Large scale agent-based modelling: a review and guidelines for model scaling.

H. R. Parry[1] and M. Bithell[2]

[1] CSIRO Entomology, Canberra, Australia
[2] Department of Geography, University of Cambridge, UK

Introduction

In agent-based simulation, the term ‘large scale’ refers not just to a simulation that contains many agents, but also refers to the problem of managing the complexity of the simulation (Parry 2009). Agent-based models are inherently complex, thus models that push the limits of resources due to their large numbers of agents or their complexity may be referred to as ‘large scale’. Another term also used for such simulations is ‘Massively Multi-agent Systems (MMAS)’ or ‘Massive Agent-based Systems (MABS)’ (Ishida *et al*., 2005; Jamali *et al*., 2008), the term ‘Massive’ being used in the general computing sense where it implies extremely large numbers (i.e. millions) of agents.

Resource limitations in agent-based simulation may be encountered as the modeller adds more agents to investigate whole system behaviour, as the modeller adds complexity to each agent in the form of rules and parameters, or the modeller may wish to examine the response of an agent in a more realistic and complex environment. Haefner (1992 pp. 156-157) had the foresight nearly 20 years ago to identify aspects of ecological individual-based models that would benefit from advanced computing: multi-species models; models of large numbers of individuals within a population; models with greater realism in the behavioural and physiological mechanisms of movement; and models of individuals with `additional individual states' (e.g. genetic variation). The introduction of a spatial dimension also adds complexity and puts demands on computing resources, yet many agent models are spatial; therefore, this chapter focuses on spatial agent-based models.

Review of large-scale modelling techniques

There have been a number of methodologies arising with which to deal with the problem of ‘large scale’ simulations in the agent-based literature, in a number of disciplines, ranging from molecular physics, social science, telecommunications and ecology, to military research. Some of these methods are given in Table 1. This chapter focuses on the two most common types of solution found in the literature: (1) Model software restructuring; (2) Computer hardware and software programming solutions, including the use of vector computers, Graphics Processing Units (GPUs) and parallel computing.

Solution	**Pro**	**Con**
Reduce the number of agents, or level of agent complexity, in order for model to run on existing hardware.	No reprogramming of model.	Assumes dynamics of a smaller or less complex system are identical to larger systems.
Revert to a population based modelling approach.	Could potentially handle any number of individuals.	Lose insights from agent approach. Effects of diversity in agent population lost. Emergent properties from simulation of non-linear interactions at agent level difficult to capture. Construction of entirely new model (not agent-based).
Invest in a larger or faster serial machine.	No reprogramming of model.	High cost. CPU speeds limited to gains of only a few percent. Most gain likely for large memory problems, but again maximum machine memory is limited. Multi-threading or parallelism would increase the utility of this approach.
Run the model on a vector computer.	Potentially more efficient as more calculations may be performed in a given time	High cost. Vector hardware not easy to obtain (although GPU may compensate this somewhat – see below). This approach works more efficiently with SIMD, possibly not so suitable for agent-based models with heterogeneous model processes.
Super-individuals (model software restructuring).	Relatively simple solution, keeping model formulation similar.	Restructuring of model. Aggregation can change dynamics. Potentially inappropriate in a spatial context (Parry and Evans 2008).
Invest in a large scale computer network and reprogram the model in parallel.	Makes available high levels of memory and processing power.	High cost (although lowering with advent of multi-core and GPU computing). Advanced computing skills required for reprogramming of model software. Algorithms need to be modified to cope with out-of-order execution on different cores. Communication efficiency between cores becomes important. Solutions required are problem dependent.

Table 1: Potential solutions to implement when faced with a 'large scale' agent-based model. Adapted from (Parry 2009)

Model software restructuring: 'super-individuals'

A relatively simple option is to implement an aggregation of the individual agents into `super-agents', such as the `super-individual' approach in ecological modelling (Scheffer *et al.*, 1995). Other terms coined for this approach in ecology are the 'Lagrangian Ensemble' method (Woods and Barkmann, 1994; Woods, 2005) and 'generalised individuals' (Metz and de Roos, 1992). A similar approach has been termed 'agent compression' in social science (Wendel and Dibble, 2007) which is derived from an earlier ecological paper (Stage *et al.*, 1993). In many ways these approaches are analogous to the concept of 'cohorts', which has been used for a long time in entomological modelling (e.g. Barlow and Dixon, 1980; Ramachandramurthi *et al.*, 1997). There are a number of examples of the super-individual method in relation to agent-based models in a wide range of literature, with examples in ecology (Schuler 2005; Parry and Evans, 2008) and social science (epidemiology) (Dibble *et al.*, 2007, Rao *et al.*, 2009). The basic concept of this approach is shown in Figure 1.

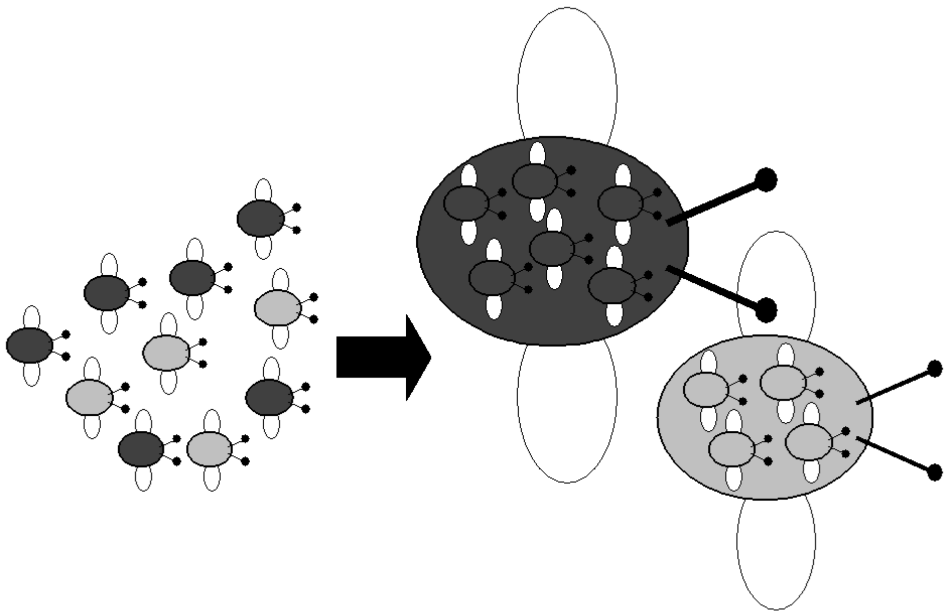


Figure 1: 'Super-agents': grouping of individuals into single objects that represent the collective (from Parry and Evans 2008).

The challenge to using a super-individual approach is relating super-individuals to individuals in time and space (Parry and Evans, 2008). Some solutions to managing super-individuals spatially have been proposed, e.g. to maintain a constant number of super-individuals within a spatial unit, so that individuals migrate from one super-individual in one cell to become part of a super-individual in another cell. However, these solutions still affect model behaviour and it comes down to a 'trade-off between error and computing costs' (Hellweger, 2008 pp 148). This approach is still likely to have some limitations when behaviour at low densities is important and there is a strong spatial effect on the individuals.

Recent work has proposed a dynamic approach to the creation of super-individuals (Wendel and Dibble, 2007). Computing techniques using compression algorithms are applied to homogenous super-individuals to selectively compress their attributes. The algorithm can maintain the integrity of the original data; however, it might be an advantage for the algorithm to combine similar pieces of information, to produce a more compact compression. The result is super-individuals that contain varying numbers of similar or identical individuals, from just a single individual to many, depending on the uniqueness of the individuals. The attributes of the individuals contained within the super-individual are monitored over time, so that if individuals differentiate themselves from the group (e.g. they change spatial location, perhaps to another grid cell) they are extracted from the super-individual and become individuals. If the attributes of the uncontained agent now matches another super-individual they may join that super-individual (e.g. they are added to a super-individual at their new spatial location). Although there is some computing overhead for this 'dynamic agent compression', it has been show that it may give some efficiency gain over an individual-based model whilst promising to preserve heterogeneity as necessary (Wendel and Dibble, 2007). In general, the fewer unique agents in the simulation the more effective this approach will be.

Model software restructuring example: spatial super-individuals

This example uses a spatially-explicit individual-based aphid model detailed in (Parry, 2006; Parry *et al*., 2006b); see also the section later in this chapter: 'Example of the use of an Agent-parallel approach'. Turning the individuals in this simulation into 'super-individuals' involved only a small alteration of the model's structure (for details see Parry and Evans, 2008). A variable was added to record the number of individuals all super-individuals actually represent. Equations that were dependent on density (such as morphology determination) were altered so that the density values were related to the real number of individuals in the simulation, not the number of super-individuals.

Movement of super-individuals followed the same rules as that of individuals; however this produced spatial clustering of the populations. The model was tested by Parry and Evans (2008), using varying populations of individuals (100, 1,000, 10,000 and 100,000 and 500,000 individuals) represented by varying numbers of super-individuals. A brief summary of the findings in this paper follow.

The super-individual model runs on a cellular landscape of 50×50 25m cells, with the initial population of apterous adult aphids initiated at the central cell.

Temporal Results

The temporal comparison of super-individuals (representing 10,000 individuals) given in Parry and Evans (2008) is shown in Figure 2. The results for 1,000 super-individuals (scale factor 10 individuals per super-individual) are the only results that fall within the 95% confidence limits of the original model for the duration of the simulation period. This is due to excessive discretization of mortality in the model for the super-individuals. Therefore super-individuals of large numbers of individuals as shown here with low scale factors may be the only acceptable way to use this approach, in this case.

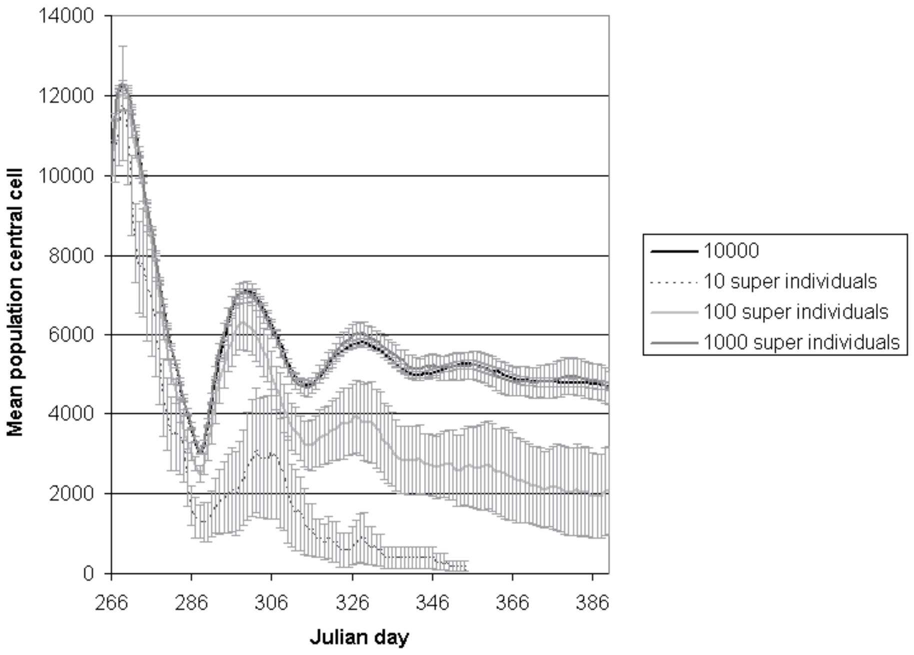
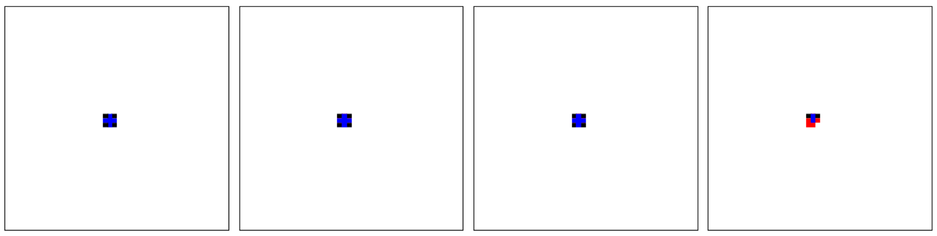


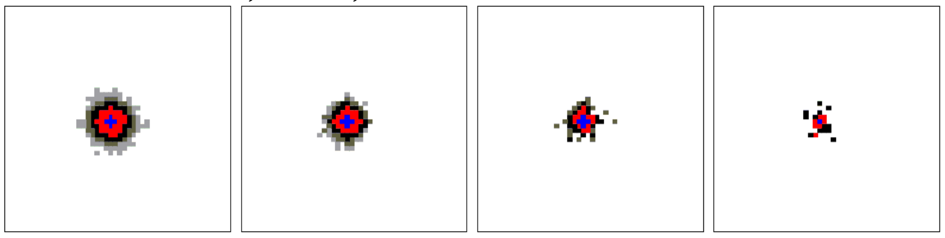
Figure 2: 10,000 individuals: comparison between individual-based simulation, 1,000 super-individual simulation (each represents 10 individuals), 100 super-individual simulation (each represents 100 individuals) and 10 super-individual simulation (each represents 1,000 individuals), showing 95% confidence limits derived from the standard error.

Spatial Results

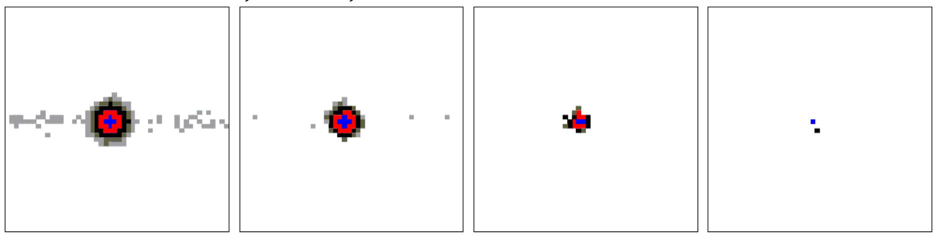
The spatial results given in Parry and Evans (2008) are summarised in Figure 3. Clustering is evident in the spatial distribution. The super-individuals are contained in fewer cells, closer to the origin, than the individual-based simulation for all instances of super-individuals, even those with a low scale factor. Thus, it is an important consideration for spatially-explicit models to test super-individual scaling approaches spatially as well as temporally, as temporal testing will not show the spatial errors that appear more sensitive.



(a) 10,000 individuals, density at 2 days: (l-r) Individual-based simulation, super-individual simulation scale factor 10, 100 and 1,000



(b) 10,000 individuals, density at 20 days: (l-r) Individual-based simulation, super-individual simulation scale factor 10, 100 and 1,000



(c) 10,000 individuals, density at 40 days: (l-r) Individual-based simulation, super-individual simulation scale factor 10, 100 and 1,000

2

Figure 3: Spatial density distributions for individual-based versus super-individual simulations (10,000 aphids) at (a) 2 days (b) 20 days and (c) 40 days. The distribution further from the central cell is influenced by the constant westerly wind direction to result in a linear movement pattern.

Computer hardware and software programming: parallel computing

Multi-core architectures

‘Parallel computing’ encompasses a wide range of computer architectures, where the common factor is that the system consists of a number of interconnected ‘cores’ (processing units), that may perform simultaneous calculations on different data (Wilkinson and Allen, 2004). These calculations may be the same or different, depending whether a ‘Single Instruction Multiple Data’ (SIMD) or ‘Multiple Instruction Multiple data’ (MIMD) approach is implemented (see glossary).

We are not concerned here with issues of parameter-space exploration or monte-carlo simulations, in which many runs of a small serial (i.e. single-CPU) code may be required. In such cases efficient use of computer clusters can be made by running identical copies of the code on many separate cores using solutions such as CONDOR (http://www.cs.wisc.edu/condor). While these are in a sense “large-scale” and make good use of multi-core or distributed computer resources on heterogeneous hardware, here we discuss the use of parallel computing to address the issue of models that require significant resources even for a single model run.

Reprogramming the model in parallel is challenging. Despite this, over the last ten years or so it has become a popular solution for agent-based modellers in many different fields of research. These range from ecology (Lorek and Sonnenschein, 1995; Abbott *et al*., 1997; Wang *et al*., 2004; Immanuel *et al*., 2005; Wang *et al*., 2005; Wang *et al*., 2006a; Wang *et al*., 2006b; Parry *et al*., 2006a) and biology (Castiglione *et al*., 1997; Da-Jun *et al*., 2004) to social and economic science (Massaioli *et al*., 2005; Takeuchi, 2005) and computer science (Popov *et al*., 2003), including artificial intelligence and robotics (Bokma *et al*., 1994; Bouzid *et al*., 2001). In the early 1990s, work in the field of molecular-dynamics (MD) simulations proved parallel platforms to be highly successful in enabling large-scale MD simulation of up to 131 million particles – equivalent to very simple 'agents' (Lomdahl *et al*., 1993). Today the same code has been tested and used to simulate up to 320 billion atoms on the BlueGene/L architecture containing 131,072 IBM PowerPC440 processors (Kadau *et al*., 2006). Agent-based simulations in ecology and social science tend to comprise more complex agents. Therefore, distributed execution resources and timelines must be managed, full encapsulation of agents must be enforced, and tight control over message-based multi-agent interactions is necessary (Gasser *et al*., 2005). Agent models can vary in complexity, but most tend to be complex especially in the key model elements of spatial structure and agent heterogeneity.

Graphics Processing Units (GPUs)

A recent advance in desktop computing through the introduction of Graphics Processing Units (GPU) has now made it even easier for modellers to take advantage of data-parallel computer architectures (Lysenko and D'Souza, 2008). The need for high levels of inter-agent communication and agent movement can make it difficult for cluster-based parallel computing to be efficient, an issue that may be addressed by tighter communication within a GPU.

Essentially GPUs are similar to Vector computers (see glossary). The structure of agent simulations (often with asynchronous updating and heterogeneous data types) could mean that running a simulation on a vector computer may make little difference to the simulation performance. This is because an agent model typically has few elements that could take advantage of SIMD: rarely the same value will be added (or subtracted) to a large number of data points (Nichols *et al*., 2008). In particular, vector processors are less successful when a program does not have a regular structure, and they don't scale to arbitrarily large problems (the upper limit on the speed of a vector program will be some multiple of the speed of the CPU (Pacheco, 1997)). GPUs offer some advantage over vector processors – their operation is single process multiple data (SPMD) rather than SIMD, so that all processing units need not be executing that same instruction as in a SIMD system (Kirk and Hwu, 2010). Although it is difficult to keep the advantages of object-oriented code in a GPU environment, there can be considerable benefits in terms of speed.

Lysenko and D'Souza (2008) manage to reformulate an agent-based model to operate on a GPU through stream processing (see glossary) by the use of large, multi-dimensional arrays to contain the complete state of an agent. Kernels are then programmed to manage the computer's resources (i.e. the GPUs) to run update functions on the arrays. A different kernel is created for each update function, which operate one at a time on the dataset. Problems are also encountered when handling mobile agents, but they can be overcome (see below). Their GPGPU (General Purpose GPU) approach required explicit use of the graphics card's texture maps and pixel colour values. Since that time, further developments have made it more straightforward to use GPUs for general computation with the advent of better hardware and libraries designed for the purpose such as NVIDIA's CUDA (http://developer.nvidia.com/object/cuda.html). These libraries relieve the programmer of some of the previous awkwardness involved in converting code for use on GPU, although awareness of the hardware layout is still required in order to get good performance. Other similar libraries such as Apple's openCL (Khronos, 2010), Intel Ct and Microsoft Direct Compute also exist but as of the time of writing seem to be in a less advanced state of development. These latter libraries also seek to incorporate some level of hardware independence and are therefore likely to be somewhat more involved to code with (Kirk and Hwu, 2010). Object-oriented Molecular Dynamics (MD) code already exists that can exploit the CUDA library (Stone *et al*., 2007), so that the prospect for making individual-based or agent-based code that exploits these libraries in the future would seem to be good. Typically for MD codes a 240 core GPU seems to be able to deliver similar performance to a 32 core CPU cluster (see for example http://codeblue.umich.edu/hoomd-blue/benchmarks.html)

Challenges of parallel computing

Several key challenges arise when implementing an agent model in parallel, which may affect the increase in performance achieved. These include load balancing between cores, synchronising events to ensure causality, monitoring of the distributed simulation state, managing communication between nodes and dynamic resource allocation (Timm and Pawlaszczyk, 2005). Good load balancing and inter-node communication with event synchronisation are central to the development of an efficient parallel simulation, a full discussion of which is in Parry (2009). Notable examples of load balancing strategies can be found in Pacheco (1997), including `block mapping' and `cyclic mapping' (see glossary).

A further major hurdle is that many (perhaps most) agent-based models are constructed with the aid of agent toolkits such as Repast or NetLogo. These toolkits may not be able to handle this conversion to another program representation (particularly an issue for GPU). Recently Minson and Theodoropoulos (2008) have used a High Level Architecture (HLA) to distribute the RePast Toolkit for a small number of highly computationally intensive agents over up to 32 cores with significant improvements in performance. Rao *et al*. (2009) express reservations about the general availability of such HLAs however. In the examples that follow we show an instance of RePast parallelised using a library (MPIJava[3]) that adds external Message Passing Interface (MPI)[4] calls to java, but use of this library required extensive restructuring of the original model code as it was designed for serial execution.

[3] Message Passing Interface for Java (MPIJava) http://www.hpjava.org/mpiJava.html is no longer available for download online. It is super-ceded by MPJ-Express http://mpj-express.org/
[4] See glossary for definition of MPI

The 'Agent-parallel' approach
This approach divides the agents between cores. The parallelisation focuses on the agents and divides them between the cores which keep track of the individual agents' properties and spatial location. Thus, each core must keep up-to-date information on the complete environment and surrounding agents. Communication with other cores is necessary to update on the actual agent densities for a given location as a result of movement, birth and death. This form of parallelisation is similar to 'functional decomposition' (Foster 1995), which divides various model processes or calculations, though not necessarily agents, between cores.

Examples from ecology:

- Aphids and Hoverflies (Parry and Evans, 2008), the example used in this chapter.
- Schools of Fish (Lorek and Sonnenschein, 1995) – includes an extension where fish are dynamically redistributed according to their neighbourhood to improve efficiency.
- Trees (one processor per tree) (Host *et al.*, 2008)
- Landscape vegetation model (functional decomposition) (Cornwell *et al.*, 2001)
- Daphnia, distributing individuals between processors as cohorts or ecotypes, similar to super-individuals (Ramachandramurthi *et al.*, 1997; Nichols *et al.*, 2008)

Examples from social science:

- Financial markets (Massaioli *et al.*, 2005)
- Crowd simulation (Lozano *et al.*, 2007)

The 'Environment-parallel' approach
This approach divides the geographical space between cores. The parallelisation focuses on a point in space (e.g. a grid cell), which is assigned to each core. The core then keeps track of all agent activity within that space. This has also been termed 'geometric' or 'domain' decomposition (Foster, 1995).

Examples from ecology:

- Parallel Individual-Based Modeling of Everglades Deer Ecology (Abbott *et al.*, 1997)
- Design and implementation of a Parallel Fish Model for South Florida (Wang *et al.*, 2004)
- Fire simulation (Wu *et al.*, 1996)
- Forest model (Chave,1999)

Examples from social science:

- Parallel implementation of the TRANSIMS micro-simulation (Nagel and Rickert, 2001)
- Abstract agent model 'StupidModel' (Lysenko and D'Souza, 2008)
- Traffic simulation (Dupuis and Chopard, 2001)
- Disaster Mitigation (Takeuchi, 2005)

Parallel computing examples: ‘agent-parallel’ and ‘environment-parallel’ approaches

Example of the use of an Agent-parallel approach

This example uses a spatial predator-prey (hoverfly-aphid) model to show how an agent-parallel model can be established. However, there were shortcomings of this approach for this particular model due to agent interaction, which is discussed.

The basic overall structure of the model system is similar to the structure used by Tenhumberg (2004) which refers to two interacting sub-models for syrphid larvae and aphids. However, in the individual-based model presented here, the movement of adult female syrphids across the landscape is also modelled. The model was constructed with the Repast 2.0 agent-based software development toolkit for Java.

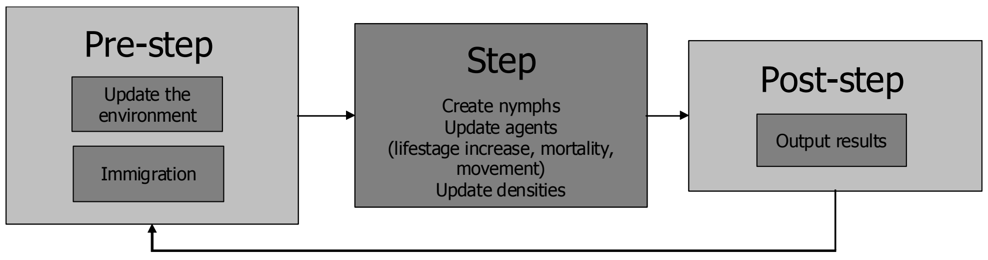
This example uses the aphid model as used to illustrate the concepts of super-individuals earlier in the chapter. The model describes the population lifecycle of an aphid, *Rhopalosiphum padi*. This includes spatial as well as temporal population dynamics within a field. Full details of the aphid sub-model can be found elsewhere (Parry, 2006; Parry *et al.*, 2006b), with a highly simplified model flow diagram shown in Figure 4.

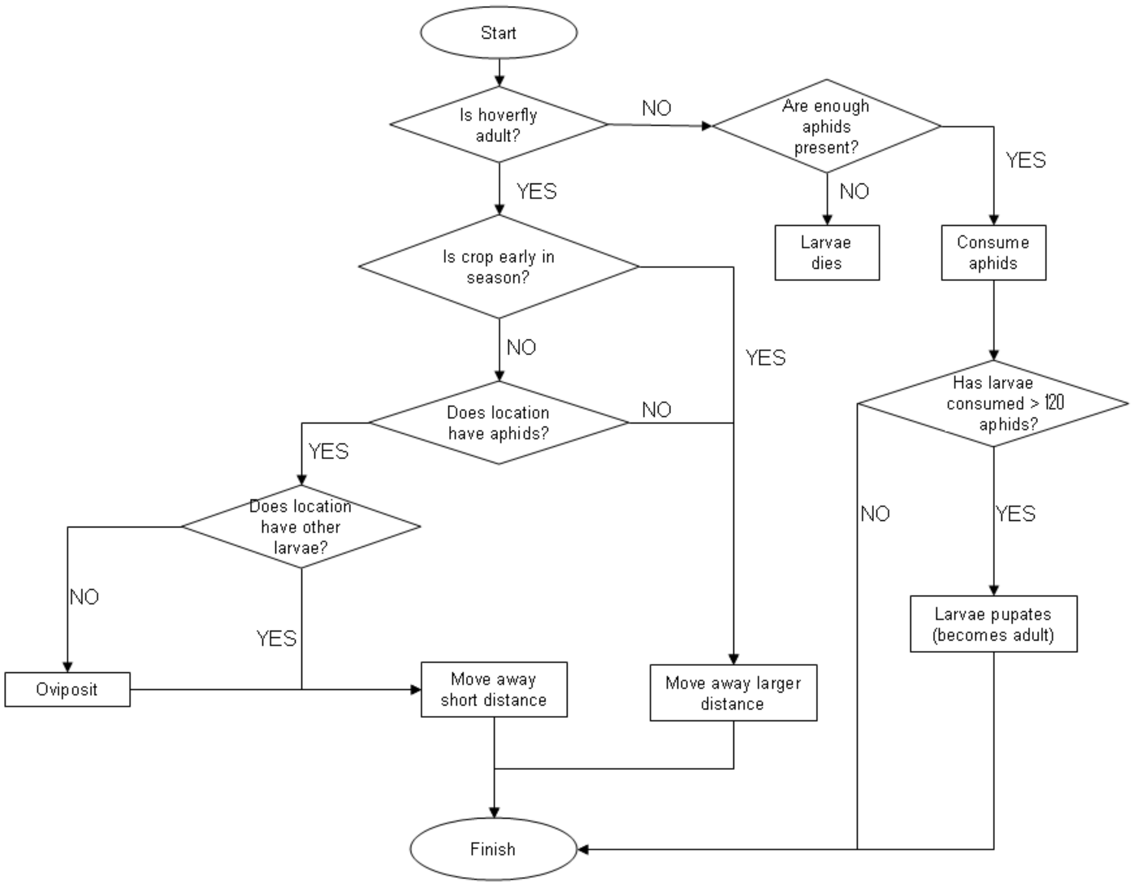
The basic rules followed in the syrphid model are given in

Figure 4: Simplified flow chart for aphid model

Figure 5, with more detail on the rules used in the hoverfly model given in appendix 1, as this sub-model is unpublished elsewhere. The two sub-models (aphids and hoverflies) are connected to one another, by the consumption of aphids by hoverfly larvae. The relationship between the two models is simplified in Figure 6.

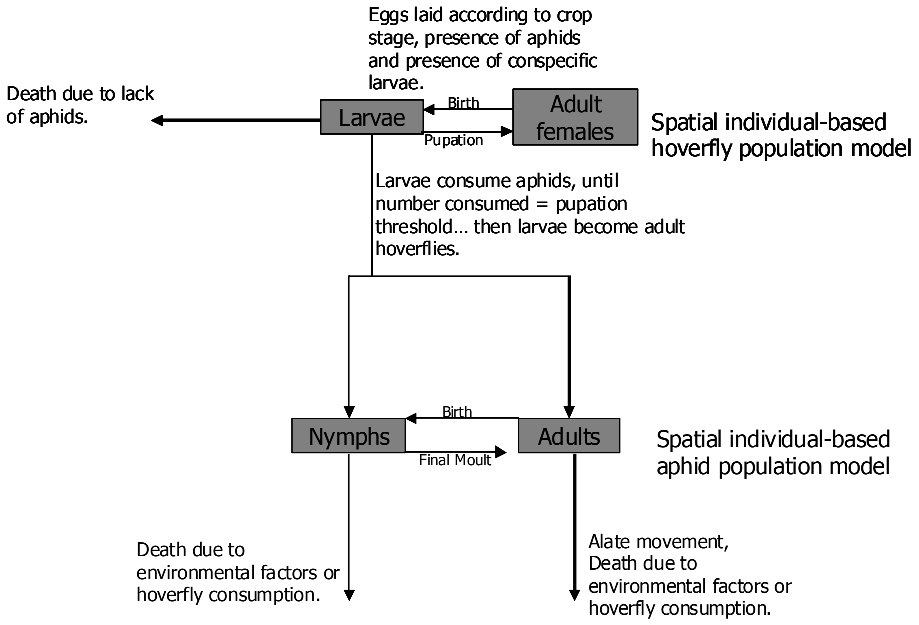
The hoverfly-aphid model is initiated with one apterous adult aphid per cell (1 m^2) in the field cells only and one female adult hoverfly per cell in the field margin cells only. The simple landscape is as shown later in this chapter, two rectangular fields split by a central margin (see Figure 11).





Start
Is hoverfly adult?
NO
YES
Are enough aphids present?
NO
YES
Larvae dies
Consume aphids
Is crop early in season?
NO
YES
Has larvae consumed > 120 aphids?
NO
YES
Does location have aphids?
NO
YES
Does location have other larvae?
NO
YES
Larvae pupates (becomes adult)
Oviposit
Move away short distance
Move away larger distance
Finish

Figure 4: Simplified flow chart for aphid model
Figure 5: Flowchart of syrphid model.



Eggs laid according to crop stage, presence of aphids and presence of conspecific larvae.
Death due to lack of aphids.
Larvae
Birth
Pupation
Adult females
Spatial individual-based hoverfly population model
Larvae consume aphids, until number consumed = pupation threshold… then larvae become adult hoverflies.
Nymphs
Birth
Final Moult
Adults
Spatial individual-based aphid population model
Death due to environmental factors or hoverfly consumption.
Alate movement, Death due to environmental factors or hoverfly consumption.

Figure 6: Hoverfly-aphid model, key processes

In order to parallelise the model to distribute the agents to different cores in a cluster, a Message Passing Interface (see glossary) for Java was used http://www.hpjava.org/mpiJava.html (no longer available for download, see footnote 3), run on a Beowulf cluster (see glossary). At each time step, agents are updated on the worker cores (See Figure 7), as the control core maintains global insect density and aphid consumption information and controls the simulation flow.

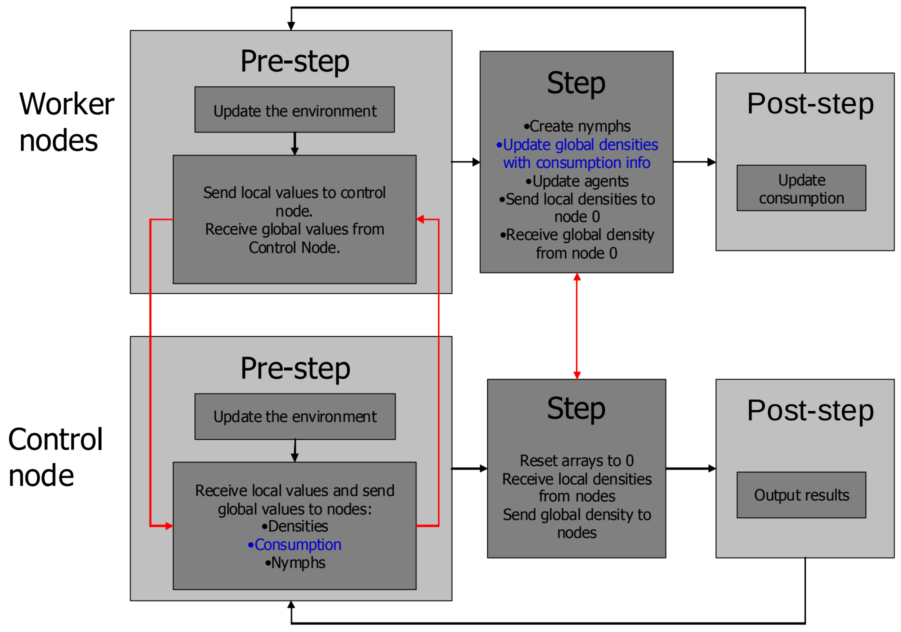

Worker nodes
Pre-step
Update the environment
Send local values to control node.
Receive global values from Control Node.
Step
•Create nymphs
•Update global densities with consumption info
•Update agents
•Send local densities to node 0
•Receive global density from node 0
Post-step
Update consumption
Control node
Pre-step
Update the environment
Receive local values and send global values to nodes:
•Densities
•Consumption
•Nymphs
Step
Reset arrays to 0
Receive local densities from nodes
Send global density to nodes
Post-step
Output results

Figure 7: Parallel model flow chart, blue text indicates interaction between the two sub-models; red arrows indicate interaction between the control core and the worker cores

Testing just the aphid model, simple tests of the parallel code versus the original model (without hoverfly larvae) showed the parallel model to replicate the original model accurately.

However, when larvae were introduced to the model, there were two major problems that meant the parallel implementation did not replicate the original, non-parallel model implementation. These arise from the added complexity of the simulation. The most complex element of the model to program was the interaction between the hoverflies and the aphids (i.e. aphid consumption). This involved additional message passing, as the hoverfly may consume aphids that reside on another processor (although in the same cell geographically). Therefore, consumption for each cell had to be totalled on the control core and then messages passed to each core to instruct the core to remove a given number of aphids in each cell. However, as these messages are only sent once per iteration, it was possible for more than one hoverfly larvae to consume the same aphid (as the hoverfly larvae would only have information from the previous model iteration on the total aphid densities within the cell, and would be unaware if an aphid had been consumed by another hoverfly larvae on another core).

The result is that occasionally the total calculated consumption of aphids per iteration per cell is greater than the total density of aphids per cell in that iteration. A simple fix was added to recalculate the total consumption, so that when the total aphid consumption is greater than the total aphid density, the consumption is reduced to the total aphid density. However, the problem still remains, and it explains lower aphid populations in the parallel model than in the non-parallel model, as shown by Figure 8.

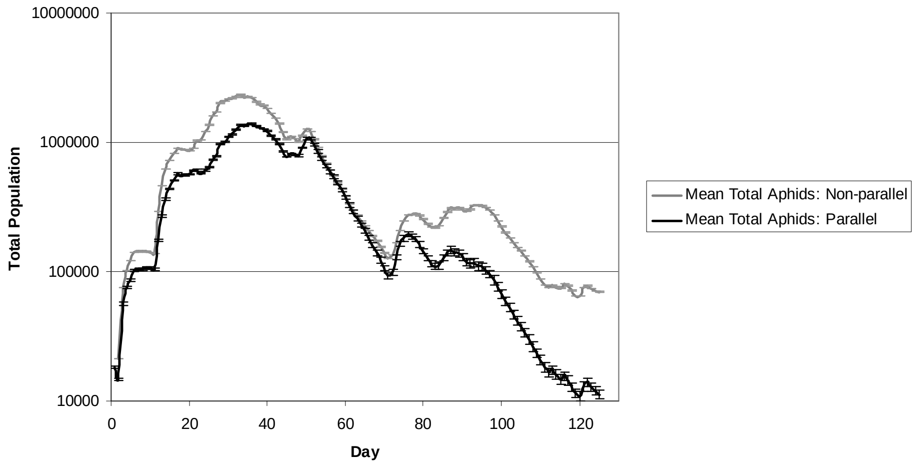


Figure 8: Comparison of the temporal dynamics of the total population of aphids between parallel and non-parallel simulation implementations (error bars show standard error).

In addition, more hoverflies are born into a cell than should be. During the same iteration different female hoverflies on different processors may perceive a cell to have no larvae present, and then both lay in that cell. However, the model rules that once larvae are present in a cell no more larvae should be laid there. The result is likely to be higher numbers of larvae throughout the simulation, which is shown by Figure 9. This also acts to reduce the aphid population below that of the non-parallel simulation.

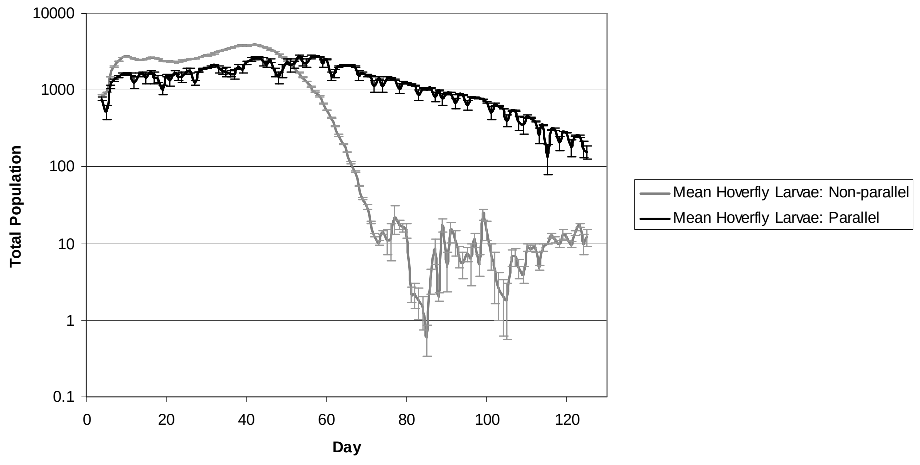


Figure 9: Comparison of the temporal dynamics of the total population of hoverfly larvae between parallel and non-parallel simulation implementations

The knock-on effect is that although higher populations of larvae are present in the non-parallel model, due to the artificial reduction in the aphid population and artificial increase in the larvae population, these larvae are less likely to reach adulthood as there are not enough aphids to consume so that they undergo the transition to adulthood in the model before dying (a combination of higher competition due to the higher larvae density and lower aphid populations due to the higher consumption rate),

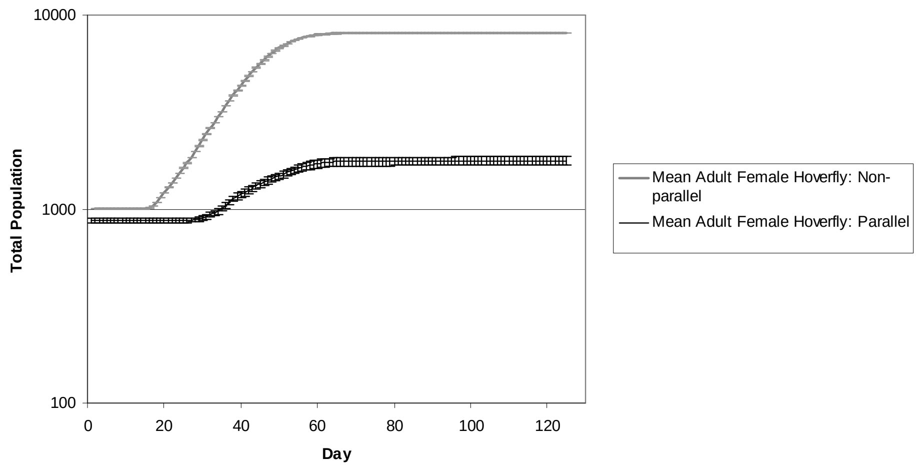


Figure **10**.

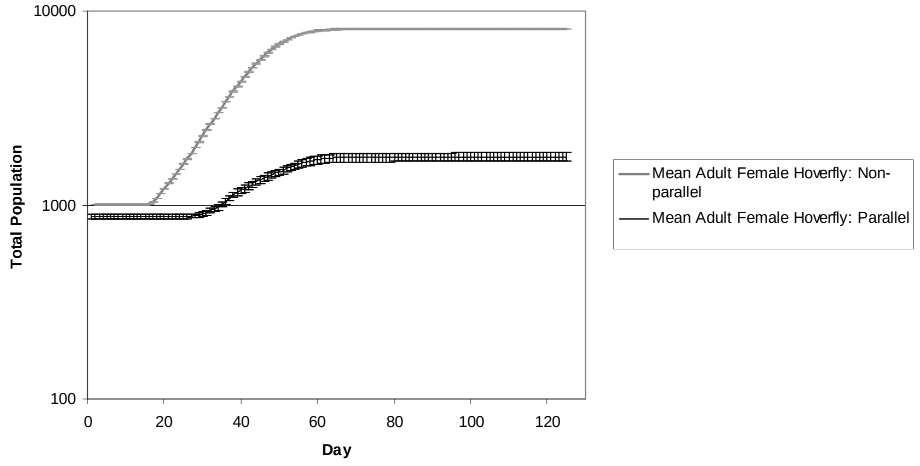


Figure 10: Comparison of the temporal dynamics of the total population of adult female hoverfly between parallel and non-parallel simulation implementations (no mortality)

These problems are not experienced in the non-parallel model, as it is straightforward to re-set the number of hoverfly larvae present within a cell during a time-step so that further hoverfly larvae are not introduced mid-iteration, and the consumption of aphids does not conflict as information on the number of aphids present can also be updated easily mid-iteration.

Therefore, the observed differences in the hoverfly population dynamics between the non-parallel and parallel simulation is attributable to these programming issues, which must be resolved before the parallel model can be used further in scenario development. However, the comparisons provide a valuable insight into the difficulties that may arise when simulating increasingly complex agent-based models in parallel. One possible solution may be the use of 'ghost' agents, as done by Nichols *et al.* (2008), however until tested with this particular model it is uncertain if this would fully resolve the issues. More generally, this indicates that as the complexity of an agent based model increases, it may be more efficient to distribute the model environment (as described in the next section on an 'Environment-parallel' approach), rather than the agents, so that local agents may interact directly and update parameters within a single model iteration.

Example of the use of an Environment-parallel approach

The environment-parallel approach is essentially a form of domain-decomposition in which spatial units are passed out for processing by remote cores, rather than individual agents. Two challenges are: firstly, to efficiently distribute the environment across cores so as to keep the processor load as even as possible and secondly, how to handle the interaction between, and movement of, the agents.

For the hoverfly-aphid model described here, handling interactions is relatively simple – the landscape (see Figure 11) is divided into a regular cellular grid, which is used to organise the search process by which hoverflies discover their prey. Note that this particle-in-cell approach need not constrain the actual spatial locations of agents, which may still take on values to a much higher level of precision than cell locations (c.f. Bithell and Macmillan (2007)) – the cells simply act as agent containers. Since the hoverfly larvae are relatively immobile their search process is approximated as involving only the cell that they currently occupy (as opposed to having to search nearby cells – this introduces further complication as noted below). Cells can then be handed off to remote cores, for processing of all parts of the model that do not involve movement beyond cell boundaries (egg-laying by hoverfly adults, predation by larvae, progression of larvae to adult hoverfly, production of young by aphids, calculation of movement by either type of insect) during the first part of the model timestep. Since all cells are independent at this point, this results in a high degree of efficiency in the use of the distributed cores (provided that the cell distribution gives equal numbers of insects per core) whilst also resolving the issues arising in the agent-parallel methodology described above.

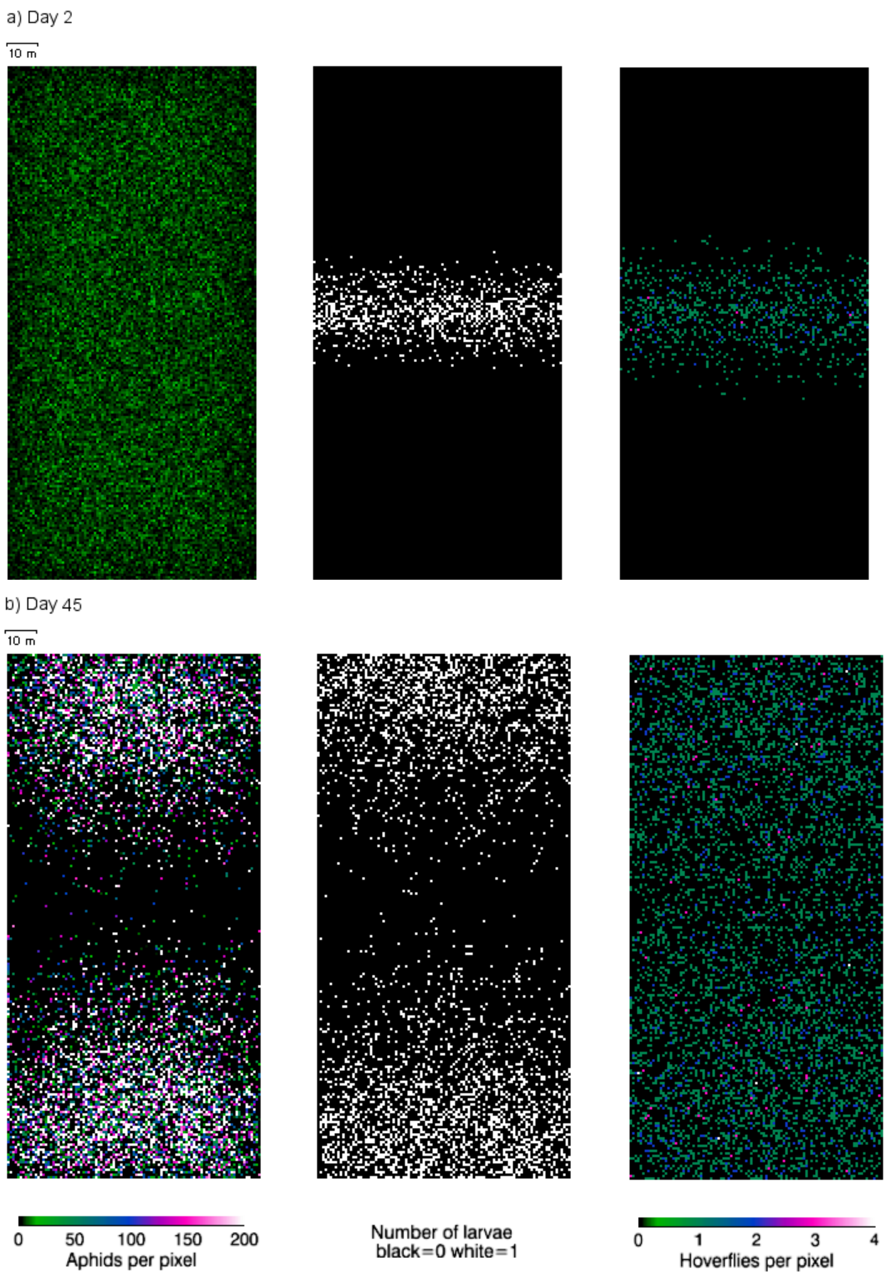
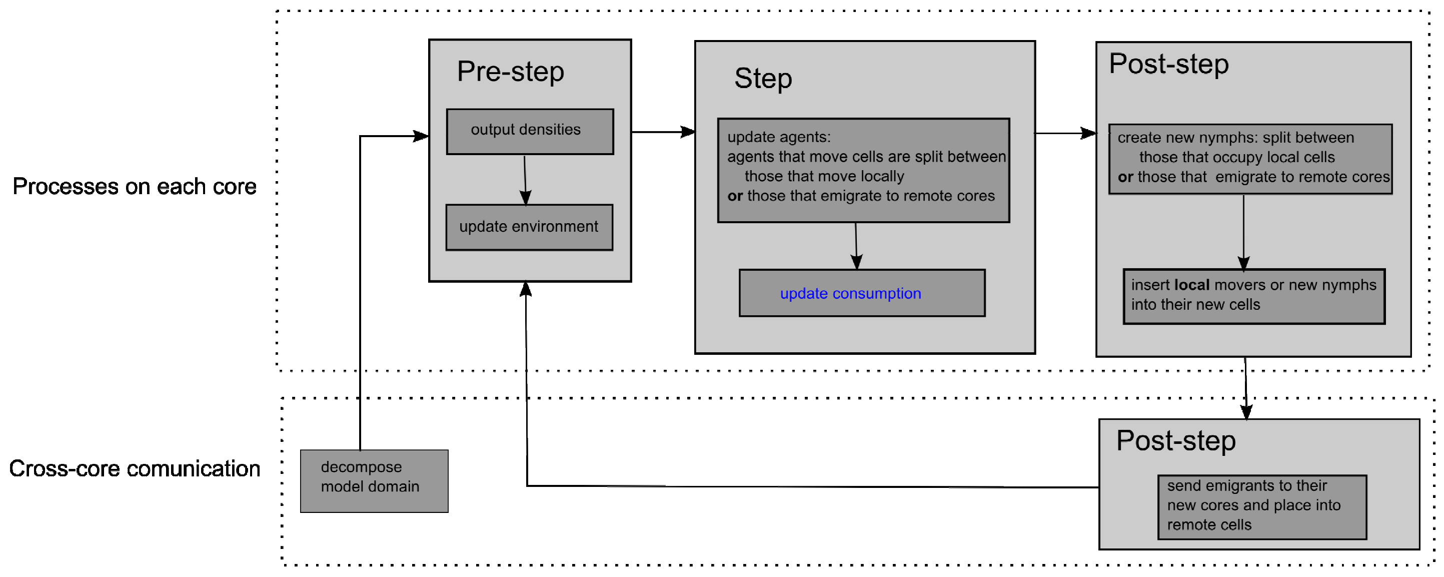


Figure 11: Snapshots of spatial distributions of aphids, hoverfly larvae and hoverfly adults showing spatial distribution over a 100m x 200m domain.

For the current simulation cells are 1m^2 – this means that typical movement per timestep (one day) exceeds the cell size (see Appendix) – insect movement may therefore necessitate transfer of agents from their current core to a remote core upon which their new cell is located. At the end of the above predation timestep, therefore, all the cells are synchronized across cores (to ensure that the same stage of calculation has been reached) and then a communication step is performed to move agents to their correct new locations (see Figure 12). As this communication step is relatively expensive, it reduces the level of speedup achievable somewhat.



Processes on each core
Pre-step
output densities
update environment
Step
update agents:
agents that move cells are split between
those that move locally
or those that emigrate to remote cores
update consumption
Post-step
create new nymphs: split between
those that occupy local cells
or those that emigrate to remote cores
insert local movers or new nymphs into their new cells
Cross-core comunication
decompose model domain
Post-step
send emigrants to their new cores and place into remote cells

Figure 12: Schematic to show the sequencing of the environment-parallel model. Note that here there is no distinction between workers and control – all cores are treated equally and all run the same set of processes.

In order to implement the above scheme the model was re-cast into C++, so that advantage could be taken of an existing data-parallel formulation (the graphcode library - Standish and Madina 2008), in which the MPI-parallel part of the code is encapsulated in the formulation of the model grid, along with a utility program (named classdesc) that allows packing and unpacking of arbitrarily structured agents for transfer between cores, making it possible to define the agent dynamics independent of the details of the MPI libraries.

The model, when re-coded into C++, produces essentially identical results (barring very small variations introduced by the use of random number generators) to the original Java version. The parallel version of the code in this case shows negligible differences from the serial version. The re-coding of the model into C++ might be expected to have efficiency gains before any parallelisation of the model (as shown for a similar individual-based model of a plant-aphid-disease system by Barnes and Hopkins 2003). However, at least for the current implementation, using java openjdk 1.6.0 and gnu C++ 4.3.2, runtimes of the serial version of the code in the two languages proved to be comparable. The parallel versions of the two implementations are not compared as the Java simulation had significant errors introduced by the

Comment [par44g 1]: When I ran these models I found the Java was much more efficient than the C++?! Java memory use = 13.5 MB and time = 98.1 sec, versus C++ memory use = 233MB and time = 4mins 20s! what figures did you get Mike?

parallelisation, as discussed in the preceding sections. An analysis of the speed-up of the Java model, when simulating aphids only, is given in Parry and Evans (2008), which also draws comparisons with the speed of the super-individual model implementation.

The environment-parallel example presented so far has two simplifications that in practice side-step two of the more awkward issues that need to be addressed in creating parallel agent code – namely a) domain decomposition is performed only once at the start of the run, where in principle it should be a dynamic process that is adaptive depending on agent density, in order to ensure a balanced load and b) the interaction between agents takes place only within a single cell, thereby limiting the necessary processes to a single core. We discuss each of these in the following sections.

a) Balancing loads in the spatially decomposed case

When the density of agents does not vary significantly across the spatial domain (or the density is uniform but the internal computation within each agent is not spatially variable) then the decomposition of the domain can be achieved at the start of the run by allocating equal area blocks of cells to different processors (see e.g. Abbott *et al* 1997). However, where there are mobile agents the density of occupation of the domain need not be uniform either spatially or temporally. Figure 11 shows two snapshots from the run of the aphid-hoverfly model **-** one at day 2 and the other after 45 days. Note that initially the aphids are completely uniformly distributed, but hoverflies and larvae are concentrated near the middle of the domain. However, once significant predation has taken place, aphids are almost entirely excluded from the domain centre, with a similar distribution to the larvae, whereas the hoverfly adults are almost uniformly spread. Since the aphids constitute the bulk of the computational load, a simple block decomposition of the domain with cores being allocated horizontal strips of cells across the domain from top to bottom would lead to cores near the domain centre spending much of their time idle compared to those nearer the upper and lower boundaries.

Since the evolution of the density field is not necessarily known from the start of the run, a re-allocation of the cell-to-core mapping should be recomputed automatically as the run proceeds. In practice this is not always a simple thing to do efficiently. Standish and Modina (2008) use the parallel graph partitioning library PARMETIS (http://glaros/dtc/umn.edu/gkhome/metis/parmetis/overview). Other methodologies exist based on space filling curves, for example (e.g. Springel 2005) – see figure 13. The latter has the advantage of being straightforward to code directly, but unlike PARMETIS does not explicitly take into account communication overhead, and has the added disadvantage of requiring a domain that can be easily mapped by a self similar structure (e.g. in the example shown the grid has to have a number of cells in each dimension that is a power of 2), making irregular regions with complex boundaries more difficult to handle.

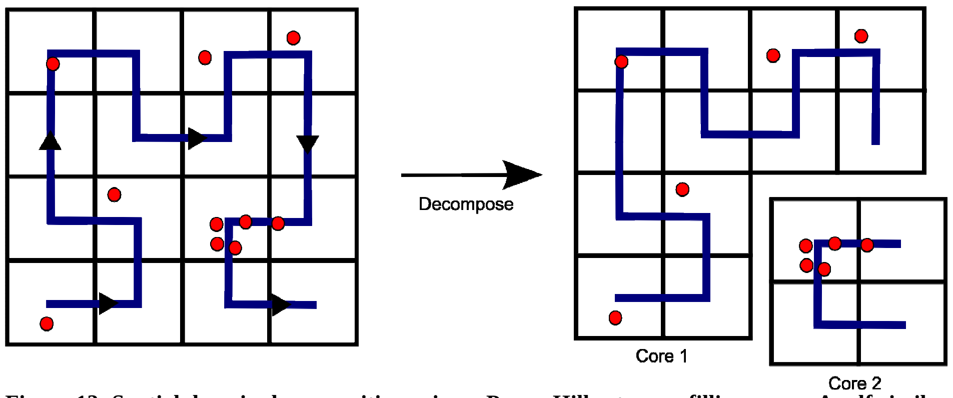


Figure 13: Spatial domain decomposition using a Peano-Hilbert space filling curve. A self-similar path is drawn connecting all the cells in the grid. The path is then traversed (as shown by the arrows), counting up the computational load, and the grid is then segmented along sections of the curve so that equal loads can be distributed to each core (here load is assumed proportional to the number of agents, shown as red dots).

In addition, any domain re-partitioning implies an overhead in re-building the allocation of cells to processor cores. How often this needs to be done and whether it is worth the time is problem dependent. For example, the C++ version of the example code on a 200m x100m domain runs 124 days on 32 cores in just 11 seconds. A much larger domain or a larger number of days would likely be required before load-balancing the code would provide a practical benefit.

b) Dealing with non-local agent interactions

As mentioned above, we can overcome the problem of predators on different cores accessing the same prey by using the environment-parallel approach when the predators do not look beyond their own local cell. However, once a region of interaction exists that extends across many cells, the problem of co-ordinating agent actions on different cores re-surfaces. Indeed the typical particle-in-cell code uses at least a four or eight cell interaction region about a central cell (see e.g. Bithell and Macmillan 2007). Once the spatial domain is split across cores, such interaction regions also get subdivided. Typically the first level required to deal with this problem is to maintain a 'halo' or 'ghost' region on each core, in which copies of the boundary cells that lie on a neighbouring core, together with *passive* copies of their contained agents, are kept on the local machine (figure 14).

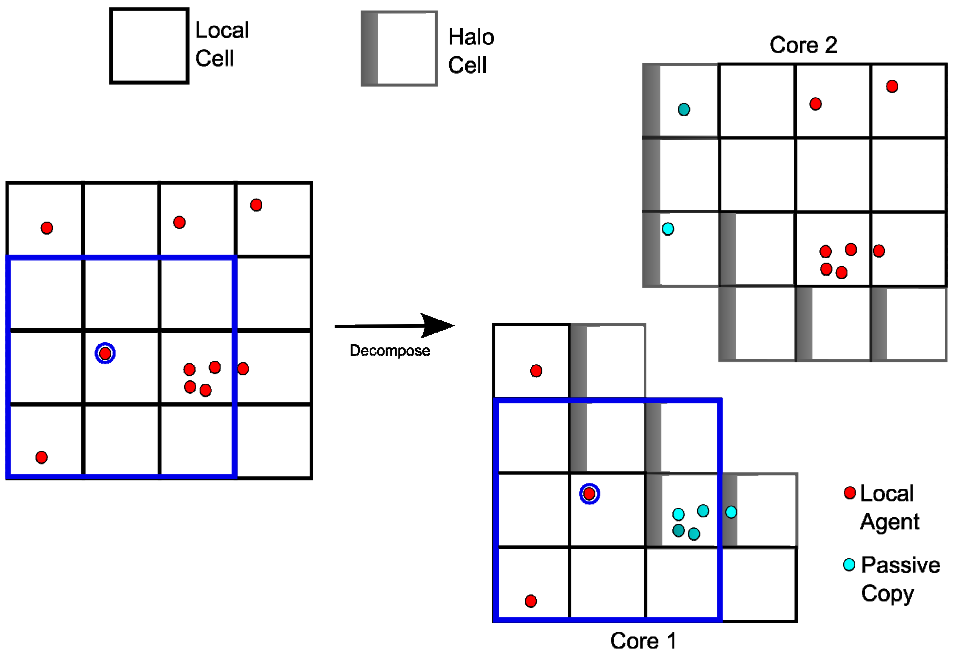


Figure 14: Domain decomposition where agents interact with others outside their own local cell. The circled agent interacts with those in its own cell, but also those in the eight-member neighbourhood outline by the blue square. On decomposition, part of this neighbourhood lies on a remote core. A halo region is therefore defined around the boundary of each decomposed part of the grid, into which passive copies of the appropriate remote cells can be placed. Locally active agents can then examine these copies in order to make decisions about interaction with the remotely stored agents. In this cased the circled agent can see one active agent on its own core, and 4 passive copies that are active on core 2. Agent copies in the halo cells are updated whenever their corresponding active counterparts on a remote core are changed.

This allows any independently computable symmetrical or uni-directional interactions to be accounted for immediately (examples would be molecular, smooth particle hydrodynamic or discrete element models, where forces encountered between interacting particles are equal and opposite, or are possibly determined by an external field, or disease models where contact with infectives leads to susceptibles acquiring disease, but the nature of the interaction is uni-directional, with no feedback to the infecting agent). Update of the passive agent copies can be performed at the end of each timestep as required. However, for typical ecological or social simulations this is unlikely to be sufficient. Figure 15 illustrates a typical case. Here agent A is a predator that can see only the prey (P) on its own core. Agent B can see a prey on its own core, but also the passive copy of the prey visible to agent A. Suppose both A and B choose to attack prey P: since the passive copy at first knows nothing of the attack of agent A, potentially A and B could independently attempt to consume the whole of P, leading to over-counting of the available prey. Any solution of this problem must additionally take account of the fact that the order of execution on different cores cannot be guaranteed.

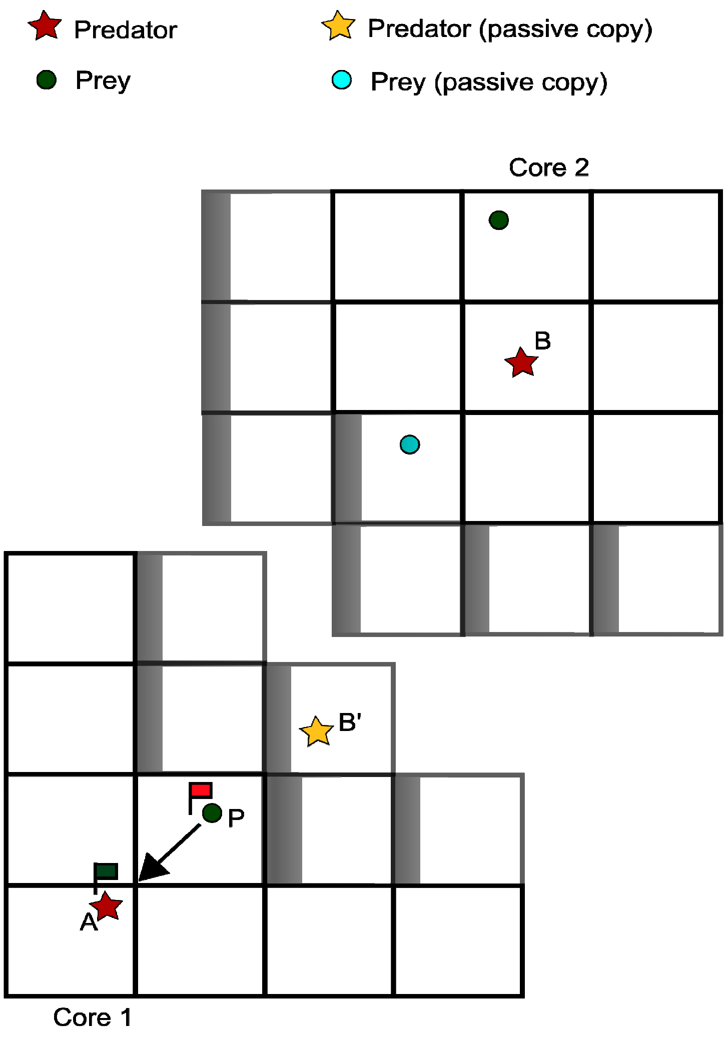


Figure 15: Predator-prey interaction taking place across cores. Prey P can see both predator A and the passive copy B'. Predator A, however, only knows about P, and not about B'. Active predator B on core 2 can see two prey, one of which is the passive copy of P. Predators and prey need to set and communicate flags to ensure consistency of action (see text). Once flags are consistent (flag on P labels it with A, flag on A labels it with P) then prey can be consumed as arrowed.

Lysenko and D'Souza (2008) encountered a similar problem in the allocation of single-occupancy spatial cells in their implementation of Stupid Model (Railsback *et al* 2005) – they overcame this using a two-pass method in which the agents initially attempted to place a flag in the cell they wish to occupy – a pre-allocated priority allowed agents to compute independently which would succeed, and on a second pass those agents with highest priority got to occupy the cells of their choice. However, in general it will not be known a priori which agent should have priority over others, requiring some form of conflict resolution to be performed: in the predator-prey case a competition between predators needs to ensue, and the outcome of this may not be known ahead of time. Mellott *et al.* (1999) discuss such a case in their implementation of deer predation by panthers (an extension of the earlier work by Abbott *et al.* (1997)). In essence a further layer of communication is needed in order to ensure consistency between the cores. Looking back at figure 15, we can envisage a three-pass algorithm in which the initial exchange is for each predator to mark itself with a flag indicating their interest in prey P. This flag is then copied across to the passive copy of the predator (in this case B') on the neighbouring core. Prey P then examines predators that are within range and runs a conflict resolution process (which may involve a more or less elaborate chase sequence involving A and B') to resolve the winner of A and B', setting a flag on itself with the identity of the winner. This flag can then also be copied across cores, and the predators can compare the flag on P with their own identity in order to find the outcome. Clearly this kind of algorithm may need to be extended in the case of more complex predator strategies (hunting as groups, for example) or more complex cognitive agents able to take account of a more extensive view of their surroundings and the available options for attack or escape. Again the result would seem to be that a general algorithm for dealing with this kind of parallel consistency issue is unlikely to be possible – the necessary solution is dictated by the problem at hand.

Potential efficiency gains

This section firstly compares the super-individual model with a parallel implementation of the aphid model only (described in Parry and Evans, 2008), which was parallelised using the agent-parallel approach described in this chapter. The aphid model parallelised well as agent-parallel because there was not the complexity of the hoverfly interactions. This shows how parallelisation and super-individuals can both help deal with increasing numbers of agents.

The second part of the section presents the efficiency gains in terms of memory and speed with increasing numbers of processors for the environment-parallel version of the aphid-hoverfly model, to illustrate how efficient this method has been in parallelising this more complex model. ~~.~~

Model speed and increasing numbers of agents

Super-individuals always improve the model speed with increasing numbers of agents (Figure 13). This improvement is linear (shown here on a log-log scale). The speed improvement is enormous for the largest simulations, where 500,000 individuals simulated with super-individuals using a scale factor of 100,000 increases the model speed by over 500 times the original speed. However, it was shown above that only large simulations with a low scale factor (10-100) may benefit from the super-individual approach, thus for these scale factors an improvement in model speed of approximately 10,000-30,000% (100-300 times) the original speed would result for simulations of 100,000 to 500,000 individuals.

Adding more processors does not necessarily increase the model speed. Figure 13 shows that for simulations run on two cores (one control core, one worker core) the simulation takes longer to run in parallel compared to the non-parallel model. Message passing time delay and the modified structure of the code are responsible. As the number of cores used increases, the speed improvement depends on the number of agents simulated. The largest improvement in comparison to the non-parallel model is when more than 500,000 agents are run across twenty-five cores, where model speed does scale linearly as the number of individuals increases. However,~~lthough~~ the parallel model is slower by comparison for lower numbers of individuals. ~~However, w~~When only five cores are used the relationship is more complex: for 100,000 agents five cores are faster than the non-parallel model, but for 500,000 the non-parallel model is faster. This is perhaps due to the balance between communication time increasing as the number of cores increases versus the decrease in time expected by increasing the number of cores. Overall, these results seem to suggest that when memory is sufficient on a single processor, it is unlikely to ever be efficient to parallelise the code.

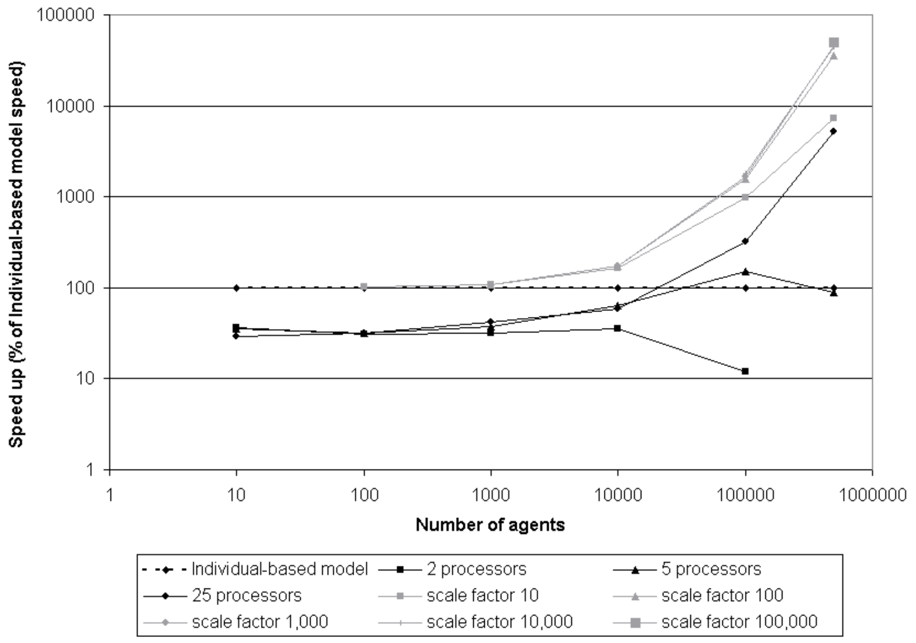


Figure 13: Plot of the percentage speed up from the individual-based (non-parallel) model against number of agents modelled: comparison between super-individuals of scale factor 10, 100, 1,000, 10,000, 100,000 and 500,000

Model memory use and increasing numbers of agents

The individual-based model has a linear increase in the memory used as agent numbers increase (shown here on a log-log scale, Figure 14).

Super-individuals always reduce the memory requirements of the simulation (Figure 14). The relationship between the number of (real) individuals in the simulation and the memory used is linear, for each scale factor (number of individuals represented by each super-individuals). The memory requirement for a simulation of super-individuals has a similar memory requirement to that of an individual-based simulation with the same number of agents as super-individuals. For simulations of 100,000 agents this can reduce the memory requirement to less than 10% of the memory required for the individual-based simulation with a scale factor of 10,000, and for simulations of 500,000 agents this may be reduced to around 1% with the same scale factor, so, when large scale factors are used, as agent numbers increase there is very little extra demand on memory.

The mean maximum memory usage by each worker core in the parallel simulations is significantly lower than the non-parallel model, for simulations using more than two cores (Figure 14). The relationship between the number of agents in the simulation and the memory used is linear for each number of processors. The two core simulation used more memory on the worker core than the non-parallel model when the simulation had 100,000 agents or above. This is probably due to the memory saved due to the separation of the GUI onto the control core being over-ridden by the slight additional memory requirements introduced by the density calculations. However, when 5 and 25 cores are used, the memory requirements on each core are very much reduced, below that of the super-individual approach in some cases. The super-individual approach uses the least memory for 500,000 individuals, apart from when only a scale factor of 10 is used (then the 25 core parallel simulation is more memory efficient).

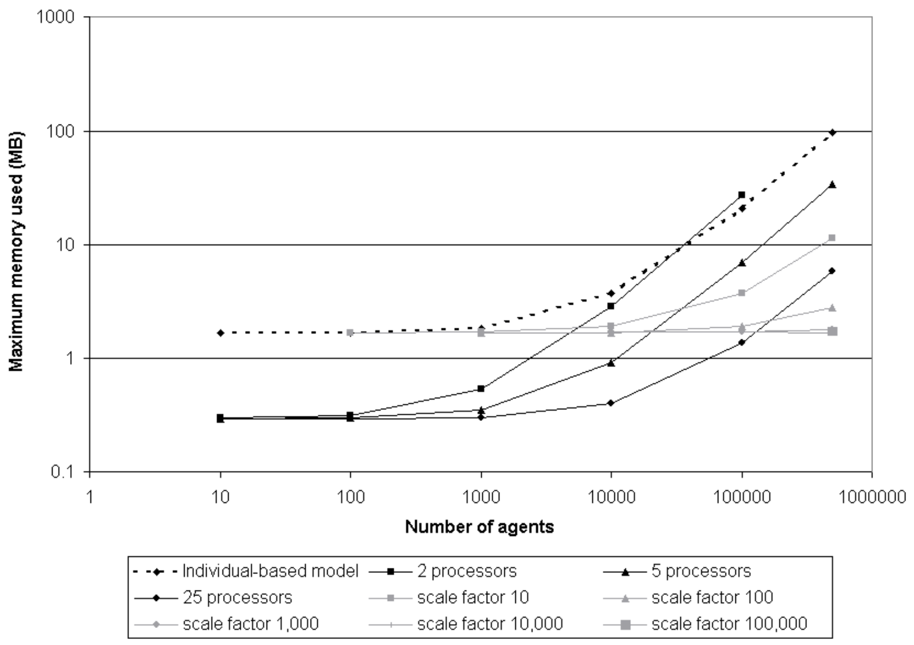


Figure 14: Plot of the mean maximum memory used in a simulation run against number of agents for the model, for different scale factors for super-individuals

Hoverfly-aphid model environment-parallel programming efficiency

The C++ programmed environment-parallel version of the hoverfly-aphid model was run on a dedicated cluster at CSIRO Black Mountain, Canberra. Each node in this network has 28x dual 3.2 GHz Xeon, with 2 or 4 Gbytes per node.

The speed-up of the model is linear in comparison to the non-parallel serial model code run on a single processor (Figure 15). In terms of memory, the model uses a little more memory per processor than the serial model when run on just two processors, however when run on more there is exponential improvement in the memory use efficency of the model (Figure 16).

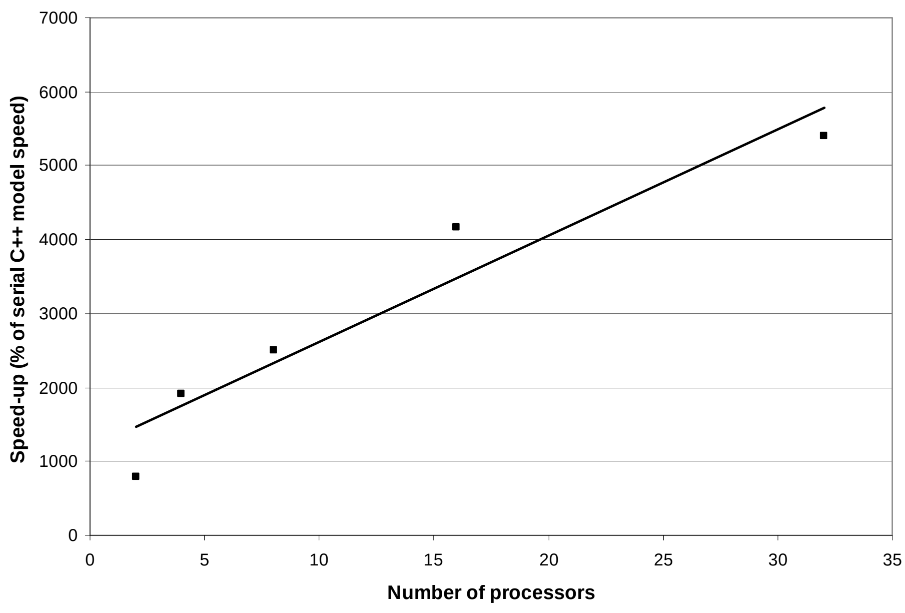


Figure 15: Environment-parallel hoverfly-aphid model: percentage speed up from the individual-based (non-parallel) model against number of processors

Formatted: Caption, Left, Space Before: 0 pt, After: 0 pt

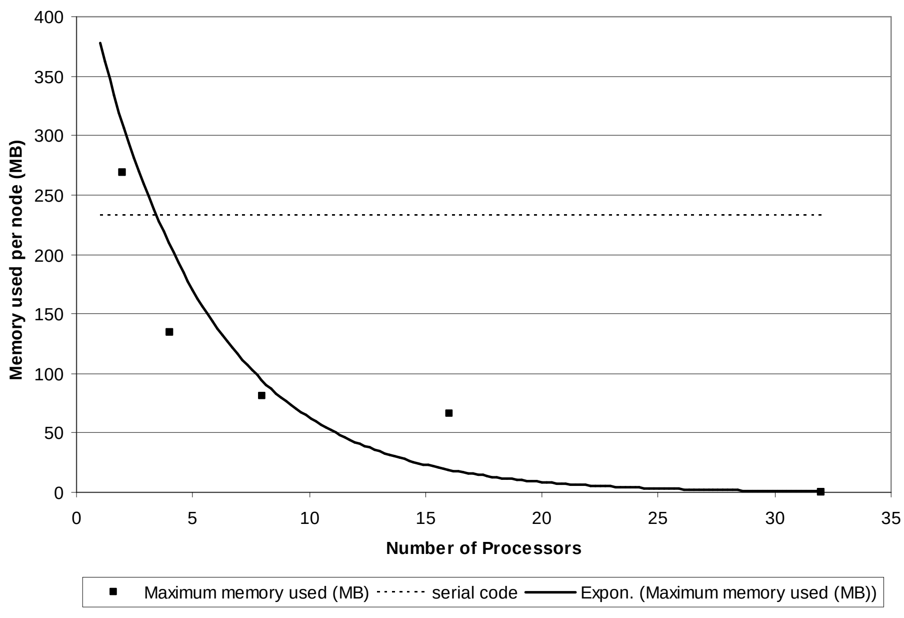


Figure 16: Environment-parallel hoverfly-aphid model: Plot of the mean maximum memory used per node (processor) in a simulation run against number of processors

Formatted: Caption

Guidelines for agent-based model scaling

There is no standard method for the development of agent-based modelling, although there are a number of agent modelling toolkits and recently some design protocols have arisen e.g. Gilbert (2007) and Grimm *et al*. (2006). Therefore, as stated in Parry (2009), there is no standard method with which a large scale agent-based model can be best developed. Instead, Parry (2009) puts forward some key questions to consider at this stage of model development (from Parry 2009 pp 152):

1. What program design do you already have and what is the limitation of this design?
    a. What is it the memory footprint for any existing implementation?
    b. What are your current run times?
2. What are your scaling requirements?
    a. How much do you need to scale now?
    b. How far do you need to scale eventually?
    c. How soon do you need to do it?
3. How simple is your model and how is it structured?
4. What are your agent complexities?
5. What are your output requirements?

The answers to these questions will help to determine the kind of solution you might seek to the problems of scale. By initially investigating the 'bottlenecks' in your model, you will be able to understand whether it is memory availability or processor speed that is limiting your model. If simple adjustments to your model code are insufficient to solve this, other solutions will then need to be sought. Perhaps a hardware upgrade may be sufficient, but if anything other than moderate scaling is required a more drastic but longer term solution might be necessary.

Question 3 is important to help decide which method may be optimal to scale up the model. Model complexity, agent interaction and spatial model environments will all pose challenges to the use of any method presented here. Some suggestions are made in this chapter as to how best to use some popular solutions when scaling a complex model, however this cannot be exhaustive and a deal of experimentation, creativity and development of solutions appropriate to the individual model is likely to be necessary.

Model outputs may also pose limits on the model, in terms of memory for data storage or the way that the output is handled (which may become critical as the model is scaled up). This should be considered when scaling-up an agent model and altering the model structure.

A protocol

In relation to the key considerations highlighted above, a simple protocol for developing a large scale agent-based simulation was defined by Parry (2009 pp 153):

1. Optimise existing code.
2. Clearly identify scaling requirements (both for now and in the future).
3. Consider simple solutions first (e.g. a hardware upgrade).
4. Consider more challenging solutions.
5. Evaluate the suitability of the chosen scaling solution on a simplified version of the model before implementing on the full model.

The main scaling solution to implement (e.g. from Table 1) is defined by the requirements of the model. Implementation of more challenging solutions should be done in stages, where perhaps a simplified version of the model is implemented on a larger scale using some of the techniques described here. Also, as demonstrated here, it is best to initially test the model with numbers lower than perhaps required for realism, to allow for faster run times when testing and experimenting with different approaches. Agent simulation development should originate with a local, flexible `prototype' and then as the model development progresses and stabilises larger scale implementations can be experimented with (Gasser *et al*. 2005). For complex solutions, such as parallel computing, a simplified model is often necessary to experiment with large numbers. Improvements to model efficiency are not necessarily linear and optimal solutions tend to be model specific, thus solutions demonstrated here will work for some agent-models but perhaps not so well for others. A key point, however is to devise a set of test cases against which the code modifications can be validated at every stage: although this should be a standard part of any software development programme, it becomes even more vital in developing parallel solutions, where subtle issues to do with timing of agent updates and access to data across cores can lead to difficult debugging problems.

Acknowledgements

Much of this work was undertaken whilst the lead author was at the Food and Environment Research Agency, UK, funded by Defra. Many thanks to the Advanced Scientific Computing team at CSIRO for their assistance in running the models on the cluster, particularly Aaron McDonagh and Tim Ho, also to Dr Andrew Evans at the University of Leeds for assistance in the early stages of this work.

Glossary

Please note this glossary is largely taken from Parry (2009)

Beowulf cluster A scalable performance computer cluster (distributed system) based on commodity hardware, on a private system network, with open source software (Linux) infrastructure (see http://www.beowulf.org/)

Block Mapping A method of partitioning an array of elements between cores of a distributed system, where the array elements are partitioned as evenly as possible into blocks of consecutive elements and assigned to processors. The size of the blocks approximates to the number of array elements divided by the number of processors.

Central Processing Unit (CPU) May be referred to as a 'core' or 'node' in parallel computing: computer hardware that executes (processes) a sequence of stored instructions (a program).

Cyclic Mapping A method of partitioning an array of elements between cores of a distributed system, where the array elements are partitioned by cycling through each core and assigning individual elements of the array to each core in turn.

Grid Computer `Grids' are comprised of a large number of disparate computers (often desktop PCs) that are treated as a virtual cluster when linked to one another via a distributed communication infrastructure (such as the internet or an intranet). Grids facilitate sharing of computing, application, data and storage resources. Grid computing crosses geographic and institutional boundaries, lacks central control, and is dynamic as cores are added or removed in an uncoordinated manner. BOINC computing is a form of distributed computing is where idle time on CPUs may be used to process information (http://boinc.berkeley.edu/)

Graphics Processing Unit (GPU) Computer hardware designed to efficiently perform computer graphics calculations, particularly for 3-dimensional objects. It operates in a similar manner to a vector computer, but is now widely available as an alternative to the standard CPU found in desktop computers.

Message passing (MP) Message passing (MP) is the principle way by which parallel clusters of machines are programmed. It is a widely-used, powerful and general method of enabling distribution and creating efficient programs (Pacheco 1997). Key advantages of using MP architectures are an ability to scale to many processors, flexibility, `future-proofing' of programs and portability (Openshaw and Turton 2000).

Message passing interface (MPI) A computing standard that is used for programming parallel systems. It is implemented as a library of code that may be used to enable message passing in a parallel computing system. Such libraries have largely been developed in C and FORTRAN, but are also used with other languages such as Java (MPJ-Express http://mpj-express.org/). It enables developers of parallel software to write parallel programs that are both portable and efficient.

Multiple Instruction Multiple Data (MIMD) Parallelisation where different algorithms are applied to different data items on different processors.

Parallel computer architecture A parallel computer architecture consists of a number of identical units that contain CPUs (Central Processing Units) which function as ordinary serial computers. These units, called cores, are connected to one another. They may transfer information and data between one another (e.g. via MPI) and simultaneously perform calculations on different data.

Single Instruction Multiple Data (SIMD) SIMD techniques exploit data level parallelism: when a large mass of data of a uniform type needs the same instruction performed on it. An example is a vector or array processor and also a GPU. An application that may take advantage of SIMD is one where the same value is being added (or subtracted) to a large number of data points.

Stream Processing is similar to a **SIMD** approach, where a mathematical operation is instructed to run on multiple data elements simultaneously.

Vector Computer/Vector Processor Vector computers contain a CPU designed to run mathematical operations on multiple data elements simultaneously (rather than sequentially). This form of processing is essentially a SIMD approach. The Cray Y-MP and the Convex C3880 are two examples of vector processors used for supercomputing in the 1980s and 1990s. Today, most recent commodity CPU designs include some vector processing instructions.

References

Abbott, C. A., M. W. Berry, E. J. Comiskey, L. J. Gross, and H.-K. Luh. 1997. Parallel Individual-Based Modeling of Everglades Deer Ecology. IEEE Computational Science and Engineering **4**:60-78.

Ankersmit, G. W., H. Dijkman, N. J. Keuning, H. Mertens, A. Sins, and H. M. Tacoma. 1986. *Episyrphus balteatus* as a predator of the aphid *Sitobion avenae* on winter wheat. Entomologia Experimentalis et Applicata **42**:271-277.

Barlow, N. D., and A. F. G. Dixon. 1980. Simulation of lime aphid population dynamics. Centre for Agricultural Publishing and Documentation, Wageningen, the Netherlands.

Barnes, D. J., and T. R. Hopkins. 2003. The impact of programming paradigms on the efficiency of an individual-based simulation model. Simulation Modelling Practice and Theory **11**:557-569.

Bithell, M. and Macmillan, W. 2007. Escape from the cell: spatial modelling with and without grids. Ecological Modelling, **200,** 59-78.

Bokma, A., A. Slade, S. Kerridge, and K. Johnson. 1994. Engineering large-scale agent- based systems with consensus. Robotics and computer-integrated manufacturing **11**:81-91.

Bouzid, M., V. Chevrier, S. Vialle, and F. Charpillet. 2001. Parallel simulation of a stochastic agent/environment interaction model. Integrated Computer-aided Engineering **8**:189-203.

Castiglione, F., M. Bernaschi, and S. Succi. 1997. Simulating the immune response on a distributed parallel computer. International Journal of Modern Physics C **8**:527-545.

Chave, J. 1999. Study of structural, successional and spatial patterns in tropical rain forests using TROLL, a spatially explicit forest model. Ecological Modelling **124**:233-254.

Cornwell, C. F., L. T. Wille, Y. G. Wu, and F. H. Sklar. 2001. Parallelization of an ecological landscape model by functional decomposition. Ecological Modelling **144**:13-20.

Da-Jun, T., F. Tang, T. A. Lee, D. Sarda, A. Krishnan, and A. Goryachev. 2004. Parallel computing platform for the agent-based modeling of multicellular biological systems. Parallel and Distributed Computing: Applications and Technologies, Lecture Notes in Computer Science **3320**:5-8.

Dibble, C., S. Wendel, and K. Carle. 2007. Simulating pandemic influenza risks of US cities. Proceedings of the 2007 Winter Simulation Conference, Vols **1-5**:1527-1529.

Dupuis, A., and B. Chopard. 2001. Parallel simulation of traffic in Geneva using cellular automata. *in* E. Kühn, editor. Virtual Shared Memory for Distributed Architecture. Nova Science Publishers, Inc., Commack, NY, USA.

Foster, I. 1995. Designing and Building Parallel Programs. Addison-Wesley, Reading, MA.

Gasser, L., K. Kakugawa, B. Chee, and M. Esteva. 2005. Smooth scaling ahead: progressive MAS simulation from single PCs to Grids. Multi-agent and multi-agent-based simulation. Joint Workshop MABS 2004 New York, NY, USA, July 19, 2004.

Gilbert, N. 2007. Agent-based Models. SAGE, London.

Grimm, V., U. Berger, F. Bastiansen, S. Eliassen, V. Ginot, J. Giske, J. Goss-Custard, T. Grand, S. Heinz, G. Huse, A. Huth, J. U. Jepsen, C. Jorgensen, W. M. Mooij, B. Muller, G. Pe'er, C. Piou, S. F. Railsback, A. M. Robbins, M. M. Robbins, E. Rossmanith, N. Ruger, E. Strand, S. Souissi, R. A. Stillman, R. Vabo, U. Visser, and D. L. DeAngelis. 2006. A standard protocol for describing individual-based and agent-based models. Ecological Modelling **198**:115-126.

Haefner, J. W. 1992. Parallel computers and individual-based models: An overview. Pages 126-164 *in* D. L. DeAngelis and L. J. Gross, editors. Individual-based models and approaches in ecology: populations, communities and ecosystems. Routledge, Chapman and Hall, New York.

Harmel, N., R. Almohamad, M.-L. Fauconnier, P. D. Jardin, F. Verheggen, M. Marlier, E. Haubruge, and F. Francis. 2007. Role of terpenes from aphid-infested potato on searching and oviposition behaviour of *Episyrphus balteatus*. Insect Science **14**:57-63.

Hellweger, F. L. 2008. Spatially explicit individual-based modeling using a fixed super-individual density. Computers and Geosciences **34**:144-152.

Hemptinne, J.-L., A. F. G. Dixon, J.-L. Doucet, and J.-E. Petersen. 1993. Optimal foraging by hoverflies (Diptera, Syrphidae) and ladybirds (Coleoptera: Coccinellidae): Mechanisms. European Journal of Entomology **90**:451-455.

Holloway, G. J., and A. R. McCaffery. 1990. Habitat utilisation and dispersion in *Eristalis pertinax* (Diptera: Syrphidae). Entomologist **109**:116-124.

Host, G. E., H. W. Stech, K. E. Lenz, K. Roskoski, and R. Mather. 2008. Forest patch modeling: using high performance computing to simulate aboveground interactions among individual trees. Functional Plant Biology **35**:976-987.

Immanuel, A., M. W. Berry, L. J. Gross, M. Palmer, and D. Wang. 2005. A parallel implementation of ALFISH: simulating hydrological compartmentalization effects on fish dynamics in the Florida Everglades. Simulation Modelling Practice and Theory **13**:55-76.

Ishida, T., L. Gasser, and H. Nakashima. 2005. Massively Multi-Agent Systems I. First International Workshop. *in* MMAS 2004. Springer-Verlag Berlin Heidelberg, Kyoto, Japan.

Jamali, N., P. Scerri and T. Suguwara. (eds) 2008. Massively Multi-Agent Technology: AAMAS Workshops, MMAS 2006, LSMAS 2006, and CCMMS 2007 Hakodate, Japan, May 9, 2006 Honolulu, HI, USA, May 15, 2007, Selected and Revised Papers, LNAI 5043, Springer-Verlag Berlin Heidelberg.

Kadau, K., T. C. Germann, and P. S. Lomdahl. 2006. Molecular dynamics comes of age: 320 billion atom simulation on BlueGene/L. International Journal of Modern Physics C **17**:1755.

Kareiva, P., and G. Odell. 1987. Swarms of predators exhibit ``preytaxis" if individual predators use area-restricted search. The American Naturalist **130**:233-270.

Khronos 2010. OpenCL implementations, tutorials and sample code. Beaverton. http://www.khronos.org/developers/resources/opencl.

Kindlmann, P., and A. F. G. Dixon. 1993. Optimal foraging in ladybird beetles (Coleoptera: Coccinellidae) and its consequences for their use in biological control. European Journal of Entomology **90**:443-450.

Kirk, D.B. and Hwu, W.W., 2010. Programming Massively parallel Processors: a hands-on approach. Morgan-Kaufmann.

Lomdahl, P. S., D. M. Beazley, P. Tamayo, and N. Gronbechjensen. 1993. Multimillion particle molecular-dynamics on the CM-5 International Journal of Modern Physics C: Physics and Computers **4**:1075-1084.

Lorek, H., and M. Sonnenschein. 1995. Using parallel computers to simulate individual-oriented models in ecology: a case study. Proceedings: ESM '95 European Simulation Multiconference, Prague, June 1995.

Lozano, M., P. Morillo, D. Lewis, D. Reiners and C. Cruz-Neira. 2007. A distributed framework for scalable large-scale crowd simulation. *In* R. Shumaker (Ed.): Virtual Reality, HCII 2007, Lecture Notes in Computer Science 4563, pp. 111-121.

Lysenko, M., and R. M. D'Souza. 2008. A Framework for Megascale Agent Based Model Simulations on Graphics Processing Units. Journal of Artificial Societies and Social Simulation **11**:10.

Massaioli, F., F. Castiglione, and M. Bernaschi. 2005. OpenMP Parallelization of Agent-Based Models. Parallel Computing **31**:1066-1081.

Mellott, L.E., Berry, M.W., Comiskey, E.J. and Gross, L.J., 1999. The design and implementation of an individual-based predator-prey model for a distributed computing environment. Simulation Practice and Theory, **7**, 47-70.

Metz, J. A. J., and A. M. de Roos. 1992. The role of physiologically structured population models within a general individual based model perspective. Pages 88-111 *in* D. L. DeAngelis and L. J. Gross, editors. Individual Based Models and Approaches in Ecology: Concepts and Models. Routledge, Chapman and Hall, New York.

Minson, R. and Theodoropoulos, G.K., 2008. Distributing RePast agent-based simulations with HLA. Concurrency Computat.: Proc. Exper. **20**, 1225-1256

Nagel, K., and M. Rickert. 2001. Parallel implementation of the TRANSIMS micro-simulation. Parallel Computing **27**:1611-1639.

Nichols, J. A., T. G. Hallam, and D. T. Dimitrov. 2008. Parallel simulation of ecological structured communities: Computational needs, hardware capabilities, and nonlinear applications. Nonlinear Analysis-Theory Methods & Applications **69**:832-842.

Openshaw, S., and I. Turton. 2000. High performance computing and the art of parallel programming : an introduction for geographers, social scientists, and engineers. Routledge, London.

Pacheco, P. S. 1997. Parallel Programming with MPI. Morgan Kauffman Publishers, San Francisco, CA.

Parry, H. R. 2006. Effects of Land Management upon Species Population Dynamics: A Spatially Explicit, Individual-based Model. PhD Thesis. University of Leeds, UK.

Parry, H. R. 2009. Agent Based Modeling, Large Scale Simulations. Pages 148-160 *in* R. A. Meyers, editor. Encyclopedia of Complexity and Systems Science. Springer, New York.

Parry, H. R., and A. J. Evans. 2008. A comparative analysis of parallel processing and super-individual methods for improving the computational performance of a large individual-based model. Ecological Modelling **214**:141-152.

Parry, H. R., A. J. Evans, and A. J. Heppenstall. 2006a. Millions of Agents: Parallel Simulations with the Repast Agent-Based Toolkit. Cybernetics and Systems 2006, Proceedings of the 18th European Meeting on Cybernetics and Systems Research.

Parry, H. R., A. J. Evans, and D. Morgan. 2006b. Aphid population response to agricultural landscape change: a spatially explicit, individual-based model. Ecological Modelling **199**:451–463.

Popov, K., V. Vlassov, M. Rafea, F. Holmgren, P. Brand, and S. Haridi. 2003. Parallel agent-based simulation on a cluster of workstations. EURO-PAR 2003 Parallel Processing **2790**:470-480.

Powell, W., S. A'Hara , R. Harling, J. M. Holland, P. Northing, C. F. G. Thomas, and K. F. A. Walters. 2004. 3D Farming: Making biodiversity work for the farmer, Report to Defra LK0915.

Rao, D.M., Chernyakhovsky, A. and Rao, V., 2009. Modelling and analysis of global epidemiology of avian influenza. Environmental Modelling and Software, **24**, 124-134.

Railsback S F, Lytinen S L and Grimm V (2005). StupidModel and Extensions: A Template and Teaching Tool for Agent-based Modeling Platforms. Webpage at http://condor.depaul.edu/~slytinen/abm/StupidModelFormulation.pdf

Ramachandramurthi, S., T. G. Hallam, and J. A. Nichols. 1997. Parallel simulation of individual-based, physiologically structured population models. Mathematical and Computer Modelling **25**:55-70.

Scheffer, M., J. M. Baveco, D. L. DeAngelis, K. A. Rose, and E. H. van Nes. 1995. Super-Individuals: a simple solution for modelling large populations on an individual basis. Ecological Modelling **80**:161-170.

Schuler, A. J. 2005. Diversity matters: dynamic simulation of distributed bacterial states in suspended growth biological wastewater treatment systems. Biotechnology and Engineering **91**:62-74.

Springel, V. 2005. The cosmological simulation code gaadget-2. Mon. Not. R. Ast. Soc. **364**, 1105-134

Stage, A. R., N. L. Crookston, and R. A. Monserud. 1993. An aggregation algorithm for increasing the efficiency of population models. Ecological Modelling **68**:257-271.

Stone, J.E., Phillips, J.C., Freddolino, P.L., Hardy, D.J., Trabuco, L.G., and Schulten, K., 2007. Accelerating Molecular Modelling Applications with Graphics Processors. J. Comput. Chem., **28**, 2618-2640.

Takeuchi, I. 2005. A massively multi-agent simulation system for disaster mitigation. *in* Massively Multi-Agent Systems I: First International Workshop MMAS 2004, Kyoto, Japan, December 2004. Springer-Verlag, Berlin Heidelberg.

Tenhumberg, B. 1995. Estimating predatory efficiency of *Episyrphus balteatus* (Diptera: Syrphidae) in cereal fields. Environmental Entomology **24**:687-691.

Tenhumberg, B. 2004. Predicting predation efficiency of biocontrol agents: linking behavior of individuals and population dynamic. *in* C. Pahl-Wostl, S. Schmidt, and T. Jakeman, editors. iEMSs 2004 International Congress: "Complexity and Integrated Resources Management". International Environmental Modelling and Software Society, Osnabrueck, Germany.

Timm, I. J., and D. Pawlaszczyk. 2005. Large Scale Multiagent Simulation on the Grid. Proceedings of the Workshop on Agent-based grid Economics (AGE 2005) at the IEEE International Symposium on Cluster Computing and the Grid (CCGRID). Cardiff University, Cardiff, UK.

Wang, D., M. W. Berry, E. A. Carr, and L. J. Gross. 2006a. A parallel fish landscape model for ecosystem modeling Simulation **82** 451-465

Wang, D., M. W. Berry, and L. J. Gross. 2006b. On parallelization of a spatially-explicit structured ecological model for integrated ecosystem simulation.

Comment [par44g 2]: Check this

Comment [par44g 3]: full

Comment [par44g 4]: Full

International Journal of High Performance Computing Applications **20**:571-581.

Wang, D., E. Carr, L. J. Gross, and M. W. Berry. 2005. Toward ecosystem modeling on computing grids. Computing in Science and Engineering **7**:44-52.

Wang, D., L. Gross, E. Carr, and M. Berry. 2004. Design and implementation of a Parallel Fish Model for South Florida. Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04).

Wendel, S., and C. Dibble. 2007. Dynamic Agent Compression. Journal of Artificial Societies and Social Simulation **10**:9.

Wilkinson, B., and M. Allen. 2004. Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers (second edition). Pearson Prentice Hall, New Jersey, USA.

Woods, J., and W. Barkmann. 1994. Simulating Plankton Ecosystems by the Lagrangian Ensemble Method. Philosophical Transactions of the Royal Society of London Series B-Biological Sciences **343**:27-31.

Woods, J. D. 2005. The Lagrangian Ensemble metamodel for simulating plankton ecosystems. Progress In Oceanography **67**:84-159.

Wu, Y. G., F. H. Sklar, K. Gopu, and K. Rutchey. 1996. Fire simulations in the Everglades Landscape using parallel programming. Ecological Modelling **93**:113-124.

Appendix: Rules for Hoverfly sub-model

Development
Development of hoverflies is highly simplified and birth and death is minimised (see below). The only development that occurs in the model is the transition of larvae to adults. In this, there is a 50% probability the hoverfly will be female (determined at birth) and male hoverflies are not included in the model from this stage onwards as their activities are assumed not to influence the distribution of larvae and thus the mortality of the aphids.

The transition from larvae to adult is modelled with the assumption that the larvae need to eat a minimum of 120 aphids in total to reach a weight at which they are able to pupate (28 mg) (Ankersmit *et al.* 1986). Thus, once this number of aphids has been consumed by an individual larva it pupates and becomes adult (if male it is then removed from the model).

Reproduction
In this model oviposition occurs once within a single 1m^2 area (i.e. grid cell) per day. This occurs providing aphids are present and the location has no other larvae. It is assumed only 1 egg is laid per day within the cell, and the egg is assumed to become larvae the next day. This is probably an underestimate, however it can easily be modified at a later stage. A suggested estimate may be up to 49 eggs within a 1m^2 area per day, based upon Harmel *et al.* (2007), where a high oviposition rate of *E. balteatus* was observed when aphid-infested potato was studied (a mean of 48.9 eggs per laying and per female). This study also found that no eggs were produced by the hoverfly on healthy aphid-free plants.

Mortality
The scenarios shown here do not include adult hoverfly mortality. Experiments with mortality in the model showed that adult mortality has a high impact upon the population dynamics of the syrphids and should be included in further developments of the model.

Mortality of larvae occurs when no aphids are present to feed them (possible if aphids are consumed or are alate and fly away), otherwise there is no mortality of larvae.

Movement and dispersal
Movement of syrphids and oviposition is key to this model. A number of rules govern the oviposition of larvae by female adult syrphids:

- Search for prey is not random (Kindlmann and Dixon 1993).
- Refrains from ovipositing in the presence of conspecific larvae (Hemptinne *et al.* 1993).
- Avoids laying eggs close to old aphid colonies, recognized by the presence of winged aphids (Hemptinne *et al.* 1993).

In this model rules govern a non-random search for prey, where eggs are only laid where aphid colonies are present and oviposition does not occur where larvae are already present. The model does not include a rule to recognise old aphid colonies at present, but this information is available in the model and could be included at a later stage.

Basic movement

A model of syrphid predator movement proposed by Kareiva and Odell (1987) is that predators move at constant speed but change direction of movement more often when satiated (area restricted search) and that increase in prey density increases the feeding rate and satiation of the predators (applied to *Uroleucon nigrotuberculatum* and *Coccinella septempunctata*). However, this may have restricted applicability to the early stages of aphid colony development (Kindlmann and Dixon 1993) and it has not been proved that this strategy is optimal (it was arbitrarily chosen).

This model will use a simplified movement rule based upon this principle - the adult female hoverflies move in a random direction, but move a greater distance if no aphids are present or the crop is early in season. It has been shown that crop growth stage and habitat type may influence syrphid movement patterns and oviposition (Powell *et al*. 2004), providing the foundations for this behavioural rule.

It is assumed that hoverflies move between 4 and 6 metres a day (given that a mark-recapture study of Holloway and McCaffery (1990) found hoverflies moved between 20-30m in a 5 day period). Thus, in the model, `focused' movement in favourable habitat (margins or late season crop) or around aphid colonies is set between 0-4 m and in unfavourable habitat (early season crop) movement is set at 4-6 m per day.

Foraging optimisation

It has been suggested that the model of Kareiva and Odell (1987) can be improved by adding terms to describe foraging optimisation (Kindlmann and Dixon 1993). This will enable the model to function at later stages of aphid colony development. The ability of the predator to assess the present and future quality of an aphid colony for their larvae should be included in the model. The effect of more than one aphid colony present in a landscape should also be considered - the presence of other colonies is likely to reduce the optimal number of eggs laid by the predator in a particular aphid colony (Kindlmann and Dixon 1993).

This is applied in the model through a simple behavioural rule: if there are aphids present within a given $1m^2$ location but other larvae are also present the hoverfly does not oviposit but moves on a short distance.

Parasitation/predation

A very simple model of aphid consumption was constructed based on the research of Ankersmit *et al*. (1986), Equation 1. More recent, complex models exist (e.g. the use of a Holling type-III function by Tenhumberg (1995)), however the nature of the model presented here at this stage does not require this level of complexity.

$$MORT = (0.3119e^{0.0337(A\times 24)} \times D + (2.512e^{0.0253(A\times 24)})$$

Equation 1: Where MORT is the predation rate per day; A is the age of the Syrphid larvae in days; D is the density of aphids per cm^2 (which is scaled down from $1m^2$ in the model).