*Res. Lett. Inf. Math. Sci.*, 2006, Vol.**1**, pp1-16
Available online at http://iims.massey.ac.nz/research/letters/



# 64-Bit Architectures and Compute Clusters for High Performance Simulations

K.A. Hawick, H.A.James & C.J.Scogings

*Institute of Information & Mathematical Sciences*
*Massey University at Albany, Auckland, New Zealand.*

Simulation of large complex systems remains one of the most demanding of high performance computer systems both in terms of raw compute performance and efficient memory management. Recent availability of 64-bit architectures has opened up the possibilities of commodity computers accessing more than the 4 Gigabyte memory limit previously enforced by 32-bit addressing. We report on some performance measurements we have made on two 64-bit architectures and their consequences for some high performance simulations. We discuss performance of our codes for simulations of artificial life models; computational physics models of point particles on lattices; and with interacting clusters of particles. We have summarised pertinent features of these codes into benchmark kernels which we discuss in the context of well-known benchmark kernels of the 32-bit era. We report on how these these findings were useful in the context of designing 64-bit compute clusters for high-performance simulations.

## 1 Introduction

We are engaged in a number of computer simulation projects , modelling various physical systems using stochastic and other techniques. A particular issue in extrapolating behaviour of a simulated system to physically realistic systems sizes and thermodynamic limits is in simulating as large a system as is possible.

Simulated system sizes are generally limited in practice by three effects.

Firstly it may simply be infeasible to run a simulation program that has available to it enough memory to hold all the simulated variables. This is quite common

for dilute lattice based models where for simulation code simplicity a great deal of empty lattice sites must be held in memory. This limit is simply one of insufficient computer memory.

Secondly it may not be feasible to reach useful run lengths (for attainment of equilibrium properties for example) for system sizes above a certain size. In this case, we may have enough computer memory but not a fast enough computer processor to usefully service a simulated system that occupies all our memory. In such cases we must deliberately scale down our ambitions. This is the speed limited case.

Thirdly we may arrive at a more complicated limiting situation. We may have found some algorithmic techniques to speed up our simulation by avoiding certain recomputations. Such lookup table techniques are useful in speeding up the simulation but inevitably use up memory. So we may be limited by a combination of processor speed and available memory.

Often we can apply parallel processing techniques either to break down our programme of work into separate jobs (simplest case) or to effectively add to our available memory by pooling memory that is owned by different processors (memory sharing case) or ultimately by finding some clever and genuinely parallel algorithm that speeds up the simulation near linearly with the number of contributing processors. Although it is possible to develop software solutions to the management of parallelism [4], the main obstacle is often still resource availability.

In practice, economic considerations and computer systems availability will typically govern what is possible, and we must make value judgements about what are the most valuable simulations we might run on limited resources. We have had some limited cycles available to us on the Helix and Double-Helix Linux Cluster Supercomputers. In this paper we describe some ideas, techniques and quantitative experiments with different architectures to determine the capabilities and value of 64-bit processor architectures, and in particular the capabilities available to us for large memory simulations.

As part of this work we have constructed a pilot cluster using Apple G5 dual processor compute nodes with large amounts of memory per node. Of particular interest to us was to study the performance and ease of use of memory addresses above the conventional 4Gigabyte limit imposed by 32-bit addressing.

In this paper we describe the installation and configuration of the "Monte" cluster (section 2), and how we have used it for a number of simulation and other high memory 64-bit applications codes (Section 5). We also studied the nodes' performance under various conditions, using various programming languages and compiler flags and options (section 3). We have tried to gather together some key performance benchmark ideas in section 4. Finally we discuss some of the issues arising from this work and its implications for building larger 64-bit architecture clusters in section 6.

## 2 Building Monte

The Monte cluster was designed as a budget constrained pilot project to determine the feasibility of using dedicated high memory compute nodes for simulations. Budgets and pricing data dates very quickly and is not worth reproducing in detail here, but the money available to us was enough for a front end node; two dual processor compute nodes; and a housing cabinet capable of holding around 24 rack mountable nodes. Previously our local compute clusters were built using ordinary desktop cases housed on cheap shelving units[2]. We were interested this time in investing in cabinet infrastructure that would support "blade" style rack mounting units and a cabinet on wheels that could be moved. We correctly anticipated a number of configuration experiments and subsequent modifications that made it convenient to house the prototype in an office environment initially, with the eventual aim of relocation to a machine room environment. At the time of writing it is still convenient and possible with good ventilation, to operate the prototype in office conditions (despite the heat of the New Zealand summer).

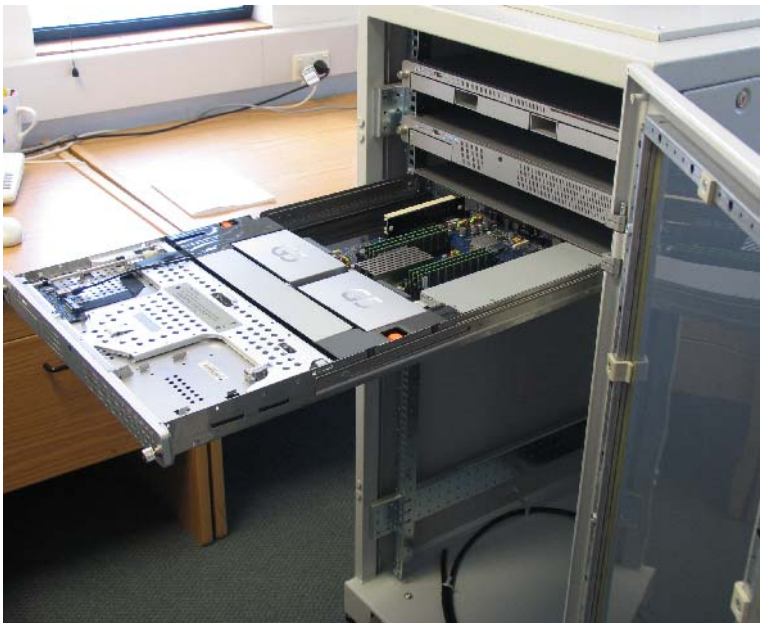

Figure 1: The 24-Unit Rack cabinet containing a partially constructed "Monte" dual-G5 compute cluster.

Figure 1 shows the prototype under construction. A slave node is pulled out of its drawer with top case removed to show off the dual processor and memory infrastructure on the board. In general we found both the case and nodes easy to construct and install and we were able to wire the system up with a Gigabit switching unit and standard lab spare cables.

The front-end node is also rack mounted and at the time of initial installation we built a Panther (Mac OSX 10.3) on the front end, with the two slave nodes upgraded to Tiger OSX 10.4 subsequently. The slave nodes have been run almost continuously generating various simulation configurations since May 2005. The front end supports a cross-mounted file system that each slave can access and the system is therefore well configured for independent jobs. The slave nodes have a total of 4.5 Gigabytes of memory each, with a further gigabyte available on the front end. This is sufficient to exceed the all critical 4Gigabyte limit imposed by 32-bit addressing. Generally we have found ourselves naturally operating individual simulations codes that routinely exceed 2 Gigabytes of memory each. This is itself a significant figure as many simulations use pointers and signed integers interchangeably and generally need careful consideration if the 2GByte signed 32-bit integer is to be used safely. The 64-bit addressing means this issues can be tackled and simulations codes tested appropriately.

The OSX operating system is based on a BSD Unix kernel and in general we have found it more robust than many Linux installations. It is also very reliable when used in tandem with cross-mounted file systems from OSX running desktop systems. Other software compatibilities have also made it an excellent platform for simulation code development and management of the resulting data files.

The processing and network hardware has proved entirely satisfactory, but as for any computer, its performance is only as good as is allowed by the operating system and supported programming language compilers.

## 3 Compiler Options

The more recent GCC compiler suite [3] allows for vast customisation of the compilation process. This is often useful when optimising programs for a particular architecture or operating system. We have extensively used the GCC compiler optimisation flags in order to achieve the best possible performance out of our simulation codes reported in section 5.

For the majority of our simulation codes we used the following compiler flags:

```
gcc-4.0 -O6 -arch ppc64 -mpowerpc64 -mtune=G5 -mcpu=G5 -DNDEBUG
```

Each of these compiler options are described below:

**-O6** Optimise the output code as much as possible wrt speed

**-arch ppc64** Make the target architecture 64-bit PowerPC

**-mpowerpc64** Enable the G5's native 64-bit long long support.

**-mtune=G5** Tune code as optimally as possible for the G5

**-mcpu=G5** Use instructions only available on G5

**-DNDEBUG** As a consequence of using the `assert.h` header file, assertion checking needs to be explicitly disabled to optimise the executable

There seems to be some disagreement in the compiler community as to what the different levels of optimisation mean. For example, in previous versions of compilers there existed options to optimise programmes in code space; this is hardly required in modern systems due to most processes being bottle-necked in terms of data space rather than code space.

One way to evaluate the effectiveness of these compiler options is to time some specific application kernels and their performance over a number of problem sizes.

## 4 Some Benchmark Kernels

There are some numerically intensive kernel operations that occur so frequently in applications codes that they provide valuable benchmarks for comparing different processor architectures and indeed operating systems and support libraries. By the term kernel we simply mean a well defined algorithm over well known data structures that can be abstracted out of an application either as a skeletal framework or as a callable library routine. It is obviously useful to discuss performance in terms of well known kernels as there will be useful comparisons that users can make against a common backdrop of experience. Kernels can typically be implemented either as code fragments that are cut-and-pasted into an application program or as library routines. There are issues connected with interfacing overheads and parameter copying and interface type mismatching that we do not wish to discuss so we have just employed cut-and-pasted codes for the kernels we do discuss.

The three considered here (with their algorithmic complexities) are:

- Matrix inversion or solving a set of $N$ linear equations ($O(N^3)$ with a small prefactor)
- Finding the Eigen values and vectors of a $N$ by $N$ matrix ($O(N^3)$ with largish prefactor)
- Finding the d-dimensional (Fast) Fourier Transform of a data block (($O(N \log N)$)

The data we present are based on Java codes, so the effects of the hardware, operating system and Java Virtual Machine implementation are all smeared together. This is nevertheless useful as it gives an amalgamated effect for what we are actually

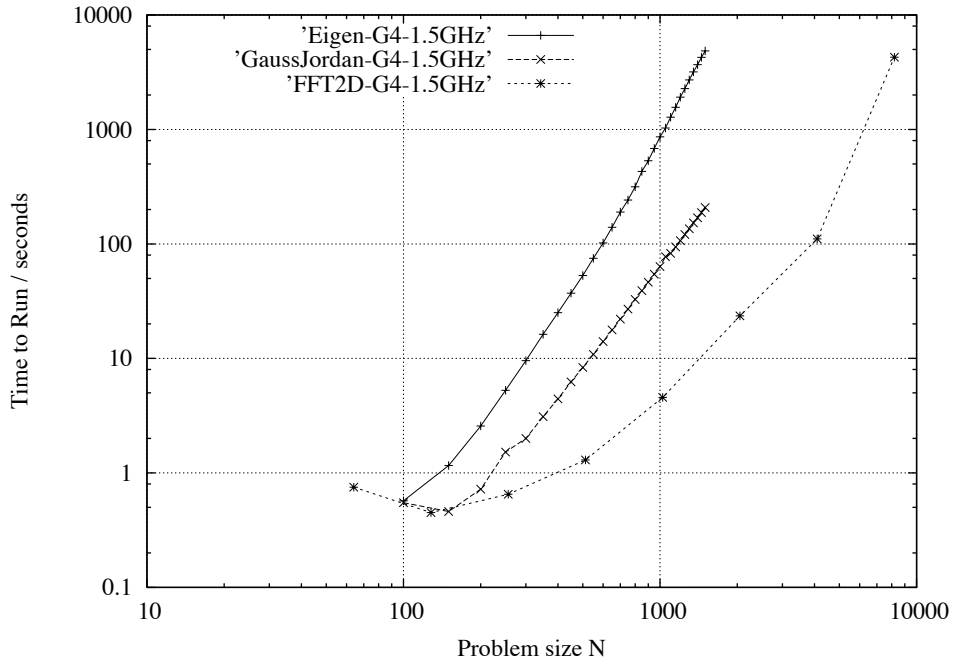


Figure 2: The three kernel codes run on a G4 Laptop (1.5GHz) running Tiger 10.4 OSX and JDK 1.5.

interested in - namely the performance of Java application/simulation programs on the platforms in question.

Figure 2 shows timing data for the three kernels running on a G4/1.5GHz Apple laptop with the Tiger 10.4 Operating system and the Java Development Kit 1.5 Beta Java Virtual Machine. The Eigen problem and Gauss Jordan elimination algorithms settle down into distinct straight lines on the log-log plot, showing the dominant complexity powers in the algorithms - with some initially complicated behaviour at low problem sizes. The FFT shows the expected $N \log N$ behaviour although there is also some exceptionally long time for the FFT algorithm as it runs out of memory on the test platform (0.5 Gigabytes for this laptop)

Figure 3 shows all the kernel data together which is mostly useful for illustrating the different power dominances and different regimes.

Figure 4 shows just the Eigen problem timing data and emphasises a cache size anomaly effect on one platform as problem size hits $N = 600$. This data can not be fitted with a polynomial of all positive coefficients - emphasising the non-trivial

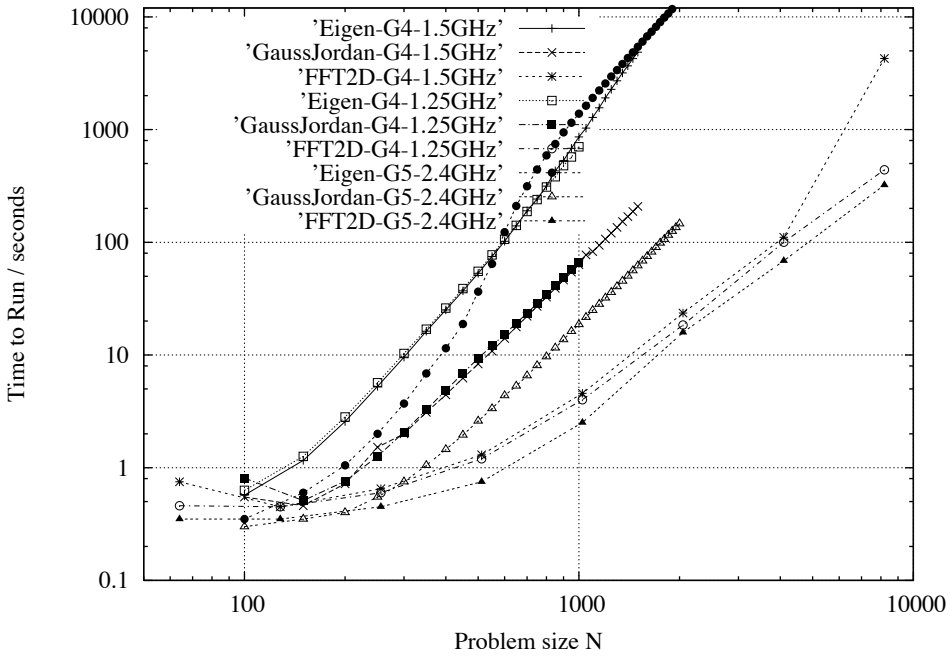


Figure 3: The three kernels run on G4 and G5 processor combinations with Tiger OSX and JDK 1.5, showing different limiting slopes (power laws) for the different algorithms. The FFT data shows a marked cache/memory limiting effect at large $N$ as it exceeds available main memory available.

nature of cache memory effects on timing complexity.

As well as the data for G4 and G5 data, we include for comparison a run of the Eigen problem on the Opteron 250 processor running Linux and JDK1.4. Within the available memory range available to us, this platform does well on performance.

## 5 Some Application Considerations

In this section we review some of the key properties of simulation applications codes we have developed, or are developing to make use of the Monte cluster and architectures like it.

For the most part the key consideration is access to high (greater than 2GBytes) of memory for the simulated systems.

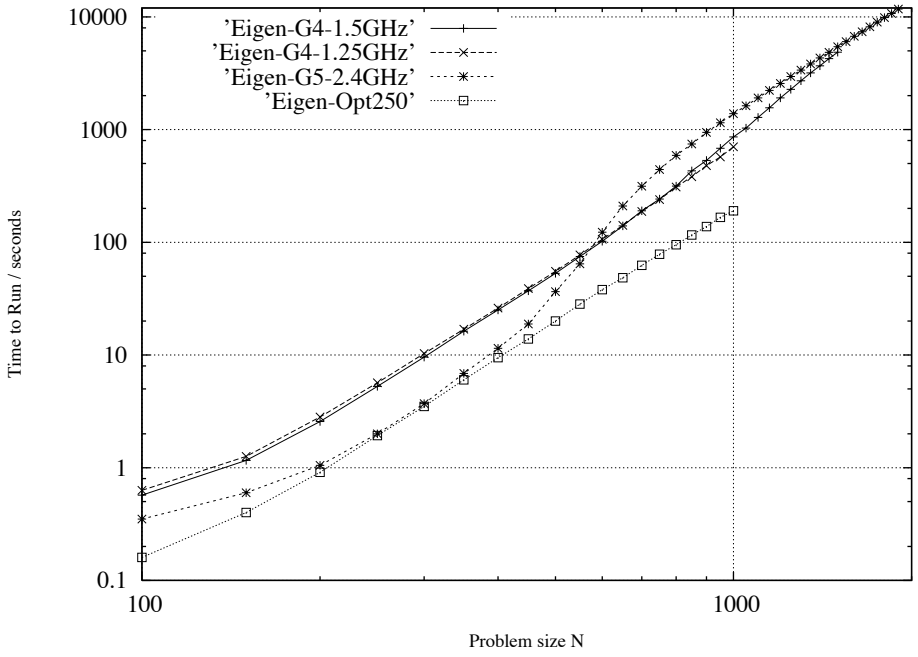


Figure 4: Eigen system times only on various platforms with JDK1.5, showing slave01(G5 2.4GHz processor) anomaly that is almost certainly a cache-size memory anomaly associated with the JVM.

In the following sub-sections we discuss the applications that we have so far run on Monte: Ising Model (section 5.1); Diffusion Limited Aggregation (section 5.2); Artificial Life (section 5.3); Sensor Networks (section 5.4); and Cluster-Cluster Aggregation (section 5.5).

## 5.1 Ising Model with Irregular Lattice

The Ising model is a simulation of a ferro-magnetic material and how it can exhibit spontaneous magnetism when slowly quenched from a high temperature state (see figure 5). Typical simulations of the Ising model feature lattices that are regular; that is in which each lattice site, or atom, is directly connected to the nearest four neighbours (in two dimensions) or six neighbours (in three dimensions).

We are studying the effects of the model when the lattice is perturbed through the introduction of 'holes' in the material or alternatively when Small-World [17] links

are added to the structure. Small-World links are akin to long distance short-cuts, making neighbours out of potentially distant atoms.

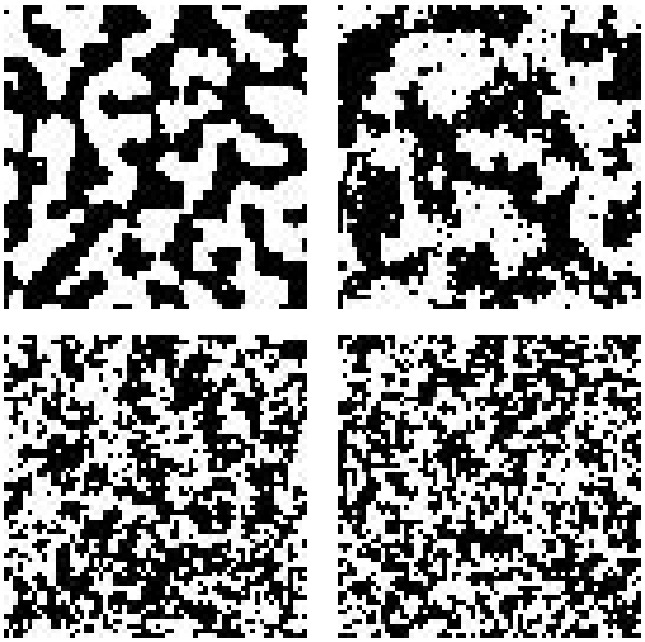

Figure 5: Four configurations of the Ising model of a two dimensional ferromagnet showing ordered and disordered magnetic domains (clusters) for various simulated temperatures.

When the Ising code is run on a regular lattice there is no need to store the locations of each of the neighbour bonds for each atom. As such most of the hardware's memory can be devoted to representing atoms' spins, however when lattices are perturbed it is necessary to have the neighbours explicitly stored in memory, thus reducing the effective number of lattice points that are representable in the same memory space. Monte's 64-bit memory addressing allows us to simulate a much larger system than would otherwise be accessible using 32-bit operating systems.

## 5.2 Diffusion Limited Aggregation

Diffusion Limited Aggregation (DLA) seeks to simulate the way in which particles, such as crystals, agglomerate through the process of diffusion. In this simulation we start with a central fixed particle, the seed, and release single particles from a large radius. The mobile particle is allowed to diffuse randomly until it touches part of the fixed structure, where it sticks.

Computationally, this simulation represents space as a large, albeit sparse, array of discrete particle locations. Monte's 64-bit memory addressing is vital for the ability

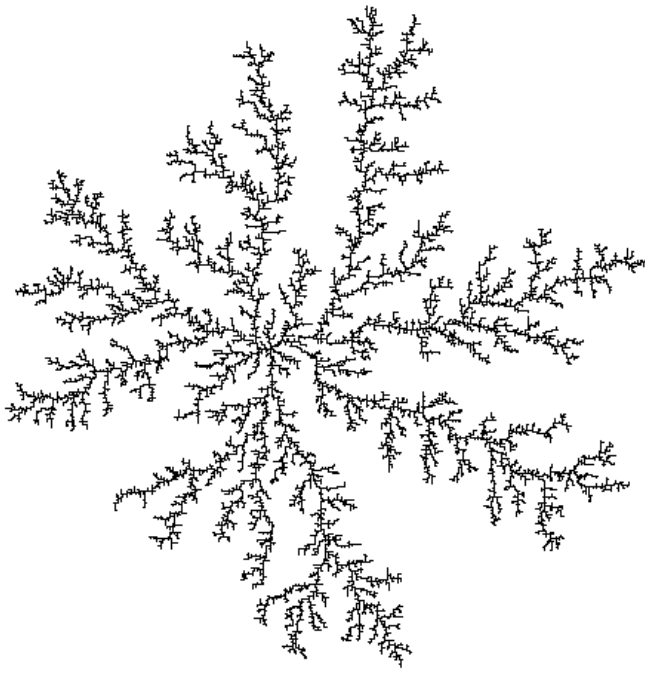

Figure 6: A two-dimensional diffusion limited aggregation cluster of approx 17,000 particles.

to simulate a large amount of space in order to grow particles with upwards of $10^{10}$ particles.

## 5.3 Artificial Life Models

We have developed a animat-based Artificial Life model for exploring the emergent behaviour from large ensembles of interacting animats.

Our experimental animat model is a simple predator-prey model; the animats nominally consist of prey ('rabbits') and predators ('foxes'). Rabbits are considered to have an unlimited amount of food 'grass'. In contrast our foxes only predate rabbits; they do not eat grass in the model. A two-phase randomized update method is used to update the system between discrete timesteps. This allows us to establish a well-defined movement phase where only spatial positions are changed and a number-changing phase where animals are born or die. Our model is based on an open system space. Animats occupy integer coordinates and in that sense the model is an automata.

The evolutionary rules of our system are relatively simple. They are consulted in

the priority order that they appear in the list below. At every time step, each animal executes one rule in order of priority. If rule 1 is executed, all other rules for the animal are ignored. If rule 1 can not be executed, rule 2 is used, and so forth. As previously mentioned, it is possible for animals to be located at the same cellular coordinates as another animal; since our animals have no notion of a third dimension, this just approximates use of a model with finer-grained cells and with longer distances of animal perception. An alternative interpretation is that our animats are Bosonic rather than Fermionic in that more than one animat can simultaneously occupy the same discrete state.

The prey or 'rabbit' rules are:

- move away from a fox if the fox is adjacent;
- breed if a rabbit is adjacent and less than 5 rabbits are nearby;
- move towards a rabbit if the rabbit is nearer than 20 spatial units;
- move to a randomly selected adjacent position.

The predator or 'fox' rules are as follows:

- eat a rabbit if the rabbit is adjacent;
- move towards a rabbit if it is nearer than 80 spatial units and this fox is hungry;
- breed if a fox is adjacent and less than 3 foxes are nearby;
- move towards a fox if it is nearer than 80 spatial and this fox is not hungry;
- move to a randomly selected adjacent position.

Animals live on an open coordinate system space. There is no 'bounding box' or array of cells. Each animal stores its own coordinates. Although we do not enforce conservation of energy in the sense that grass is always available to rabbits, we do ensure transactional semantics to ensure no rabbit is eaten more than once.

The model is described in more detail in [13, 14, 10].

The main performance issues are associated with the $O(N^2)$ searching requirements for animal-animal interactions. This gives rise to large memory requirements, particularly if we use memory-intensive Objects for each animat. Some performance improvement can be obtained through the use of truncated interaction distances and judicious use of look-up tableau. However, many of the obvious performance improvement techniques do provide a positive trade-off against even higher memory requirements.

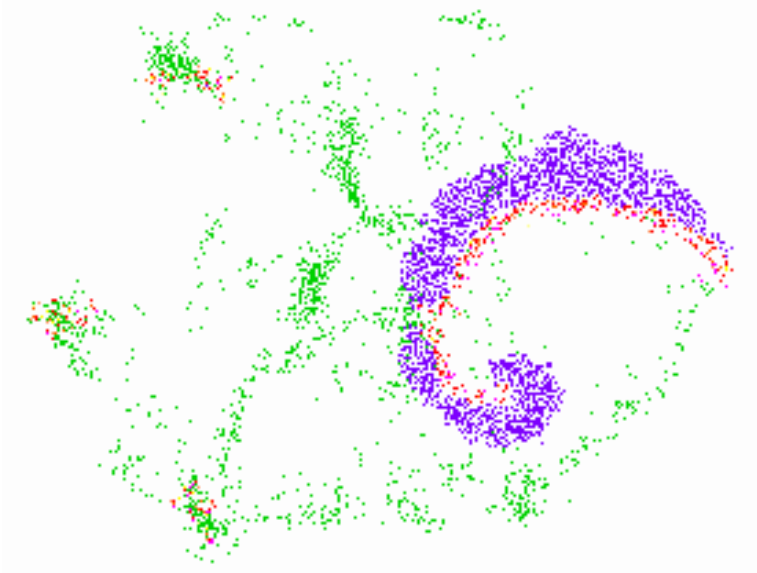

Figure 7: Visualisation of the predator-prey model state. Dark pixels (blue) are the prey ("rabbits"). Lighter pixels (red) represent predators ("foxes"). The rabbits are relying on safety in numbers while fleeing from the attacking foxes. Very light pixels (green) show the location of fox corpses when they died of starvation.

## 5.4 Sensor Net Coverage

The field of *ad-hoc* networks is an important and active one with many new applications arising from the viability of commodity priced deployable devices such as personal digital assistants (PDAs) and other mobile agents or devices. There are however some non-trivial problems in optimising *ad-hoc* networks in terms of component cost and performance and reliability. The recent interest in small-world network effects has highlighted the applicability of both graph theory and scaling theory to the analysis of networked systems. We described some novel *ad-hoc* network scenarios involving small-world network effects and show the influence of 'shortcuts' on the behaviour and properties of *ad-hoc* networks, comprising of wireless agents and sensor networks in [7].

The performance limitations of graph-based simulations in this arena are due to the fundamental high computational complexities of the requisite algorithms. Although some memory-performance trade-off techniques and heuristics can be applied these calculations are still heavily constrained by the hardware limitations of a system like Monte.

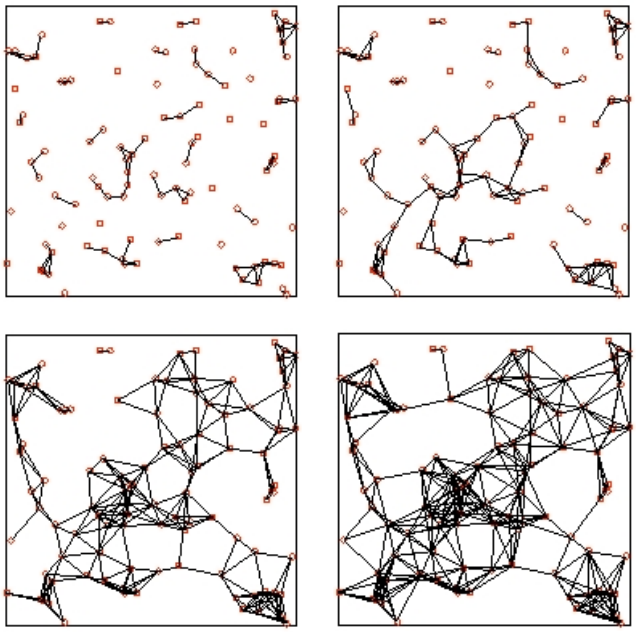

Figure 8: 100 Nodes of a circular region of influence, each of radii 0.080, 0.099, 0.150 and 0.180, in a unit square.

## 5.5 Cluster-Cluster Aggregation

In a similar thread of research to that reported in section 5.2, we have also been experimenting with a Diffusion Limited Cluster Aggregation (DLCA) [15, 11] code. Whereas the DLA code features a single stationary particle and at each step a single particle is released and allowed to diffuse until it aggregates with the existing particle aggregate, the DLCA code maintains a 'soup' of particles which are allowed to individually diffuse within the boundaries of our simulation. Akin to the DLA model, when the particles collide they aggregate. Thus, the system may initially contain many clusters of very small size. Over time the clusters aggregate to form larger and larger clusters. Clusters diffuse in certain directions according to their relative mass. A snapshot of our system is shown in figure 9.

The current state of the art in DLCA simulations has produced systems with hundreds of clusters with many million particles in each. For this reason it is necessary to be able to produce simulations of very large 2-, 3- and higher-dimensional systems. As can be gleaned from the discussion above, it is not merely sufficient that we represent our very large lattice: there are quite a few subsidiary data structures that can grow quite large, too. This requires the ability to access those regions of memory higher than the 4GB limit imposed on current 32-bit operating systems, or the 2GB addressable by signed 32-bit pointers. We have been successfully simulating very large systems using our Macintosh G5 processing nodes running the

Figure 9: Snapshot of our system at simulation time 508 when run with Peclet number of 1.0. A right to left drift force is applied, which biases the random choice of particle movement.

64-bit Tiger operating system that we were not able using previous versions. Also required is the ability to instantiate a single variable with sufficient range to be able to effectively use memory above the 2GB limit. In our 64-bit version of this code we treat `size_t` structures as if they are `int`s for the purposes of array indexing, etc.

## 6 Conclusions

We have discussed a wide range of simulation applications with different system requirements. In general, the large-scale statistical simulation ensembles that we require to investigate emergent and complex-systems behaviours mandate a computational platform with high amounts of memory and as fast a processor clock speed that is economically available.

Generally, processor clock speed availabilities are a hostage to Moore's Law, whereas there is usually considerable discretion available to configure a platform with memory. We, like many other system procurers, are limited by the economic cost of memory, rather than any particular fundamental limits set by the hardware. This situation is likely to continue even with bus addressing effects associated with the transition from 32- to 64-bit addressing. Languages like C/C++ typically have types such as `long long int` and `long double` which support use of the new 64-bit word lengths from the software and algorithms perspective. It then becomes a matter of importance for programmers of applications such as the ones we describe in this paper, to be aware of the issues involved in addressing data structures that transcend the 2GB (signed) and 4GB (unsigned) memory barriers.

## Acknowledgements

Thanks to IIMS for buying Monte.

## References

[1] Apple Computer Ltd. *G5 Processor Particulars* available at http://www.apple.com/powermac

[2] Barczak, A.L.C., Messom, C.H., Johnson, M.J., *Performance Characteristics of a Cost-Effective Medium-Sized Beowulf Cluster Supercomputer*, Research Letters in the Information and Mathematical Sciences Volume 5, June 2003, ISSN 1175-2777

[3] Free Software Foundation. *Gnu Compiler Collection* available at http://www.gnu.org

[4] Gan-El. M. and Hawick, K.A., *Parallel containers - a tool for applying parallel computing applications on clusters.* Research Letters in the Information and Mathematical Sciences ISSN 1175-2777, Information and Mathematical Sciences, Massey University, Albany, North Shore 102-904, Auckland, New Zealand, May 2004. CSTN-005.

[5] Hawick, K.A. and James, H.A., *Large Scale Spatial Simulation Optimisations*, Technical Report CSTN-032, February 2006.

[6] Hawick, K.A. and James, H.A., *Performance, scalability and Object-Orientation in Discrete Graph-based Simulation Models*, and in: Proc. Int. Conf on Modeling, Simulation and Visualization Methods (MSV'05), June 2005, Las Vegas, USA.

[7] Hawick, K.A. and James, H.A., *Small-World Effects in Wireless Sensor Networks*, International Journal of Wireless and Mobile Computing. Special issue on Mobile Systems, E-Commerce and Agent Technology. Vol 1, Issue 7, 2005.

[8] Hawick, K.A., James, H.A. and Scogings, C.J., *Grid-Boxing for Spatial Simulation Performance Optimisation*, in Proc 39th Annual Simulation Symposium - 2-6 April 2006, Huntsville Alabama, USA, Pub. IEEE Computer Society.

[9] Hawick, K.A., James, H.A. and Scogings, C.J., *High-Performance Spatial Simulations and Optimisations on 64-Bit Architectures* Technical Note CSTN-026, October 2005.

[10] Hawick, K.A., Scogings, C.J. and James, H.A., *Defensive Spiral Emergence in a Predator-Prey Model*, In Proceedings of Complexity 2004, Cairns, Australia,

December 2004, Edited by Russel Stonier, Quinglong Han and Wei Li, PP 662-674.

[11] Hellén, E.K.O., Salmi, P.E., and Alava, M.J., *Cluster Persistence in One-Dimensional Diffusion-Limited Cluster-Cluster Aggregation*, Physical Review E 66, 051108 (2002). http://de.arXiv.org/abs/cond-mat/0206139

[12] James, H.A. and Hawick, K.A., *Trends in Cluster Computing Scheduling and the Missing Cycles*, in Proc. Int. Conf on Parallel and Distributed Processing techniques and Applications (PDPTA'05), June 2005, Las Vegas, USA.

[13] James, H.A., Scogings, C. and Hawick, K.A., *A Framework and Simulation Engine for Studying Artificial Life.* Research Letters in the Information and Mathematical Sciences Volume 6, ISSN 1175-2777, May 2004.

[14] James, H.A., Scogings, C. and Hawick, K.A., *Parallel Synchronisation issues in Simulating Artificial Life*, in Proc IASTED Int. Conf on Parallel and Distributed Computing and Systems (PDCS), MIT, Cambridge, November 2004.

[15] Peltomäki, M., Hellén, E.K.O., and Alava, M.J., *No self-similar aggregates with sedimentation*, J. Stat Mech, JSTAT (2004) P09002.

[16] Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P., *Numerical Recipes in C++* Second edition, Cambridge University Press, 1988.

[17] Watts, D.J. and Strogatz, S.H., *Collective dynamics of 'small-world' networks*, Nature (393), 4 June 1998.