A Simulation of Oxygen Diffusion within the High-Temperature Superconductor $Y Ba_2 Cu_3 O_{6+x}$ using the Monte-Carlo technique

by

Ian Keith Robinson

An original study submitted as part of the degree in Computational Physics

undertaken within Salford University's Dept of Physics

Abstract

The effect of chemical potential upon oxygen diffusion within the superconducting Cu-O laminar planes of $YBa_2Cu_3O_{6+x}$ has been investigated using Monte Carlo simulations based upon the Ising model. Two programs were written and extensively tested to perform this modelling. GLAUBER.FOR and KAWASAKI.FOR employing Glauber and Kawasaki dynamics respectively.

The results show evidence of phase changes occuring as the lattice structure evolves with increasing oxygen concentration and the formation of a mixed phase region. There is opportunity for further study of what appear to be minor phase changes developing at lower temperatures.

Contents

Contents

1
Introduction

1.1 Historical Background

The mathematically ideal would appear to be a rare quality within the universe. Physics students wrestle with the equations of simple harmonic motion, 2-body gravitational systems the gas laws and so on. Only to learn that these are idealised representations of a far more messy, less clear cut world. Superconductivity stands out as salient exception, materials exist which appear to permit, not a vanishingly small resistance but absolutely no impediment to the flow of electric current.

Superconductivity was first identified in 1911 when the Dutch physicist Heike Onnes noted that the resistivity of mercury appeared to suddenly vanish as the metal was cooled to below 4.2 K. Over the following six decades researchers discovered a number of metallic elements and compounds which shared this property. Despite the marked benefits which would flow from the widespread use of superconducting transmission cables, motors and magnets, practical applications remained esoteric due to the cost of liquid Helium required to cool to such low temperatures.

	T_c /K
Zn	0.87
Al	1.14
Pb	7.19
Nb	9.5
Nb-Ti	9
Nb_3Sn	18
Nb_3Ge	23

Table 1.1 The critical temperatures for a range of metallic elements and binary alloys

In 1986 IBM researchers discovered superconductivity in cuprate oxides and found evidence for transition temperatures in excess of 30 K. This lead to an explosion of interest in the superconducting properties of copper oxide ceramics. In 1987 research groups in Houston and Alabama made a key discovery, that the $YBa_2Cu_3O_7$ ceramic had a transition temperature of 92 K. Since the boiling point of nitrogen is 77 K then the possibility arises of more widespread use of superconductors cooled by this relatively cheap and plentiful liquid.

	T_c /K
$La_{2-x}Sr_xCuO_4$	38
$YBa_2Cu_3O_7$	92
$Bi_2Ca_2Sr_2Cu_3O_{10}$	110
$Ti_2Ca_2Ba_2Cu_3O_{10}$	125

Table 1.2 The critical temperatures for a range of high temperature superconducting materials.

1.2 Crystalline Structure of YBCO

Yttrium Barium Copper Oxide (YBCO) possesses an anisotropic structure of copper oxide planes separated by Barium Oxide and Yttrium ions (*figs 1.1*). This planar structure is common to most high temperature superconductors, specifically the Copper oxide layers in which superconduction takes place.

A model of the structure places two conducting copper oxide planes separated by an yttrium site. These are blanketed by the copper oxide chains with an intervening barium oxide layer.

-------------------- Cu-O chain --------------------

BaO

-------------------- CuO_2 plane --------------------

Y

-------------------- CuO_2 plane --------------------

BaO

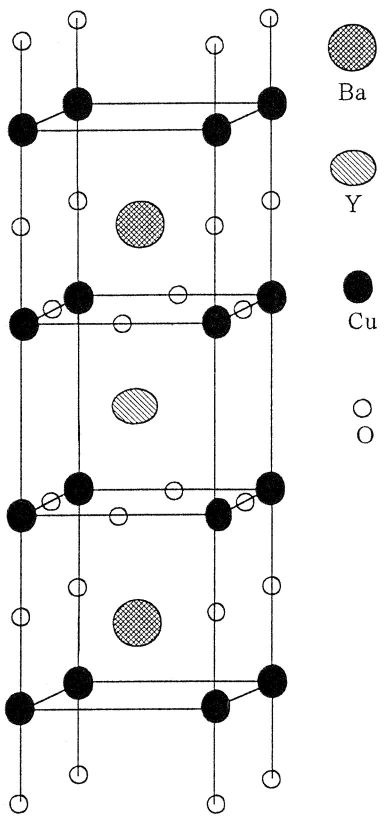
-------------------- Cu-O chain --------------------

fig 1.1 Schematic diagram of $YBa_2Cu_3O_{6.93}$ *(ref 2 p177)*

The copper oxide chains act as charge reservoirs. Adding oxygen upsets the charge balance causing electrons to migrate from the copper oxide planes. The resulting holes can form Cooper pairs below T_c resulting in superconduction as per the BCS model. Thus the excess oxygen fraction directly effects the superconducting properties of the material.

A key point here is that the excess oxygen atoms introduced into the structure form copper-oxygen chains (fig 1.2). At high temperatures the oxygen atoms are randomly distributed over the lattice. As the temperature is lowered ordering takes place due to long range Coulomb interactions. The cations are effectively frozen in place over the temperature range of interest and only the anions (oxygens) are able to migrate.

Oxygen ordering takes place in the (001) Copper Oxygen basal planes which approximate to a square lattice . As Khachaturyan *et al* note that correlation between distinct Cu-O planes is weak (they are some 12 Angstroms apart) diffusion within the layers is thus orders of magnitude greater than that between layers [6]. Accordingly I have modelled diffusion only within Cu-O planes.



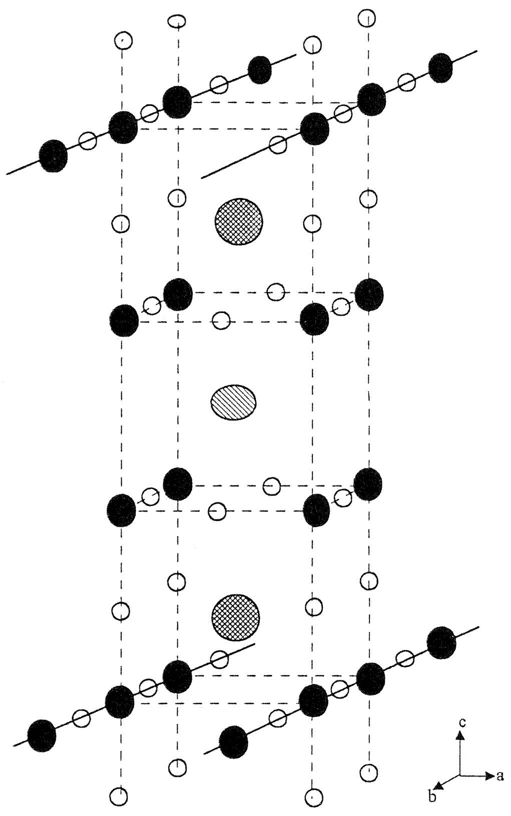
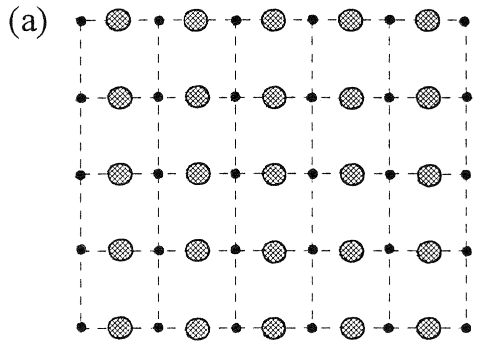


fig 1.2 Crystal structure of $YBa_2Cu_3O_6$

fig 1.3 Crystal structure of $YBa_2Cu_3O_7$

The process of oxygen ordering converts the random (tetragonal) structure to one of two orthorhombic phases OI and OII (fig 1.4).



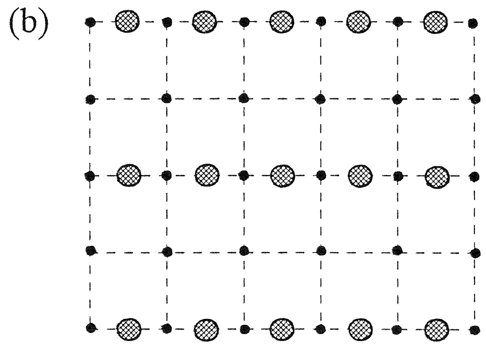


fig 1.4 a) Orthorhombic I

b) Orthorhombic II

This stoichiometric factor is of significance in determining the superconducting properties of the material.

$YBa_2Cu_3O_{6+x}$ for $0.0 < x < 0.4$ acts as an insulator.
for $0.4 < x < 1.0$ acts as a superconductor.

Diffraction studies have indicated that most oxygen vacancies occur within the copper oxide (a-b) basal plane. The aim of this work has been to model oxygen flow through this material and compare these results with those of phenomenological studies carried out at Salford.

1.3 Preparation of YBCO

Pure phase crystals are typically produced by reacting together precursors for many hours at a temperature close to the lowest of their melting points. For example one may mix powdered Y_2O_3 and BaO_2 and react in air or oxygen at >900 °C for > 5 hours to form $YBa_2Cu_3O_6$. Cycles of grinding and reacting may be required to produce a suitable superconducting powder. These grains may then be sintered together at high temperature to produce a bulk sample.

A key problem is controlling the excess oxygen fraction critical to the material's superconducting properties. Excess oxygen may be introduced during manufacture by manipulating melt times, quenching, and the partial pressure of oxygen in the surrounding atmosphere. The team at Salford University are investigating oxygen uptake by YBCO crystals at fixed temperatures for a range of surrounding oxygen partial pressures.

1.4 Brief Discussion of Superconductivity

There has yet to be developed a comprehensive theoretical model of superconductivity in these new ceramic compounds. It is likely that any such model will draw upon the well established models which deal with the older metallic elements and non-laminal compounds.

The BCS Theory

In 1957 J Bordeen, L Cooper and R Schrieffer presented a general microscopic theory of (low temperature) superconductivity. Here they postulated that below the transition temperature T_c there exists a net attraction between pairs of electrons in contrast to the more usual Coulomb repulsion. One may picture this by imagining that an electron fractionally distorts the crystal lattice through which it travels. An electron some distance behind has an enhanced likelihood of following this wake before it relaxes, thus introducing a correlation between the motion of the pair.

This results in a condensed state where electrons weakly couple with those of opposite spin and momenta. The net momentum of such a pair is zero and hence their associated De Broglie wavelength is, in principle infinite; or at least as large as the sample. Since any potential scattering centres are many orders of magnitude smaller than λ then no scattering should occur. Thus without a mechanism to impede the flow of electrons the sample should exhibit zero resistance.

Unlike individual electrons which follow Fermi-Dirac statistics the Cooper pairs act as Bosons obeying Bose-Einstein statistics. Electrons within a conventional metallic conductor are each described by their individual wave function. Cooper pairs by comparison are all permitted to occupy the same quantum state and thus are all described by a single wavefunction.

$$\Psi(\mathbf{r}) = \sqrt{\eta}\,(\mathbf{r})e^{i\varphi(\mathbf{r})}$$

As all Cooper pairs are in the same state current flow becomes a macroscopic quantity unaffected by microscopic scattering points within the lattice.

2
Theoretical Background

2.1 Principles of Diffusion

It would seem that a universal property of matter is the tendency towards minimum free energy. Material systems tend to move towards stable equilibria by particulate migration so as to eliminate spatial variations and thus minimise free energy. This is the cause of diffusion which occurs in all systems above absolute zero. Diffusion rates vary markedly, from seconds in the case of gases to millennia for crystalline solids.

In the mid 19th century Adolf Fick, noting that diffusion was analogous to heat conduction, formulated a definition of diffusivity now known as Fick's 1st law.

$$J = -D_{chem}\frac{dc}{dx} \qquad 2.1$$

Here he assumed i) net Brownian flow from high to low concentration
ii) net flow is proportional to concentration gradient.

where J = net flow/unit time
= net particles crossing unit area in unit time
D_{chem} is a system dependant constant

From the conservation of particles we may derive;

$$\frac{dJ}{dx} = \frac{-dc}{dt} \qquad 2.2$$

From 2.1 and 2.2 we may obtain Fick's 2nd law

$$\frac{dc}{dt} = D_{chem}\frac{d^2c}{dx^2} \qquad 2.3$$

In 3 dimensions

$$\frac{dc}{dt} = D_{chem}\nabla^2 C \qquad 2.4$$

The aim of these experiments has been to model flow into YBCO and determine a value for D_{chem}.

For lattice diffusion in 2d the oxygen mobility M may be determined from the Nernst-Einstein equation

$$M = l^2/6\tau kT$$

Where τ is the mean site residence time, and l^2 is the mean squared distance moved in one jump. Mobility here is a measure of

Le Claire (1970) defines a tracer diffusion coefficient as

$$D_t = f_t l^2/6\tau$$

which represents the enhanced probability that an atom having made a jump jumps back to its original site. f_t tends to unity as the concentration of diffusing particles tends to zero.
Hence we may write

$$M = D_t/kT$$

thus

$$D_{chem} = Mc\frac{d\mu}{dc}$$

2.2 The Monte Carlo Method

The Monte Carlo method is a powerful tool for analysis of complex, many variable systems. It has enjoyed particular success in modelling atomic systems and within quantum electrodynamics.

In the case of atomic systems a mathematical determination of the varying phase transitions may be impractical, particularly if one is dealing with more than one atomic species. Monte Carlo simulations bypass almost all of this mathematics by modelling an assembly of mutually interacting points (atoms). Since pairwise atomic interactions are easily quantified then in theory one merely needs to model a sufficiently large assembly of atoms in order to produce accurate results. One can view such a Monte Carlo simulation as sampling a massive number of points within a system's phase space representing possible configurations. A given 'fitness' criteria, such as entropy, will then cause the system to migrate towards a minimal free energy configuration

The simulations presented here are based upon the locally anisotropic two dimensional lattice gas model proposed by Wille, Berera and De Fontaine [8]. This is identical to the Ising model of antiferromagnetism and can also model binary alloys. We may consider the modelling of oxygen diffusion into the laminal basal planes of YBCO as equivalent to a binary alloy with the mobile oxygen atoms representing one atomic species and the lattice vacancies as the other.

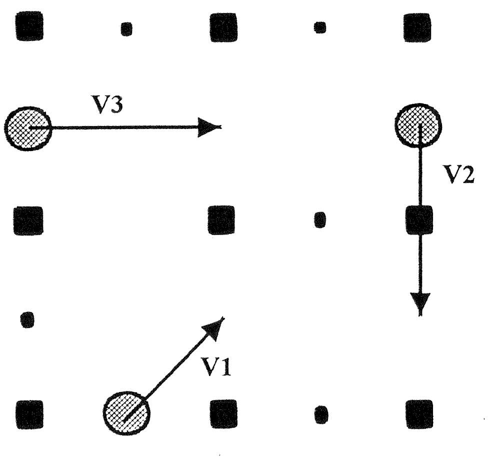


fig 2.1 The three n.n exchanges

Mobile atoms may make random jumps to vacant lattice sites. The probability of a particular jump occurring is dependent upon the resulting free energy change. One could thus create an environment whereby the model tends to the least energetic stable configuration.
The mean field approximation assumes that surrounding lattice sites are filled to mean concentration which means that we only need to take account of n.n or n.n.n interactions. This greatly reduces computational load.

In this case we assume three interaction parameters V_1, V_2 and V_3. We can then describe the Hamiltonian of the total internal energy of the system

$$H = -V_1 \sum_{ij}^{nn} n_i n_j \; - V_2 \sum_{ij}^{nnn} n_i n_j \; - V \sum_{ij}^{nnn} n_i n_j$$

where V_1 represents a n.n interaction, V_2 a n.n.n interaction mediated by an intermediate copper atom and V_3 a n.n.n jump not mediated by a copper atom (fig 2.1).

The configuration partition function Z may be written as

$$z = \sum_{x} -H(x)/K_b$$

Here we sum over all possible system configurations x, the Hamiltonians of each configuration H(x). K_b is the Boltzmann's constant and T a temperature term.

Thus the free energy $\quad G = U - TS$

where $\quad$ G = free energy, U = internal energy and TS is an entropy term

we may write

$$G = -K_b \, T \ln(z)$$

Given the massive number of configurations possible within a physical sample Z is too complex to evaluate formally. Approximations are available (Bragg-Williams and Cluster Variation model) however these are likely to adversely affect accuracy in the relatively complex scenario here. Thus Monte Carlo methods may be employed to carry out a numerical simulation of the YBCO system.

2.3 The Simulations

The objective of this work was to simulate oxygen diffusion into the laminal Cu-O planes of YBCO. The specific aim being to produce graphs of oxygen concentration against temperature and concentration against pressure. These may be compared with those produced by direct experimentation at Salford.
Programs were written to simulate oxygen diffusion into a sample from an external heat-bath. The time to equilibrium concentration for fixed driving potentials was measured along with the variation of final oxygen concentration as a function of driving potential. Both of these simulations were performed for a range of temperatures.

3
The Computer Simulation

Two separate programs were developed to simulate the diffusion of oxygen with the Cu-O layers of YBaCuO, **Glauber** and **Kawasaki**. Both simulated a grand canonical assembly of oxygen atoms migrating from an external heat bath into an initially 'empty' lattice of zero stochiometric coefficient employing the Monte Carlo technique for computational investigation of many body systems.
Glauber allowed jumps over arbitrary distance whilst **Kawasaki** permitted only nearest neighbour exchanges.

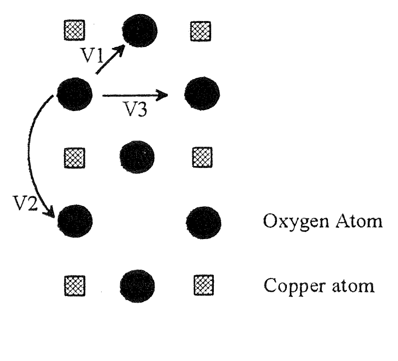
Two additional programs were written to test the performance of the random number generating routine used on the department's Alpha machine.

3.1 Lattice Simulations

In each program the lattice was modelled as a 50 by 50 array of lattice sites with the copper atoms fixed in position. The programs represented these by integer arrays with different integers used to denote atomic species.
The simulation process involved randomly selecting an oxygen atom and randomly choosing a lattice site to jump to. The sum of the atom's interaction potentials with its neighbours was compared with that assuming that it had made a jump to the vacant n.n site. If the interaction potential at the vacant site was lower than at the current site the atom moved. If the potential was higher the jump could still take place though with a probability which exponentially diminished with the increase in interaction potential difference. Governed by

$$p = e^{-\Delta E/Kt}$$

The three n.n interaction potentials are shown in fig 3.1



V1	the spatially n.n interaction
V2	n.n across the unit cell centre
V3	n.n across an intervening Copper atom.

The values used were

V1	= 6.9 mRy	
V2	= -2.4 mRy	
V3	= 1.1 mRy	[9]

fig 3.1 The three forms of O-O interaction

The mean site interaction energies for the different phases should be:-

OI	2V(2) + 2V(3)	-2.6 mRy
OII	2V(2)	-4.8 mRy
All sites occupied		25 mRy

The programs were tested with lattices pre-loaded with these phases and they returned the expected values for mean site interaction energies.

3.2 Simulating Oxygen migration into a sample

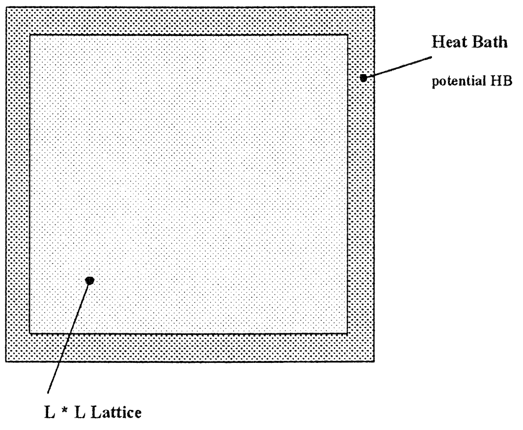


fig 3.3 The Heat bath

The program Glauber measures oxygen diffusion into an initially empty lattice surrounded by an infinite reservoir of oxygen atoms at a chemical potential HB. The program may monitor oxygen diffusion (as measured by the change in concentration within the lattice) for either fixed temperature and varying chemical potential or fixed potential and varying temperature.

To reduce run time the lattice may be preloaded to a desired concentration.

Glauber is a computationally intensive program with long run times before the lattice settles to an equilibrium concentration. This is particularly problematic at low temperatures where the probability of jumping up a potential gradient is greatly diminished. Two techniques were considered to reduce run time;

i. One approach used involved randomly pre-populating the lattice with oxygen atoms. This greatly reduces run time and seems to have no effect upon the final 'settled' lattice concentration.

ii. Jumps could be allowed over arbitrary distance i.e.; not just to n.n sites. Whereas this use of Glauber dynamics does not accurately model the physical diffusion process the lattice approaches equilibrium some orders of magnitude more quickly.

The program could be configured to monitor final concentration and mean site interaction energy against the driving chemical potential of the heat bath at fixed temperature or varying temperature for fixed chemical potential. The final values of the dependant variables concentration and mean site energy were given as the mean and standard deviations over the previous 100 samples.

I employed Glauber dynamics for monitoring the variation of final (equilibrium) concentration against chemical potential for fixed temperature. Whilst a second program configuration 'Kawasaki', allowing only n.n jumps, was employed to measure the flow rate as a function of driving potential for fixed temperature.

4
Testing the Simulations

Writing programs of this type is comparatively simple. The difficulty lies in ensuring that they function as intended. Hundreds of hours were spent in testing, then correcting errors and fine tuning performance, particularly with respect to the edge effects caused by the wrap around. A summary of the testing is presented here.

4.1 Testing the Random Number generator

A computer can generate a single number at random. One could set up a loop to cycle through digits stopping at a keypress. Since a person cannot reduce the time between strokes to much less than 0.1 seconds and given the speed at which the machine can cycle through a set of digits we can safely assume that a digit has been chosen at random. However digital computers are deterministic devices and as such are incapable of generating strings of genuinely random numbers.

Monte Carlo simulations of the type described here necessitate the serial generation of large quantities of random numbers, both Glauber and Kawasaki used the Nag library routine G05DAF.

Such routines typically develop an initial seed number from the system time. Subsequent pseudo random numbers are developed from an algorithm such as the power residue method.

$$x_n = cx_{n-1}(\text{mod } N)$$

where N and c are constants and x_0 is the initial seed number.

Given that computers are deterministic devices and the algorithm used such sequences can only be described as pseudorandom, a salient point being that the sequence will repeat after some interval.
There would seem to be no general theoretical technique for predicting the performance of most pseudorandom number generating algorithms particularly with regard to the statistical properties of the numbers generated. The sequence length is generally less problematic. For the algorithm above this is maximised when

	$c = 8n \pm 3$	where n is any positive integer
and	x_0	is any odd integer.

I decided to carry out two comparatively simple tests of the G05DAF routine which should detect any gross deviation from a random sequence. Firstly I looked at the distribution of numbers generated. Secondly I looked for repeated sequences and pseudorepitition.

4.1.1 Frequency Distribution

If one generates M numbers in the range 1 to N and tally those which lie in each of the equal intervals 1/N one would expect a uniform distribution across the intervals. One would require that the individual deviations from the mean tally in each interval should obey the appropriate statistical laws.

The probability of finding x numbers within any given subinterval is given by

$$e^{-(x-\bar{x})^2/2\sigma^2}$$

where $\bar{x}$ is the mean number in all intervals, and σ is the standard deviation given by

$$\sigma = \sqrt{\bar{x}}\,.$$

Given that we expect a Gaussian distribution one would expect to find 68% of the tallies to be within one standard deviation point of the mean. Also one would expect the rms. deviation for all subintervals to approach σ for large M.

$$d_{rms} = \sqrt{\frac{1}{M}\sum^{M} d_i^2} = \sqrt{\frac{1}{M}\sum^{M}(x_i - \bar{x})^2}$$

To investigate the frequency distribution a program, Histogrm was written to generate 10 million numbers in the range 0 to 100. These were tallied into 100 equal intervals (fig 4.1) .

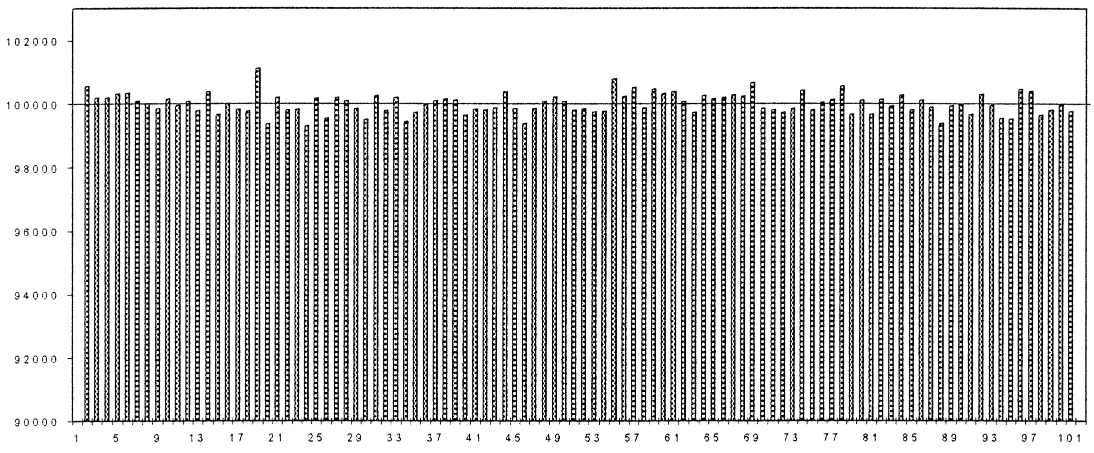


fig 4.1 histogram showing frequency distribution of 10,000,000 numbers

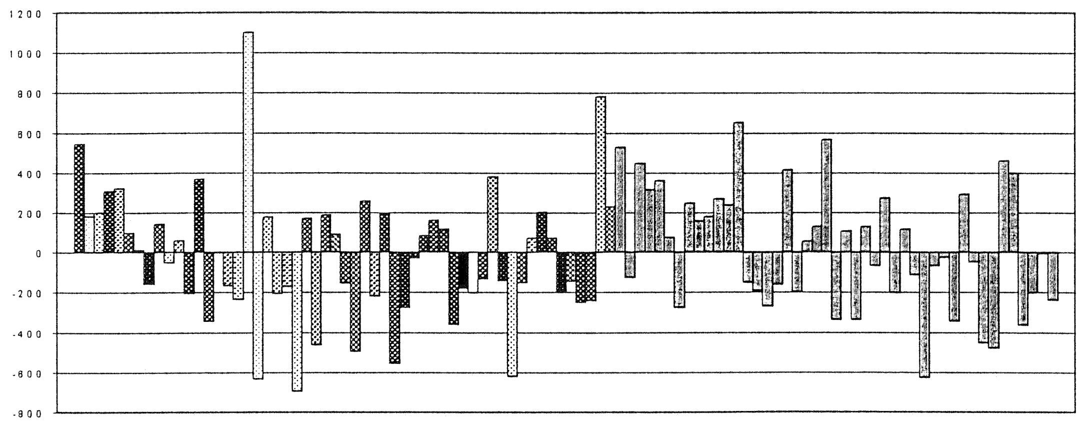


fig 4.2 *Deviations from* $\bar{x}$ (100,000) msd = 321.9 71% of values within 1 sd of $\bar{x}$

Here the msd of 321.9 is within 2% of the predicted figure of $\sigma = \sqrt{100000} = 316.2$. A cumulative frequency plot of no of deviations falling within specified ranges of the mean is shown in fig 4.2 The weighting towards the upper range may be a result of a single 'large' deviation in the number of values returned between 17 and 18, though here the absolute count was only 1% or 3 sd units above the mean.

In conclusion the G05DAF number generator produced a reasonably uniform spread of values with 71% falling within 1 sd unit of the mean. The sample was distributed with 52% of the values below the mean and 48% above. Given these it seems reasonable to assume that the distribution approximates to Gaussian form.

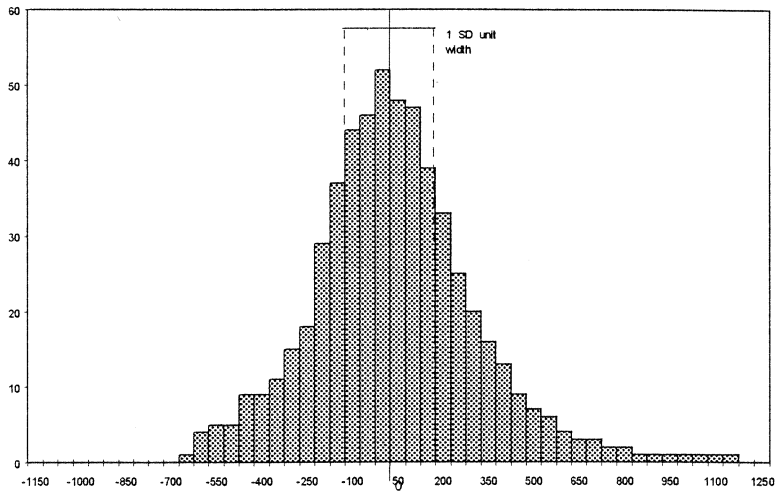


fig 4.3 Plot of no of deviations against distance from mean

4.1.2 Serial Correlations

The second series of tests on G05DAF attempted to identify any tendency towards repetition of the random sequence. G05DAF should have an extremely long period and it was not expected to cycle within the limit of processor time available. I chose to check for more subtle signs of partial repetition and the tendency towards serial correlation. A comprehensive series of tests would be unfeasible. One could look for series of numbers consistently above or below the mean, series which increment in a particular fashion, series which tend to lie within a sub-range of those required and so on. The program Repitit monitored two aspects of the random series. The first simply measured the maximum length of any repeated series. The second test was more subtle and looked for a generalised tendency towards repetition.
The program generated 10^7 integers in the range 1 to 100. The first 1000 of which were systematically compared with the rest and the length of the maximum repeated string was measured.

For any number x_i in a string of n random integers with m degrees of freedom one would expect the probability of it equalling x_{i+di} to be given by

$$p(x_i = x_{i+di}) = \frac{1}{m}$$

and the probability of any series of length s beginning with x_i to be equal to that at an distance di to be

$$p = \frac{1}{m^s}$$

Therefore in any series of n numbers we would expect to find a maximum repeated string length of

$$n = m^s$$

On two runs of 10^8 numbers Histogrm found a maximum repetition length of 4 in line with predictions.

The second test was devised to look for a tendency towards correlation. It takes the initial 1000 numbers generated and compares them with the subsequent series and counts the quantity of number pairs (Correlations) which it finds.

x_i	x_{i+1}	x_{i+2}	x_{i+3}	x_{i+4}	x_{i+5}	x_{i+6}	x_{i+7}	.....	....
x_{i+di}	x_{i+di+1}	x_{i+di+2}	x_{i+di+3}	x_{i+di+4}	x_{i+di+5}	x_{i+di+6}	x_{i+di+7}	.....	....

The program stores the first 1000 numbers generated in an array. These are then compared with all subsequent series commencing x_i for i = 1001 to n - 1000. With the quantity of correlations recorded.

I am unaware of this being a standard form of correlation test. It was based upon a technique used in cryptoanalysis and developed by the American cryptographer William Freedman in about 1920. It is notable that the worlds first electronic computers applied this technique during the Second World War to help break the German Enigma and Japanese Purple ciphers.

The probability of any integer x being equal to another x_i is given by

$$p(x = x_i) = \frac{1}{m}$$

assuming m degrees of freedom and a uniform distribution. So for the case of m = 100 one expects to see 10 number pairs per 1000 number series compared.

The program records a histogram of the number of correlations. Additionally if the number of correlations exceeds an arbitrary threshold value (24) this is recorded against the offset di. to help detect any periodicity in correlation peaks.

This test should detect partial or interrupted repetition whereby for example a repetition occurs where every 10th number is repeated and the intervening ones do not.

Fig 4.4 shows the correlation histogram produced. It appears to be of a normal distribution peaking at the expected value of 10 correlations. The mean correlation is within 0.003% of the expected value of 10 implying no general tendency towards repetition within the sample of 10^8 numbers.

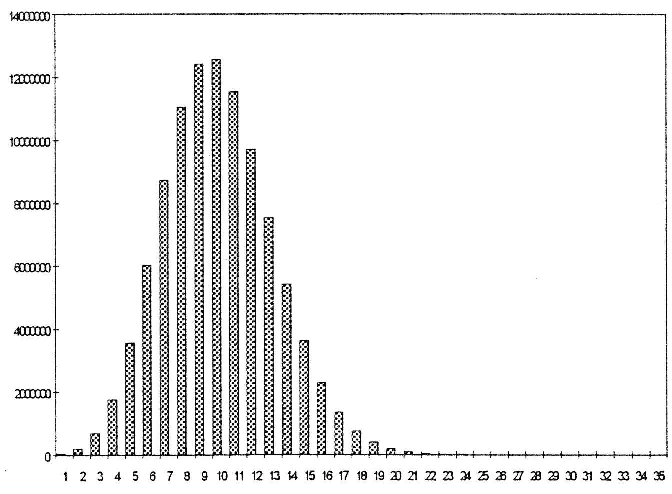


fig 4.4 Frequency of number of correlations

Seeking signs of periodicity between high correlation peaks fig 4.5 shows the sequence separation between the first 256 high correlations found. The mean distance between high correlations over the entire sample was 54674 with a standard deviation of 58004 again implying no marked periodicity in these high correlation sequences.

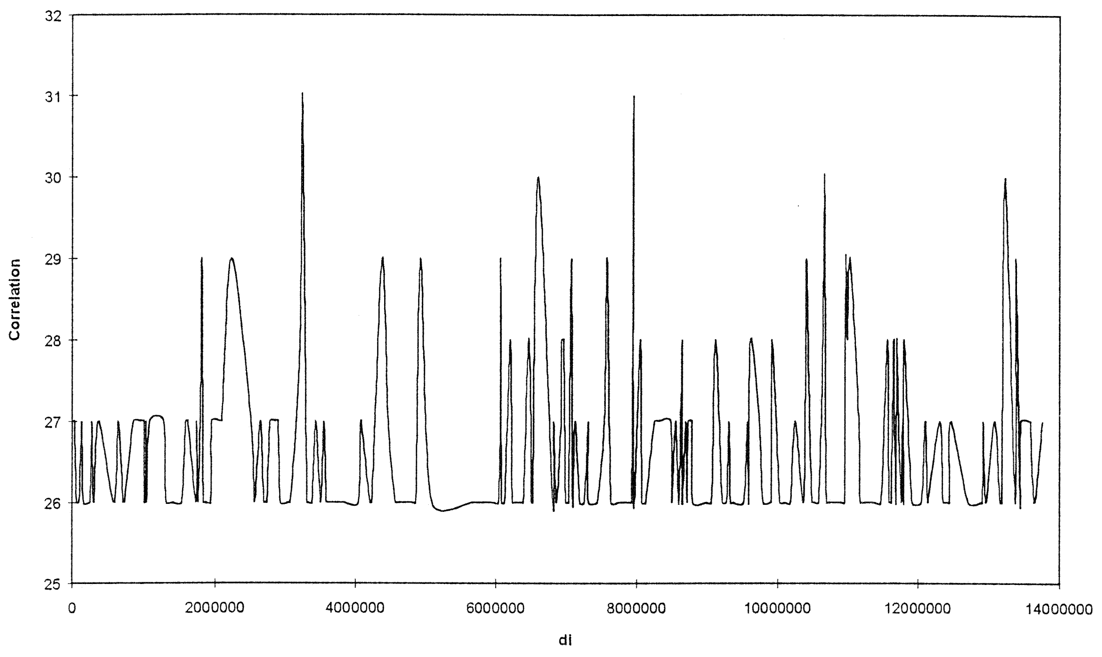


fig 4.5 Plot of first 250 high correlation value against distance

In summary runs found no marked deviation for the gross statistical properties which one would expect to see with a random sequence. The tests though were far from exhaustive.

Histogram simply measured the bulk distribution of numbers and would not detect sequences of numbers which tended towards a subset of the range. A more sophisticated check would seek serial frequency discrepancy looking for series of numbers which deviated from a uniform distribution.

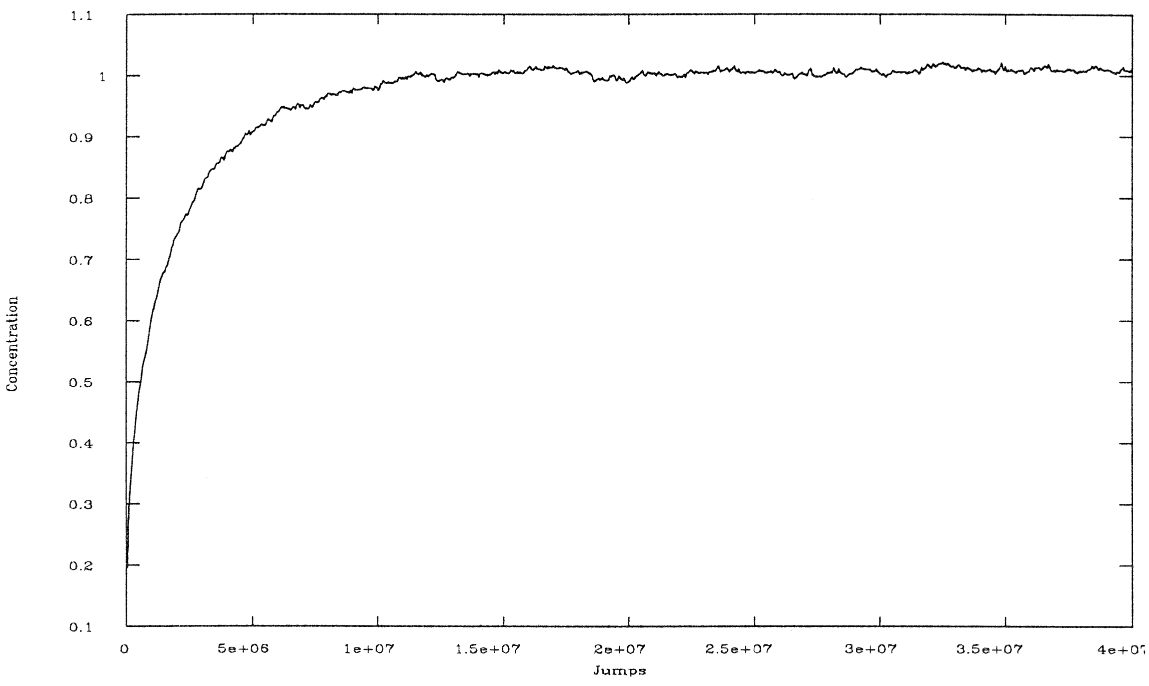
Repitit was based on a more powerful algorithm examining a general tendency towards serial correlation along with identical repeated sequences. Its salient limitation lies in the fact that it compares the first 1000 numbers generated with subsequent numbers. G05DAF commences a generation of random numbers at some arbitrary point within the total series. It is possible that the initial 1000 numbers lie within a near random sequence and if so any subsequent serial correlations are likely to be overlooked. One technique to overcome this would be to use a longer initial sequence though this would increase processing time. To test N values against an initial sequence length of m requires m x N comparisons to be performed. Whereas such a linear growth rate is generally preferable to exponential growth the length of pseudo random sequences prior to repetition precluded investigation of the series length.

4.2 Testing the Kawasaki (Diffusion) Program

4.2.1 Time to Equilibrium

The computational intensity of the Kawasaki program results from the massive number of exchanges required before the system settles down to equilibrium. As the jump rate is temperature dependent this effect is more marked at lower temperatures. Since only nearest neighbour interactions are permitted Oxygen atoms have to diffuse into the centre sample from the edges which greatly increases the time to equilibrium.
Of particular interest is the temperature range below kT/V(1) = 0.1 in which the O1 phase predominates. Investigation of this region is hampered by very long run times.
The runs below were all performed on a 100 by 100 lattice.

The X2 run
Here kt/v(1) = 0.7 and the chemical potential of the heat bath HB was set to the comparatively high value of 10.



4.6 C vs jumps for HB = 10, kt/v(1) = 0.7

Fig 4.6 shows the variation in concentration against the number of attempted n.n exchanges (jumps) within a 101 by 101 element lattice.
It is clear that even at this high temperature the system requires approx. 10^7 attempted exchanges to reach equilibrium. Fig 4.7 shows the variation in mean site energy, the variability in which suggests that activity is still proceeding when the lattice has settled to a stable configuration.

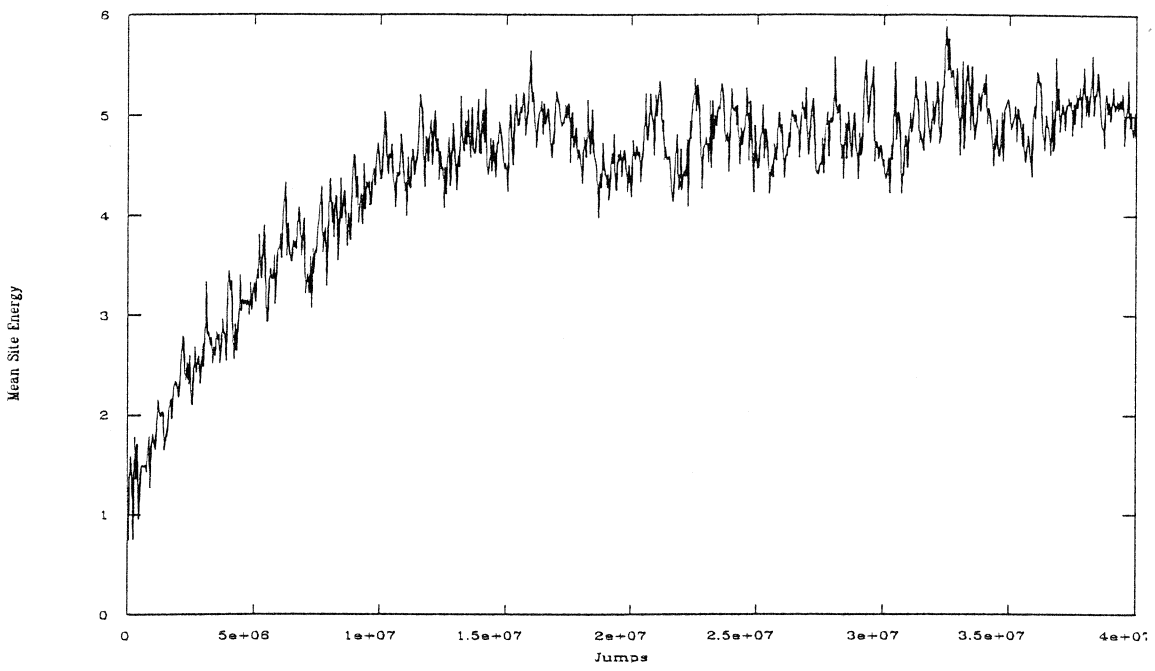


fig 4.7 E vs jumps for HB = 10, kt/v(1) = 0.7

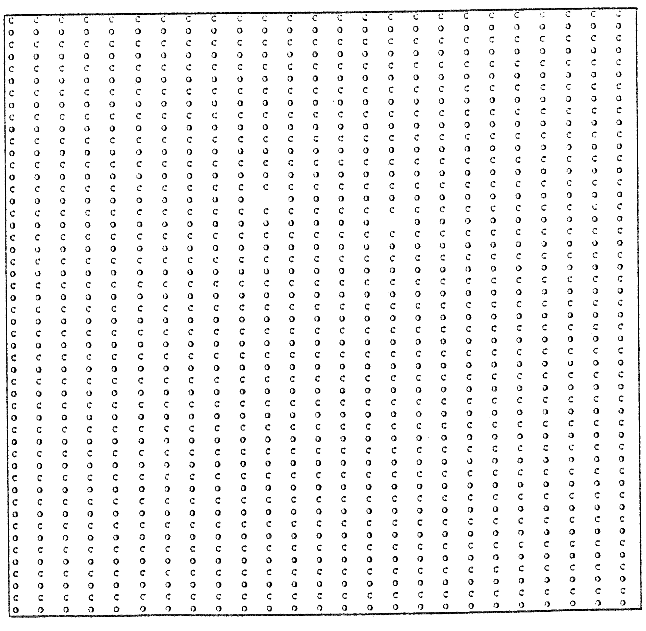


fig 4.8 Final lattice snapshot + = Cu atoms O = oxygen atoms

The X9 Run

A massive run was performed to determine the equilibrium point at kT/V(1) = 0.07

Fig 4.9 implies that 10^9 attempted jumps are required to reach stability at this temperature compared to 10^7 jumps at kT/V(1) = 0.7.

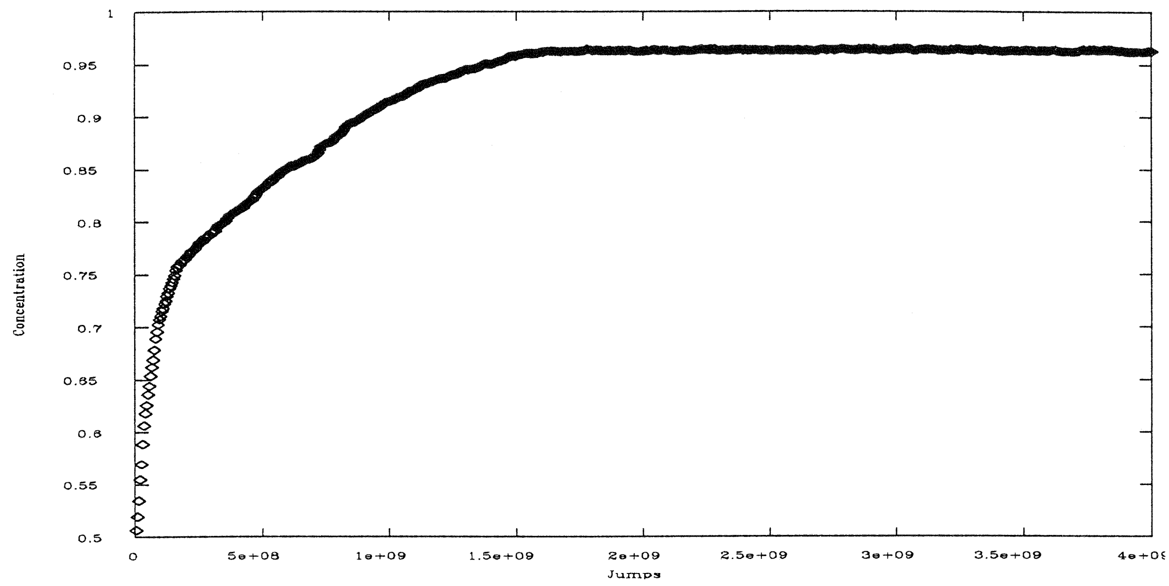


fig 4.9 C vs jumps for HB = 10, kt/v(1) = 0.07

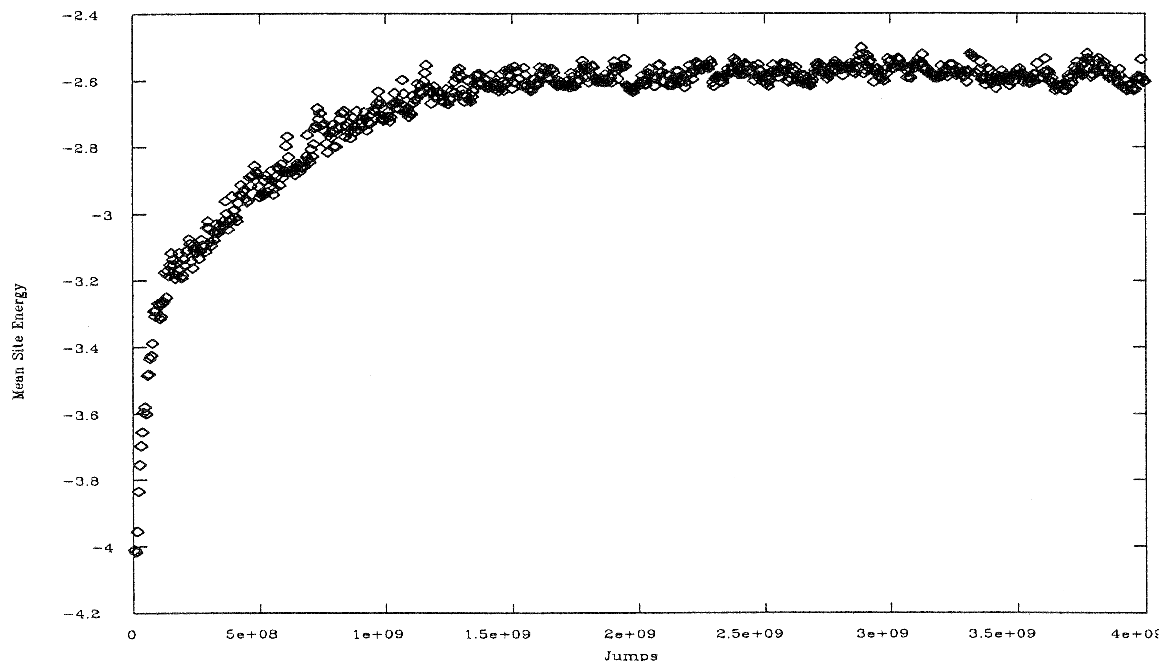


fig 4.10 E vs jumps for HB = 10, kt/v(1) = 0.07

Fig 4.10 indicates rather lower relative variability in the mean site energy than that for kT/V(1)= 0.7 possibly as a result of lower jump rates due to the lower temperature.

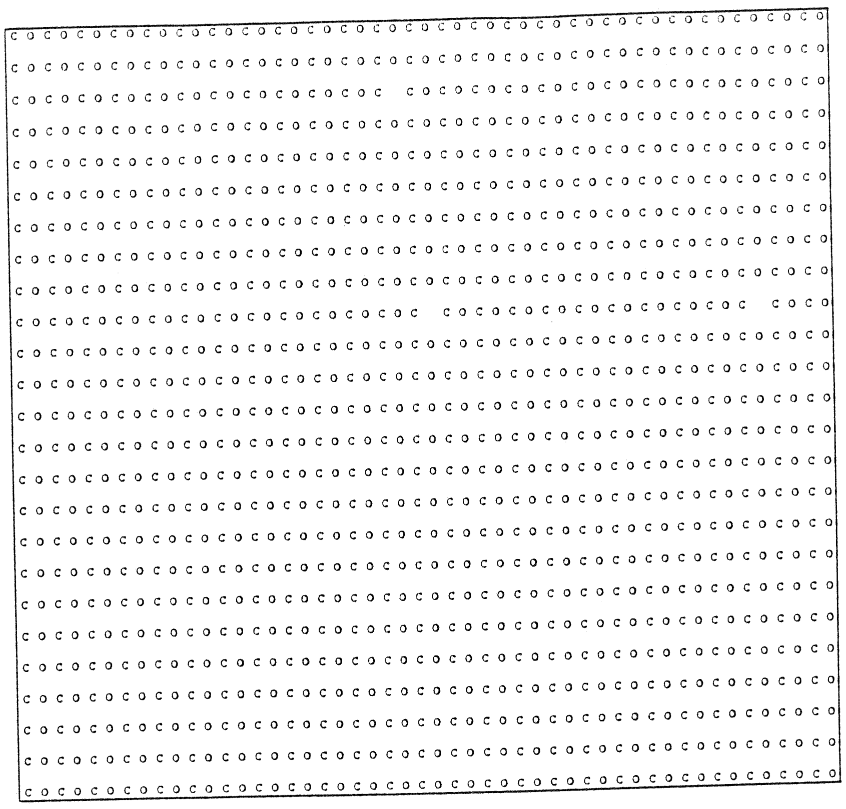


fig 4.11 Final lattice snapshot + = Cu atoms O = oxygen atoms

4.2.2 Summary

At a high temperature of KT/V(1)= 0.7 the lattice requires of the order of 1.5 x 10^7 jumps to reach equilibrium. At a low temperature of KT/V(1)= 0.07 the lattice requires of the order of 2 x 10^9 jumps to reach equilibrium

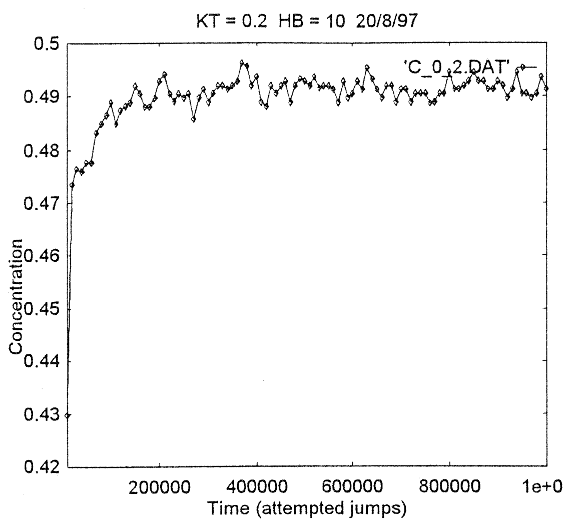
4.3 Testing the Glauber Program

4.3.1 Time to Equilibrium

Glauber permits atomic exchanges over arbitrary distances and thus reaches equilibrium much more rapidly than Kawasaki. The runs were performed on a 50 by 50 lattice to further reduce run time.

Run 1

Run 1 monitored the oxygen concentration against time (attempted site exchanges) at a driving potential (HB) of 10 and a (high) temperature of kt/V(1)= 0.2 (figs 4.12, 4.13). These imply that approx. 1 million attempted site exchanges are required to reach equilibrium.



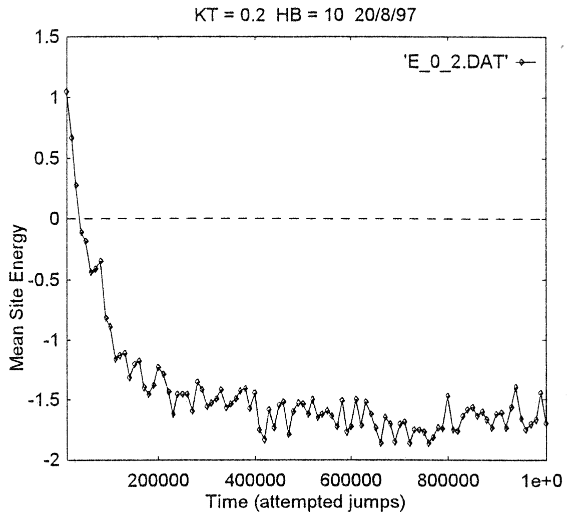
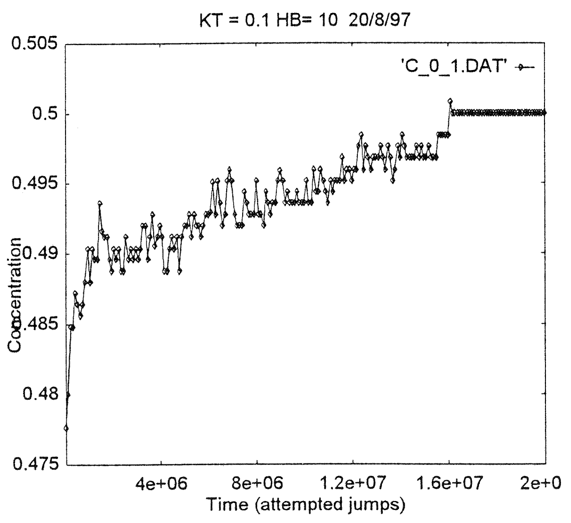


fig 4.12

fig 4.13

Run 2

Run 2 monitored the oxygen concentration against time (attempted site exchanges) at a driving potential of 10 and an intermediate temperature of kt/V(1)= 0.1 (figs 4.14, 4.15).



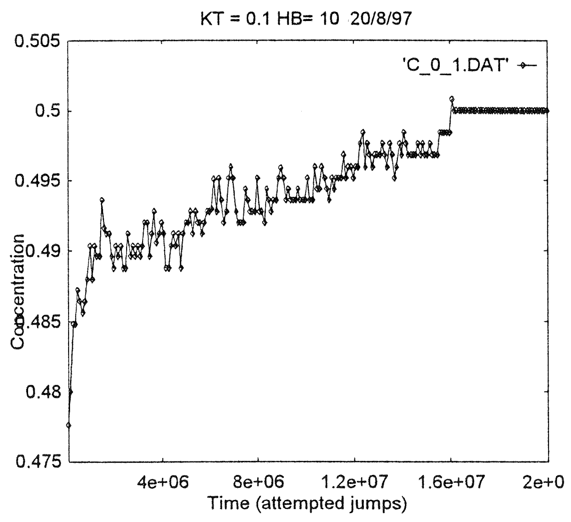


fig. 4.14

fig. 4.15

Run 3

Run 3 monitored the oxygen concentration against time (attempted site exchanges) at a driving potential (HB) of 10 and a (very) low temperature of kt/V(1)= 0.007 (figs. 4.16, 4.17)

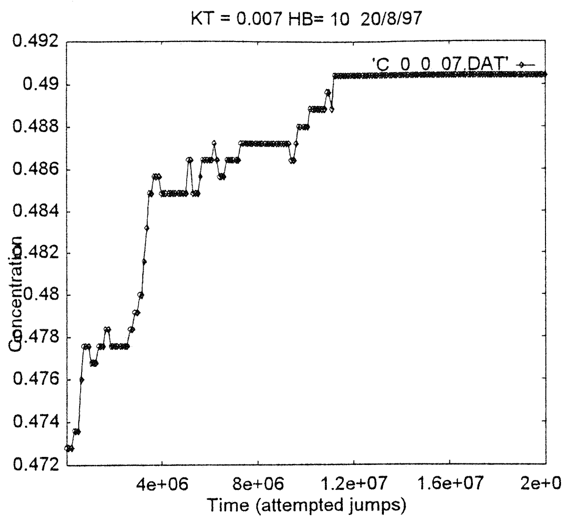


fig. 4.16

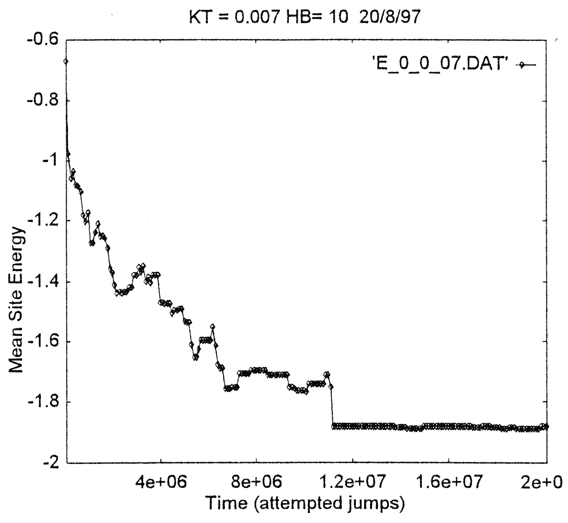


fig. 4.17

4.3.2 Summary

Temperature kT/V(1)	Approximate Time to Equilibrium
0.2	1×10^6
0.1	1.6×10^7
0.07	1.2×10^7

4.4 Overall

G05DAF routine appears to generate strings of pseudo-random numbers which show no marked repetition over runs of 10^7

The Glauber program reaches equilibrium some 100 times faster than the Kawasaki program at low temperatures.

Lattice images generated by both programs show similar structures developing at comparable temperatures.

5
Results

5.1 Phase Changes within the simulations employing Glauber Dynamics

Here the final equilibrium concentration of the lattice was plotted against the chemical potential of the heat bath. Runs were performed at 4 temperatures, kt/V(1) = 0.2, 0.1, 0.07, and 0.01. Some interesting structure in the plots was observed at the lower temperatures including the formation of the OII phase.

5.1.1 High Temperature

Here the temperature KT/V(1) = 0.2. the driving potential of the heat bath (HB) was varied from -10 to +40 in steps of 0.5. After each increment in driving potential the lattice was run for 2 million attempted site exchanges to stabilise. at the end of each increment the standard deviations of the results (concentration and mean site energy) were calculated. However the values were too small to resolve on the graphs. Concentration is normalised to a fully filled lattice, i.e.. the concentration of the OI phase = 0.5.

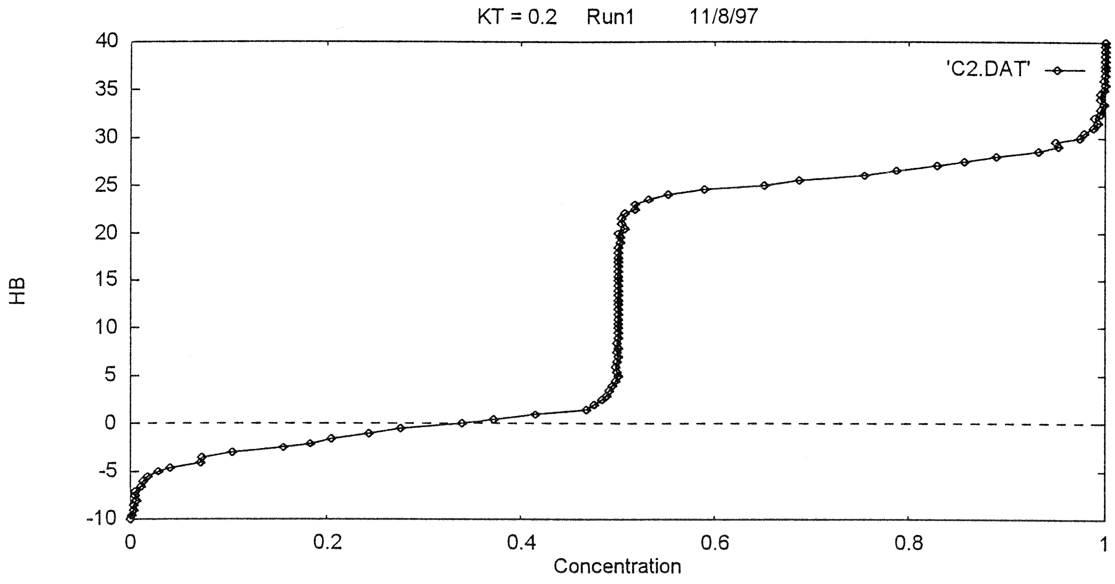


fig 5.1 variation of concentration with driving potential kt/v(1)=0.2

In fig 5.1 we can see that the lattice gradually fills to C=0.5 at which point a highly stable OI phase forms. This resists further ingress of oxygen atoms until the driving potential reaches ~22 when the lattice continues to fill to saturation. Little other structure is immediately apparent.

The plot of mean site n.n interaction energy against driving potential presents a similar structure. The site energy falls to -2.4 mRy, characteristic of the OI phase, mirroring the concentration curve. As the OI phase is disrupted by ingress of oxygen atoms the energy gradually rise to the 25 mRy expected for a fully filled lattice.

Fig 5.3 shows an image of the lattice at HB = 10 exhibiting an expected single region OI phase.

However fig 5.4, an image at HB = 0 shows a mixed phase OI and OII sample. These OII regions are lacking at HB = -5 and +5. I had not expected the OII phase to appear at such a high temperature.

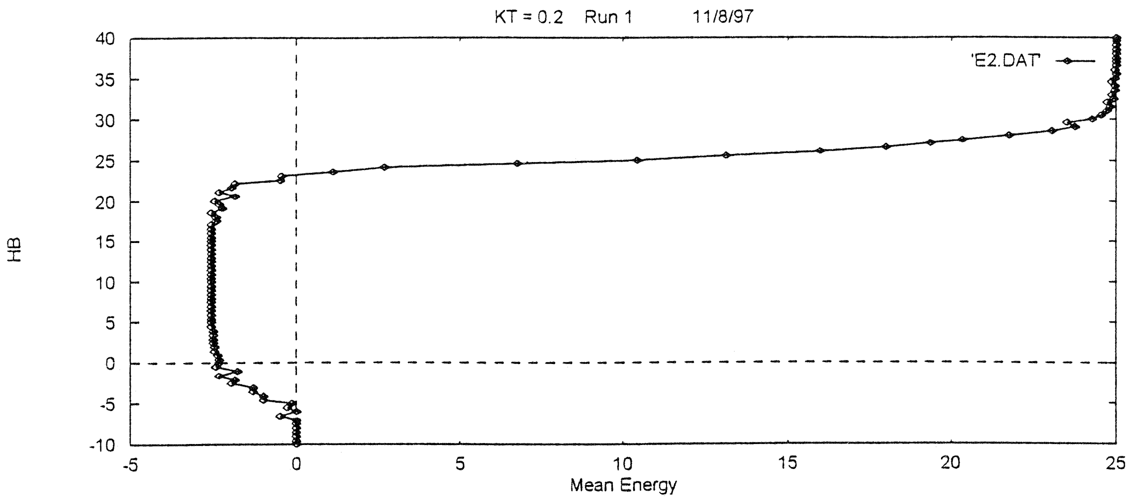
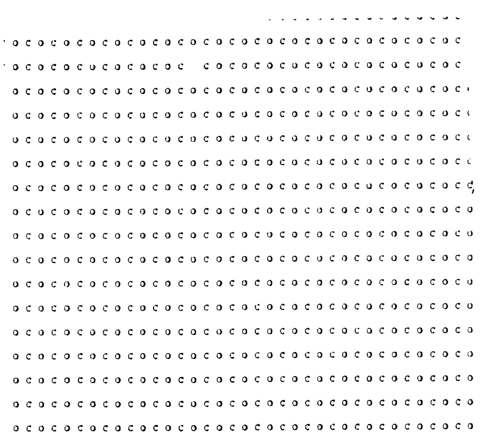


fig 5.2 variation of mean site interaction energy with driving potential



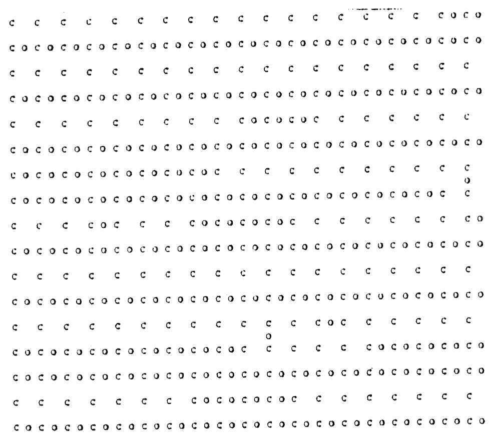


fig 5.3 Lattice at HB = +5 *fig 5.4 Lattice at HB = 0*

5.1.2 Medium Temperature

Temperature KT/V(1) = 0.1. HB was varied from -10 to +40 in steps of 0.25. After each increment in driving potential the lattice was run for 4 million attempted site exchanges to stabilise. All other comments as 5.1.1.
Fig 5.5 has the same general shape as fig 5.1 dominated by a stable OI phase. However there appears to be some discontinuity at HB = -1 and +27. A plot of mean site energy against HB (fig 5.6) shows similar discontinuity at these points. At HB = -1.5 the site energy has dropped to approx. - 4.8 characteristic of the OII region.

A picture of the lattice at HB = -1.5 (fig 5.7) shows that the lattice has initially ordered into the OII phase. By HB = -0.5 (fig 5.8) this has changed to a mixed phase OI and OII. At HB = +0.5 (fig 5.9) this has settled into the OI phase .

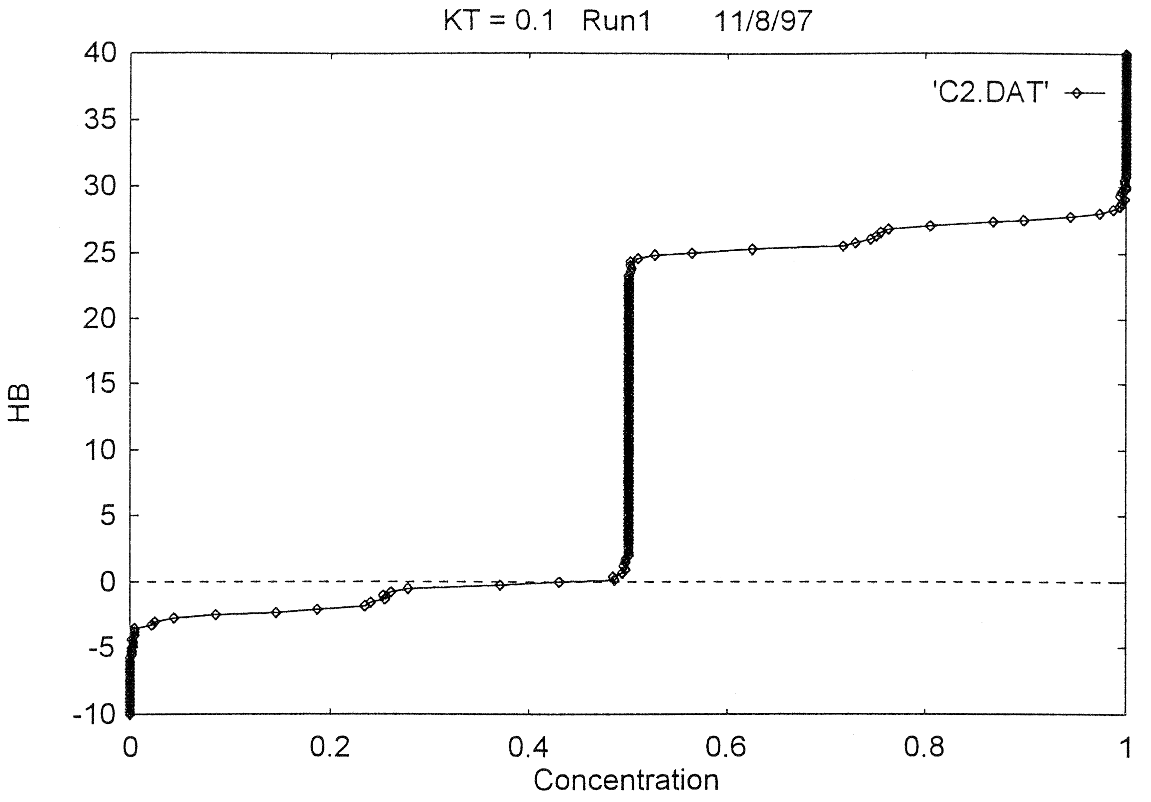


fig 5.5 variation of concentration with driving potential kt/v(1)=0.1

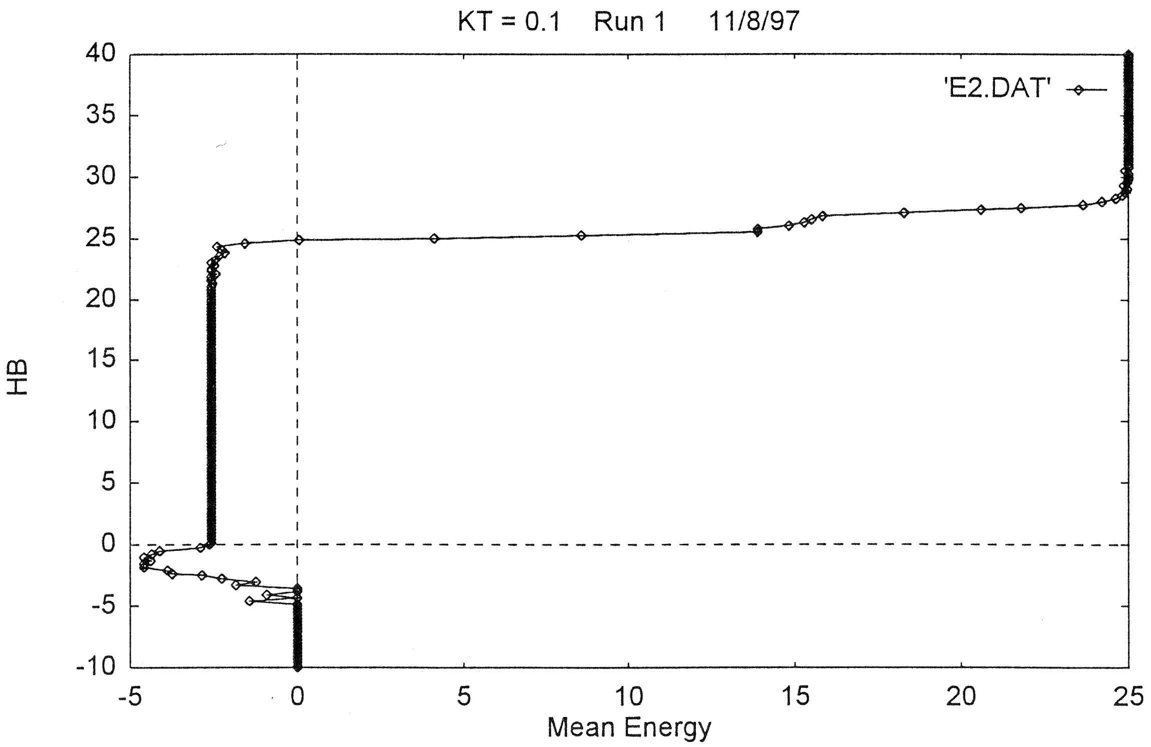


fig 5.6 variation of mean energy with driving potential kt/v(1)=0.1

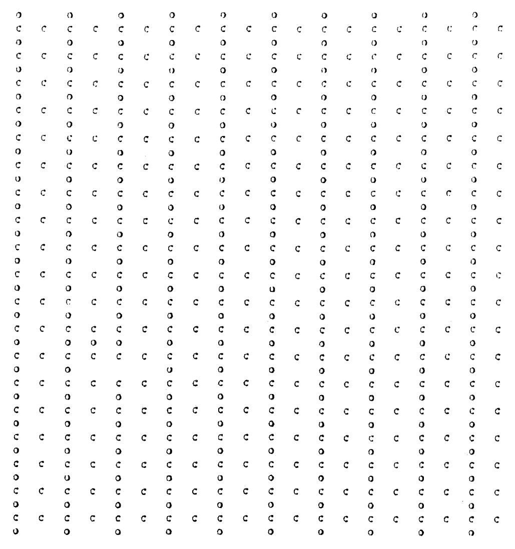


fig 5.7 Lattice at HB = -1.5

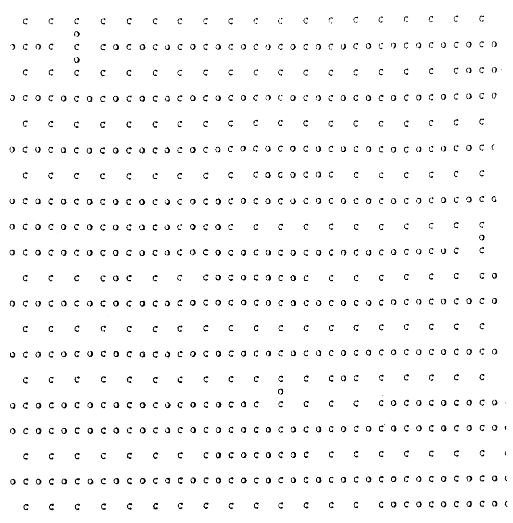


fig 5.8 Lattice at HB = -0.5

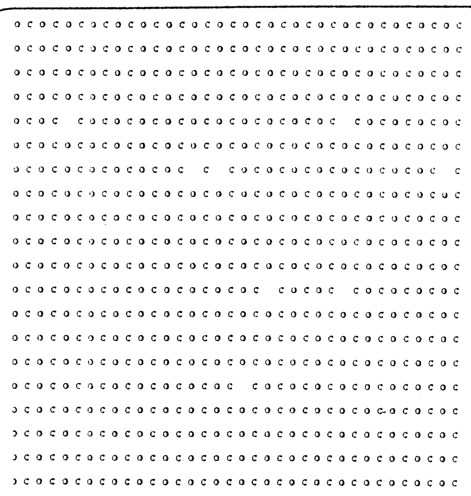


fig 5.9 Lattice at HB = +5

A more detailed investigate of this first discontinuity as carried out with a run over the range HB = -5.5 to +2 incrementing by +0.1.

In fig 5.10 one sees a lessening in oxygen uptake between ~ HB -2.2 and -0.5 which may be attributed to the increased stability of the OII phase. In fig 5.11 we see the mean site energy drop to that of the OII phase -4.6 mRy.

Note: The energy peak at HB = -4.7 has been investigated and appears due to the chance configuration of a few atoms.

fig 5.10

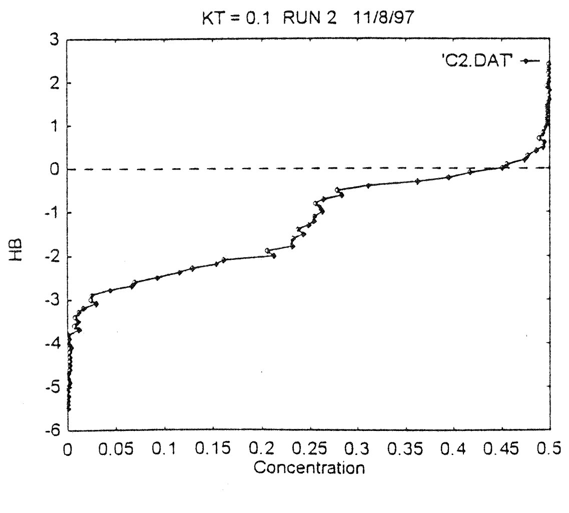
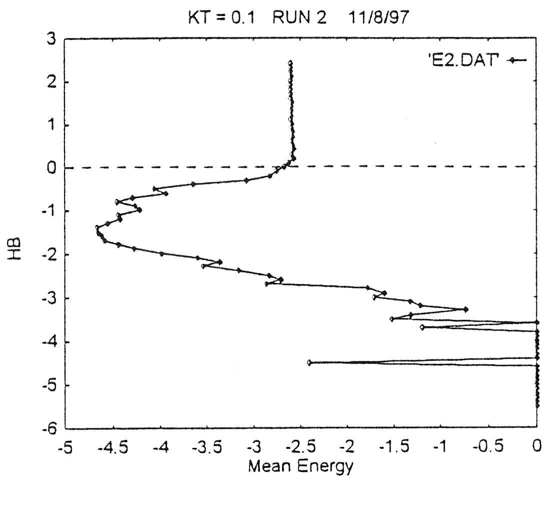
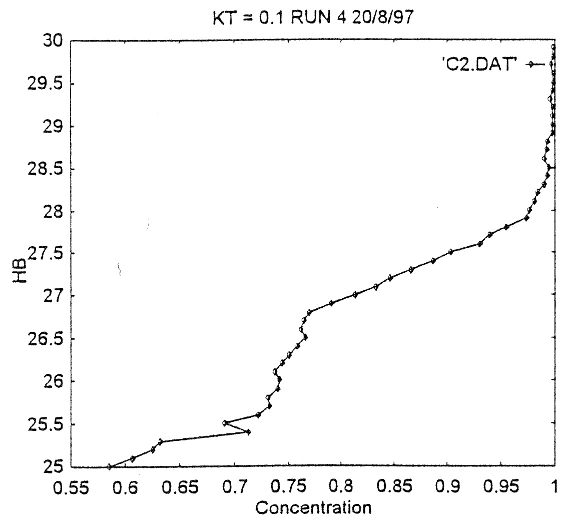


fig 5.11



The upper discontinuity around HB = 26 was also investigated with a higher resolution run. Plots on concentration and mean energy both show clear signs of some form of state change, figs 5.12 and 5.13.



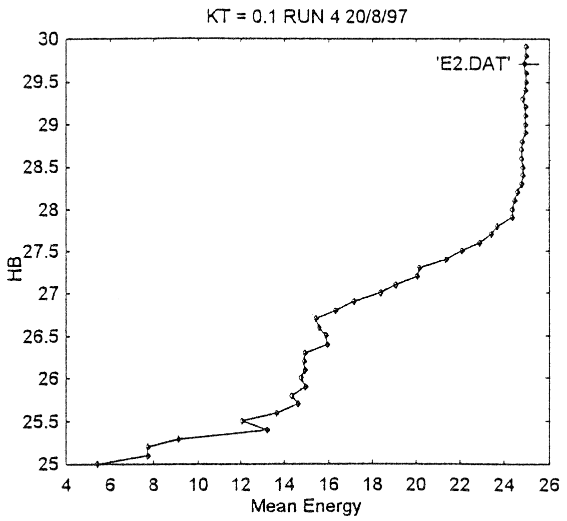
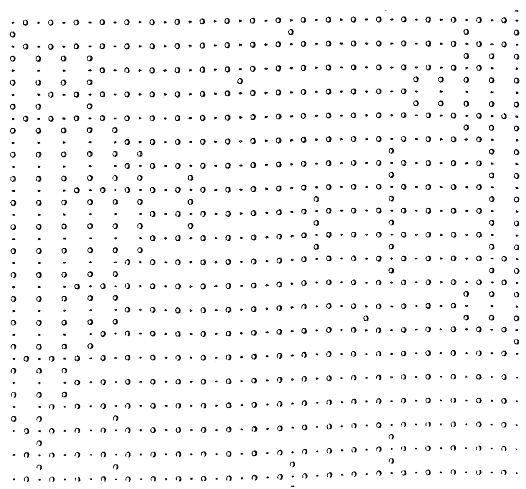


fig 5.12

fig 5.13

Snapshots of the evolution of the lattice reveal the formation of an unusual phase (figs 5.14 to 5.18).

At HB = 25 the sample exhibits a multiple region OI phase, to be expected from the energy plot in fig 5.6. As the driving potential increases to 25.5 the lattice absorbs more oxygen atoms. these preferentially sit on an OII phase running at right angles to and across the existing OI lattice. At HB = 26 this secondary phase is virtually filled. By HB = 26.5 incoming atoms are having to site at the remaining free interstitial sites. By HB = 28 almost all interstitial sites are filled and the mean site energy approaches 25 mRy, that expected for a fully filled lattice.



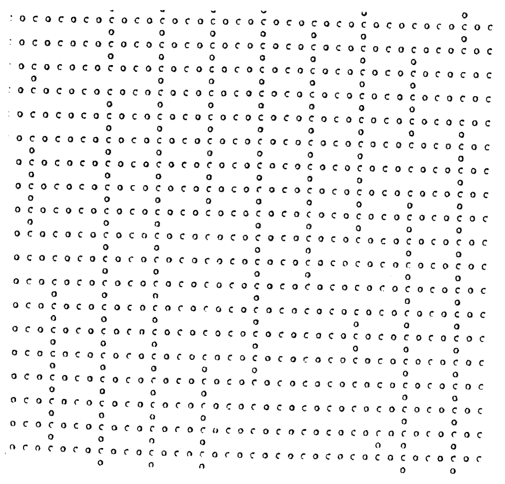


fig 5.14

fig 5.15

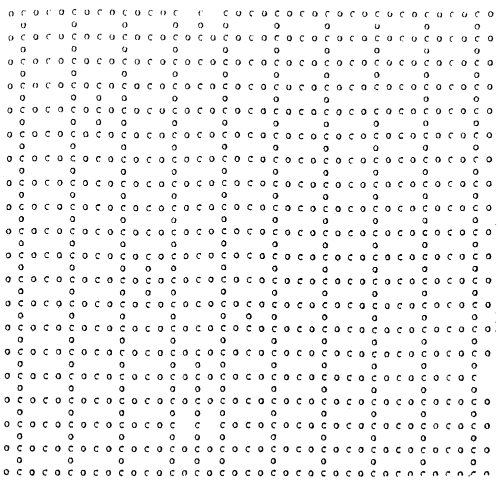


fig 5.16

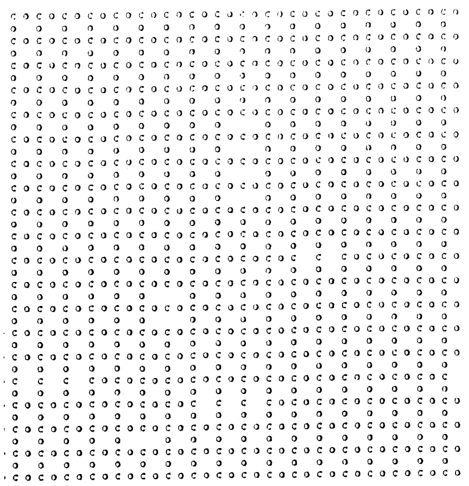
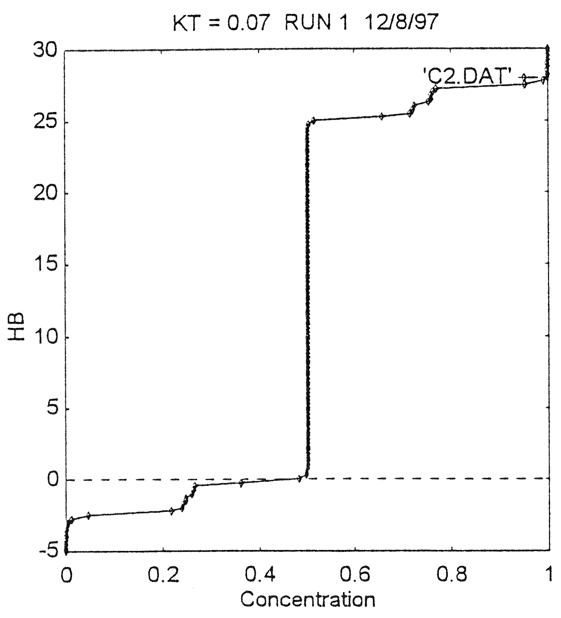


fig 5.17

5.1.3 Low Temperatures

kT/V(1) = 0.07

Temperature kT/V(1) = 0.07. HB was varied from -10 to +40 in steps of 0.25. After each increment in driving potential the lattice was run for 6 million attempted site exchanges to stabilise. All other comments as 5.1.1.



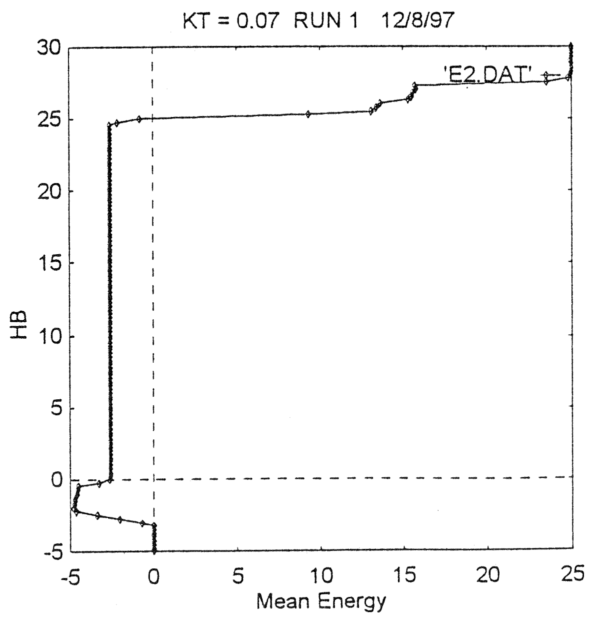


fig 5.18

fig 5.19

At this lower temperature the apparent phase changes at around HB = -2 and above 25 become more abrupt. This was investigated with a lower temperature run.

kT/V(1) = 0.01

Temperature kT/V(1) = 0.01. HB was varied from -1 to +30 in steps of 0.1. After each increment in driving potential the lattice was run for 6 million attempted site exchanges to stabilise. All other comments as section 5.1.1.

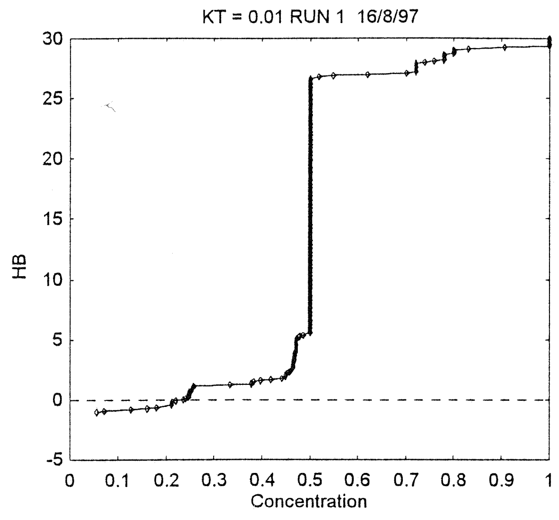


fig 5.20

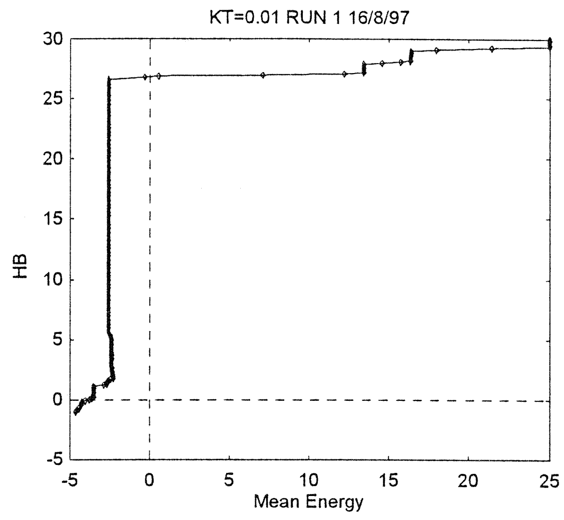


fig 5.21

As with the run at kT/V(1) = 0.07 there appears to be one or more phase changes above HB = 25.

5.1.4 Summary of Glauber Runs

High Temperatures:

The phase change diagram is dominated by an OI phase which is stable over a wide range of driving potential. There is some evidence of a mixed OI/OII at a temperature below this.

Medium Temperatures:

Again a stable OI phase dominates now over a greater range of driving potential. However it appears that two additional phase changes occur below and above this region. The lower potential changes appears to be when the OII region evolves into an OI lattice. The upper phase change occurs as an OII region develops overlying the OI region.

Low Temperatures:

The upper and lower phases changes become more clearly defined and there is limited evidence of further phase changes appearing. The central OI phase change is now stable over an even greater range of chemical potential.

5.2 Attempts to monitor oxygen uptake as a function of Temperature

5.2.1 Glauber Dynamics

Glauber dynamics may not be the best model of atomic diffusion in this case but the rapid times to equilibrium permit limited investigation of oxygen uptake.

A series of runs were performed to investigate potential correlations between oxygen uptake and temperature. Summarised below in fig 5.22 - 5.25.

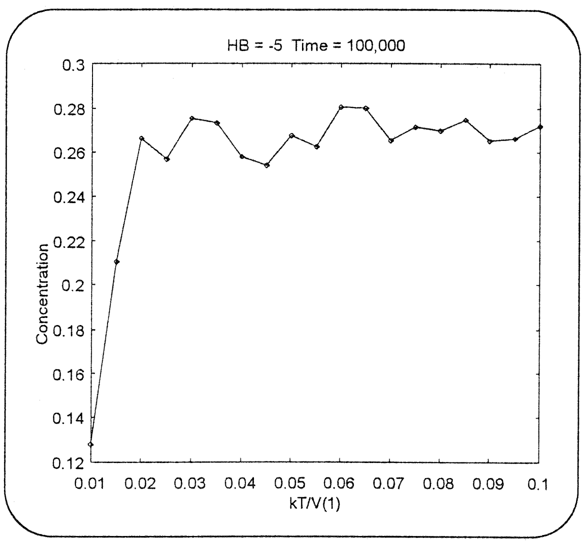


fig 5.22

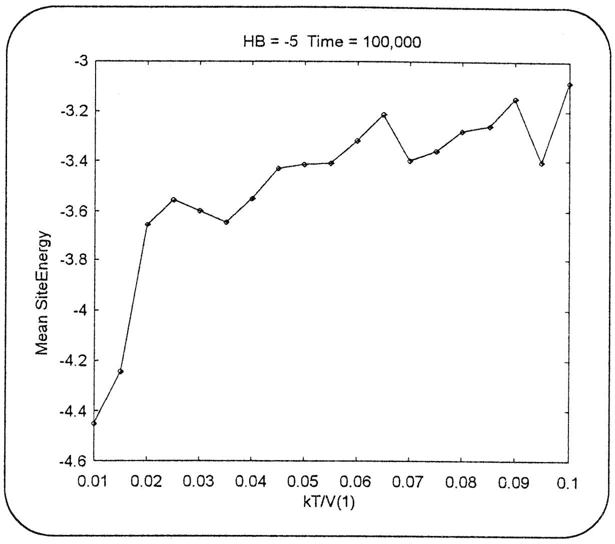


fig 5.23

This run was performed at a relatively low driving potential below that which favours formation of the OI phase. Further runs carried out at higher potentials showed no discernible features as the lattice rapidly settled into a stable OI phase.

The question arises of how to monitor oxygen uptake by the lattice. The experiments detailed in the first part of this chapter (5.1) correlated the final 'settled' concentration of the lattice against driving chemical potential using Glauber dynamics. Clear structures can be seen indicative of some form of phase changes. There is probably insufficient data in these first runs to extract relationships between final oxygen concentration and temperature at a range of driving potentials.

Three questions arise

- How does oxygen uptake vary with time at fixed temperature and driving potential?
- Is the equilibrium concentration of the lattice affected by temperature?
- Is Glauber dynamic a reasonable analogue of this situation or should Kawasaki dynamics be employed?

Time constraints prevent detailed investigation of all three questions however I will attempt some investigation of each.

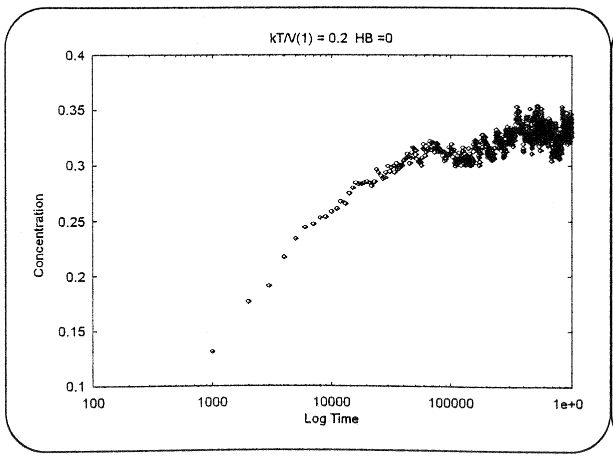
5.2.2 Variation of Oxygen Concentration with Time using Kawasaki and Glauber Dynamics

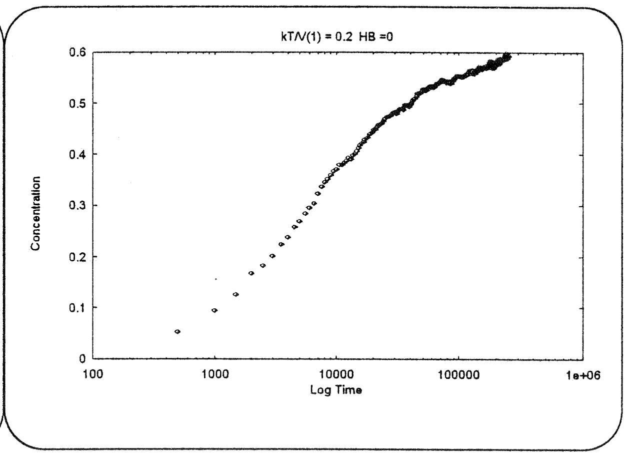
Six runs were performed at a driving potential below that which favours formation of the stable OI phase(HB = 0). Runs were performed at three different temperatures high, intermediate and low using both Kawasaki and Glauber dynamics. The results presented below.

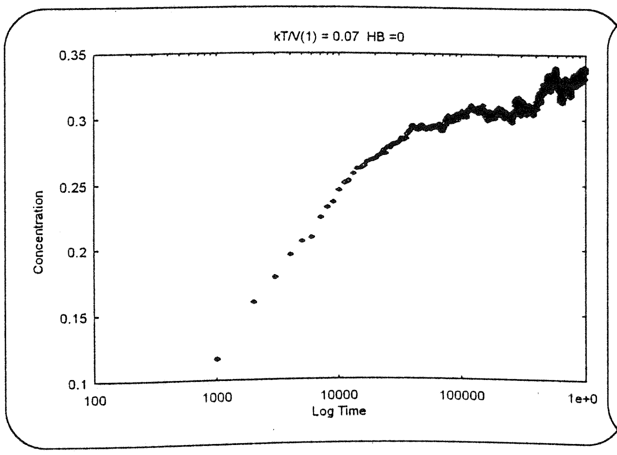
	kT/V(1)	Glauber	Kawasaki
High T	0.2		
Medium T	0.07		
Low T	0.01		

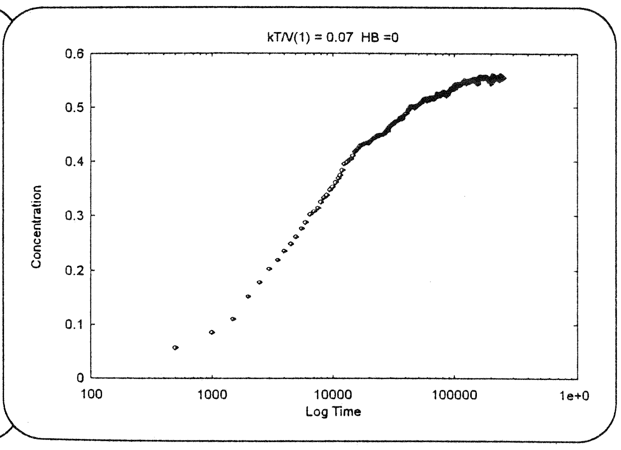
The excessive time to settling using Kawasaki dynamics was reduced by increasing the probability that the program would select an atom from within the heat bath to enter the lattice rather than movement within the lattice. Run times were thus reduced by a factor of 10 to 100.

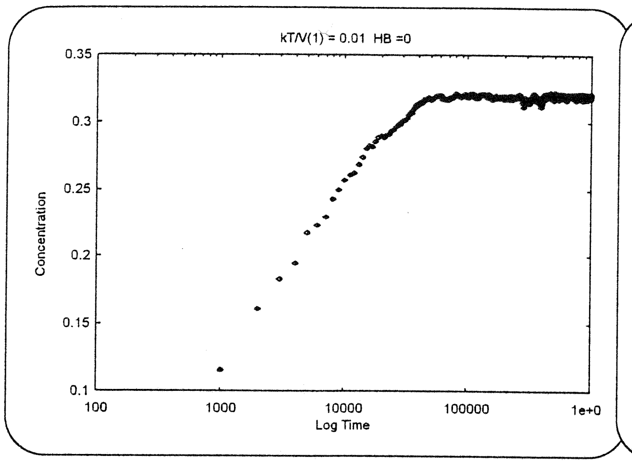
Glauber Dynamics **Kawasaki Dynamaics**

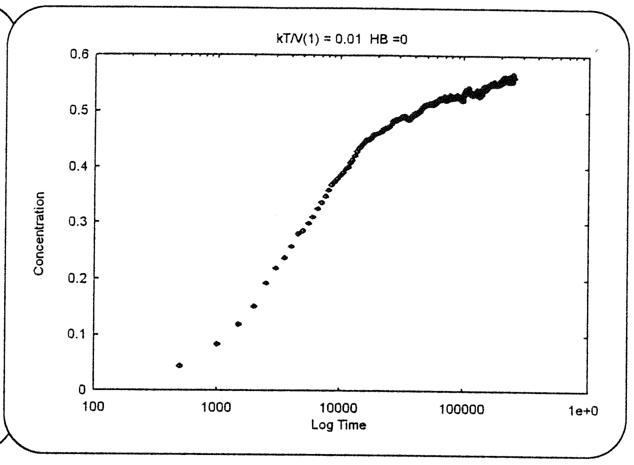












5.2.3 Summary

Firstly there appears to be a difference between the final concentration of the lattices between simulations using Glauber dynamics and Kawasaki dynamics. In the former case the concentration settled at $\simeq 32\%$ of the OII phase while in the latter figure was $\simeq 58\%$. In either case there appeared little correlation between the temperature and final concentration.

kT/V(1)	G: Final Conc %	G: T to 25% Fill	K: Final Conc %	K: T to 25% Fill
0.2	34	8×10^3	59	4.5×10^3
0.07	34	1×10^4	55	5×10^3
0.01	32	1×10^4	56	4×10^3

Direct comparison of rates between the two programs is inappropriate due the accelerated fill techniques used in the Kawasaki program. However it is puzzling that the two programs settle at such different concentrations and I am tempted to think that this is a computational artifact arising from subtle differences between the two.

Conclusions

Two programs have been developed which simulate diffusion with a laminal lattice. These programs have been extensively tested and are fairly well optimised. This was my first extended software project and I have certainly learnt from mistakes. Almost two years were wasted in developing and testing programs which proved to be based on sloppy consideration of the physics. In addition the software developed rather than being strictly planned from the outset. Whilst this form of development is probably typical for small projects it is far from desirable. I now try to ensure that my computing students do not make the same mistake.

The programs generated phase diagrams which appeared to contain some interesting structures with evidence of minor phase changes below and above the OII region. These changes were consistent over a range of initial conditions.

An interesting phase was observed at very high concentrations where OI and OII regions lay across each other. In retrospect this could have been predicted from consideration of the inter-atomic interactions.

Unfortunately it has not been possible for me to relate my results to physical studies. This would be a rather extended project though. I regret this as computational work in isolation is unlikely to offer much insight in this field.

Suggestions for further work

The two programs could be merged into a single engine easily configurable by the user. This should remove any artefacts caused by slight code differences and permit benchmark comparisons between Kawasaki and Glauber dynamics as techniques for monitoring atomic diffusion.

Attempts could be make to link the computational work with the experimental in an extended study.

Ian Robinson April 1998

References

[1] A Monte-Carlo Study into the Oxygen Ordering Processes of the High Temperature Superconductor $Y_1 Ba_2 Cu_3 O_x$

Sean Pittaway

[2] Introduction to Superconductivity and High T_c Materials

M Cyrot, D Payuna

[3] The use of Monte Carlo simulations in the study of a real lattice gas and its application to the α Pd-D system

R A Bond, D K Ross
J.Phys.F Met. Phys. 12(1982) 597-609

[4] Tracer and chemical diffusion of hydrogen in BCC metals

D A Faux, D K Ross

[5] Model simulation study of the structural stability of oxygen order in the $Y Ba_2Cu_3O_{6+x}$ superconductor

J V Anderson, H Bohr, O G Mouritsen
Computational Materials Science 1992 p25-32

[6] Structure Transformations in $YBa_2Cu_3O_{6+d}$ caused by oxygen ordering

S Semenovskaya, A G Khachaturyan
PhysicaD 1993 p205

[7] Oxygen-ordering phenomena in $YBa_2Cu_3O_{6+x}$ studied by Monte Carlo simulation

T Figg, J V Anderson et al
PhysicaD 1993

[8] Physics Review Letters 60, p1065 1988

Wille, Berera, De Fontaine

[9] Proceedings of the Stanford Conference on Superconductivity, P Alto

P Stern
Physica C, 42, 165 (1989)

Appendices

Appendix A

Technical documentation
Kawasaki.for and Glauber.for

Both programs simulate oxygen flow within a laminal lattice mimicking the basal planes of YtBaCuO. The only difference being that Kawasaki permits only nearest neighbour exchanges whilst Glauber allows them over arbitrary distance.

The programs simulate oxygen diffusion into the laminal planes of YBaCuO from an infinite external heat bath. They start by setting up a virtual lattice in a 2d integer array. Fixed Copper atoms are represented by the number -1 and (forbidden) cell centres as -10. Mobile oxygen atoms are represented by +10. The user can randomly prefill the lattice to a required concentration so as to reduce run times.

The programs pick a mobile oxygen at random, from within the lattice or the heatbath. Then select a nearest neighbour site at random. If this is empty the oxygen atom's interaction potentials with its nearest neighbours is summed. Then the oxygen is temporarily moved to the proposed 'jumpto' site and the interaction potentials again summed. If the proposed jump is down an energy gradient it always proceeds. If it requires a rise in interaction potentials then the jump may proceed but with a probability related to the energy rise required (see 'Jump' below).
The programs record the mean site interaction energy and concentration along with their standards deviations over the previous 100 jumps. They may also be configured to output 'snapshots' of the lattice at desired points throughout a run.

Kawasaki and Glauber are highly modular and the main body may be configured to perform runs with varying or fixed chemical potentials of the heat bath (HB) or temperature (kT/V(1)).

Program Modules

Main	Setup	Initialise
IniCu	IniOxy	Clearlatt
Picksite	PickOxy	Jumpto
Kinos	SitexEnergy	Pair
ELattice	Jump	Flow
Convert	ChkLatt	

Setup

Collects together user defined parameters for easy editing. These are the n.n interaction values *V(1)*, *V(2)* and *V(3)*. The temperature *kT* defined as faction of *V(1) and t*he Oxygen concentration as a fraction of a fully filled lattice (*Conc*).
Note: The lattice size cannot be manipulated here and is controlled by the parameter *L* at the beginning of the main body of the program.

Initialise

Prepares the lattice for a fresh run. Calls **Clearlatt** to erase all lattice data, particularly when performing multiple runs. **IniCu** to set up the fixed Copper atoms and **Populate** which randomly populates empty sites with Oxygen atoms to concentration *Conc* (*C* atoms).

Note: Initialise always calls the module **Count_OX** to determine the number of lattice sites availible for oxygen occupancy. Thus saving the user the task of calculating the number of free sites.

Kinos

Kinos is the core module of the program and simulates the movement of mobile atoms around the lattice. Its somewhat awkward structure is an attempt to optimise handling of wrap around and to deal with those atoms moving between the lattice and the external heat bath. **Kinos** randomly chooses an oxygen atom using **PickOxy** determining its interaction energy using **SitexEnergy**. Then **Jumpto** is called to pick a n.n site as the proposed destination. If the destination is within the lattice the site is checked for a vacancy. The lattice is reset assuming that the jump has occurred to permit calculation of the interaction energy at the destination then **Jump** is called to determine whether or not a jump occurs.

If the destination is within the Heat bath (i or j = L+1) then the interaction energy is set simply as *HB*.

SitexEnergy

Determines the Site interaction energy. Sums *V(1)*, *V(2)* and *V(3)* interactions using **Pair** to classify individual n.n interactions.

ELattice

Elattice returns the mean site interaction energy per oxygen atom. One would expect this value to drop as the lattice configuration becomes more homogeneous and settles into a lower energy configuration.

Jump

Jump decides whether or not a jump is to be allowed based upon the difference in interaction energies at the source and destination site. Jumps to a lower energy site are always allowed whilst those to a higher energy destination takes place with a probability exponentially diminishing with increasing energy difference. The normalised jump probability is found from

$$p = e^{\Delta E/Kt}$$

Chklatt

Checks the positions of oxygen atoms in the lattice against those stored in the array *Oxygens* to check inconsistencies indicative of programming errors.

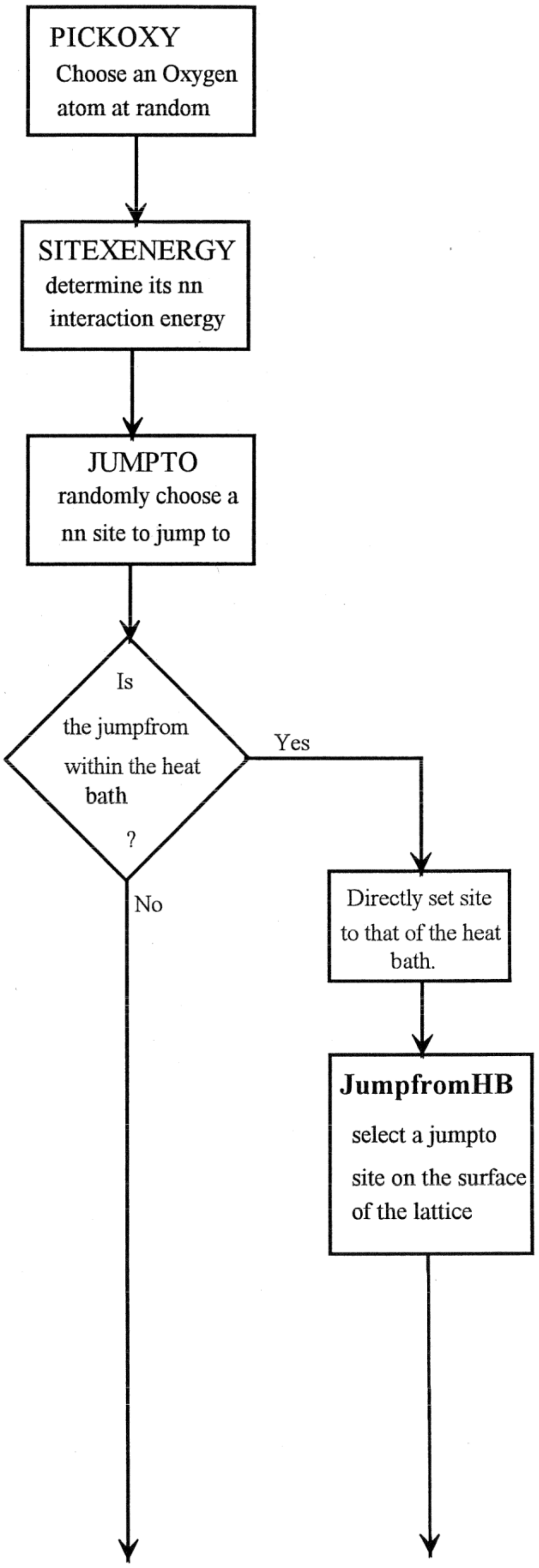
Convert

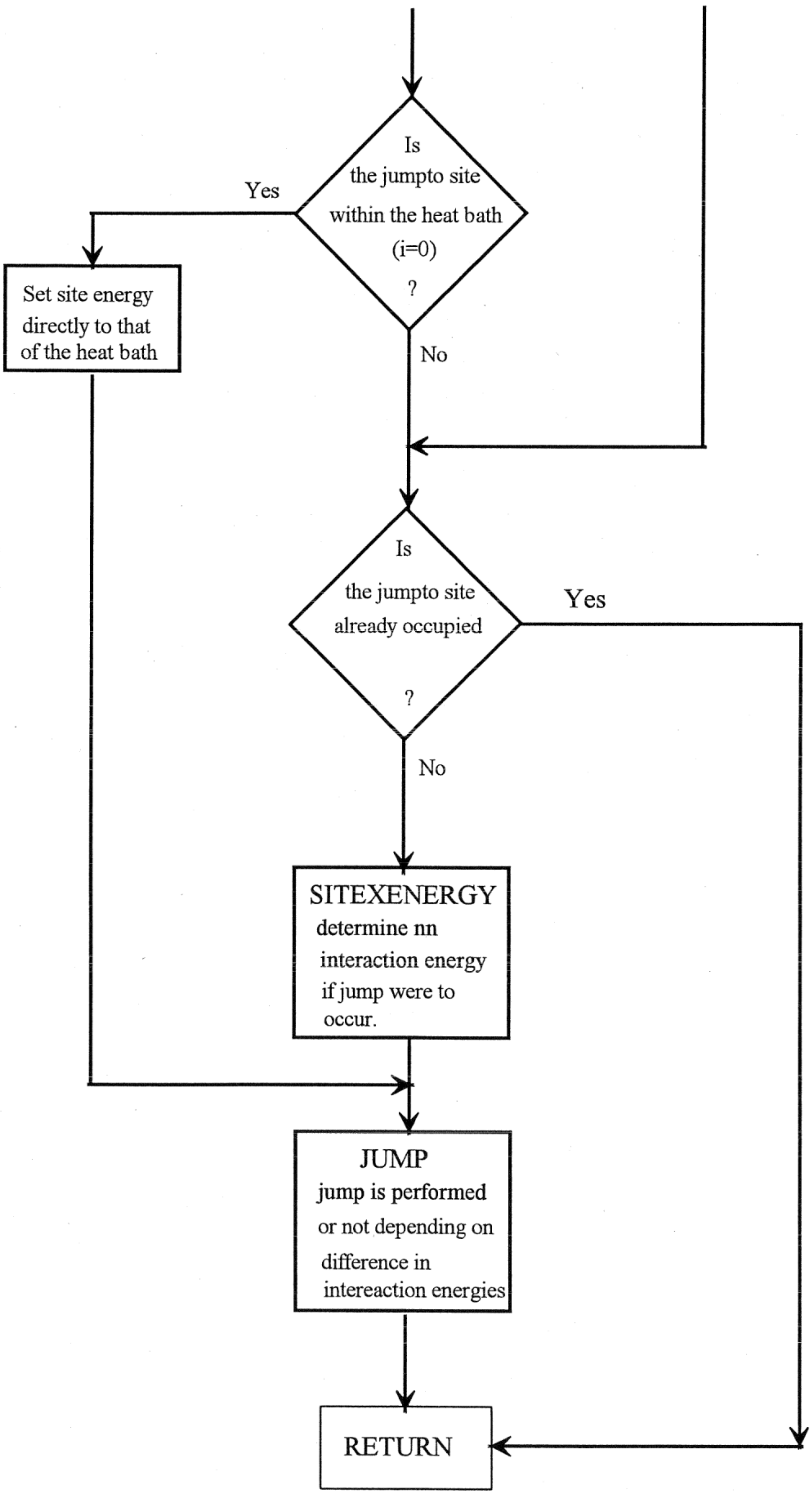
Generates a datafile containing a (crude) ASCII picture of the lattice with copper atoms represented by the letter C and oxygen atoms by the letter O.

Global Variables

	Type	Description
Lattice	Int 2d Array	L by L array representing lattice sites. Cu atoms are represented by -1, each oxygen by a unique integer =>1 and empty sites by 0.
L	Int	Parameter variable denoting linear dimension of array Lattice, typically set to 100.
C	Int	Number of Oxygen atoms within the lattice.
Conc	Real	oxygen concentration as a fraction of fully filled lattice
Att	Int	Number of jump attempts
Act	Int	Number of successful jumps
MCC	Int	Records time elapsed in Monte Carlo cycles
Limit	Int	User defined cap on no of MCCs executed = length of run.
V	Real 1d array	V(1),(2) and (3) store the pairwise oxygen interaction energies.
Conc	Real	Oxygen concentration normalised w.r.t proportion of OII phase.
KT	Real	Temperature term.
Elat	Real	Mean oxygen site energy.
CvsHB	Real	Stores variation of concentration with variation in chemical potential of the heat bath
EvsHB	Real	Stores variation of mean site energy with variation on driving chemical potential.

Kinos Routine


PICKOXY
Choose an Oxygen atom at random
SITEXENERGY
determine its nn interaction energy
JUMPTO
randomly choose a nn site to jump to
Is the jumpfrom within the heat bath ?
Yes
No
Directly set site to that of the heat bath.
JumpfromHB
select a jumpto site on the surface of the lattice


Is the jumpto site within the heat bath (i=0) ?
Yes
No
Set site energy directly to that of the heat bath
Is the jumpto site already occupied ?
Yes
No
SITEXENERGY determine nn interaction energy if jump were to occur.
JUMP jump is performed or not depending on difference in intereaction energies
RETURN

Appendix C

Technical documentation
Histogrm.For + Repitit.for

Histogrm.for

Histogram is a simple program which generates a random number (Q) in the range 1 to 100. This is converted into an integer and that element of the Results array is incremented by one. The array Results is written to a datafile HResults.txt .

Main Variables

	Type	**Description**
N	Int	No of random numbers to be generated
P	Int	Random number in the range 1 to 100
Results	Int 2d array	stores histogram of random numbers generated

Repitit.for

Repitit.for is slightly more complex than histogram. It generates N random numbers of which the first 1000 are stored in the first row of a 2d array. The next 1000 numbers generated are stored in the second row of the array. Subsequent random numbers are generated individually and stored in the second row of the array in cyclic order. i.e.. the first is placed at site 1, to second at site 2 and so on cycling back to position 1 at the end of the array. Every time a new number is inserted at the end of the list the elements of the first row, the initial string of 1000 numbers, are compared with those of the second row with element i of the first row being compared with i + dX of the second where dX is the position of the last new number inserted. this has the effect of comparing the first l000 with the last 1000 numbers generated. Any number pairs are classed as a correlation and a counter J is incremented. Another counter Flag is used to store the length of the last common sequence. When a dissimilar pair is found Flag is reset to zero. A counter MFlag stores the maximum value of Flag and thus the maximum repeated sequence found in the entire run of N numbers.
Given the quantity of numbers generated it is impractical to store all of the J results. If J exceeds an arbitrary value (in these runs 24) J and the total offset between the initial list and the start of the current 1000 numbers are stored in an array ResultsA.

At the end of the run ResultsA is written to a datafile RepititA.txt. A note of the run length and the value of MFlag are written to the file Flags.txt.
The program also generates a Histogram of the correlations (J values). this is written to a file REPDISTB.txt .
Should no correlations be found above the threshold the nonsensical values -1,-1 are written to RepititA.

Main Variables

	Type	**Description**
N	Int	No of random numbers to be generated
P	Int	Random number in the range 1 to 100
J	Int	No of number pairs found
Flag	Int	Length of last repeated sequence
Mflag	Int	Length of maximum repeated sequence
List	Int 1d array	Stores sequence of 2000 random numbers
ResultsA	Int 1d array	Stores correlations vs dX (offset)
Distrib	Int 1d array	Stores histogram of correlations

Appendix D

User Documentation

Glauber.For

As with Kawasaki.for Glauber may be configured for two types of investigation, the measurement of diffusion rate against the driving chemical potential of the heat bath at fixed temperature or the measurement of diffusion rate against temperature at fixed driving potential. Unlike Kawasaki it is impractical to measure both in a single run on a workstation owing to the computational load.
Changing the size of the lattice is performed by changing the parameter value L in line 9 to generate an L*L lattice.

The loop statement on line 25 may be altered to produce a series of runs at differing values of chemical potential (HB) or differing temperatures (kT). The length of a run is controlled by two nested loop statements on lines 29 and 33. That on line 33 controls the run time between readings whilst that on 29 controls to number of readings to be taken with an upper limit of 10,000.

Three data files are generated;
CvsHB.dat lists driving chemical potential against final concentration and its standard deviation over the last 10 readings.

EvsHb.dat lists chemical potential against mean site interaction energy and its standard deviation over the last 10 readings.

Picture.txt stores an ASCII representation of the lattice at the end of the runs.

For a configuration of varying temperature the data written to these needs to be altered. Line 51 should read CvsHB(run,1) = kT and line 54 EvsHB(run,1) = kT.

Histogram.For

The only value a user is likely to wish to enter is that of the number of random values to be generated which is controlled by the variable N on line 14.

Repitit.For

A user is only likely to wish to change two parameters here, the number of random values to be generated and the arbitrary threshold above which a discrete record is kept of correlations vs distance from the initial list.
The variable N on line 18 describes the length of the run in numbers to be generated.
The threshold value is contained within the IF(....) statement on line 75. One expects to see an average of 10 correlations per sequence and the default threshold is 24 correlations.

Appendix E

Program Listings

i) Kawasaki.for

ii) Glauber.for

iii) Histogram.for

iv) Glauber.for

```
c               KAWASAKI.FOR
c
c
c   +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
c   A program to simulate oxygen diffusion into the Cu-O laminal
c   planes of YBaCuO using Kawasaki dynamics (n.n neighbour jumps only).
c   +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
c   MAIN:

    INTEGER L
c   l = unit cell dimensions ((n*3)-1)
    PARAMETER (L = 50)

    INTEGER LATTICE(L,L),OX,X,Y,RUN
    REAL KT,HB,V(3),ENERGY(1000,2),Cresults(1000,2),Emean,CONC
    REAL CvsHB(1000,3),EvsHB(1000,3),Efinal,Cfinal,ESD,CSD

    REAL G05DAF
    EXTERNAL G05DAF

    RUN = 0

    CALL SETUP(V,KT,CONC)
    CALL INITIALISE(LATTICE,L,OX,CONC,V,KT)

    HB = 10

c   run the lattice for a while in the initial state to stabilise it
5       DO 7 X = 1,10000000
              CALL KINOS(LATTICE,L,V,KT,HB)
7       CONTINUE

c   this particular configuration varies the chenical potential of the
c   heat bath (HB) whilst keeping the temperature constant.

9   DO 50 HB = 26.5,29.5,0.02
c   run is used to space the sampling  ..see line labelled 46..
    RUN =  RUN + 1

10       DO 11 X = 1,9900
              CALL KINOS(LATTICE,L,V,KT,HB)
11       CONTINUE

c       final 100 runs to calculate means + msd's of results
12       DO 45 X = 1,100
              CALL KINOS(LATTICE,L,V,KT,HB)
              CALL COUNT(LATTICE,L,OX,CONC)
              Cresults(X,1) = X
              Cresults(X,2) = CONC

              CALL MEANENERGY(LATTICE,L,Emean,KT,V)
              ENERGY(X,1) = X
              ENERGY(X,2) = Emean
45       CONTINUE

c   calculate mean conc + energy (+msd's) over last 100 jump attemps
    CALL FINALENERGY(ENERGY,Efinal,ESD,X)
    CALL FINALCONC(Cresults,Cfinal,CSD,X)
```

```
c     store the final energy,conc and their sd's in the two data arrays
46    CvsHB(RUN,1) = HB
      CvsHB(RUN,2) = Cfinal
      CvsHB(RUN,3) = CSD
      EvsHB(RUN,1) = HB
      EvsHB(RUN,2) = Efinal
      EvsHB(RUN,3) = ESD


c     call up the routines to output images of the lattice
c     best to use the integer RUN rather than an IF equality containing
c     the real no's HB or temperature KT.
      IF (RUN.EQ.1) THEN
            CALL CONVERT0(LATTICE,L)
      ENDIF
      IF (RUN.EQ.20) THEN
            CALL CONVERT1(LATTICE,L)
      ENDIF
      IF (RUN.EQ.21) THEN
            CALL CONVERT2(LATTICE,L)
      ENDIF
      IF (RUN.EQ.22) THEN
            CALL CONVERT3(LATTICE,L)
      ENDIF
      IF (RUN.EQ.24) THEN
            CALL CONVERT4(LATTICE,L)
      ENDIF
      IF (RUN.EQ.65) THEN
            CALL CONVERT5(LATTICE,L)
      ENDIF
      IF (RUN.EQ.70) THEN
            CALL CONVERT6(LATTICE,L)
      ENDIF
      IF (RUN.EQ.75) THEN
           CALL CONVERT7(LATTICE,L)
      ENDIF
      IF (RUN.EQ.80) THEN
            CALL CONVERT8(LATTICE,L)
      ENDIF
      IF (RUN.EQ.281) THEN
            CALL CONVERT9(LATTICE,L)
      ENDIF


50    CONTINUE

c     output conc and energy results to two data files
      CALL RESULTS(CvsHB,EvsHB,RUN)

      END

c     ==================================================================
c          subroutine to initialise user defined parameters
      SUBROUTINE SETUP(V,KT,CONC)
      REAL V(3),KT,CONC
```

```
c     set n.n interaction parameters
      V(1) = 6.90
      V(2) = -2.40
      V(3) = 1.10

c     set temperature (if not varying in the main loop above)
      KT = V(1)*0.01

c     set initial concentration .. usefull for long low-temperature runs
      Conc = 0.00

      RETURN
      END

c     ==================================================================

c           subroutine to clear and initialise the lattice array
      SUBROUTINE INITIALISE(LATTICE,L,OX,CONC,V,KT)
      INTEGER LATTICE,L,OX
      REAL CONC,V(3),KT

c     empty the lattice array
      CALL CLEARLATT(LATTICE,L)

c     add fixed copper atoms
      CALL INICU(LATTICE,L)

c     count the number of intersitial sites (max no of oxygens)
      CALL COUNT_OX(LATTICE,L,OX)

c     prepopulate lattice to initial concentration 'conc'
      CALL POPULATE(LATTICE,L,OX,CONC)

      RETURN
      END
C     ==================================================================
c     subroutine to delete contents of lattice
      SUBROUTINE CLEARLATT(LATTICE,L)
      INTEGER L,LATTICE(L,L),I,J


c     clears the lattice
      DO 130 I = 1,L
            DO 120 J = I,L
                  LATTICE(I,J) = 0
120         CONTINUE
130   CONTINUE

      RETURN
      END

c     ==================================================================
c           subroutine to initialise the copper atoms and block cell
c                       centers

      SUBROUTINE INICU(LATTICE,L)
      INTEGER  L,LATTICE(L,L),I,J
```

```
c        setting the cu atoms (as -1), row by row

      DO 20 J = 1,L,2
            DO 10 I = 1,L,2

                  LATTICE(I,J) = -1
10          CONTINUE
20    CONTINUE

c        setting the site centres as (-10)
      DO 40 J = 2,L,2
            DO 30 I = 2,L,2
                  LATTICE(I,J) = -10
30          CONTINUE
40    CONTINUE

      RETURN
      END

c     ====================================================================
C     subroutine to count the number of interstitial sites
      SUBROUTINE COUNT_OX(LATTICE,L,OX)
      INTEGER L,LATTICE(L,L),OX,I,J

c     ox is the total no of sites for mobile oxygens within the lattice
      OX = 0

      DO 40 J = 1,L
            DO  30 I = 1,L
                  IF (LATTICE(I,J).GT.-1) THEN
                        OX = OX + 1
                  ENDIF
30          CONTINUE
40    CONTINUE


      RETURN
      END

c     ====================================================================
c     subroutine to randomly populate the lattice to concentration CONC
      SUBROUTINE POPULATE(LATTICE,L,OX,CONC)
      INTEGER L,OX,LATTICE(L,L),N,I,J,A
      REAL CONC
      REAL G05DAF

c     n = no of oxygens required for concentration of 'conc'
c     a = no of oxygens placed so far

      N = INT(CONC*OX)
      A = 0

c     randomly pick a site using subroutine picksite
110   CALL PICKSITE(L,I,J)

c     test for out of range values as picksite can select a site within
c     the heat bath
      IF(I.GT.(L).OR.I.LT.1.OR.J.LT.1.OR.J.GT.(L)) THEN
```

```
            GOTO 110
      ENDIF

c     if site is vacant then place an oxygen
      IF (LATTICE(I,J).EQ.0) THEN
            LATTICE(I,J) = 10
            A = A+1
      ENDIF


      IF(A.LT.N) THEN
            GOTO 110
      ENDIF

      RETURN
      END

c     =====================================================================
c           subroutine to move the atoms
      SUBROUTINE KINOS(LATTICE,L,V,KT,HB)
      INTEGER L,LATTICE(L,L),I,J,P,Q
      REAL V(3),E1,E2,ESITE,KT,HB
      REAL G05DAF


c     i,j are the coordinates of the jumpfrom site
c     p,q are the coordinates of the jumpto site
      E1 = 0.0
      E2 = 0.0
      ATT = ATT+1

      CALL PICKOXY(LATTICE,L,I,J,IFAIL)
c     if jump is from h:b then set energy as kt directly,call jumpfromhb
      IF(I.LT.1.OR.I.GT.L.OR.J.LT.1.OR.J.GT.L) THEN
            E1 = HB
            CALL JUMPFROMHB(LATTICE,L,I,J,P,Q)
            GOTO 305
      ENDIF
c           determine energy of 1st site
      CALL SITEXENERGY(LATTICE,L,I,J,E1,V,KT)
301   CALL JUMPTO(LATTICE,L,I,J,P,Q)

c     test whether jump is into heatbath or sink and set e2 directly
      IF(P.LT.1.OR.P.GT.L.OR.Q.LT.1.OR.Q.GT.L) THEN
            E2 = HB
            GOTO 310
      ENDIF

c              test for vacancy at jumpto site
305   IF(LATTICE(P,Q).NE.0) THEN
            GOTO 390
      ENDIF

c           reset lattice as if jump has occured before calculating e2
      LATTICE(P,Q) = 10

c     if jump from hb no need to set hb site to zero (outside lat range)
      IF(I.LT.1.OR.I.GT.L.OR.J.LT.1.OR.J.GT.L) THEN
```

```
            CALL SITEXENERGY(LATTICE,L,P,Q,E2,V,KT)
            LATTICE(P,Q) = 0
            GOTO 310
      ENDIF

      LATTICE(I,J) = 0
      CALL SITEXENERGY(LATTICE,L,P,Q,E2,V,KT)
      LATTICE(I,J) = 10
      LATTICE(P,Q) = 0

310   CALL JUMP(LATTICE,L,KT,I,J,P,Q,E1,E2)

390   RETURN
      END
c     ================================================================
c           subroutine to perform the jump (or not)
      SUBROUTINE JUMP(LATTICE,L,KT,I,J,P,Q,E1,E2)
      INTEGER L,LATTICE(L,L),I,J,P,Q
      REAL E1,E2,R,PROB,KT,DE
      REAL G05DAF


      DE = E1-E2
c           if dE >= 0 then jump
      IF(dE.GE.-0.001) THEN
            GOTO 510
      ENDIF

      R = G05DAF(0.,1.)

      PROB = EXP(dE/(KT))
c           as dE becomes more negative, prob decreases hence jump
c           likelyhood dereases.

      IF(PROB.GE.R) THEN
            GOTO 510
      ENDIF

c     no jump so return

      GOTO 520

c     do jump
510   ACT = ACT + 1

c     reset lattice (if i,j   or p,q are within it and not in h:b)
      IF (I.LT.1.OR.I.GT.L.OR.J.LT.1.OR.J.GT.L) THEN
            GOTO 515
      ENDIF
      LATTICE(I,J) = 0
515   IF(P.LT.1.OR.P.GT.L.OR.Q.LT.1.OR.Q.GT.L) THEN
            GOTO 520
      ENDIF
      LATTICE(P,Q) = 10
520   RETURN
      END

c     ================================================================
c         subroutine to sum a site's pairwise interaction energy
      SUBROUTINE SITEXENERGY(LATTICE,L,I,J,ESITE,V,KT)
      INTEGER L,LATTICE(L,L),I,J,X,Y,NN
      REAL ESITE,V(3),KT

      ESITE = 0.0

c         i,j,p,q are all in the range 1:L

c     ***for oxygens at edge of lattice PAIR assumes that nn oxygen
c     interactions occur only within the lattice (cf: surface tension)
c     sum V(1) terms
      NN = 0

      X = I + 1
      Y = J + 1
      CALL PAIR(LATTICE,L,I,J,X,Y,NN)

      X = I + 1
      Y = J - 1
      CALL PAIR(LATTICE,L,I,J,X,Y,NN)

      X = I - 1
      Y = J + 1
      CALL PAIR(LATTICE,L,I,J,X,Y,NN)

      X = I - 1
      Y = J - 1
      CALL PAIR(LATTICE,L,I,J,X,Y,NN)

      ESITE = ESITE + (V(1)*NN)

c     sum V(2) terms
      NN = 0

c     look for nn copper atoms
      X = I + 1
      Y = J
      IF(X.GT.L) THEN
            X = 1
      ENDIF
      IF (LATTICE(X,Y).EQ.-1) THEN
c         nn = cu
            X = I + 2
            CALL PAIR(LATTICE,L,I,J,X,Y,NN)
      ENDIF

      X = I - 1
      Y = J
      IF(X.LT.1) THEN
            X = L
      ENDIF
      IF (LATTICE(X,Y).EQ.-1) THEN
c         nn = cu
            X = I - 2
            CALL PAIR(LATTICE,L,I,J,X,Y,NN)
      ENDIF
      X = I
      Y = J + 1
      IF (Y.GT.L) THEN
            Y = 1
      ENDIF
      IF (LATTICE(X,Y).EQ.-1) THEN
c        nn = cu
            Y = J + 2
            CALL PAIR(LATTICE,L,I,J,X,Y,NN)
      ENDIF

      X = I
      Y = J - 1
      IF (Y.LT.1) THEN
            Y = L
      ENDIF
      IF (LATTICE(X,Y).EQ.-1) THEN
c        nn = cu
            Y = J - 2
            CALL PAIR(LATTICE,L,I,J,X,Y,NN)
      ENDIF

      ESITE = ESITE + (V(2)*NN)

c     sum V(3) terms
      NN = 0

      X = I + 1
      Y = J
      IF (X.GT.L) THEN
            X = 1
      ENDIF
      IF (LATTICE(X,Y).EQ.-10) THEN
c        nn = site centre
            X = I+2
            CALL PAIR(LATTICE,L,I,J,X,Y,NN)
      ENDIF

      X = I - 1
      Y = J
      IF (X.LT.1) THEN
            X = L
      ENDIF
      IF (LATTICE(X,Y).EQ.-10) THEN
c        nn = site centre
            X = I - 2
            CALL PAIR(LATTICE,L,I,J,X,Y,NN)
      ENDIF

      X = I
      Y = J + 1
      IF (Y.GT.L) THEN
            Y = 1
      ENDIF
      IF (LATTICE(X,Y).EQ.-10) THEN
c        nn = site centre
            Y = J+2
            CALL PAIR(LATTICE,L,I,J,X,Y,NN)
      ENDIF

      X = I
      Y = J - 1
      IF (Y.LT.1) THEN
              Y = L
      ENDIF
      IF (LATTICE(X,Y).EQ.-10) THEN
c           nn = site centre
              Y = J - 2
              CALL PAIR(LATTICE,L,I,J,X,Y,NN)
      ENDIF

      ESITE = ESITE + (V(3)*NN)

150   RETURN
      END
c     ===============================================================
c           subroutine to identify o=o pairs
      SUBROUTINE PAIR(LATTICE,L,I,J,X,Y,NN)
      INTEGER L,LATTICE(L,L),I,J,X,Y,NN

c     simply looks at lattice(i,j) and lattice(x,y) and returns nn=nn+1
c     if O=O pair and nn=nn for all others

c           2 possible configurations
c     1)   assumes nn interactions occur solely within lattice this
c           causes a 'surface tension' effect
c     2)   assumes nn interactions can occur between lattice and hb
c           no surface tension, runs faster at low temps

c     test for nn within hb
      IF(X.LT.1.OR.X.GT.L.OR.Y.LT.1.OR.Y.GT.L) THEN
c 120         nn outside lattice (abort)
c 130   GOTO 195

c     next line allows interactions between oxygens within the lattice
c     and those within the hb
140         if (lattice(i,j).eq.10) then
150         nn = nn +1

160   ENDIF
      ENDIF

c           test if both atoms are an oxygen
      IF(LATTICE(I,J).GT.0.AND.LATTICE(X,Y).GT.0) THEN
                        NN = NN + 1
c                 o=o pair
      ENDIF
195   RETURN
      END

c     ===============================================================
      SUBROUTINE PICKOXY(LATTICE,L,I,J)
      INTEGER L,LATTICE(L,L),I,J
      REAL G05DAF
```

```
210   CALL PICKSITE(L,I,J)

c        if i,j are within h:b then = oxygen by default
      IF (I.LT.1.OR.I.GT.L.OR.J.LT.1.OR.J.GT.L) THEN
            GOTO 220
      ENDIF

c        check if the site contains an oxygen, if not recall picksite
      IF (LATTICE(I,J).LT.1) THEN
                  GOTO 210
      ENDIF

220   RETURN
      END

c     ==================================================================
c        subroutine to pick an atom (or site) at random
      SUBROUTINE PICKSITE(L,I,J)
      INTEGER I,J,L
      REAL G05DAF

c        generates rnd in the range 0=(L+1)
10    I = INT(G05DAF(0.,1.)*(L+2))
      J = INT(G05DAF(0.,1.)*(L+2))

c        test for out of range values
      IF(I.GT.(L+1).OR.I.LT.0.OR.J.LT.0.OR.J.GT.(L+1)) THEN
            GOTO 10
      ENDIF

      RETURN
      END

c     ==================================================================
c        subroutine to propose a landing site at random
      SUBROUTINE JUMPTO(LATTICE,L,I,J,P,Q)
      INTEGER L,LATTICE(L,L),I,J,P,Q,X
      REAL R
      REAL G05DAF


      P = I
      Q = J
c        picking a direction at random
c        first two lateral/vertical jump vectors are dependant upon
c        the oxygens position vis a vis the coppers

c     wraps around when determining whether i=1 n.n is a copper atom
      X = I-1
      IF (X.LT.1) THEN
            X=L - X
      ENDIF

      R = G05DAF(0.,6.)
      IF (R.LT.1.0) THEN
            IF (LATTICE(X,J).EQ.-1) THEN
                  P = I
                  Q = J+2
```

```
                ELSE
                        P = I+2
                        Q = J
                ENDIF
        ELSE IF(R.LT.2.0) THEN
                IF(LATTICE(X,J).EQ.-1) THEN
                        P = I
                        Q = J - 2
                ELSE
                        P = I - 2
                        Q = J
                ENDIF

                ELSE IF (R.LT.3.0) THEN
                        P = I+1
                        Q = J+1
                ELSE IF (R.LT.4.0) THEN
                        P = I+1
                        Q = J-1
                ELSE  IF (R.LT.5.0) THEN
                        P = I-1
                        Q = J+1
                ELSE
                        P = I-1
                        Q = J-1
        ENDIF

        RETURN
        END
c       ================================================================
c             subroutine to propose a landing site at random
        SUBROUTINE JUMPFROMHB(LATTICE,L,I,J,P,Q)
        INTEGER L,LATTICE(L,L),I,J,P,Q,X
        REAL R
        REAL G05DAF


        P = I
        Q = J

c       check whether the i,j are at the lattice corners=only 1 jumpto
c       site availible
        IF(I.LT.1.AND.J.LT.1) THEN
                P = 1
                Q = 1
                GOTO 410
        ELSE IF(I.LT.1.AND.J.GT.L) THEN
                P = 1
                Q = L
                GOTO 410
        ELSE IF(I.GT.L.AND.J.LT.1) THEN
                P = L
                Q = 1
                GOTO 410
        ELSE IF(I.GT.L.AND.J.GT.L) THEN
                P = L
                Q = L
                GOTO 410
```

```
      ENDIF

c        picking a direction at random
c     if the jump is from within the hb subroutine picks a jumpto site
c     from one of the 3 nn within the lattice

      R = G05DAF(0.,3.)

      IF(I.LT.1) THEN
            IF(R.LT.1.0) THEN
            P = I+1
            Q = J+1
            ELSE IF (R.LT.2.0) THEN
            P = I+1
            Q = J
            ELSE
            P = I+1
            Q = J-1
            ENDIF
      ENDIF

      IF(I.GT.L) THEN
            IF(R.LT.1.0) THEN
            P = I-1
            Q = J+1
            ELSE IF(R.LT.2.0) THEN
            P = I-1
            Q = J
            ELSE
            P = I-1
            Q = J-1
            ENDIF
      ENDIF

      IF(J.LT.1) THEN
            IF(R.LT.1.0) THEN
            P = I+1
            Q = J+1
            ELSE IF(R.LT.2.0) THEN
            P = I
            Q = J+1
            ELSE
            P = I-1
            Q = J+1
            ENDIF
      ENDIF

      IF(J.GT.L) THEN
            IF(R.LT.1.0) THEN
            P = I+1
            Q = J-1
            ELSE IF(R.LT.2.0) THEN
            P = I
            Q = J-1
            ELSE
            P = I-1
            Q = J-1
            ENDIF
      ENDIF

410   RETURN
      END

c     ================================================================

c     subroutine to determine mean oxygen atom site energy
      SUBROUTINE MEANENERGY(LATTICE,L,Emean,KT,V)
      INTEGER L,LATTICE(L,L),I,J,N
      REAL Emean,ESITE,KT,V(3)

c     n = no of oxygens
c     sume = total site=energy of oxygens
c     Emean = mean site energy of oxygens

      Emean = 0.00
      SUME = 0.00
      N = 0

      DO 620 I = 3,L-3
      DO 610 J = 3,L-3
      IF (LATTICE(I,J).EQ.10) THEN
              N = N+1
              CALL SITEXENERGY(LATTICE,L,I,J,ESITE,V,KT)
              SUME = SUME + ESITE
      ENDIF
610   CONTINUE
620   CONTINUE

      IF(N.LT.1) THEN
      Emean = 0.00
              ELSE
              Emean = SUME/REAL(N)
      ENDIF

      RETURN
      END

c     ================================================================
c           subroutine to calculate the mean value of mean site energy
c     and its standard deviation over the last 10 runs
      SUBROUTINE FINALENERGY(ENERGY,Efinal,ESD,Y)
      REAL ENERGY(1000,2),Efinal,ESD,SUM,DEV
      INTEGER Y,X
      SUM = 0.00
      DEV = 0.00

      DO 750 X = 1,100
                SUM = SUM + ENERGY(X,2)
750   CONTINUE
      Efinal = SUM/100.00

      DO 760 X= 1,100
              DEV = DEV + ((Efinal-ENERGY(X,2))**2)
760   CONTINUE

      ESD = SQRT(DEV/100.00)
      RETURN
      END

c     ==================================================================
c           subroutine to calculate the mean value of the conc and its
c     standard deviation over the last 10 runs
      SUBROUTINE FINALCONC(Cresults,Cfinal,CSD,Y)
      REAL Cresults(1000,2),Cfinal,CSD,SUM,DEV
      INTEGER Y,X
      SUM = 0.0
      DEV = 0.0

      DO 850 X = 1,100
            SUM = SUM+Cresults(X,2)
850   CONTINUE

      Cfinal = SUM/100.00

      DO 860 X = 1,100
            DEV = DEV + ((Cfinal - Cresults(X,2))**2)
860   CONTINUE
      CSD = SQRT(DEV/100.0)

      RETURN
      END

c     ==================================================================
c           subroutine to store the results arrays CvsHB and EvsHB in
c     two data files c.txt and e.txt.
      SUBROUTINE RESULTS(CvsHB,EvsHB,RUN)
      REAL CvsHB(1000,3), EvsHB(1000,3)
      INTEGER RUN,X


      OPEN (UNIT = 6,FILE = 'C1.DAT',STATUS = 'NEW')
      OPEN (UNIT = 7,FILE = 'E1.DAT',STATUS = 'NEW')

      DO 910 X = 1,RUN
            WRITE(6,980)CvsHB(X,1),CvsHB(X,2),CvsHB(X,3)
            WRITE(7,990)EvsHB(X,1),EvsHB(X,2),EvsHB(X,3)
910   CONTINUE

      CLOSE(6)
      CLOSE(7)

980   FORMAT(1X,F16.4,5X,F7.4,5X,F6.5)
990   FORMAT(1X,F16.4,5X,F7.4,5X,F6.5)
      RETURN
      END

c     ==================================================================
c     subroutine to check the positions of the copper atoms
      SUBROUTINE CHKLATT(LATTICE,L,IFAIL)
      INTEGER L,LATTICE(L,L),IFAIL,I,J

      PRINT *,'CHECKING POSITIONS OF COPPERS AND BLOCKED CELL CENTRES'
      DO 620 J = 1,L,2
      DO 610 I = 1,L,2
      IF (LATTICE(I,J).NE.-1) THEN
            IFAIL = 4
            print *,'chklatt copper error at  ',i,j,' =  ',lattice(i,j)
      ENDIF
610   CONTINUE
620   CONTINUE

      DO 640 J = 2,L,2
      DO 630 I = 2,L,2
      IF (LATTICE(I,J).NE.-10) THEN
            IFAIL = 4
            print *,'chklatt centre error at  ',i,j,' =  ',lattice(i,j)
      ENDIF
630   CONTINUE
640   CONTINUE
      RETURN
      END
c     ==============================================================
c     subroutine to count the no of oxygen atoms within the lattice
      SUBROUTINE COUNT(LATTICE,L,OX,CONC)
      INTEGER L,LATTICE(L,L),I,J,OX,N
      REAL CONC

      N = 0
      DO 730 I = 1,L
      DO 720 J = 1,L
      IF (LATTICE(I,J).GT.0) THEN
            N = N + 1
      ENDIF

720   CONTINUE
730   CONTINUE
c           conc is expressed as a fraction of a fully filled lattice
c           i.e  conc of OI phase is 0.5
      CONC = (REAL(N)/REAL(OX))

      RETURN
      END

c     ----------------------------------------------------------------
c     subroutine to save lattice array to a pretty picture in
c                 an external ascii file

      SUBROUTINE CONVERT0(LATTICE,L)
      INTEGER L,LATTICE(L,L),I,J
      CHARACTER*1 PICTURE(50,50)


      DO 520 I = 1,L
      DO 510 J = 1,L
      IF (LATTICE(I,J).EQ.-10) THEN
            PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).EQ.-1) THEN
            PICTURE(I,J) = 'C'
      ELSE IF (LATTICE(I,J).EQ.0) THEN
            PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).GT.0) THEN
```

```
            PICTURE(I,J) = 'O'
      ENDIF
 510  CONTINUE
 520  CONTINUE

      OPEN (UNIT=2,FILE ='PIC_0.TXT',STATUS='NEW')
      WRITE (2,560)PICTURE
      CLOSE (2)

 560  FORMAT(50(A1))
      RETURN
      END

c     ----------------------------------------------------------------
c     subroutine to save lattice array to a pretty picture in
c                     an external ascii file

      SUBROUTINE CONVERT1(LATTICE,L)
      INTEGER L,LATTICE(L,L),I,J
      CHARACTER*1 PICTURE(50,50)


      DO 520 I = 1,L
      DO 510 J = 1,L
      IF (LATTICE(I,J).EQ.-10) THEN
            PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).EQ.-1) THEN
            PICTURE(I,J) = 'C'
      ELSE IF (LATTICE(I,J).EQ.0) THEN
            PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).GT.0) THEN
            PICTURE(I,J) = 'O'
      ENDIF
 510  CONTINUE
 520  CONTINUE

      OPEN (UNIT=2,FILE ='PIC_1.TXT',STATUS='NEW')
      WRITE (2,560)PICTURE
      CLOSE (2)

 560  FORMAT(50(A1))
      RETURN
      END
c     ----------------------------------------------------------------
c     subroutine to save lattice array to a pretty picture in
c                     an external ascii file

      SUBROUTINE CONVERT2(LATTICE,L)
      INTEGER L,LATTICE(L,L),I,J
      CHARACTER*1 PICTURE(50,50)


      DO 520 I = 1,L
      DO 510 J = 1,L
      IF (LATTICE(I,J).EQ.-10) THEN
            PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).EQ.-1) THEN
            PICTURE(I,J) = 'C'
      ELSE IF (LATTICE(I,J).EQ.0) THEN
              PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).GT.0) THEN
              PICTURE(I,J) = 'O'
      ENDIF
510   CONTINUE
520   CONTINUE

      OPEN (UNIT=2,FILE ='PIC_2.TXT',STATUS='NEW')
      WRITE (2,560)PICTURE
      CLOSE (2)

560   FORMAT(50(A1))
      RETURN
      END
c     -----------------------------------------------------------------
c     subroutine to save lattice array to a pretty picture in
c                    an external ascii file

      SUBROUTINE CONVERT3(LATTICE,L)
      INTEGER L,LATTICE(L,L),I,J
      CHARACTER*1 PICTURE(50,50)


      DO 520 I = 1,L
      DO 510 J = 1,L
      IF (LATTICE(I,J).EQ.-10) THEN
              PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).EQ.-1) THEN
              PICTURE(I,J) = 'C'
      ELSE IF (LATTICE(I,J).EQ.0) THEN
              PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).GT.0) THEN
              PICTURE(I,J) = 'O'
      ENDIF
510   CONTINUE
520   CONTINUE

      OPEN (UNIT=2,FILE ='PIC_3.TXT',STATUS='NEW')
      WRITE (2,560)PICTURE
      CLOSE (2)

560   FORMAT(50(A1))
      RETURN
      END
c     -----------------------------------------------------------------
c     subroutine to save lattice array to a pretty picture in
c                    an external ascii file

      SUBROUTINE CONVERT4(LATTICE,L)
      INTEGER L,LATTICE(L,L),I,J
      CHARACTER*1 PICTURE(50,50)


      DO 520 I = 1,L
      DO 510 J = 1,L
      IF (LATTICE(I,J).EQ.-10) THEN
              PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).EQ.-1) THEN
              PICTURE(I,J) = 'C'
      ELSE IF (LATTICE(I,J).EQ.0) THEN
              PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).GT.0) THEN
              PICTURE(I,J) = 'O'
      ENDIF
510   CONTINUE
520   CONTINUE

      OPEN (UNIT=2,FILE ='PIC_4.TXT',STATUS='NEW')
      WRITE (2,560)PICTURE
      CLOSE (2)

560   FORMAT(50(A1))
      RETURN
      END
c     ----------------------------------------------------------------
c     subroutine to save lattice array to a pretty picture in
c                 an external ascii file

      SUBROUTINE CONVERT5(LATTICE,L)
      INTEGER L,LATTICE(L,L),I,J
      CHARACTER*1 PICTURE(50,50)


      DO 520 I = 1,L
      DO 510 J = 1,L
      IF (LATTICE(I,J).EQ.-10) THEN
              PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).EQ.-1) THEN
              PICTURE(I,J) = 'C'
      ELSE IF (LATTICE(I,J).EQ.0) THEN
              PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).GT.0) THEN
              PICTURE(I,J) = 'O'
      ENDIF
510   CONTINUE
520   CONTINUE

      OPEN (UNIT=2,FILE ='PIC_5.TXT',STATUS='NEW')
      WRITE (2,560)PICTURE
      CLOSE (2)

560   FORMAT(50(A1))
      RETURN
      END
c     ----------------------------------------------------------------
c     subroutine to save lattice array to a pretty picture in
c                 an external ascii file

      SUBROUTINE CONVERT6(LATTICE,L)
      INTEGER L,LATTICE(L,L),I,J
      CHARACTER*1 PICTURE(50,50)


      DO 520 I = 1,L
      DO 510 J = 1,L
      IF (LATTICE(I,J).EQ.-10) THEN
              PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).EQ.-1) THEN
              PICTURE(I,J) = 'C'
      ELSE IF (LATTICE(I,J).EQ.0) THEN
              PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).GT.0) THEN
              PICTURE(I,J) = 'O'
      ENDIF
510   CONTINUE
520   CONTINUE

      OPEN (UNIT=2,FILE ='PIC_6.TXT',STATUS='NEW')
      WRITE (2,560)PICTURE
      CLOSE (2)

560   FORMAT(50(A1))
      RETURN
      END
c     -----------------------------------------------------------------
c     subroutine to save lattice array to a pretty picture in
c                   an external ascii file

      SUBROUTINE CONVERT7(LATTICE,L)
      INTEGER L,LATTICE(L,L),I,J
      CHARACTER*1 PICTURE(50,50)


      DO 520 I = 1,L
      DO 510 J = 1,L
      IF (LATTICE(I,J).EQ.-10) THEN
              PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).EQ.-1) THEN
              PICTURE(I,J) = 'C'
      ELSE IF (LATTICE(I,J).EQ.0) THEN
              PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).GT.0) THEN
              PICTURE(I,J) = 'O'
      ENDIF
510   CONTINUE
520   CONTINUE

      OPEN (UNIT=2,FILE ='PIC_7.TXT',STATUS='NEW')
      WRITE (2,560)PICTURE
      CLOSE (2)

560   FORMAT(50(A1))
      RETURN
      END
c     -----------------------------------------------------------------
c     subroutine to save lattice array to a pretty picture in
c                   an external ascii file

      SUBROUTINE CONVERT8(LATTICE,L)
      INTEGER L,LATTICE(L,L),I,J
      CHARACTER*1 PICTURE(50,50)
      DO 520 I = 1,L
      DO 510 J = 1,L
      IF (LATTICE(I,J).EQ.-10) THEN
              PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).EQ.-1) THEN
              PICTURE(I,J) = 'C'
      ELSE IF (LATTICE(I,J).EQ.0) THEN
              PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).GT.0) THEN
              PICTURE(I,J) = 'O'
      ENDIF
510   CONTINUE
520   CONTINUE

      OPEN (UNIT=2,FILE ='PIC_8.TXT',STATUS='NEW')
      WRITE (2,560)PICTURE
      CLOSE (2)

560   FORMAT(50(A1))
      RETURN
      END
c     ----------------------------------------------------------------
c     subroutine to save lattice array to a pretty picture in
c                     an external ascii file

      SUBROUTINE CONVERT9(LATTICE,L)
      INTEGER L,LATTICE(L,L),I,J
      CHARACTER*1 PICTURE(50,50)


      DO 520 I = 1,L
      DO 510 J = 1,L
      IF (LATTICE(I,J).EQ.-10) THEN
              PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).EQ.-1) THEN
              PICTURE(I,J) = 'C'
      ELSE IF (LATTICE(I,J).EQ.0) THEN
              PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).GT.0) THEN
              PICTURE(I,J) = 'O'
      ENDIF
510   CONTINUE
520   CONTINUE

      OPEN (UNIT=2,FILE ='PIC_9.TXT',STATUS='NEW')
      WRITE (2,560)PICTURE
      CLOSE (2)

560   FORMAT(50(A1))
      RETURN
      END

c     ==================================================================

C     function to permit compiling on a pc running salford fortran77
      REAL FUNCTION G05DAF(A,B)
      REAL A,B,X
      REAL *4 FUNCTION RANDOM(Z)
      INTEGER *4Z
      X = RANDOM(0)
      G05DAF = (RANDOM(0)*B)
      RETURN
      END
c     ===================================================================
c                    GLAUBER.FOR
c
c     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
c     A program to simulate oxygen diffusion into the Cu-O laminal
c     planes of YBaCuO using Glauber dynamics (jumps over arbitary distance)
c     and record variation of
c     oxygen concentration and E (mean site interaction energy) against
c     chenical potential of the surrounding heat bath.
c     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
c     MAIN:

      INTEGER L
c     l = unit cell dimensions ((n*3)-1)
      PARAMETER (L = 50)

      INTEGER LATTICE(L,L),OX,X,Y,RUN
      REAL KT,HB,V(3),ENERGY(1000,2),Cresults(1000,2),Emean,CONC
      REAL CvsHB(1000,3),EvsHB(1000,3),Efinal,Cfinal,ESD,CSD

      REAL G05DAF
      EXTERNAL G05DAF

      RUN = 0

      CALL SETUP(V,KT,CONC)
      CALL INITIALISE(LATTICE,L,OX,CONC,V,KT)

      HB = 10

c     run the lattice for a while in the initial state to stabilise it
5         DO 7 X = 1,10000000
                    CALL KINOS(LATTICE,L,V,KT,HB)
7         CONTINUE

c     this particular configuration varies the chenical potential of the
c     heat bath (HB) whilst keeping the temperature constant.

9     DO 50 HB = 26.5,29.5,0.02
c     run is used to space the sampling  ..see line labelled 46..
      RUN =  RUN + 1

10        DO 11 X = 1,9900
                    CALL KINOS(LATTICE,L,V,KT,HB)
11        CONTINUE

c         final 100 runs to calculate means + msd's of results
12        DO 45 X = 1,100
                    CALL KINOS(LATTICE,L,V,KT,HB)
                    CALL COUNT(LATTICE,L,OX,CONC)
                    Cresults(X,1) = X
                    Cresults(X,2) = CONC

                    CALL MEANENERGY(LATTICE,L,Emean,KT,V)
                    ENERGY(X,1) = X
                    ENERGY(X,2) = Emean
45        CONTINUE

c     calculate mean conc + energy (+msd's) over last 100 jump attemps
```

```
      CALL FINALENERGY(ENERGY,Efinal,ESD,X)
      CALL FINALCONC(Cresults,Cfinal,CSD,X)

c     store the final energy,conc and their sd's in the two data arrays
46     CvsHB(RUN,1) = HB
      CvsHB(RUN,2) = Cfinal
      CvsHB(RUN,3) = CSD
      EvsHB(RUN,1) = HB
      EvsHB(RUN,2) = Efinal
      EvsHB(RUN,3) = ESD



c     call up the routines to output images of the lattice
c     best to use the integer RUN rather than an IF equality containing
c     the real no's HB or temperature KT.
      IF (RUN.EQ.1) THEN
            CALL CONVERT0(LATTICE,L)
      ENDIF
      IF (RUN.EQ.20) THEN
            CALL CONVERT1(LATTICE,L)
      ENDIF
      IF (RUN.EQ.21) THEN
            CALL CONVERT2(LATTICE,L)
      ENDIF
      IF (RUN.EQ.22) THEN
            CALL CONVERT3(LATTICE,L)
      ENDIF
      IF (RUN.EQ.24) THEN
            CALL CONVERT4(LATTICE,L)
      ENDIF
      IF (RUN.EQ.65) THEN
            CALL CONVERT5(LATTICE,L)
      ENDIF
      IF (RUN.EQ.70) THEN
            CALL CONVERT6(LATTICE,L)
      ENDIF
      IF (RUN.EQ.75) THEN
           CALL CONVERT7(LATTICE,L)
      ENDIF
      IF (RUN.EQ.80) THEN
            CALL CONVERT8(LATTICE,L)
      ENDIF
      IF (RUN.EQ.281) THEN
            CALL CONVERT9(LATTICE,L)
      ENDIF


50    CONTINUE

c     output conc and energy results to two data files
      CALL RESULTS(CvsHB,EvsHB,RUN)

      END

c     ==================================================================
c           subroutine to initialise user defined parameters
      SUBROUTINE SETUP(V,KT,CONC)
      REAL V(3),KT,CONC

c     set n.n interaction parameters
      V(1) = 6.90
      V(2) = -2.40
      V(3) = 1.10

c     set temperature (if not varying in the main loop above)
      KT = V(1)*0.01

c     set initial concentration .. usefull for long low-temperature runs
      Conc = 0.00

      RETURN
      END

c     =================================================================

c          subroutine to clear and initialise the lattice array
      SUBROUTINE INITIALISE(LATTICE,L,OX,CONC,V,KT)
      INTEGER LATTICE,L,OX
      REAL CONC,V(3),KT

c     empty the lattice array
      CALL CLEARLATT(LATTICE,L)

c     add fixed copper atoms
      CALL INICU(LATTICE,L)

c     count the number of intersitial sites (max no of oxygens)
      CALL COUNT_OX(LATTICE,L,OX)

c     prepopulate lattice to initial concentration 'conc'
      CALL POPULATE(LATTICE,L,OX,CONC)

      RETURN
      END
C     =================================================================
c     subroutine to delete contents of lattice
      SUBROUTINE CLEARLATT(LATTICE,L)
      INTEGER L,LATTICE(L,L),I,J


c     clears the lattice
      DO 130 I = 1,L
            DO 120 J = I,L
                  LATTICE(I,J) = 0
120         CONTINUE
130   CONTINUE

      RETURN
      END

c     =================================================================
c          subroutine to initialise the copper atoms and block cell
c                    centers

      SUBROUTINE INICU(LATTICE,L)
      INTEGER  L,LATTICE(L,L),I,J

c          setting the cu atoms (as -1), row by row

      DO 20 J = 1,L,2
            DO 10 I = 1,L,2

                  LATTICE(I,J) = -1
10          CONTINUE
20    CONTINUE

c          setting the site centres as (-10)
      DO 40 J = 2,L,2
            DO 30 I = 2,L,2
                  LATTICE(I,J) = -10
30          CONTINUE
40    CONTINUE

      RETURN
      END

c     ==================================================================
C     subroutine to count the number of interstitial sites
      SUBROUTINE COUNT_OX(LATTICE,L,OX)
      INTEGER L,LATTICE(L,L),OX,I,J

c     ox is the total no of sites for mobile oxygens within the lattice
      OX = 0

      DO 40 J = 1,L
            DO  30 I = 1,L
                  IF (LATTICE(I,J).GT.-1) THEN
                        OX = OX + 1
                  ENDIF
30          CONTINUE
40    CONTINUE


      RETURN
      END

c     ==================================================================
c     subroutine to randomly populate the lattice to concentration CONC
      SUBROUTINE POPULATE(LATTICE,L,OX,CONC)
      INTEGER L,OX,LATTICE(L,L),N,I,J,A
      REAL CONC
      REAL G05DAF

c     n = no of oxygens required for concentration of 'conc'
c     a = no of oxygens placed so far

      N = INT(CONC*OX)
      A = 0

c     randomly pick a site using subroutine picksite
110   CALL PICKSITE(L,I,J)

c     test for out of range values as picksite can select a site within
```

```
c     the heat bath
      IF(I.GT.(L).OR.I.LT.1.OR.J.LT.1.OR.J.GT.(L)) THEN
            GOTO 110
      ENDIF

c     if site is vacant then place an oxygen
      IF (LATTICE(I,J).EQ.0) THEN
            LATTICE(I,J) = 10
            A = A+1
      ENDIF


      IF(A.LT.N) THEN
            GOTO 110
      ENDIF

      RETURN
      END

c     ==================================================================
c          subroutine to move the atoms      (the biggie)
c          lots of naughty 'gotos'
      SUBROUTINE KINOS(LATTICE,L,V,KT,HB)
      INTEGER L,LATTICE(L,L),I,J,P,Q
      REAL V(3),E1,E2,ESITE,KT,HB
      REAL G05DAF

c     the complexity of this routine arises from the desire to keep as
c     many of the subsiduary routines as simple as possible. It has to
c     deal with 4 separate types of jump all of which require different
c     handling. Jumps from heat bath to lattice, lattice to heat bath,
c     within the heat bath (aborted), and within the lattice.

c     i,j are the coordinates of the 'jumpfrom' site
c     p,q are the coordinates of the proposed 'jumpto' site
c     e1,e2 are the interaction potentials of oxygens at the two sites
      E1 = 0.000
      E2 = 0.000


c     pick an oxygen atom .. may be within lattice or the heat bath
      CALL PICKOXY(LATTICE,L,I,J)

c     if jump is from h:b then set energy as that of hb directly
      IF(I.LT.1.OR.I.GT.L.OR.J.LT.1.OR.J.GT.L) THEN
            E1 = HB
c          select a site to jumpto
            CALL JUMPTO(LATTICE,L,I,J,P,Q)
            GOTO 305
      ENDIF

c     determine interqaction potential of 1st site
      CALL SITEXENERGY(LATTICE,L,I,J,E1,V,KT)

c     select a site to jumpto
301   CALL JUMPTO(LATTICE,L,I,J,P,Q)

c     test whether jump is into heatbath and set e2 directly as hb
```

```
305  IF(P.LT.1.OR.P.GT.L.OR.Q.LT.1.OR.Q.GT.L) THEN
            E2 = HB
c         if jump is limited to heat bath then abort
                    IF(I.LT.1.OR.I.GT.L.OR.J.LT.1.OR.J.GT.L) THEN
                          GOTO 390
                    ENDIF
            GOTO 310
      ENDIF

c           test for vacancy at jumpto site
      IF(LATTICE(P,Q).NE.0) THEN
            GOTO 390
      ENDIF

c     set lattice as if jump has occured before calculating e2
      LATTICE(P,Q) = 10

c     if jump from hb no need to set hb site to zero (outside lat range)
      IF(I.LT.1.OR.I.GT.L.OR.J.LT.1.OR.J.GT.L) THEN
            CALL SITEXENERGY(LATTICE,L,P,Q,E2,V,KT)
            LATTICE(P,Q) = 0
            GOTO 310
      ENDIF

      LATTICE(I,J) = 0
      CALL SITEXENERGY(LATTICE,L,P,Q,E2,V,KT)
      LATTICE(I,J) = 10
      LATTICE(P,Q) = 0

c     let's try and do it
310   CALL JUMP(LATTICE,L,KT,I,J,P,Q,E1,E2)

390   RETURN
      END
c     ==================================================================
c         subroutine to perform the jump (or not)
      SUBROUTINE JUMP(LATTICE,L,KT,I,J,P,Q,E1,E2)
      INTEGER L,LATTICE(L,L),I,J,P,Q
      REAL E1,E2,R,PROB,KT,DE
      REAL G05DAF


c     calculate difference in site energies between the two sites
      DE = E1-E2

c     if dE >= 0.00 then jumpto site has lower energy so always jump
c     -0.001 required otherwise jumps of dE = 0 fail 50% of times
      IF(dE.GE.-0.001) THEN
c         jump
            GOTO 510
      ENDIF


c     de is <0 so jump is probabilistic ... generate random number R
      R = G05DAF(0.,1.)


c     normalised probability of jump proceeding
```

```
      PROB = EXP(dE/(KT))
c          as dE becomes more negative, prob decreases hence jump
c          likelyhood dereases.

c     if jump probability greater than random value R then jump proceeds
      IF(PROB.GE.R) THEN
            GOTO 510
      ENDIF

c     if rpogram reaches here then no jump so return

      GOTO 520

c     do jump
c     reset lattice (if i,j   or p,q are within it and not in h:b)
510      IF (I.LT.1.OR.I.GT.L.OR.J.LT.1.OR.J.GT.L) THEN
            GOTO 515
      ENDIF
c     previous site becomes vacant
      LATTICE(I,J) = 0

515   IF(P.LT.1.OR.P.GT.L.OR.Q.LT.1.OR.Q.GT.L) THEN
c          oxygen has returned to heat bath, nothing else to do
            GOTO 520
      ENDIF

c     place oxygen at new site
      LATTICE(P,Q) = 10

520   RETURN
      END

c     ================================================================
c          subroutine to sum a site's pairwise interaction energy
      SUBROUTINE SITEXENERGY(LATTICE,L,I,J,ESITE,V,KT)
      INTEGER L,LATTICE(L,L),I,J,X,Y,NN
      REAL ESITE,V(3),KT

      ESITE = 0.0

c          i,j,p,q are all in the range 1:L

c     sum V(1) terms
      NN = 0

      X = I + 1
      Y = J + 1
      CALL PAIR(LATTICE,L,I,J,X,Y,NN)

      X = I + 1
      Y = J - 1
      CALL PAIR(LATTICE,L,I,J,X,Y,NN)

      X = I - 1
      Y = J + 1
      CALL PAIR(LATTICE,L,I,J,X,Y,NN)

      X = I - 1
      Y = J - 1
      CALL PAIR(LATTICE,L,I,J,X,Y,NN)

      ESITE = ESITE + (V(1)*NN)

c     sum V(2) terms
      NN = 0

c     look for nn copper atoms
      X = I + 1
      Y = J
      IF(X.GT.L) THEN
            X = 1
      ENDIF
      IF (LATTICE(X,Y).EQ.-1) THEN
c         nn = cu
            X = I + 2
            CALL PAIR(LATTICE,L,I,J,X,Y,NN)
      ENDIF

      X = I - 1
      Y = J
      IF(X.LT.1) THEN
            X = L
      ENDIF
      IF (LATTICE(X,Y).EQ.-1) THEN
c         nn = cu
            X = I - 2
            CALL PAIR(LATTICE,L,I,J,X,Y,NN)
      ENDIF

      X = I
      Y = J + 1
      IF (Y.GT.L) THEN
            Y = 1
      ENDIF
      IF (LATTICE(X,Y).EQ.-1) THEN
c         nn = cu
            Y = J + 2
            CALL PAIR(LATTICE,L,I,J,X,Y,NN)
      ENDIF

      X = I
      Y = J - 1
      IF (Y.LT.1) THEN
            Y = L
      ENDIF
      IF (LATTICE(X,Y).EQ.-1) THEN
c         nn = cu
            Y = J - 2
            CALL PAIR(LATTICE,L,I,J,X,Y,NN)
      ENDIF

      ESITE = ESITE + (V(2)*NN)

c     sum V(3) terms
      NN = 0
      X = I + 1
      Y = J
      IF (X.GT.L) THEN
              X = 1
      ENDIF
      IF (LATTICE(X,Y).EQ.-10) THEN
c          nn = site centre
              X = I+2
              CALL PAIR(LATTICE,L,I,J,X,Y,NN)
      ENDIF

      X = I - 1
      Y = J
      IF (X.LT.1) THEN
              X = L
      ENDIF
      IF (LATTICE(X,Y).EQ.-10) THEN
c          nn = site centre
              X = I - 2
              CALL PAIR(LATTICE,L,I,J,X,Y,NN)
      ENDIF

      X = I
      Y = J + 1
      IF (Y.GT.L) THEN
              Y = 1
      ENDIF
      IF (LATTICE(X,Y).EQ.-10) THEN
c          nn = site centre
              Y = J+2
              CALL PAIR(LATTICE,L,I,J,X,Y,NN)
      ENDIF

      X = I
      Y = J - 1
      IF (Y.LT.1) THEN
              Y = L
      ENDIF
      IF (LATTICE(X,Y).EQ.-10) THEN
c          nn = site centre
              Y = J - 2
              CALL PAIR(LATTICE,L,I,J,X,Y,NN)
      ENDIF

      ESITE = ESITE + (V(3)*NN)

150   RETURN
      END
c     ==============================================================
c          subroutine to identify o=o pairs
      SUBROUTINE PAIR(LATTICE,L,I,J,X,Y,NN)
      INTEGER L,LATTICE(L,L),I,J,X,Y,NN

c     simply looks at lattice(i,j) and lattice(x,y) and returns nn=nn+1
c     if O=O pair and nn=nn for all others


c     next section performs wrap around
```

```
      IF(X.LT.1) THEN
              X=L-X
      ENDIF
      IF(X.GT.L) THEN
              X=X-L
      ENDIF
      IF(Y.LT.1) THEN
              Y=L-Y
      ENDIF
      IF(Y.GT.L) THEN
              Y=Y-L
      ENDIF

c          test if both atoms are an oxygen
      IF(LATTICE(I,J).GT.0.AND.LATTICE(X,Y).GT.0) THEN
                      NN = NN + 1
c               o=o pair
      ENDIF
195   RETURN
      END

c     ==================================================================
c     subroutine to randomly select an oxygen atom
      SUBROUTINE PICKOXY(LATTICE,L,I,J)
      INTEGER L,LATTICE(L,L),I,J
      REAL G05DAF

c     pick a site
210   CALL PICKSITE(L,I,J)

c          if i,j are within h:b then = oxygen by default
      IF (I.LT.1.OR.I.GT.L.OR.J.LT.1.OR.J.GT.L) THEN
              GOTO 220
      ENDIF

c     check if the site contains an oxygen, if not recall picksite
      IF (LATTICE(I,J).LT.1) THEN
                      GOTO 210
      ENDIF

220   RETURN
      END

c     ==================================================================
c     subroutine to pick an site at random
      SUBROUTINE PICKSITE(L,I,J)
      INTEGER I,J,L
      REAL G05DAF

c     nb: site may be within the lattice or heat bath.

c          generates rnd in the range 0 to (L+1)
10    I = INT(G05DAF(0.,1.)*(L+2))
      J = INT(G05DAF(0.,1.)*(L+2))

c          test for out of range values
      IF(I.GT.(L+1).OR.I.LT.0.OR.J.LT.0.OR.J.GT.(L+1)) THEN
              GOTO 10
      ENDIF

      RETURN
      END

c     ==============================================================
c     subroutine to propose a landing site at random
      SUBROUTINE JUMPTO(LATTICE,L,I,J,P,Q)
      INTEGER L,LATTICE(L,L),I,J,P,Q,X
      REAL R
      REAL G05DAF

c     uses Glauber dynamics and thus picks a jumpto site anywhere
      Call PICKSITE(L,P,Q)

      RETURN
      END
c     ==============================================================

c     subroutine to determine mean oxygen atom site energy
      SUBROUTINE MEANENERGY(LATTICE,L,Emean,KT,V)
      INTEGER L,LATTICE(L,L),I,J,N
      REAL Emean,ESITE,KT,V(3)

c     n = no of oxygens
c     sume = total site=energy of oxygens
c     Emean = mean site energy of oxygens

      Emean = 0.00
      SUME = 0.00
      N = 0

      DO 620 I = 3,L-3
      DO 610 J = 3,L-3
      IF (LATTICE(I,J).EQ.10) THEN
            N = N+1
            CALL SITEXENERGY(LATTICE,L,I,J,ESITE,V,KT)
            SUME = SUME + ESITE
      ENDIF
610   CONTINUE
620   CONTINUE

      IF(N.LT.1) THEN
      Emean = 0.00
            ELSE
            Emean = SUME/REAL(N)
      ENDIF

      RETURN
      END

c     ==============================================================
c           subroutine to calculate the mean value of mean site energy
c     and its standard deviation over the last 10 runs
      SUBROUTINE FINALENERGY(ENERGY,Efinal,ESD,Y)
      REAL ENERGY(1000,2),Efinal,ESD,SUM,DEV
      INTEGER Y,X
      SUM = 0.00
      DEV = 0.00

      DO 750 X = 1,100
              SUM = SUM + ENERGY(X,2)
750   CONTINUE
      Efinal = SUM/100.00

      DO 760 X= 1,100
            DEV = DEV + ((Efinal-ENERGY(X,2))**2)
760   CONTINUE

      ESD = SQRT(DEV/100.00)

      RETURN
      END

c     ==================================================================
c         subroutine to calculate the mean value of the conc and its
c     standard deviation over the last 10 runs
      SUBROUTINE FINALCONC(Cresults,Cfinal,CSD,Y)
      REAL Cresults(1000,2),Cfinal,CSD,SUM,DEV
      INTEGER Y,X
      SUM = 0.0
      DEV = 0.0

      DO 850 X = 1,100
            SUM = SUM+Cresults(X,2)
850   CONTINUE

      Cfinal = SUM/100.00

      DO 860 X = 1,100
            DEV = DEV + ((Cfinal - Cresults(X,2))**2)
860   CONTINUE
      CSD = SQRT(DEV/100.0)

      RETURN
      END

c     ==================================================================
c         subroutine to store the results arrays CvsHB and EvsHB in
c     two data files c.txt and e.txt.
      SUBROUTINE RESULTS(CvsHB,EvsHB,RUN)
      REAL CvsHB(1000,3), EvsHB(1000,3)
      INTEGER RUN,X


      OPEN (UNIT = 6,FILE = 'C1.DAT',STATUS = 'NEW')
      OPEN (UNIT = 7,FILE = 'E1.DAT',STATUS = 'NEW')

      DO 910 X = 1,RUN
            WRITE(6,980)CvsHB(X,1),CvsHB(X,2),CvsHB(X,3)
            WRITE(7,990)EvsHB(X,1),EvsHB(X,2),EvsHB(X,3)
910   CONTINUE

      CLOSE(6)
      CLOSE(7)
```

```
980  FORMAT(1X,F16.4,5X,F7.4,5X,F6.5)
990  FORMAT(1X,F16.4,5X,F7.4,5X,F6.5)
     RETURN
     END

c    ================================================================
c    test subroutine to check the positions of the copper atoms
     SUBROUTINE CHKLATT(LATTICE,L)
     INTEGER L,LATTICE(L,L),I,J


     DO 620 J = 1,L,2
     DO 610 I = 1,L,2
     IF (LATTICE(I,J).NE.-1) THEN
             print *,'chklatt copper error at  ',i,j,'  =  ',lattice(i,j)
     ENDIF
610  CONTINUE
620  CONTINUE

     DO 640 J = 2,L,2
     DO 630 I = 2,L,2
     IF (LATTICE(I,J).NE.-10) THEN
             print *,'chklatt centre error at  ',i,j,'  =  ',lattice(i,j)
     ENDIF
630  CONTINUE
640  CONTINUE
     RETURN
     END
c    ================================================================
c    subroutine to count the no of oxygen atoms within the lattice
     SUBROUTINE COUNT(LATTICE,L,OX,CONC)
     INTEGER L,LATTICE(L,L),I,J,OX,N
     REAL CONC

     N = 0
     DO 730 I = 1,L
     DO 720 J = 1,L
     IF (LATTICE(I,J).GT.0) THEN
             N = N + 1
     ENDIF

720  CONTINUE
730  CONTINUE
c         conc is expressed as a fraction of a fully filled lattice
c         i.e  conc of OI phase is 0.5
     CONC = (REAL(N)/REAL(OX))

     RETURN
     END

c    ----------------------------------------------------------------
c    subroutine to save lattice array to a pretty picture in
c                    an external ascii file

     SUBROUTINE CONVERT0(LATTICE,L)
     INTEGER L,LATTICE(L,L),I,J
     CHARACTER*1 PICTURE(50,50)
```

```
      DO 520 I = 1,L
      DO 510 J = 1,L
      IF (LATTICE(I,J).EQ.-10) THEN
            PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).EQ.-1) THEN
            PICTURE(I,J) = 'C'
      ELSE IF (LATTICE(I,J).EQ.0) THEN
            PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).GT.0) THEN
            PICTURE(I,J) = 'O'
      ENDIF
510   CONTINUE
520   CONTINUE

      OPEN (UNIT=2,FILE ='PIC_0.TXT',STATUS='NEW')
      WRITE (2,560)PICTURE
      CLOSE (2)

560   FORMAT(50(A1))
      RETURN
      END

c   ---------------------------------------------------------------
c   subroutine to save lattice array to a pretty picture in
c                  an external ascii file

      SUBROUTINE CONVERT1(LATTICE,L)
      INTEGER L,LATTICE(L,L),I,J
      CHARACTER*1 PICTURE(50,50)


      DO 520 I = 1,L
      DO 510 J = 1,L
      IF (LATTICE(I,J).EQ.-10) THEN
            PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).EQ.-1) THEN
            PICTURE(I,J) = 'C'
      ELSE IF (LATTICE(I,J).EQ.0) THEN
            PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).GT.0) THEN
            PICTURE(I,J) = 'O'
      ENDIF
510   CONTINUE
520   CONTINUE

      OPEN (UNIT=2,FILE ='PIC_1.TXT',STATUS='NEW')
      WRITE (2,560)PICTURE
      CLOSE (2)

560   FORMAT(50(A1))
      RETURN
      END
c   ---------------------------------------------------------------
c   subroutine to save lattice array to a pretty picture in
c                  an external ascii file

      SUBROUTINE CONVERT2(LATTICE,L)
      INTEGER L,LATTICE(L,L),I,J
      CHARACTER*1 PICTURE(50,50)


      DO 520 I = 1,L
      DO 510 J = 1,L
      IF (LATTICE(I,J).EQ.-10) THEN
            PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).EQ.-1) THEN
            PICTURE(I,J) = 'C'
      ELSE IF (LATTICE(I,J).EQ.0) THEN
            PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).GT.0) THEN
            PICTURE(I,J) = 'O'
      ENDIF
510   CONTINUE
520   CONTINUE

      OPEN (UNIT=2,FILE ='PIC_2.TXT',STATUS='NEW')
      WRITE (2,560)PICTURE
      CLOSE (2)

560   FORMAT(50(A1))
      RETURN
      END
c     -----------------------------------------------------------------
c     subroutine to save lattice array to a pretty picture in
c                 an external ascii file

      SUBROUTINE CONVERT3(LATTICE,L)
      INTEGER L,LATTICE(L,L),I,J
      CHARACTER*1 PICTURE(50,50)


      DO 520 I = 1,L
      DO 510 J = 1,L
      IF (LATTICE(I,J).EQ.-10) THEN
            PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).EQ.-1) THEN
            PICTURE(I,J) = 'C'
      ELSE IF (LATTICE(I,J).EQ.0) THEN
            PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).GT.0) THEN
            PICTURE(I,J) = 'O'
      ENDIF
510   CONTINUE
520   CONTINUE

      OPEN (UNIT=2,FILE ='PIC_3.TXT',STATUS='NEW')
      WRITE (2,560)PICTURE
      CLOSE (2)

560   FORMAT(50(A1))
      RETURN
      END
c     -----------------------------------------------------------------
c     subroutine to save lattice array to a pretty picture in
c                 an external ascii file
```

```
      SUBROUTINE CONVERT4(LATTICE,L)
      INTEGER L,LATTICE(L,L),I,J
      CHARACTER*1 PICTURE(50,50)


      DO 520 I = 1,L
      DO 510 J = 1,L
      IF (LATTICE(I,J).EQ.-10) THEN
              PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).EQ.-1) THEN
              PICTURE(I,J) = 'C'
      ELSE IF (LATTICE(I,J).EQ.0) THEN
              PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).GT.0) THEN
              PICTURE(I,J) = 'O'
      ENDIF
  510 CONTINUE
  520 CONTINUE

      OPEN (UNIT=2,FILE ='PIC_4.TXT',STATUS='NEW')
      WRITE (2,560)PICTURE
      CLOSE (2)

  560 FORMAT(50(A1))
      RETURN
      END
c     ------------------------------------------------------------
c     subroutine to save lattice array to a pretty picture in
c                   an external ascii file

      SUBROUTINE CONVERT5(LATTICE,L)
      INTEGER L,LATTICE(L,L),I,J
      CHARACTER*1 PICTURE(50,50)


      DO 520 I = 1,L
      DO 510 J = 1,L
      IF (LATTICE(I,J).EQ.-10) THEN
              PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).EQ.-1) THEN
              PICTURE(I,J) = 'C'
      ELSE IF (LATTICE(I,J).EQ.0) THEN
              PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).GT.0) THEN
              PICTURE(I,J) = 'O'
      ENDIF
  510 CONTINUE
  520 CONTINUE

      OPEN (UNIT=2,FILE ='PIC_5.TXT',STATUS='NEW')
      WRITE (2,560)PICTURE
      CLOSE (2)

  560 FORMAT(50(A1))
      RETURN
      END
c     ------------------------------------------------------------
```

```
c     subroutine to save lattice array to a pretty picture in
c                    an external ascii file

      SUBROUTINE CONVERT6(LATTICE,L)
      INTEGER L,LATTICE(L,L),I,J
      CHARACTER*1 PICTURE(50,50)


      DO 520 I = 1,L
      DO 510 J = 1,L
      IF (LATTICE(I,J).EQ.-10) THEN
            PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).EQ.-1) THEN
            PICTURE(I,J) = 'C'
      ELSE IF (LATTICE(I,J).EQ.0) THEN
            PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).GT.0) THEN
            PICTURE(I,J) = 'O'
      ENDIF
510   CONTINUE
520   CONTINUE

      OPEN (UNIT=2,FILE ='PIC_6.TXT',STATUS='NEW')
      WRITE (2,560)PICTURE
      CLOSE (2)

560   FORMAT(50(A1))
      RETURN
      END
c     -----------------------------------------------------------------
c     subroutine to save lattice array to a pretty picture in
c                    an external ascii file

      SUBROUTINE CONVERT7(LATTICE,L)
      INTEGER L,LATTICE(L,L),I,J
      CHARACTER*1 PICTURE(50,50)


      DO 520 I = 1,L
      DO 510 J = 1,L
      IF (LATTICE(I,J).EQ.-10) THEN
            PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).EQ.-1) THEN
            PICTURE(I,J) = 'C'
      ELSE IF (LATTICE(I,J).EQ.0) THEN
            PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).GT.0) THEN
            PICTURE(I,J) = 'O'
      ENDIF
510   CONTINUE
520   CONTINUE

      OPEN (UNIT=2,FILE ='PIC_7.TXT',STATUS='NEW')
      WRITE (2,560)PICTURE
      CLOSE (2)

560   FORMAT(50(A1))
      RETURN
      END
c     -----------------------------------------------------------------
c     subroutine to save lattice array to a pretty picture in
c                     an external ascii file

      SUBROUTINE CONVERT8(LATTICE,L)
      INTEGER L,LATTICE(L,L),I,J
      CHARACTER*1 PICTURE(50,50)


      DO 520 I = 1,L
      DO 510 J = 1,L
      IF (LATTICE(I,J).EQ.-10) THEN
            PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).EQ.-1) THEN
            PICTURE(I,J) = 'C'
      ELSE IF (LATTICE(I,J).EQ.0) THEN
            PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).GT.0) THEN
            PICTURE(I,J) = 'O'
      ENDIF
510   CONTINUE
520   CONTINUE

      OPEN (UNIT=2,FILE ='PIC_8.TXT',STATUS='NEW')
      WRITE (2,560)PICTURE
      CLOSE (2)

560   FORMAT(50(A1))
      RETURN
      END
c     -----------------------------------------------------------------
c     subroutine to save lattice array to a pretty picture in
c                     an external ascii file

      SUBROUTINE CONVERT9(LATTICE,L)
      INTEGER L,LATTICE(L,L),I,J
      CHARACTER*1 PICTURE(50,50)


      DO 520 I = 1,L
      DO 510 J = 1,L
      IF (LATTICE(I,J).EQ.-10) THEN
            PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).EQ.-1) THEN
            PICTURE(I,J) = 'C'
      ELSE IF (LATTICE(I,J).EQ.0) THEN
            PICTURE(I,J) = ' '
      ELSE IF (LATTICE(I,J).GT.0) THEN
            PICTURE(I,J) = 'O'
      ENDIF
510   CONTINUE
520   CONTINUE

      OPEN (UNIT=2,FILE ='PIC_9.TXT',STATUS='NEW')
      WRITE (2,560)PICTURE
      CLOSE (2)
```

```
560  FORMAT(50(A1))
     RETURN
     END

c    =================================================================

C     function to permit compiling on a pc running salford fortran77
     REAL FUNCTION G05DAF(A,B)
     REAL A,B,X
     REAL *4 FUNCTION RANDOM(Z)
     INTEGER *4Z
     X = RANDOM(0)
     G05DAF = (RANDOM(0)*B)
     RETURN
     END

c
=========================================================================
ccc                   Histogrm.for

ccc  a program to test the distribution of random numbers generated by
ccc  the FORTRAN library subroutine G05DAF
ccc  ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

ccc  MAIN

      INTEGER RESULTS(1:2,0:101),I,Q,N
      REAL G05DAF
      EXTERNAL G05DAF

      DATA RESULTS/204*0/
      N = 100000000


      DO 10 I = 1,N
            Q = INT(G05DAF(1.,101.))

ccc         test for out of range values
            IF (Q.LT.1) THEN
                  RESULTS(2,0) = RESULTS(2,0) + 1
            ELSE IF(Q.GT.100) THEN
                  RESULTS(2,101) = RESULTS(2,101) + 1
            ENDIF

ccc  increment the appropriate tally
      RESULTS(2,Q) = RESULTS(2,Q) + 1

10   CONTINUE

      DO 20 I = 0,101,1
            RESULTS(1,I) = I
20   CONTINUE

      OPEN(UNIT=1,FILE = 'H10mill.TXT',STATUS = 'NEW')
      WRITE(1,100)RESULTS
      CLOSE(1)

100  FORMAT(1x,I6,'     ',I11)

      END
```

```
ccc                 REPITIT.FOR

ccc  ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
ccc  a program to test for repeated sequences within a large sequence
ccc  of random numbers generated by the FORTRAN library routine G05DAF
ccc  ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

ccc  MAIN

     INTEGER LIST(1000,2),RESULTSA(10000,2),dX,X,Y,Z,N,HITS,FLAG,MFLAG
     INTEGER RUN,J,DISTRIB(100),B,C

     REAL G05DAF
     EXTERNAL G05DAF

     DATA RESULTSA/20000*0/DISTRIB/100*0/LIST/2000*0/

     N = 10000
     HITS = 1
     FLAG = 0
     MFLAG = 0
     dX = 0
     Z = 1

ccc      generate a list of random numbers and store them in LIST
     DO 10 X = 1,1000
           LIST(X,1) = INT(G05DAF(1.,100.))
10   CONTINUE

     DO 15 X = 1,1000
           LIST(X,2) = INT(G05DAF(1.,100.))
15   CONTINUE

     DO 35 B = 0,N,1
     DO 30 C = 1,1000

           RUN = (B*1000) + C
           J = 0
           FLAG = 0

           LIST(Z,2)= INT(G05DAF(1.,100.))
           Z = Z+1
           IF(Z.GT.1000)THEN
                 Z = 1
           ENDIF

           dX = dX+1
           IF(dX.GT.999)THEN
                 dX = 0
           ENDIF


           DO 20 X = 1,1000
                 Y = X + dX
                 IF(Y.GT.1000) THEN
                       Y = Y-1000
                 ENDIF
```

```
                    IF(FLAG.GT.MFLAG) THEN
                          MFLAG = FLAG
                    ENDIF

                    IF(LIST(X,1).EQ.LIST(Y,2)) THEN
                          FLAG = FLAG + 1
                          J = J+1
                          ELSE
                             FLAG = 0
                    ENDIF
20          CONTINUE

c        store histogran of correlations in distrib
              DISTRIB(J) = DISTRIB(J) + 1


              IF(J.GT.24) THEN
                    RESULTSA(HITS,1) = RUN + 1000
                    RESULTSA(HITS,2) = J
                    HITS = HITS + 1
              ENDIF
30     CONTINUE


35     CONTINUE

   OPEN(UNIT=2,FILE='FLAGS.TXT',STATUS='NEW')
          WRITE(2,*)'MAX REPEATED SEQUENCE LENGTH =',MFLAG
          WRITE(2,*)'ON A RUN OF',(N*1000),'  RANDOM NUMBERS'
   CLOSE(2)

   OPEN(UNIT = 1,FILE='REPITA.TXT',STATUS='NEW')
          IF(HITS.GT.1) THEN
                DO 90 X = 1,HITS
                      WRITE(1,100)RESULTSA(X,1),RESULTSA(X,2)
90               CONTINUE
          ELSE
                WRITE(1,100)-1,-1
          ENDIF
   CLOSE(1)

   OPEN(UNIT=3,FILE='REPDISTB.TXT',STATUS = 'NEW')
          WRITE(3,110)DISTRIB
   CLOSE(3)

100  FORMAT(1X,I9,'      ',I6)
110  FORMAT(1X,I8)
   END
```