chem. Phys. 67 2585 (1977).

[8] W. J. McNeill and W. G. Madden, J. Chem. Phys. 76 6221 (1982).

[9] J. Vieillard-Baron, J. Chem. Phys. 56 4729 (1972).

[10] J. W. Perram, M. S. Wertheim, J. L. Lebowitz, and G. O. Williams, Chem. Phys. Lett. 105 277 (1984).

[11] L. Verlet, Phys. Rev. 159 98 (1967).

[12] M. P. Allen and D. Frenkel, submitted for publication.

A COMMENT ON DEBUGGING MONTE CARLO PROGRAMS

M. Mezei

The production of correct computer code is a major difficulty in any computational project. For a long code, extensive testing is required, special cases where the result is known should be tried. A particular difficulty arises in the debugging of a Monte Carlo program: since the results are legitimately subject to statistical uncertainties, no precise match with existing results can be expected.

This note will describe an approach to the debugging of Monte Carlo programs for simulating an assembly of particles. The key idea is the extensive use of consistency checks as follows. In most problems, there are several expressions or algorithms that compute a given quantity and usually the one that is (thought to be) optimal is chosen. The other ones, however, can still be used for a consistency check. In the rest of this note the self tests employed in a Metropolis Monte Carlo [1] program using pairwise additive energy and single particle moves with force biased displacements [2] will be described (see also Ref. [3]).

The fundamental test is on the calculation and updating of the total energy of a system of N molecules, E:

$$E = \sum_{i<j}^{N} e_{ij} \qquad (1)$$

where e_{ij} is the pair interaction energy between molecules i and j. E is updated whenever a particle i' is moved. For this update only the terms involving the moved particle i' are to be considered since the rest remain unchanged:

$$E^{new} = E^{old} - B^{old}_{i'} + B^{new}_{i'} \qquad (2)$$

where

$$B_i = \sum_{i \neq j}^{N} e_{ij} \qquad (3)$$

is the binding energy of molecule i and the superscripts old and new refer to the status before and after the move, respectively. The B_i values are also carried and kept updated whenever a move is accepted (for all molecules $i \neq i'$):

$$B^{new}_i = B^{old}_i - e^{old}_{i',j} + e^{new}_{i',j} \qquad (4)$$

This way (since the e_{ij} values are not stored), Eq. (2) can be used without recomputing the $e_{i',j}$ values before the move and the recomputation is only needed for accepted moves (to update the B_is) [4]. The energy E is related to the B_is as

$$E = \sum_{i=1}^{N} B_i/2 \qquad (5)$$

Therefore, the first test compares the result of eq. (4) with the energy carried. The second test looks at the binding energy $B_{i'}$ carried by the program and compares it with the value directly computed from Eq. (3). The third test recomputes E using Eq. (1) and compares it with the energy carried. Notice that the first test requires no energy calculation at all, the second requires the computation of $\sim N$ energies and the third requires the computation of $\sim N^2/2$ energies. All of these tests should give equality up to the precision of the computer.

If the program calculates forces (torques) and the virial sum then analogous tests can be performed on them. The virial sum is updated in analogy of the total energy, and thus the third test is applicable. The components of the forces (torques) acting on a molecule are obtained and updated in analogy to the binding energy B_i and thus the second test can be applied to them.

The tests described above do not check the calculation of e_{ij} and its derivatives but rather the operations involved in composing the various molecular and supermolecular sums. For programs calculating forces, there is a possibility of testing simultaneously the calculation of e_{ij} and its derivatives by performing a numerical differentiation and comparing the derivatives computed. This test, however, is not as strong a the ones described above since in general when the difference quotient is a good approximation to the derivative the increment is very small. But a too small

increment will give an imprecise quotient due to the limited precision of the machine. The test is more powerful when performed on a system consisting of a few molecules only. In our experience, on a 32-bit word-length machine only 2-3 digit agreement can be obtained.

Condensed phase simulations are usually performed under periodic boundary conditions. While most applications choose the rectangular unit cell (that is very simple to implement), there are advantages to using other shapes: truncated octahedron, hexagonal prism or the Wigner-Seitz cell of the face-centred cubic close packing. The efficient implementation will use an algorithm for the determination of the image cell a given point is in that is specifically developed for the boundary condition chosen. An inefficient, but general algorithm would simply find the cell whose centre is the nearest to the point considered. This general algorithm, however, can be used to check any of the specific ones.

Quite often, a Monte Carlo program maintains a list of neighbours (to reduce the energy calculation) or the number of neighbours (for calculation of coordination numbers). Again, these data are computed from scratch at the beginning only and are updated thereafter. In the self-testing mode, however, they can be recomputed from scratch and compared with the lists carried. For the neighbour lists the symmetry should also be checked: if i is a neighbour of j, then j must be a neighbour of i and vice versa.

In the actual realisation a subroutine is maintained in the program for these tests. This subroutine is called only optionally, primarily during the testing period but periodically thereafter as well (since some bugs may manifest themselves only after longer runs or under some special input parameter combinations). Any time the program is modified, the self tests are called.

In concluding, I would like to apologise to the reader for the obviousness of the material described. The reason for the presentation is not so much to point out the existence of these identities or alternative algorithms but rather to report that they proved to extremely useful in detecting and locating bugs during the development of our Monte Carlo programs. In several instances, prior to performing the consistency checks, the program apparently "worked" and there was no obvious indication of malfunctioning. When a failure occured, the combination of failed tests usually gave strong indication as to the location of the error in the program and quite often the point of breakdown in the

calculation singled out special circumstances under which the error occured, further narrowing the search. It is suggested therefore that these tests (and possibly any other that is applicable to the problem at hand) should be routinely incorporated into Monte Carlo programs as a debugging aid for the developer and as a confidence builder for the user. As a final warning, however, one should add that all consistency checks serve only as necessary conditions for the correctness of the program and other means of testing should be employed as well.

Acknowledgements

This research was supported by NIH grant GM 24914 and NSF grant CHE-8293501. Several useful discussions with Prof. K. Bencsath are gratefully acknowledged.

References

[1] N.A. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller and E. Teller, J. Chem. Phys., 21, 1087 (1953).

[2] M. Rao, C.S. Pangali and B.J. Berne, Molec. Phys., 37, 1779 (1979).

[3] D.L. Beveridge, M. Mezei, P.K. Mehrotra, F.T. Marchese, G. Ravi-Shanker, T. Vasu and S. Swaminathan, in "Molecular-Based Study of Fluids", J.M. Haile and G.A. Mansoori, eds., Advances in Chemistry, Vol 204, American Chemical Society, Washington (1983).

[4] P.K. Mehrotra, unpublished result.

PROTEIN CRYSTAL DATA ANALYSIS

A Study Weekend jointly organised by the Collaborative Computing Project in Protein Crystallography (CCP4) and Daresbury Laboratory

23-24 January, 1987
Lecture Theatre, Daresbury Laboratory

The collection and analysis of protein crystallographic data is the basis from which structural information on biological systems, at the molecular level, is derived. The Study Weekend will address particularly the computational aspects of this topic. The processing of film data, measured via monochromatic oscillation and white beam Laue methods, will be discussed. Special attention will be given to the acquisition and processing of data collected on electronic area detectors using conventional or synchrotron x-rays and neutron beams; progress on the EEC initiative on this topic will be reported.

Speakers include:

G. Bricogne	(LURE)	A. Leslie	(Imperial College, London)
D.W.J. Cruickshank	(York/Daresbury)	P.A. Machin	(Daresbury)
P.R. Evans	(MRC, Cambridge)	M.Z. Papiz	(Daresbury)
R. Fourme/R. Kahn	(LURE)	J. Pflugrath	(Munich)
T.J. Greenhough	(Keele/Daresbury)	T. Richmond	(MRC, Cambridge)
J. Hajdu	(Oxford)	R. Stansfield	(Edinburgh)
M.M. Harding	(Liverpool)	D. Stuart	(Oxford)
J.R. Helliwell	(York/Daresbury)	D.J. Thomas	(MRC, Cambridge)
A.J. Howard	(Genex Corporation)	A.J. Wonacott	(Imperial College, London)

There is a registration fee of £43.00 excluding accommodation. The fee will include all meals during the meeting with the exception of lunch on Friday. Accommodation has been arranged at a local hotel for the Friday night and the charges are £20.00 per night for a single room; £18.50 per night for a shared room (Bed & Breakfast). Please note that the number of places at the meeting is limited so early application is advisable.

How to apply:

Enquiries about the programme should be made to Dr. J.R. Helliwell, Department of Physics, University of York, Heslington, York YO1 5DD, Tel.: 0904-430000 (Ext.5507) or to P. Machin, SERC Daresbury Laboratory, Tel.: 0925-603350. Additional information and application forms are available from Mrs. S.A. Lowndes at Daresbury, Tel.: 0925-603305, who will be pleased to answer queries about the organisation of the meeting.

DARESBURY

SERC
DARESBURY LABORATORY
WARRINGTON WA4 4AD
Tel. 0925 603000 Telex 629609

MAIN JAMMD MINOM

2NTJMA