Grid-Boxing for Spatial Simulation Performance Optimisation

K. A. Hawick and H. A. James and C. J. Scogings

2006

Computer simulations of complex systems such as physical aggregation processes or swarming and collective behaviour of life-forms, often require order N-squared computational complexity for N microscopic components. This is a significant handicap to simulating systems large enough to compare with real-world experimental data. We discuss space partitioning methods for two such simulation codes and demonstrate complexity improvements by taking advantage of information about locations and interaction distances of the microscopic model components. The space of our models are partitioned into grid-boxes. We present results for a diffusion-limited cluster-cluster aggregation code and for an artrificial life simulation code. We discuss the data structures necessary to support such algorithms and show how they can be implemented to obtain high performance and maximal simulation productivity for a given computational resource. There are some subtlties in whether such spatial partitioning algorithms should produce a computational complexity of N to some power between 1 and 2 or whether they should be order N log N. We discuss these effects in the context of our data.

Keywords: complexity; particle-in-grid; performance optimisation

BiBTeX reference:

```
@INPROCEEDINGS{CSTN-027,
        author = {K. A. Hawick and H. A. James and C. J. Scogings},
        title = {Grid-Boxing for Spatial Simulation Performance Optimisation},
        booktitle = {Proc. 39th Annual Simulation Symposium},
        year = {2006},
        editor = {T. Znati},
        pages = {98-106},
        address = {Huntsville, Alabama, USA},
        month = {2-6 April},
        organization = {The Society for Modeling and Simulation International},
        publisher = {IEEE Computer Society},
        keywords = {complexity; particle-in-grid; performance optimisation}
}
```

Grid-Boxing for Spatial Simulation Performance Optimisation

K.A. Hawick, H.A. James and C.J. Scogings
Institute of Information and Mathematical Sciences
Massey University – Albany
North Shore 102-904, Auckland, New Zealand
Email: {k.a.hawick,h.a.james,c.scogings}@massey.ac.nz
Tel: +64 9 414 0800   Fax: +64 9 441 8181

Abstract— **Computer simulations of complex systems such as physical aggregation processes or swarming and collective behaviour of life-forms, often require order N-squared computational complexity for N microscopic components. This is a significant handicap to simulating systems large enough to compare with real-world experimental data. We discuss space partitioning methods for two such simulation codes and demonstrate complexity improvements by taking advantage of information about locations and interaction distances of the microscopic model components. We present results for a diffusion limited cluster-cluster aggregation code and for an artificial life simulation code. We discuss the data structures necessary to support such algorithms and show how they can be implemented to obtain high performance and maximal simulation productivity for a given computational resource. There are some subtleties in whether such spatial partitioning algorithms should produce a computational complexity of N to some power between 1 and 2 or whether they should be order N log N. We discuss these effects in the context of our data.**

Keywords: complexity; particle-in-grid; performance optimisation.

I. INTRODUCTION

Computational science demands continual improvement in the capability to simulate larger and more realistic simulation models. A particular problem of high computational cost is simulating a system of microscopic components such as a particle system as used in movie and game special-effects rendering. This problem also occurs in defence and military simulations. The issue of optimising proximity detection has been considered in a general manner elsewhere in the defence literature [22], [23]. In this paper we focus on some specific algorithmic approaches.

There are a number of algorithms in the literature for optimising the performance of N-Body problems both using serial and parallel code. Some of these algorithms, including the Barnes-Hut [2], Fast Multipole [5] and Parallel Multipole Tree [4] are reviewed in [3]. The Barnes-Hut and Fast-Multipole algorithms are well established method for reducing the computational complexity of a particle model [20] from $O(N^2)$ to around $O(N \log N)$ by managing the particles in an oct-tree, or quad-tree for a two dimensional system, and applying an **approximation** for the net force on a particle from all other particles, respectively. The interactions are truncated to finite distance and with an appropriate hierarchical structure of spatial cells, the computationally costly interactions need not all be computed.

These algorithms unfortunately are not applicable to all interaction models and unless the density patterns in a system are highly non-uniform the additional book-keeping needed to maintain the necessary data structures can outweigh the advantages. In this paper we describe a method inspired by Barnes-Hut that reduces the computational complexity of spatially discrete simulation models with N microscopic components that interact over short to medium range but which have particle densities that are still rela-

tively space-filling. The approach involves a spatial partitioning of the model system into grid boxes that are all of the same size at any time, but which size can be adapted slowly in simulation time to always contain roughly the same number of microscopic particles. As we discuss, halo effects aside, this means that the computational complexity is considerably reduced to some power p of the number of particles N with $1 < p < 2$. In the models we describe this is **not an approximation**. It is a data-structuring and code book-keeping algorithm that speeds up execution but which gives identical results to the normal algorithm.

In section II we describe two simulations codes: one is an artificial-life model and the other a diffusing cluster-cluster aggregation model. We describe the data structures for implementing grid-boxing in section III. We present some results showing the optimal grid box sizes in section IV and a discussion of the most efficient parameter regimes in section V. We indicate some lines of future inquiry and our conclusions in section VI.

II. TWO SIMULATION MODELS

Different simulation code details manifest themselves in different measures of performance accruing from the various tradeoffs achievable in terms of the computational complexities of different parts of the code. It is therefore useful to apply the grid-boxing ideas and algorithm to more than one simulation program. In this section we describe a diffusion limited cluster-cluster aggregation code [13], [17] (DLCA) and an animat-based artificial life (ALife) simulation code [7], [10], [11], [14], [15]. The DLCA code represents a simpler basic model with grounding in simple physics and chemistry. The ALife model is based on a smart and adaptive predator-prey scenario. Both models are based on microscopic components which interacts through short to medium range forces.

A. Diffusion-Limited Cluster-Cluster Aggregation

We have been developing a Diffusion-Limited Cluster-Cluster Aggregation (DLCA) [9] code in order to simulate the aggregation of particles into clusters in a fixed-boundary system in 2-, 3-, 4- and 5-dimensions. The dimensional dependence of critical and growth properties of the model are

themselves of interest. When particles in our system encounter one another, they stick together, forming clusters which monotonically grow in size. Clusters never break apart. In this text we use the term 'cluster' bearing in mind that a cluster consists of one or more aggregated particles.

Similar to many N-Body problems, our code represents particles as points within a discrete space and clusters as groups of multiple particles. In order to update the system we consider how each of the clusters may move, with certain probabilistic chance move in each direction and dimension, with computed probabilities weighted by model parameters such as temperature and gravitational drift force. Thus, when a cluster moves, we must test whether it has collided with another cluster, thus aggregating and forming a larger cluster. The naive approach, therefore is to test every other particle in every other cluster for collision, each time one is moved. This is the so-called all-pairs algorithm. This leads to an effective algorithmic complexity for our system of $O(N^2)$. In the presence of a very dilute system, where N is low, this is not a huge inconvenience. However, when N is large, this rapidly becomes a huge overhead.

We take the simpler approach of simply dividing space into equal-size boxes that can contain (sometimes vastly) differing numbers of clusters. In fact, the only thing that is constant in our system is the area of space contained in each box structure. For ease of management of computation, our n-dimensional boxes have equal size in each dimension.

At programme initialisation, particles and hence clusters are arranged randomly in our d-dimensional space. Our gridding algorithm then inspects the physical arrangement of clusters in each grid box, taking note of which clusters have elements in each grid box. Note that larger clusters may span multiple grid boxes.

Upon each iteration, each cluster is sampled, in a random order with equal probability, for potential movement. When a cluster is moved within the system, instead of using the all-pairs algorithm, we restrict checking to only those clusters in grid squares neighbouring those inhabited by the cluster in question. This means that for small-to-moderate cluster sizes, the number of grid boxes (and hence other clusters) that must be checked for collisions

is far smaller than in an $O(N^2)$ algorithm. It does however, involve an overhead requirement for maintaining up to date grid box information at every step.

At each step we record the number of particles in the largest cluster in the system. In an effort to reduce computational complexity, we try to ensure that each cluster is able to be stored within a single grid box (although the cluster may actually straddle a number of grid boxes): this reduces the halo of grid boxes that we must search in order to correctly update our system.
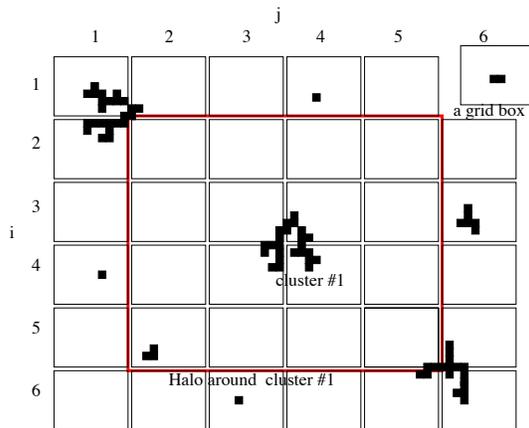

Fig. 1. Grid boxes and halos and domain decomposition. The simulation lattice is divided evenly into a number of rectangular grid boxes. Clusters may span multiple grid boxes. Instead of searching *every other* cluster to test for possible interactions, we restrict the search to the grid boxes that a cluster currently occupies and its surrounding halo.

Seeking to maintain a small halo of neighbouring grid boxes to search as clusters get larger requires us to re-partition grid box space as soon as one cluster gets too large. For convenience we simply double the size of the grid box in each dimension (thus increasing its size by a factor of 2^d) until it is at least big enough to hold the largest cluster. Repartitioning grid box space involves recomputing which grid box each cluster now inhabits. When the size of the largest cluster means that it is no longer efficient to maintain the grid boxes, grid-boxing is discarded and the code executes as an all-pairs code. It is not obvious that using scale factor for the grid box size of other than 2 would be worthwhile.

Some slight further optimisation is also possible if we compare dimensional components of the distance between two clusters. For example if we immediately compare the partial distance in the first dimen-

sion it is possible to immediately eliminate some possible proximities without recourse to computing the full Pythagorean distance.

B. Animat Model of Artificial Life

Our Animat [6], [21] model of artificial life treats different predator-prey species as cellular automata. Different species 'animats' inhabit an unbounded space which can either resemble a 'soup' or have definite physical spatial boundaries. Some very popular Artificial Life systems include [1], [8], [16], [18].

Animat-based Artificial Life system have the interesting property that complex behaviours can emerge [19], at a macroscopic scale for the whole system, from the relatively simple microscopic rules that govern the way that animats can interact. Different models may restrict the modification or even evolution of animat behaviour rules during the execution of a simulation.

Our system features two different types (species) of animat: a predator ('fox') and prey ('rabbits'). Each animat has a built-in set of rules and at each iteration of the model (time step) the animat selects one rule to execute. The rules have conditions and are placed in the set in order of priority, thus if the conditions for rule 1 are met then rule 1 will be executed. If the conditions for rule 1 are not met then the conditions for rule 2 are checked and so on. Typical rules include:

- if hungry move towards the nearest rabbit (a rule for foxes)
- if not hungry move towards the nearest fox (a rule for foxes)
- if a fox is adjacent move directly away from it (a rule for rabbits)
- if no fox is adjacent move towards the nearest rabbit (a rule for rabbits)

Some rules ensure survival of an individual by moving towards prey or moving away from predators and some rules ensure the survival of the species by moving towards a potential mate of the same species. Thus every animat requires the location of its nearest neighbours (of both species). In addition, the chances of successful breeding depend on the number of animals in the immediate vicinity of the breeding pair. Neither species will breed in a

situation where there is overcrowding and predators will not breed unless there is sufficient prey nearby.

This means that at every time step, every animat has to locate not only its nearest neighbours but also has to count the number of animats in its local area. In early versions of the model, animats were all maintained in a single list and thus the only way to locate neighbours was to perform an exhaustive search on the list. This process was extremely time consuming and the time taken to locate neighbours rapidly increased as the number of animats increased. This is clearly shown by the 'no grid' line in the graph in Figure 9.

The time increase is caused by checking every other animat. However, this is unnecessary as animats only react to other animats within their local area. Thus the checking process can be made more efficient by using a grid system and maintaining a list of animats in each grid box. If this is done, animats only have to check the other animats within the same grid box although animats that happen to be near the edge of a box also have to check the list of animats in the neighbouring box(es).

Because the local area of interest is fixed, the size of a grid box remains fixed for the duration of the model. It is important to ensure that grid box size is not significantly smaller than local area size as this would force animats to check a large number of surrounding grid boxes. On the other hand if the grid box is significantly larger than a local area each box would contain a large number of animats and the process of locating neighbours would once again be checking a large number of animats outside the local area.

III. DATA STRUCTURES

Our code uses a cluster data structure as shown on figure 2. The physical lattice uses conventional Cartesian coordinates (x, y, z) or $(x_1, x_2, x_3, x_i, .., x_d)$ for a d dimensional system. The lattice is of size L_i in each dimension and we can encode the x_i into a single integer storage coordinate which we term a k index. The k index values are effectively pointers into our $\mathcal{R}^d$ lattice space and make it easy to group clusters in terms of arrays of k values.

Figure 3 shows an indirect addressing scheme that is useful for hyper-cubic symmetry lattice models
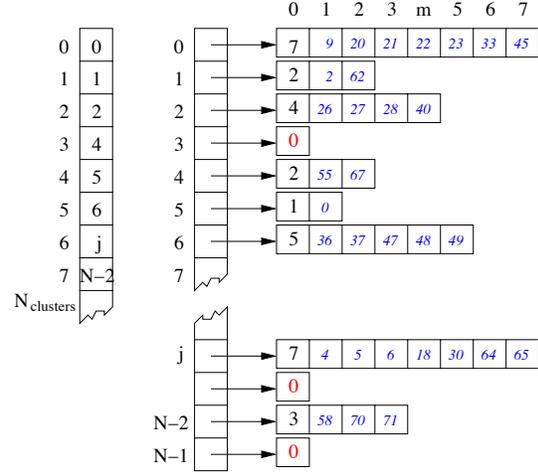

Fig. 2. Internal representation of clusters using hierarchies of arrays. As shown on the right hand side, the actual cluster data for a cluster with n particles is stored in an array of size $(n + 1)$. The zeroth element contains the size of the cluster. the remaining elements store the 'k' values of the cluster contents. Some clusters may be zero-sized through being merged with other clusters. For this reason the absolute cluster numbers (centre array) may be a sparse array. We indirectly reference this through the left-hand array to make it easy to iterate across each cluster in the system.

in arbitrary dimension d. The lattice lengths $L_i, i = 1, ..., d$ in each dimension are fixed and hence particle positions $x_i, i = 1, ..., d$ on the lattice can be encoded as a single integer $k = x_1 + x_2 \times L_1 + x_3 \times L_2 \times L_1$ and so forth. These "k-pointers" can be used to identify the location of the m'th member particle in the j'th cluster.

IV. SCALING ANALYSIS AND RESULTS

In this section we present and discuss some scaling results for firstly the Diffusion-Limited Cluster-Cluster Aggregation(DLCA) model and secondly the Animat ALife model.

A. DLCA Scaling Results

Figure 1 shows the basic partitioning scheme for a projection of a simulation onto a 2-dimensional space. The DLCA model is restricted to a finite or closed space with periodic boundary conditions. We typically run the DLCA model until clusters have aggregated to a small number of final clusters, usually between 100 and 1000 in number. When

Fig. 4. DLCA simulation nClusters=2365. Using gridding reduces the average number of clusters tested at each update step. This histogram shows that the most frequent case in this simulation, at this The average time taken to do an update step reduces by a factor of approximately 30.

Fig. 3. In a 12×6 system, where L[1]=12 and L[2]=6, we assign each lattice point a unique value called a 'k' value. Simple functions can convert between 'k' values and traditional (x_1, x_2) vectors. Using 'k' values in the body of our code removes any dependencies of the code on the actual dimensionality of the simulation being studies. In this figure grey boxes represent those site which are filled and white boxes are empty. In out code we use the convention that in any array the first element is the size of the array.

starting with millions of individual particles, this can take on the order of hundreds of processor hours.

In contrast, the Animat model uses open boundary conditions and animals can roam in any direction. In practice we only run the model for finite times and animals can therefore only stray so far in any one direction in finite lifetimes. Rogue individual animats in fact die of starvation if they roam too far from the herds, which also severely limits the spatial size of the system.

The grid scheme involves choosing an initial grid box size in each dimension so that for any calculations on a particular animat or cluster only requires information about animats in the same grid box or in the immediately neighbouring grid boxes. Our animat model has a maximal interaction distance of 60 spatial cells. This governs good choices of the grid size.

Figure 4 shows a histogram of the number of clusters in the DLCA simulation required to search through in order to update the positional information for a cluster in the system. This histogram is of simulation step 100, which is very near to the start of any interesting simulation; there are currently 2365 clusters represented inside the system. The position
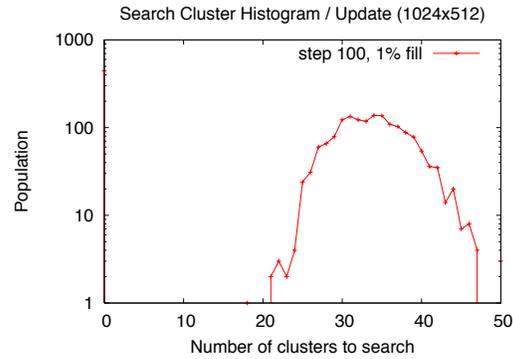
of the distribution in the figure shows that, on average, the system must search through approximately 30 other clusters in order to update the present cluster. The maximum number of clusters that must be searched through in this current configuration is 52. This is far less than performing a $N^2 = 2365^2 \approx 5.5 \times 10^6$ cluster search in order to test for potential interactions. Not shown on the diagram is the number of clusters with zero neighbours: there are 439 of these clusters that do not require any other neighbouring grid sites to be checked. This, again, is a huge efficiency advantage.
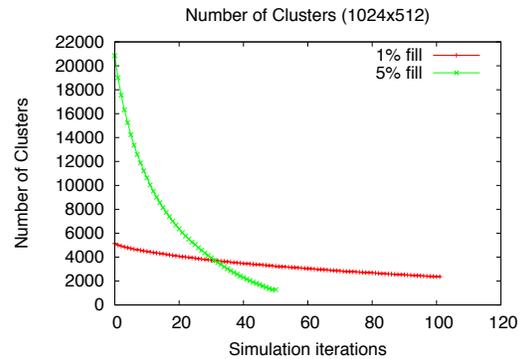

Fig. 5. Number of DLCA clusters over simulation iterations in a 1024x512 system with initial fill fractions of 1% and 5%. The number of clusters falls more rapidly in the concentrated system as there is a higher chance of colliding with another cluster.

Figure 5 shows the fall in the number of DLCA clusters in our sample 1024x512 system over time for an initial fill fraction of 1% and 5%. It is sensible that the 5% fill fraction simulation will have a larger

number of clusters in the initial configuration. It is also sensible that due to random particle movements the probability of clusters aggregating is higher – simply because there are more of them in the system. It can be seen that in the 5% fill fraction simulation the number of clusters fall quite quickly but soon plateaus as clusters get larger and the chance of larger clusters colliding in space falls. In contrast, the number of clusters in the 1% fill fraction simulation falls quite slowly, due to the lesser change of particles colliding in the dilute space.
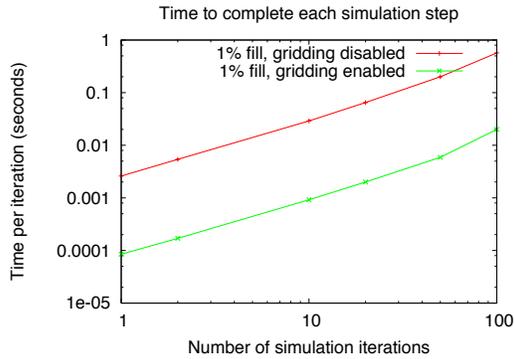

Fig. 6. Log-log graph of the time taken to complete each update step factored by the number of clusters in the system at each data point.

Figure 6 shows the time that the DLCA code takes to perform each update step factored by the number of clusters in the system at the time the measurements are taken. The slope of the graph is positive, indicating that as the simulation progresses, even though the total number of clusters in the system is declining (see figure 5), the searches through the clusters is more complex because of cluster aggregation. In the case of the gridded simulation, the size of the grid box is based on the size of the largest cluster in the system. As clusters aggregate, grid boxes are expanded so that each cluster can fit entirely within one grid box. However, this does not mean that a cluster will be placed entirely within *one* grid box – they may overlap a number of boxes. Thus, the number of neighbouring grid boxes that need to be checked for nearby cluster is still significant. We see from this figure that using a grid box approach leads to a time saving per factored update step of approximately a factor of 30 compared with not using grid-boxing, even incorporating the time taken to maintain the grid structures.
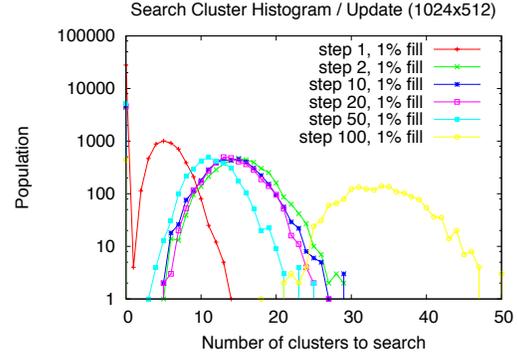

Fig. 7. Histograms of the number of neighbouring clusters searched by the gridding algorithm in our DLCA code for given time steps in a system with initial fill fraction of 1%. The histograms actually represent the number of grid boxes whose contained clusters have the given number of nearby neighbours.

Figures 7 and 8 show an overlayed histogram of the number of clusters that the gridding algorithm has chosen to search at each step. Each curve represents the distribution of cluster neighbours in internal state of the system at a given time-step across the grid boxes in the system. Figure 7 represents an initial fill fraction of 1% while figure 8 is for an initial fill fraction of 5%. In figure 7 the curves for steps 1 and 2 are almost co-incident: notice the large number of clusters with no (zero) nearby neighbours to search (27925 grid boxes are either empty or contain only a solitary cluster with no nearby neighbours). Clearly, there is more chance of having larger initial clusters and more neighbours in a simulation with higher initial fill fraction. Thus in step 1 using 1% fill, the majority of clusters in grid boxes have zero then four neighbours; in step 1 using 5% fill most clusters have 64 neighbours to search for interactions – a direct consequence of grid box sizes being adjusted for the largest cluster made so far.

B. Scaling Results for Animat Model

Figure 9 shows the total time taken to perform the neighbour checks for all animats in the model. Different grid sizes are used and compared with the time taken when no grid system is used. The smallest grid box size is 60 pixels as this is the vision range of an animat. If a smaller grid size were used most animats would be forced to check a number of surrounding grids including some of those not immediately adjacent.
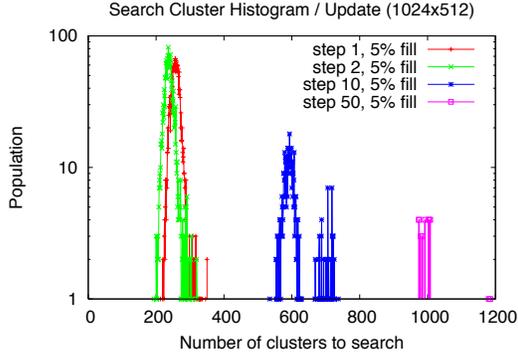
Fig. 8. Histograms of the number of neighbouring clusters searched by the gridding algorithm in our DLCA code for given time steps in a system with initial fill fraction of 5%. The histograms actually represent the number of grid boxes whose contained clusters have the given number of nearby neighbours. Clearly, there is more chance of having larger initial clusters and more neighbours in a simulation with higher initial fill fraction.
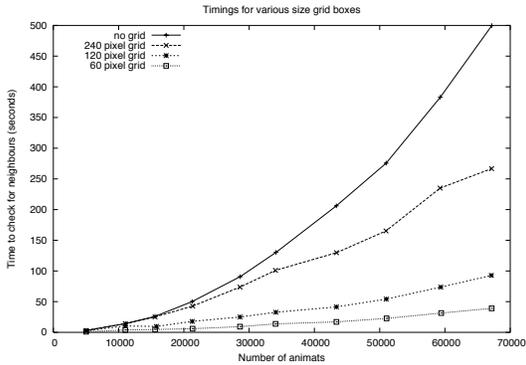

Fig. 9. Animat Model computational complexity curves

Size of Grid (pixels)	Fitted Gradient
No Grid	1.96384 ± 0.0112363
240×240	1.70614 ± 0.0359479
120×120	1.3346 ± 0.0637962
60×60	1.35362 ± 0.0576574

TABLE I

DATA FITTED TO THE ANIMAT MODEL

for the smaller grid sizes are closer to 1, indicating that the grid box system is an order of magnitude more efficient than having no grid.

V. REGIMES OF EFFICIENCY

It appears from our experiments that optimal choice of the grid-boxing size depends crucially on the system models' parameter regime. More specifically it depends on the spatial density of model system components, which of course itself depends on whatever physical model parameters that are under investigation. Generally the whole point of a numerical experiment is to determine these sorts of dependencies and it is therefore not feasible to optimise the grid-boxing sizes *a priori*. Nevertheless, the common *modus operandi* of such experiments in practice involves an initial exploration of the model parameter space followed by a more detailed or fine-grained numerical scan across parameter space, possibly involving a large number of "runs" in the case of a models that have a dependence on their initial conditions. For example, a large number of model runs might be carried out for statistical refinement purposes. Running $N = 100$ different model configurations to obtain $\sqrt{N} = 10\%$ statistics is a common approach. The computational cost of these N runs can be significantly lowered if the grid-boxing sizes have been chosen to best suit the model regimes under investigation.

We can analyse the grid size choices generally, independent of any particular model, in terms of an average density and the corresponding volumes for the whole system, the grid-box and for clusters of particles in the model system.

Figure 10 shows some examples of grid-boxing for one, two and three dimensions. In general if we take our model cell to be of unit length, and all the system length sides to the the same with value L then the average density of "particles" in the model system is $\rho_d = N_{filled}/L^d$, with the integer

The plot shows the time taken by grids of 60 pixels, 120 pixels, 240 pixels and no grid system. As the grid size is increased, it takes longer to check for neighbours as each grid contains more animats that need to be checked. If a grid is too big, it has the same effect as no grid at all as all animats are positioned within the same grid box. This effect is the reason why the no grid and 240 pixel grid are the same up to approximately 15,000 animats.

Table I provides the results of calculating the gradients of the lines shown in the graph in Figure 9. The line corresponding to "no grid" has a calculated gradient very close to 2. This is to be expected as, in the absence of grids, every animat has to locate every other animat in order to find its nearest neighbours and this process is $O(N^2)$. The gradients
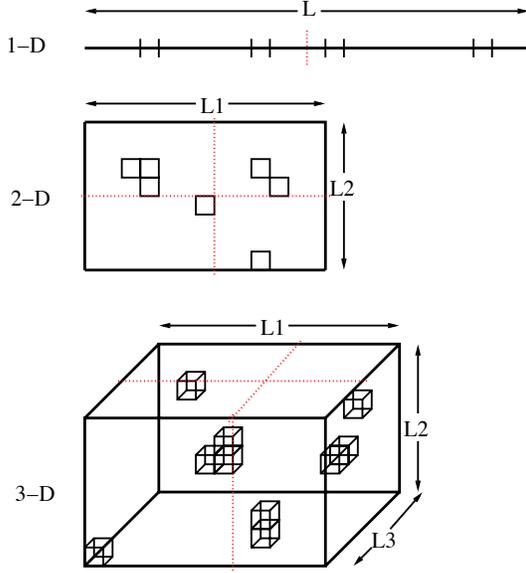
Fig. 10. Examples of grid-boxing in 1-, 2-, and 3-dimensions. We divide each dimension into grid boxes of the same size. Note that there may not necessarily be the same number of boxes in each dimension. Clusters inhabit a grid box (or may span more than one). As our codes use locally-based interactions only, it is necessary only to check surrounding grid boxes for possible interactions, thus saving an $O(N^2)$ search.

dimension $d = 1, 2, 3$. This is to be compared with a grid-box which we can take to have volume A^d for some grid-box length size A. We can then describe our composite clusters of particles in the model as having characteristic length scale $l_c \approx \rho^{\frac{1}{d_f}}$, where d_f is a characteristic effective dimension describing the model and need not be an integer. We may need to consider the maximum value of $l_c^{\max}$ present in a given configuration rather than mean values. We can then identify the following regimes for grid-boxing:

- the **dilute limit** when ρ is small and hence l_c is small compared to both L and to A. In such a regime we will need to consider only the immediate halo of a given grid-box to identify all possible (locally) interacting clusters.
- the **intermediate case** where the normalised ρ is similar to the volume of a grid box A^d and where a normal halo of grid boxes is still sufficient providing d_f ius close to d. If this last condition is not true and there are large straggling fractal clusters, their tendrils may cross the normal halo and we would need to consider maximum cluster lengths in each

dimension separately to chack that the grid-box halo does not omit any possible cluster-cluster interactions.
- the **dense limit** where clusters are typically larger than the halo of grid-boxes and in this case we need to either make the grid-box length larger, or abandon grid-boxing in favour of checking the whole system. In general it is only useful to have grid-boxing when $3A < L$ for all dimensions, since $3A$ is the minimum halo length.

It is likely that when $3A$ is close to L the book-keeping computational overheads will mean it is not worth applying grid boxing. A heuristic can be used to determine this for a given model code.

This analysis is based on an assumption that the population of interacting clusters is relatively sharp around some mean value, and that the spatial density fluctuations across the whole system are relatively insignificant. These assumptions are likely to be wrong in a model system at or near criticality where there is a phase transition. In such cases it is potentially worthwhile to construct halos on an individual basis, possibly spiralling around the central grid box in question checking each neighbouring box. In such cases it is worthwhile maintaining a grid box size that is ostensiby too small compared with $l_c^{\max}$. A small grid box can be used effectively to narrow the search for neighbours of small clusters, even although it may not help with the largest cluster. Once again a model dependent heuristic tradeoff analysis is the only way to optimise this situation. Generally, providing there are a significant number of clusters encountered that **are** smaller in size than the current grid box size, then it is worth while leaving the grid box at this size. A "significant" number in this context might be for example between one quarter and one half of the number of clusters present in the model system.

VI. SUMMARY AND CONCLUSIONS

We have reported some promising initial results for our grid-boxing spatial partitioning algorithm applied to two different simulation codes. It is worth emphasising that this approach is not an approximation but rather is an effective way to restructure the simulation algorithm to improve performance. We believe this approach is quite a valuable and general one for microscopically based spatial simulations

such as those for defence force management or emergency response scenarios.

We have shown the effectiveness of our technique for two model codes. The approach is effective because we already know what parameter regime - and hence effective system density regime - we are operating in. We are therefore able to tune the grid box size accordingly. This is shown to be important as an arbitrary choice of grid box size can in fact unnecessarily add to the computational book-keeping overhead and actually slow down the simulation calculations. Fortunately it is relatively straightforward to build in self-tuning of the grid-boxing into the simulations codes to adjust this at runtime.

Finally, we are presently experimenting with an adaptation of our algorithms for a parallel computing system. In general to date it has proved more efficient to conduct our simulation runs as independent jobs on a cluster with no parallel interactions. However as we attempt very large scale simulation systems it is more difficult to arrange to have enough memory available on a single processor and it may become attrcative to partition the grid boxes on a parallel "owner computes" basis. We discuss high-memory simulation optimisations in the context of 64-bit architectures elsewhere [12].

ACKNOWLEDGEMENTS

REFERENCES

[1] Adami, C., Avida (Digital Life Laboratory) Available at http://dllab.caltech.edu/avida

[2] Barnes, J.E. and Hut, P., A hierarchical $O(N \log N)$ force calculation algorithm. *Nature*, 324(4):446-449, December 1986.

[3] Blelloch, G. and Narlikar, G., A Practical Comparison of N-Body Algorithms, in *Parallel Algorithms*. Series in Discrete Mathematics and Theoretical Computer Science, Volume 30, American Mathematical Society, 1997.

[4] Board, J.A., Hakura, Z.S., Elliot, W.S., and Rankin, W.T., Scalable variants of multipole-accelerated algorithms for molecular dynamics. Technical Report 94-006, Duke University Dept of Electrical Engineering, 1994.

[5] Greengard, L., *The rapid evaluation of potential fields in particle systems*. The MIT Press, 1987.

[6] Hawick, K.A., James, H.A. and Scogings, C.J., Web site, containing model Snapshots and Movie sequences: *http://www.massey.ac.nz/˜kahawick/alife* Massey University 2004-5

[7] Hawick, K.A., Scogings, C.J. and James, H.A., Defensive Spiral Emergence in a Predator-Prey Model in Proc. Complexity 2004, Cairns, Australia, December 2004.

[8] Holland, J., ECHO Available from http://www.santafe.edu/-projects/echo/echo.html

[9] Hawick, K.A. and James, H.A., Growth of Inhomogeneities in Sedimentary Diffusion-Limited Cluster Aggregation (DLCA), Unpublished, 2005 - Technical Note CSTN-012, February 2005.

[10] Hawick, K.A., James, H.A. and Scogings, C.J., A Zoology of Emergent Patterns in a Predator-Prey Model Computational Science Technical Note CSTN-015, Massey University , March 2005.

[11] Hawick, K.A., James, H.A. and Scogings, C.J., Roles of Rule-Priority Evolution in Animat Models, to appear *Proc ACAL05*, Sydney, 5-8 December 2005.

[12] Hawick, K.A., James, H.A. and Scogings, C.J., High Performance Simulations Kernels and Optimisation on 64-Bit Architectures, TechnicalNote CSTN-026, October 2005. http://www.massey.ac.nz/ kahawick/cstn/026/cstn-026.html

[13] 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

[14] James, H.A., Scogings, C.J. and Hawick, K.A., A Framework and Simulation Engine for Studying Artificial Life in Research Letters in the Information and Mathematical Sciences, Vol 6, May 2004, pp143–155, ISSN 1175-2777. Available from http://iims.massey.ac.nz/research/letters/volume6/

[15] James, H.A., Scogings, C.J. and Hawick, K.A., Parallel Synchronisation issues in Simulating Artificial Life in Proc. 16th IASTED Int. Conf. on Parallel and Distributed Computing and Systems (PDCS), pp. 815–820, Boston, November 2004.

[16] Levy, S., "Artificial Life" Penguin Books, 1992. ISBN 0-14-023105-6

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

[18] Ray, T., Tierra Available at http://www.isd.adr.co.jp/˜ray/tierra

[19] Ronald, E.M.A., Sipper, M. and Capcarrère, M.S., Testing for Emergence in Artificial Life, In Advances in Artificial Life: Proc. 5th European Conference on Artifical Life (ECAL'99), Switzerland, 1999 Ed. Dario Floreano and Jean-Daniel Nicoud and Francesco Mondada, Pages 13-20, Pub Springer-Verlag ISBN 3-540-66452-1.

[20] Salmon, J.K., *Parallel hierarchical N-body methods.*, PhD thesis Caltech University, 1991.

[21] Wilson, S., The Animat Path to AI in *From Animals to Animats 1: Proceedings of The First International Conference on Simulation of Adaptive Behavior*, (pp. 15-21); Meyer, J-A. & Wilson, S. (eds), Cambridge, MA: The MIT Press/Bradford Books (1991).

[22] IEEE 93 Standard for Information Technology, Application Protocol for Distributed Interactive Simulation. and IEEE 1278.1-1995 -Standard for Distributed Interactive Simulation - Application protocols

[23] US Defense Modelling and Simulation Office (DMSO) - High Level Architecture (HLA), https://www.dmso.mil/public/transition/hla/