Chapter 14
Large Scale Agent-Based Modelling: A Review and Guidelines for Model Scaling

Hazel R. Parry and Mike Bithell

Abstract This chapter provides a review and examples of approaches to model scaling when constructing large agent-based models. A comparison is made between an aggregate 'super-individual' approach, as run on a single processor machine, and two different approaches to parallelisation of agent models run on multi-core hardware. Super-individuals provide a straightforward solution without much alteration of the model formulation and result in large improvements in model efficiency (speed and memory use). However, there are significant challenges to using a super-individual approach when relating super-individuals to individuals in time and space. Parallel computing approaches accept the requirement for large amounts of memory or CPU and attempt to solve the problem by distributing the calculation over many computational units. This requires some modification of the model software and algorithms to distribute the model components across multiple computational cores. This can be achieved in a number of different ways, two of which we illustrate further for the case of spatial models, an 'agent-parallel' and an 'environment-parallel' approach. However, the success of such approaches may also be affected by the complexity of the model (such as multiple agent types and agent interactions), as we illustrate by adding a predator to our example simulation. Between these two parallelisation approaches to the case study, the environment-parallel version of the model, written in C++ instead of Java, proved more efficient and successful at handling parallel processing of complex agent interactions. In conclusion, we use our experiences of creating large agent-based simulations to provide some general guidelines for best practice in agent-based model scaling.

H.R. Parry (✉)
Department of Entomology, Commonwealth Scientific and Industrial Research Organisation (CSIRO), Canberra, Australia
e-mail: Hazel.Parry@csiro.au

M. Bithell
Department of Geography, University of Cambridge, Cambridge, UK
e-mail: mike.bithell@geog.cam.ac.uk

A.J. Heppenstall et al. (eds.), *Agent-Based Models of Geographical Systems*,
DOI 10.1007/978-90-481-8927-4_14, © Springer Science+Business Media B.V. 2012

14.1 Introduction

In agent-based simulation (ABS), 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). 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 ABS 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 when the modeller wishes 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-based models (ABMs) are spatial.

In this chapter we focus on spatial ABMs. We compare the aggregate 'super-individual' approach as run on a single processor machine with two different approaches to parallelisation of agent models run on multi-core hardware, using Message-Passing Interface (MPI) libraries to achieve communication between cores. We use a model of insect population dynamics to provide specific examples of each approach. We point out the potential pitfalls that arise from aggregation of individuals in a spatial context and from communication complications that arise when moving from serial to parallel code. The advantages and disadvantages of each approach for speeding up computation and managing memory use will be discussed.

14.2 Review of Large-Scale Modelling Techniques

A number of methodologies have arisen 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 14.1. This chapter focuses on the last two entries in the table, as the 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.

Table 14.1 Potential solutions to implement when faced with a 'large scale' ABM (Adapted from Parry 2009)

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 sufficiently identical to larger systems, or that there is a simple scaling relationship deducible from the reduced model
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 (CPU speeds no longer increasing with Moore's law). 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 (see last entry in the table)
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 Graphics Processing Units (GPU) may compensate this somewhat – see below). This approach works more efficiently with SIMD (see glossary), possibly not so suitable for ABMs 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

14.3 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 ABMs 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 Fig. 14.1.

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 or cell, 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). 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 can be an advantage for the algorithm to combine similar pieces of information to produce a more compact representation. 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 spatial cell), they are extracted from the super-individual and become separate individuals. If the attributes of the uncontained agent now match 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.

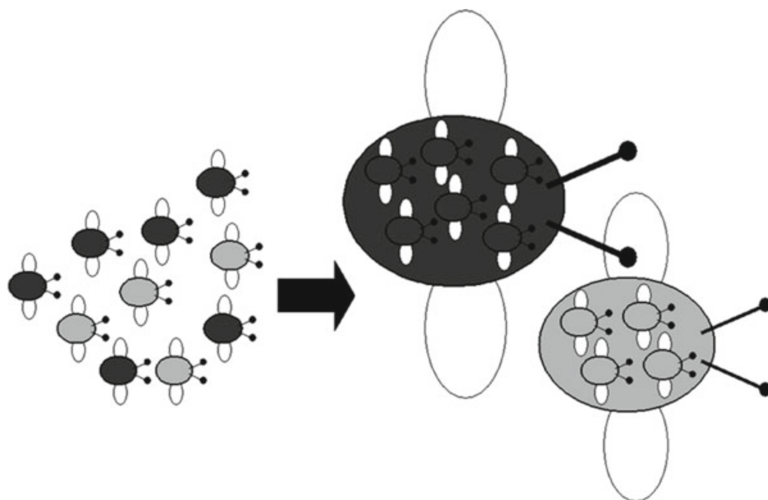

Fig. 14.1 'Super-agents': grouping of individuals into single objects that represent the collective (Taken from Parry and Evans 2008)

14.4 Parallel Computing

Instead of attempting to reduce the computational load by agent-aggregation, parallel approaches accept the requirement for large amounts of memory or CPU and attempt to solve the problem by distributing the calculation over many computational units. One sense in which such distribution can be used is for 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 machines 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 a model in parallel is challenging. Despite this, over the last 10 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, 2005, 2006a, b; Immanuel et al. 2005; 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). ABS 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). ABMs can vary in complexity, but most tend to be complex, especially in the key model elements of spatial structure and agent heterogeneity.

14.4.1 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), which may perform simultaneous calculations on different data (Wilkinson and Allen 2004). These calculations may be the same or different, depending upon whether a 'Single Instruction Multiple Data' (SIMD) or 'Multiple Instruction Multiple data' (MIMD) approach is implemented (see glossary). Large-scale shared-memory vector processing machines operating via SIMD are now something of a rarity (although individual processors will usually use such methods internally). On the other hand, desktop machines now typically have multi-core processors (with each core essentially acting as a separate CPU), and large-scale high performance computer (HPC) clusters built from such machines with fast low-latency network inter-connects allow the same code to be tested on a desktop and then deployed to a larger system with little or no modification. As there is no longer a trend toward increasing individual CPU speeds, increases in computing power are mostly coming from higher numbers of cores per chip, so that building parallel applications will be a necessary part of exploiting hardware improvements. By designing models that exploit local desktop parallelism and scale to HPC machines, one can not only benefit from desktop speed improvements but also thoroughly test parallelization before making larger runs on more expensive systems. In practice MPI-based applications fulfil this role well, but alternative architectures are beginning to compete with this approach.

14.4.2 Graphics Processing Units (GPUs)

Recent advances in the power of Graphics Processing Units (GPU) now make it easier for modellers to take advantage of data-parallel computer architectures on desktop machines (Lysenko and D'Souza 2008). Multi-core graphics cards can be used not just for display purposes, but also for more general numerical computing tasks (sometimes referred to as GPGPU (General Purpose GPU)). 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 as these devices have been designed with very high memory bandwidth (although this comes at the cost of higher memory latency).

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 ABM 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 do not 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.

The architecture of GPUs is rather different from traditional cluster systems. Groups of stream processors are arranged with their own local shared memory, plus access to global memory that resides on the GPU. To make use of this, data must be copied from the CPU-accessible memory into the graphics card. Then the data can be processed by invoking one of a number of 'Kernel functions' that run on the GPU. Lysenko and D'Souza (2008) reformulated two ABMs (Sugar-scape and Stupid Model) to operate on a GPU by the use of large, multi-dimensional arrays to contain the complete state of an agent. Kernels were programmed to run update functions on these arrays. A different kernel was created for each update function, which operated one at a time on the dataset. Some careful coding was required when handling mobile agents (see below), but good performance was obtained for models with a few millions of agents on a domain of up to $2{,}048 \times 2{,}048$ cells. However, their approach required explicit use of the graphics card's texture maps and pixel colour values – such technical details make it awkward for the general programmer to easily access and exploit hardware of this type. 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 a 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 than CUDA (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). Simulations of cell-level biological systems using FLAME (Richmond et al. 2009a, b), a finite-state machine agent architecture designed specifically to exploit parallel hardware, seem to bear out the potential for simulation speedup that a GPU can offer. However, problems with very large memory requirements may still be challenging for these architectures (as of the time of writing the largest GPU memories are of order 4 GB). The solution in such cases is likely to require running on multiple GPUs, possibly distributed over many independent nodes, with the result that the message passing techniques described below will still be needed.

14.4.3 Challenges of Parallel Computing

Several key challenges arise when implementing an ABM 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) ABMs 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[1]) that adds external Message Passing Interface (MPI)[2] calls to Java, but use of this library required extensive restructuring of the original model code, as it was originally designed for serial execution. Since this work was carried out, a facility for making MPI-parallel models using C++ has been added to RePast. Conversion of existing Java code to C++ is usually fairly straightforward, (we will use an alternative C++ library later in this chapter) but the algorithmic considerations regarding the changes needed to ensure correct functioning of parallel code discussed below are still relevant.

---

[1]Message Passing Interface for Java (MPIJava) http://www.hpjava.org/mpiJava.html is no longer available for download online. It has been super-ceded by MPJ-Express http://mpj-express.org/

[2]See glossary for definition of MPI

14.4.4 Approaches to Agent Parallelism

Parallel agent modelling requires that agent computation is distributed in a way that allows model updates to be carried out on many computational cores simultaneously. This can be achieved in a number of different ways, two of which we will illustrate further for the case of spatial models. In both cases the idea is to send the whole data-structure involved with each agent out to processor cores for updating. In the 'agent parallel' approach, this is done without reference to any spatial structure, but is needed for carrying out update tasks. The 'environment parallel' approach instead divides up the spatial domain between cores and carries the agents associated with each spatial unit along with the spatial sub-division.

14.4.4.1 The 'Agent-Parallel' Approach

This approach 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 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. The advantage is that load balancing is more straightforward, as cores can be loaded with agents symmetrically so that each core bears as nearly as possible an equal share of the computation. However, since the spatial data are not included in this process, an extra overhead is implied in ensuring that spatially localized agent interactions are dealt with consistently, as co-location on a core does not guarantee co-location in space.

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).

14.4.4.2 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). Local spatial interactions between agents are now likely also to be local to a single core, with potentially easier co-ordination of agent updates. However, when the agents are highly mobile, or when the density of agents varies spatially over time, balancing the load between cores becomes more of an issue, as the allocation of agents to cores must be re-calculated at intervals that depend upon the model dynamics.

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 modelling (Chave 1999).

Examples from social science:

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

14.5 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 Sect. 14.6.1. 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 that 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.

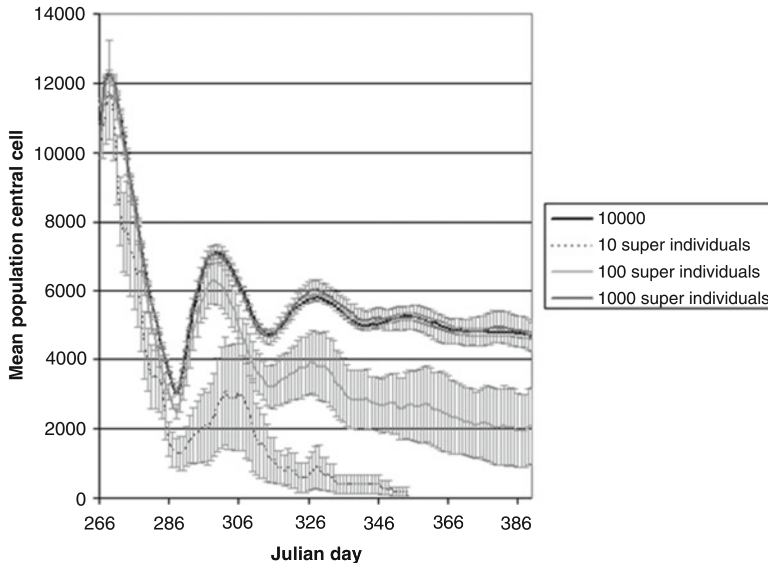

Fig. 14.2 10,000 individuals: comparison between individual-based simulation, 1,000 super-individual simulation (each represents 10 individuals), 100 super-individual simulation (each represents 100 individuals) and 10 super-individual simulation (each represents 1,000 individuals), showing 95% confidence limits derived from the standard error (Taken from Parry and Evans 2008)

14.5.1 Temporal Results

The temporal comparison of super-individuals (representing 10,000 individuals) given in Parry and Evans (2008) is shown in Fig. 14.2. The results for 1,000 super-individuals (scale factor ten 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 composed 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.

14.5.2 Spatial Results

The spatial results given in Parry and Evans (2008) are summarised in Fig. 14.3. 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 more sensitive spatial errors.

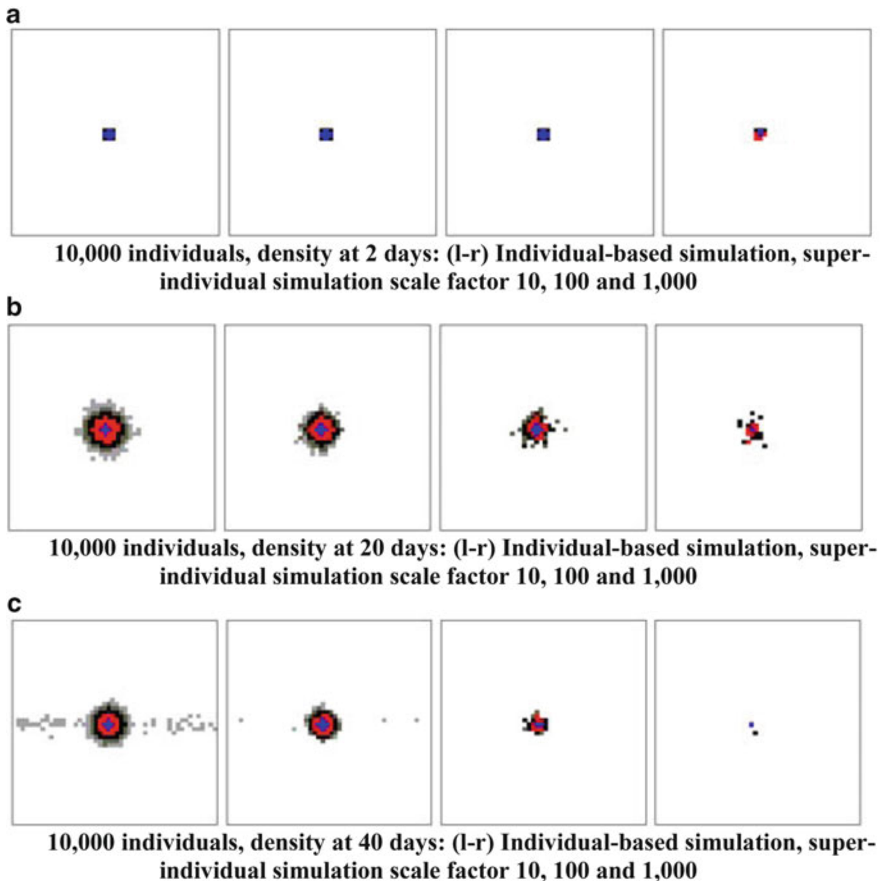

Fig. 14.3 Spatial density distributions for individual-based versus super-individual simulations (10,000 aphids) at (**a**) 2 days (**b**) 20 days and (**c**) 40 days. The distribution further from the central cell is influenced by the constant westerly wind direction to result in a linear movement pattern (Taken from Parry and Evans 2008)

14.6 Parallel Computing Examples: 'Agent-Parallel' and 'Environment-Parallel' Approaches

14.6.1 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. The model was constructed with the RePast 2.0 agent-based software development toolkit for Java (http://repast.sourceforge.net/). The example illustrates how spatial interactions between predators and prey can lead to difficulties in reproducing the results from serial code.

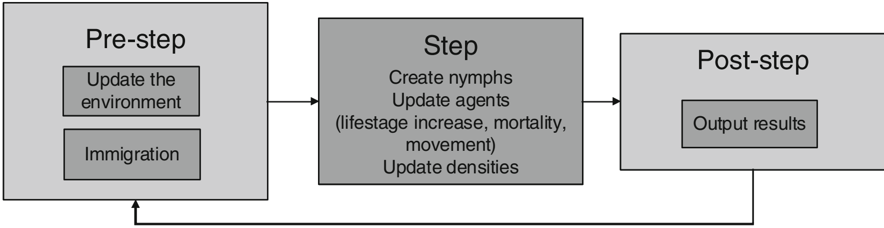

Fig. 14.4 Simplified flow chart for the aphid model

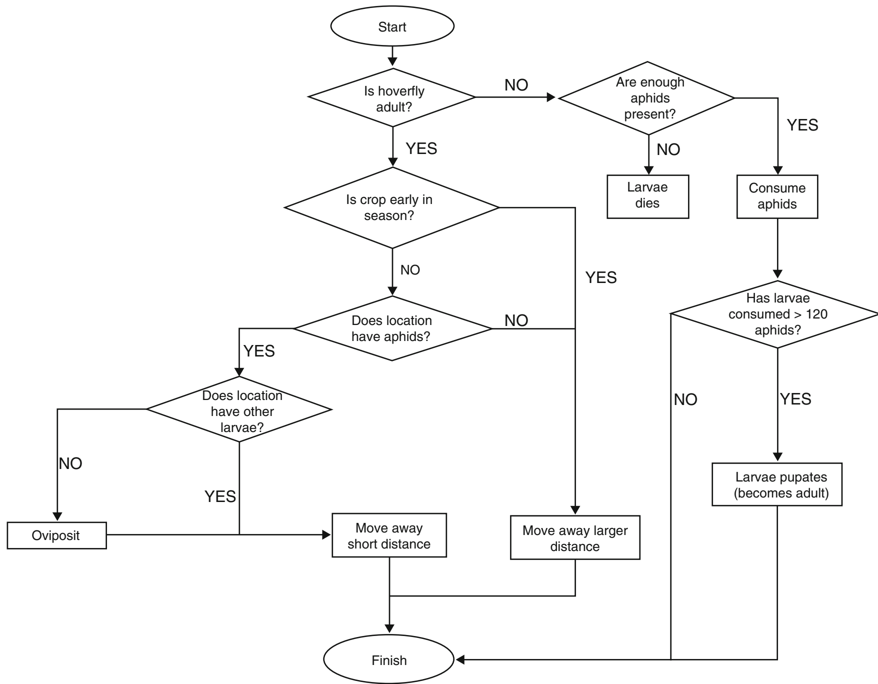

Fig. 14.5 Flowchart of the syrphid model

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. The model describes the population lifecycle of an aphid, *Rhopalosiphum padi*. However, in the individual-based model presented here, the movement of adult female syrphids across the landscape is also modelled. 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 Fig. 14.4.

The basic rules followed in the syrphid model are given in Fig. 14.5, with more detail on the rules used in the hoverfly model given in the Appendix, as this sub-model is unpublished elsewhere. The two sub-models (aphids and hoverflies) are

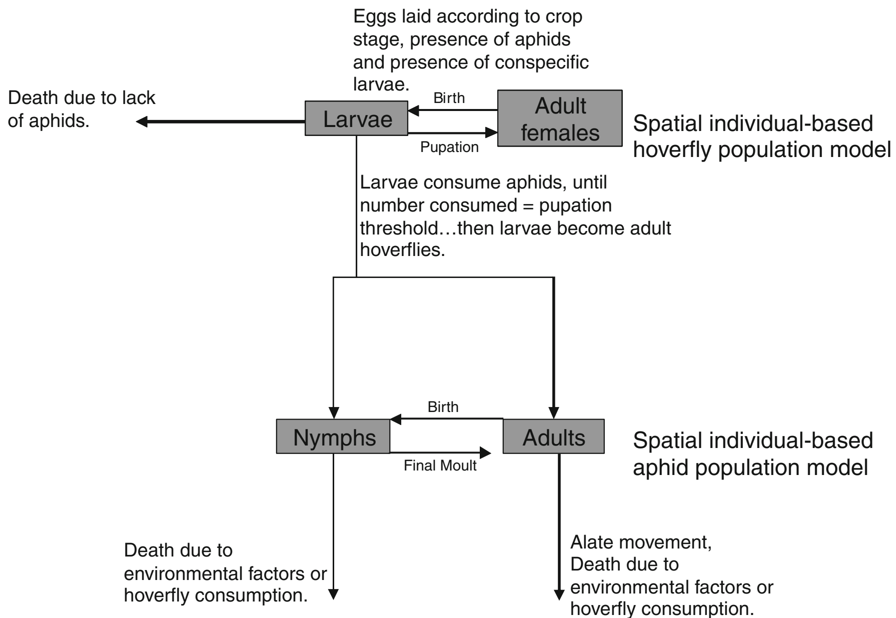

Fig. 14.6 Key processes of the hoverfly-aphid model

connected to one another, by the consumption of aphids by hoverfly larvae. The relationship between the two models is simplified in Fig. 14.6.

The simple model landscape is as shown later in this chapter, two rectangular fields split by a central margin (see Fig. 14.11). The space is divided into a set of square cells, each of area 1 m^2. The model is initiated with one apterous adult aphid in each field cell and one female adult hoverfly in each cell at the field margin.

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 1), run on a Beowulf cluster (see glossary). At each time step, agents are updated on the worker cores (see Fig. 14.7), as the control core maintains global insect density and aphid consumption information and controls the simulation flow.

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 serial model accurately.

However, when hoverfly larvae were introduced, the parallel implementation did not replicate the original, non-parallel version. The added complexity of including predators gave rise to two major problems. 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 might attempt to consume aphids allocated to 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

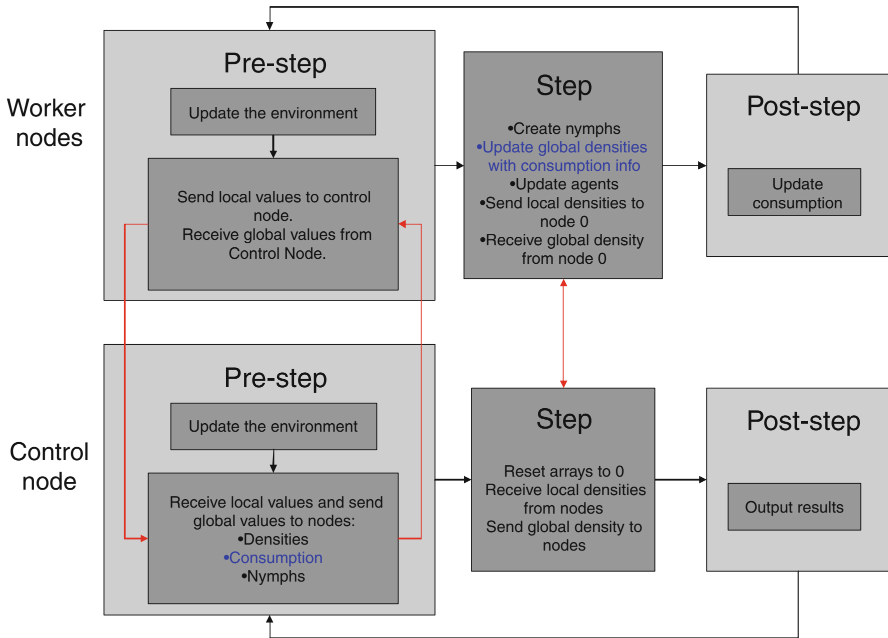

Fig. 14.7 Parallel model flow chart where blue text indicates interaction between the two sub-models and red arrows indicate interaction between the control core and the worker cores

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 larva on another core).

The result was that, occasionally, the total calculated consumption of aphids per iteration per cell was 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 was greater than the total aphid density, the consumption was reduced to the total aphid density. However, the problem still remained, and it gave rise to lower aphid populations in the parallel model than in the non-parallel model, as shown by Fig. 14.8.

In addition, more hoverflies were 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 state 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, as shown in Fig. 14.9. This also acts to reduce the aphid population below that of the non-parallel simulation.

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

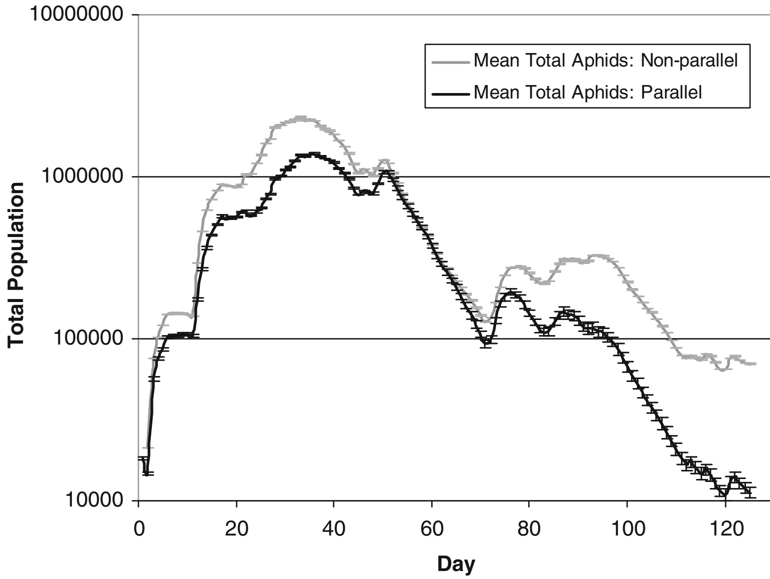

Fig. 14.8 Comparison of the temporal dynamics of the total population of aphids between parallel and non-parallel simulation implementations (error bars show standard error)

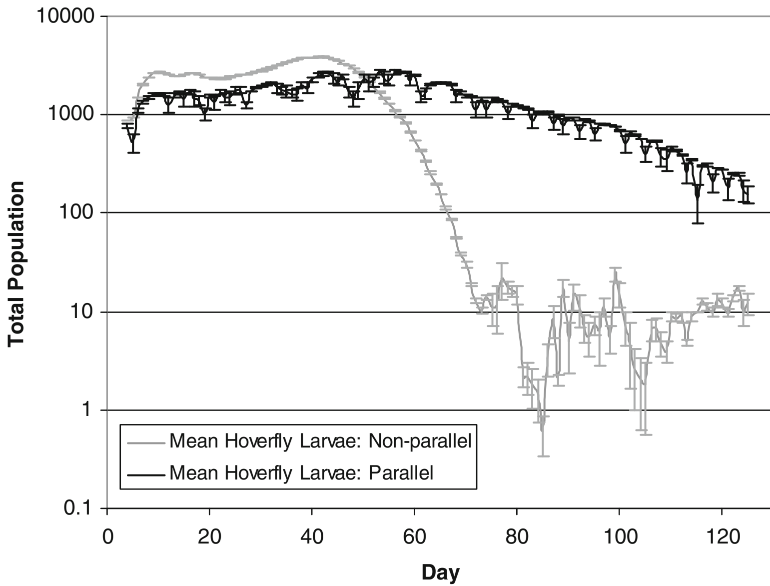

Fig. 14.9 Comparison of the temporal dynamics of the total population of hoverfly larvae between parallel and non-parallel simulation implementations

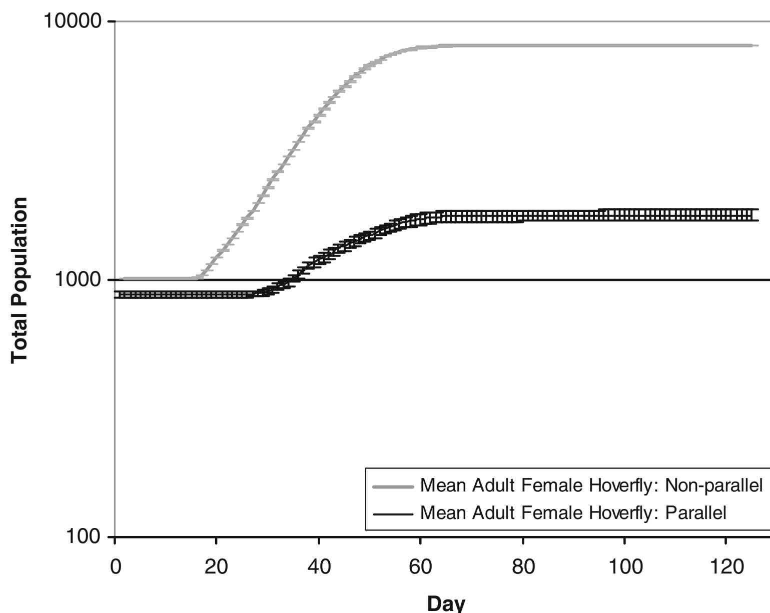

Fig. 14.10 Comparison of the temporal dynamics of the total population of adult female hoverfly between parallel and non-parallel simulation implementations (no mortality)

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) (Fig. 14.10).

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.

Such programming issues need to be resolved before the agent-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 ABMs 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 ABM increases, it may be more efficient to distribute the model environment (as described in the next section), rather than the agents, so that local agents may interact directly and update parameters within a single model iteration.

14.6.2 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 Fig. 14.11) 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 can 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.

For the current simulation, cells are 1 m^2 – this means that typical movement per timestep (1 day) exceeds the cell size (see the 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 Fig. 14.12). As this communication step is relatively expensive, it reduces the level of speedup achievable somewhat.

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 serial 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

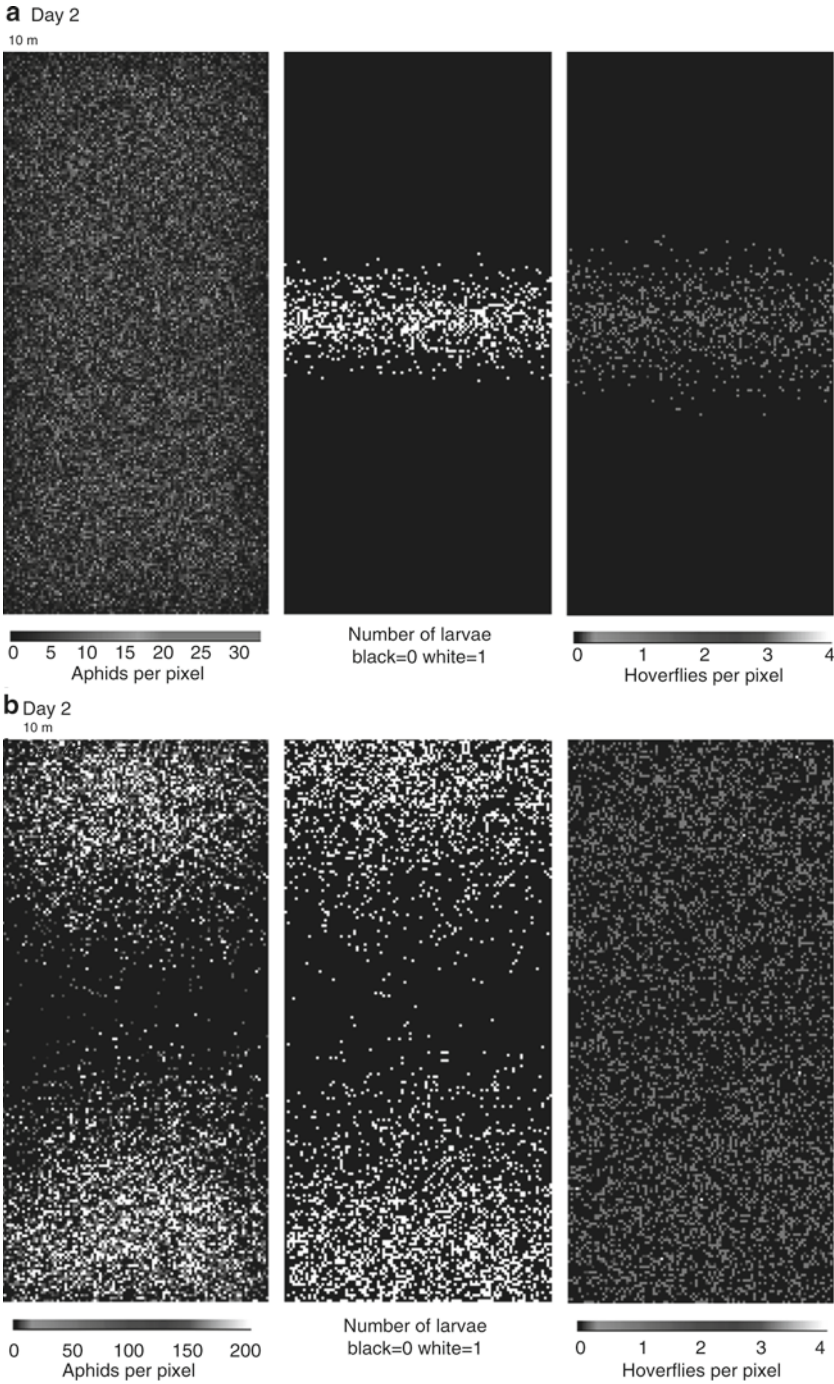

Fig. 14.11 Snapshots of spatial distributions of aphids, hoverfly larvae and hoverfly adults showing spatial distribution over a 100 m × 200 m domain

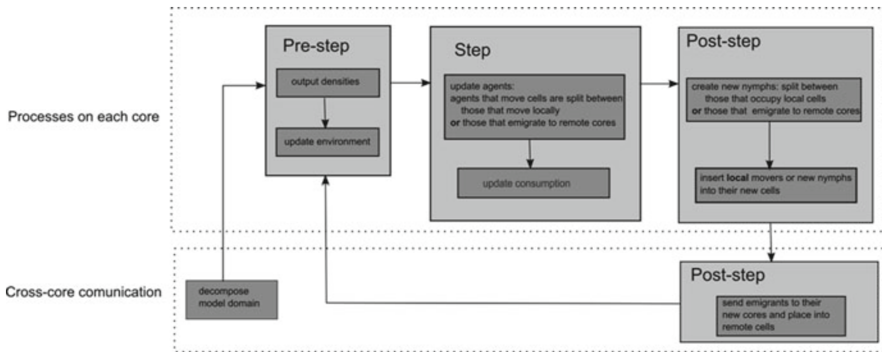

Fig. 14.12 Schematic to show the sequencing of the environment-parallel model. Note that here there is no distinction between workers and control – all cores are treated equally and all run the same set of processes

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 parallelisation, as discussed in the preceding sections. An analysis of the speed-up of the Java model (when simulating aphids only) is given later in this chapter, which also draws comparisons with the speed of the super-individual model implementation and the efficiency of the C++ environment-parallel model.

While the environment-parallel version of the model successfully reproduced the results of the serial code, the 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 14.11 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

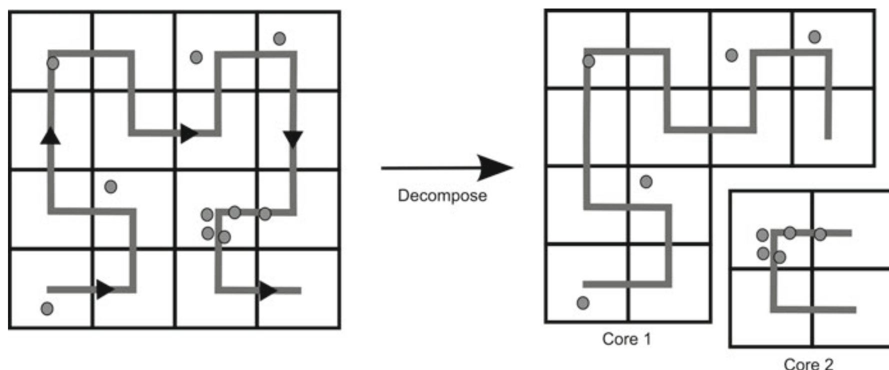

Fig. 14.13 Spatial domain decomposition using a Peano-Hilbert space filling curve. A self-similar path is drawn connecting all the cells in the grid. The path is then traversed (as shown by the arrows), counting up the computational load, and the grid is then segmented along sections of the curve so that equal loads can be distributed to each core (here load is assumed proportional to the number of agents, shown as red dots)

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 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 Madina (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, e.g. Springel (2005) – see Fig. 14.13. 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.

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 200 × 100 m domain runs 124 days on 32 cores in just 7 s. A much larger domain or a larger number of days would likely be required before load-balancing the code would provide a practical benefit.

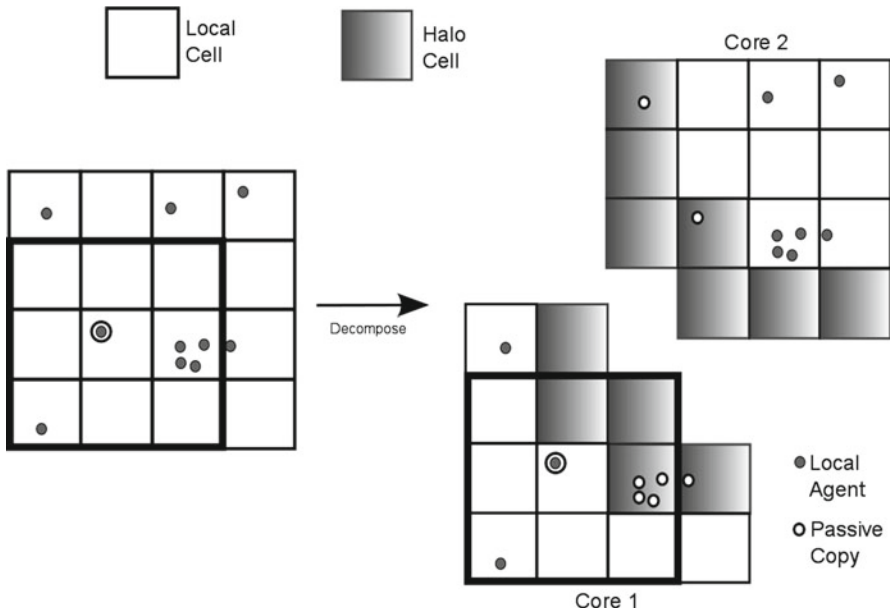

Fig. 14.14 Domain decomposition where agents interact with others outside their own local cell. The circled agent interacts with those in its own cell, but also those in the eight-member neighbourhood outlined by the blue square. On decomposition, part of this neighbourhood lies on a remote core. A halo 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 case, 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

(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 (Fig. 14.14).

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

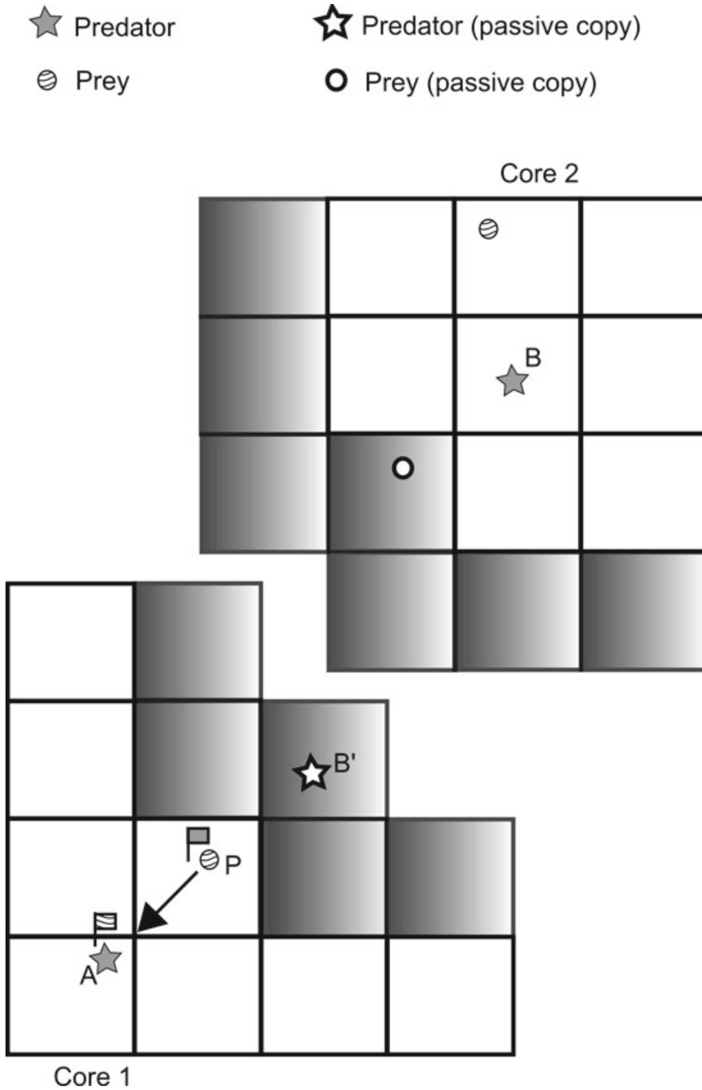

Fig. 14.15 Predator–prey interaction taking place across cores. *Prey P* can see both *predator A* and the passive copy *B'*. *Predator A*, however, only knows about *P*, and not about *B'*. Active *predator B* on core 2 can see two prey, one of which is the passive copy of *P*. Predators and prey need 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 indicated by the *arrow*

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 14.15 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.

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 Fig. 14.15, 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.

14.7 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). The aphid-only model parallelised well using the agent-parallel method as it lacked the complexity of the hoverfly interactions. This shows how parallelisation and super-individuals can both help deal with increasing numbers of agents.

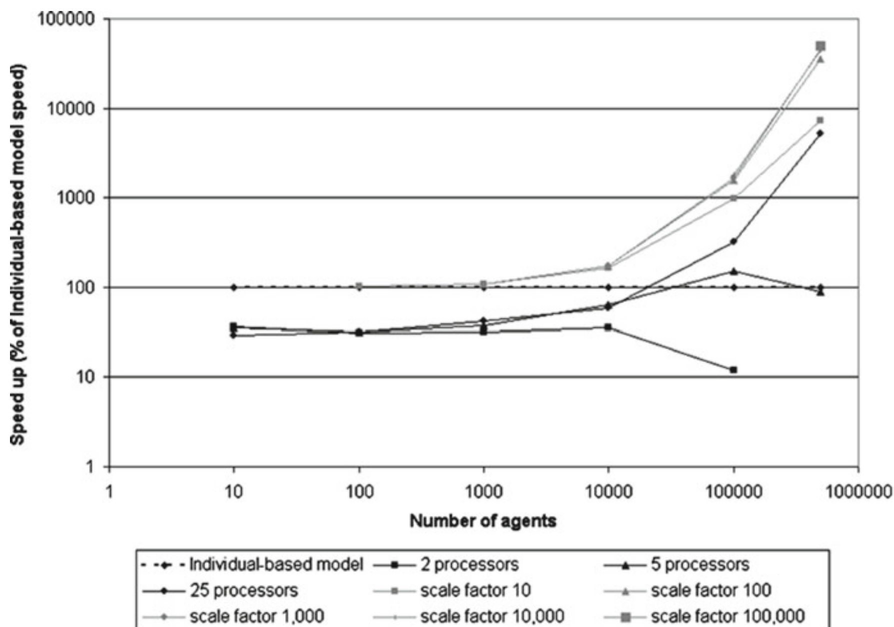

Fig. 14.16 Plot of the percentage speed up from the individual-based (non-parallel) model against number of agents modelled: comparison between super-individuals of scale factor 10, 100, 1,000, 10,000, 100,000 and 500,000

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.

14.7.1 Model Speed and Increasing Numbers of Agents

Super-individuals always improve the model speed with increasing numbers of agents (Fig. 14.16). This improvement is linear (shown here on a log-log scale). The speed improvement is enormous for the largest simulations: 500,000 individuals simulated with super-individuals using a scale factor of 100,000 increases the model speed by over 500 times. 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 100,000–500,000 individuals.

For the agent-parallel implementation, adding more processors does not necessarily increase the model speed. Figure 14.16 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 25 cores, where model speed does scale linearly as the number of individuals increases. However, the parallel model is slower than the serial code for fewer than about 30,000 individuals. 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 be efficient to parallelise the code unless the number of individuals is sufficiently large.

14.7.2 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, Fig. 14.17).

Super-individuals always reduce the memory requirements of the simulation (Fig. 14.17). 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-individual). 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. For simulations of 500,000 agents, this may be reduced to around 1% with the same scale factor. Thus, when large scale factors are used and as agent numbers increase, there is very little extra demand on memory.

The mean maximum memory usage by each worker core in the agent-parallel simulations is significantly lower than the non-parallel model, for simulations using more than two cores (Fig. 14.17). The relationship between the number of agents in the simulation and the memory used is linear for each number of cores. 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 visualization of the output 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 (after which the 25 core parallel simulation is more memory efficient).

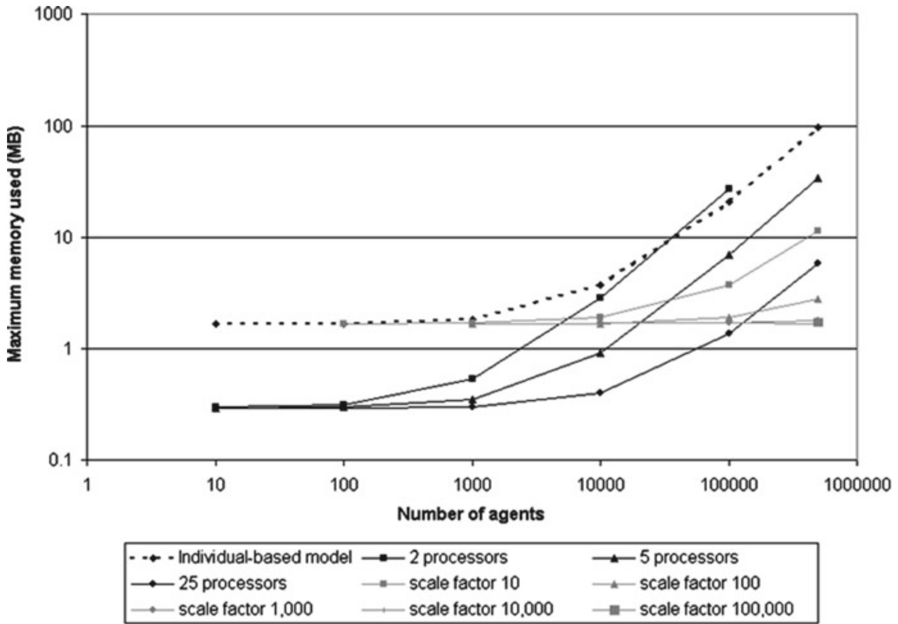

Fig. 14.17 Plot of the mean maximum memory used in a simulation run against number of agents for the model, for different scale factors for super-individuals

14.7.3 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 approximates a power law for up to 32 cores in comparison to the non-parallel serial model code run on a single processor (Fig. 14.18). At 64 processors the speed-up drops, probably due to the overhead required for each processor to run the model and the time taken for processors to communicate now exceeding the time take for the distributed model to run (at 32 processors the model takes less than 7 s to run) – if tested with a longer or larger (more agents) run of the model, 64 processors would perhaps continue to show increased efficiency as this would remove the effect of this overhead. In terms of memory, the parallel model uses much less memory per processor than the serial implementation, again approximately following a power-law decay up to 32 processors (Fig. 14.19). Overall, of the two parallel approaches, the environment-parallel version of the model, written in C++ instead of Java, proved more efficient and successful at handling parallel processing of complex agent interactions in this case study.

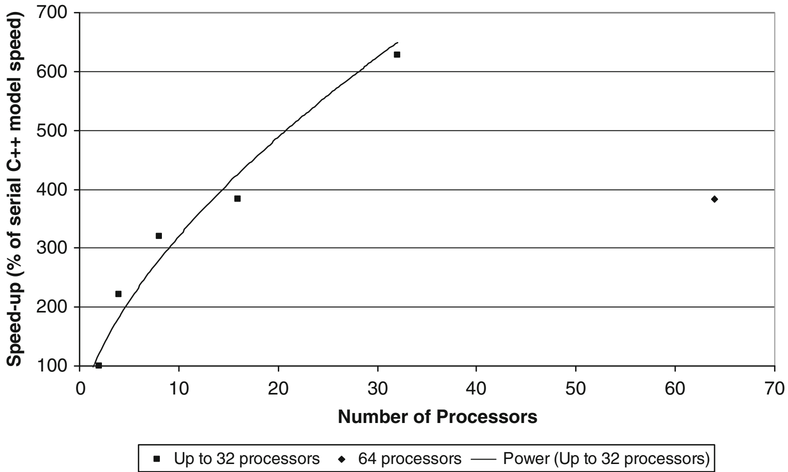

Fig. 14.18 Environment-parallel hoverfly-aphid model: percentage speed-up from the individual-based (non-parallel) model against number of processors. Under 32 processors, this approximates a power law relationship, as shown

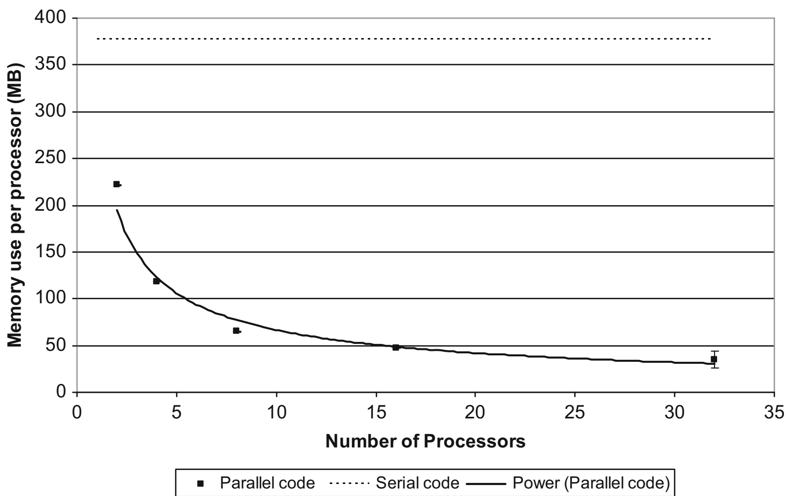

Fig. 14.19 Environment-parallel hoverfly-aphid model: Plot of the mean maximum memory used per processor in a simulation run against number of processors

14.8 Guidelines for Agent-Based Model Scaling

There is no standard method for the development of ABMs, although there are a number of agent modelling toolkits and recently some design protocols have arisen, e.g. Gilbert (2007), Grimm et al. (2006) and Grimm and Railsback (2012). Therefore,

as stated in Parry (2009), there is no standard method with which a large scale ABM 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 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 great 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 ABM and altering the model structure.

14.8.1 A Protocol

In relation to the key considerations highlighted above, a simple protocol for developing a large scale ABS 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 it on the full model.

The main scaling solution to implement (e.g. from Table 14.1) 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 ABMs 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 and the School of Geography at the University of Leeds, 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, and also to Dr Andrew Evans at the University of Leeds for assistance in the early stages of this work.

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 that 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 an adult (if male, it is then removed from the model).

Reproduction

In this model oviposition occurs once within a single 1 m^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 one egg is laid per day within the cell, and the egg is assumed to become a larva 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 1 m^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 m a day (given that a mark-recapture study of Holloway and McCaffery (1990) found hoverflies moved between 20–30 m 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 and 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 1 m^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):

$$MORT = (0.3119e^{0.0337(A\times 24)} \times D + (2.512e^{0.0253(A\times 24)}) \qquad (14.1)$$

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

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 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) and 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 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., Berry, M. W., Comiskey, E. J., Gross, L. J., & Luh, H.-K. (1997). Parallel individual-based modeling of Everglades deer ecology. *IEEE Computational Science and Engineering, 4*, 60–78.

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

Barlow, N. D., & Dixon, A. F. G. (1980). *Simulation of lime aphid population dynamics*. Wageningen: Centre for Agricultural Publishing and Documentation.

Barnes, D. J., & Hopkins, T. R. (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., & Macmillan, W. (2007). Escape from the cell: Spatial modelling with and without grids. *Ecological Modelling, 200*, 59–78.

Bokma, A., Slade, A., Kerridge, S., & Johnson, K. (1994). Engineering large-scale agent-based systems with consensus. *Robotics and Computer-Integrated Manufacturing, 11*, 81–91.

Bouzid, M., Chevrier, V., Vialle, S., & Charpillet, F. (2001). Parallel simulation of a stochastic agent/environment interaction model. *Integrated Computer-Aided Engineering, 8*, 189–203.

Castiglione, F., Bernaschi, M., & Succi, S. (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., Wille, L. T., Wu, Y. G., & Sklar, F. H. (2001). Parallelization of an ecological landscape model by functional decomposition. *Ecological Modelling, 144*, 13–20.

Da-Jun, T., Tang, F., Lee, T. A., Sarda, D., Krishnan, A., & Goryachev, A. (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., Wendel, S., & Carle, K. (2007). Simulating pandemic influenza risks of US cities. In *Proceedings of the 2007 winter simulation conference*, Vols 1–5 (pp. 1527–1529). New York: IEEE Press.

Dupuis, A., & Chopard, B. (2001). Parallel simulation of traffic in Geneva using cellular automata. In E. Kühn (Ed.), *Virtual shared memory for distributed architecture*. Commack: Nova Science Publishers, Inc.

Foster, I. (1995). *Designing and building parallel programs*. Reading: Addison-Wesley.

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

Gilbert, N. (2007). *Agent-based models*. London, UK: Sage.

Grimm, V., & Railsback, S. F. (2012). Designing, formulating and communicating agent-based models. In A. J. Heppenstall, A. T. Crooks, L. M. See, & M. Batty (Eds.), *Agent-based models of geographical systems*. Dordrecht: Springer. pp. 361–377.

Grimm, V., Berger, U., Bastiansen, F., Eliassen, S., Ginot, V., Giske, J., et al. (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. In D. L. DeAngelis & L. J. Gross (Eds.), *Individual-based models and approaches in ecology: Populations, communities and ecosystems* (pp. 126–164). New York: Routledge, Chapman and Hall.

Harmel, N., Almohamad, R., Fauconnier, M.-L., Jardin, P. D., Verheggen, F., Marlier, M., et al. (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., Dixon, A. F. G., Doucet, J.-L., & Petersen, J.-E. (1993). Optimal foraging by hoverflies (Diptera, Syrphidae) and ladybirds (Coleoptera: Coccinellidae): Mechanisms. *European Journal of Entomology, 90*, 451–455.

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

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

Immanuel, A., Berry, M. W., Gross, L. J., Palmer, M., & Wang, D. (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., Gasser, L., & Nakashima, H. (2005). Massively multi-agent systems I. First international workshop, in MMAS 2004. Heidelberg: Springer-Verlag.

Jamali, N., Scerri, P., & Suguwara, T. (Eds.) (2008). Massively multi-agent technology: AAMAS workshops, MMAS 2006, LSMAS 2006, and CCMMS 2007 Hakodate, May 9, 2006 Honolulu, May 15, 2007, Selected and Revised Papers, LNAI 5043, Heidelberg: Springer-Verlag.

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

Kareiva, P., & Odell, G. (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., & Dixon, A. F. G. (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., & Hwu, W. W. (2010). *Programming massively parallel processors: A hands-on approach*. Burlington: Morgan-Kaufmann.

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

Lorek, H., & Sonnenschein, M. (1995). Using parallel computers to simulate individual-oriented models in ecology: A case study. In *Proceedings, ESM '95 European Simulation Multiconference*, Prague, June 1995.

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

Lysenko, M., & D'Souza, R. M. (2008). A framework for megascale agent-based model simulations on graphics processing units. *Journal of Artificial Societies and Social Simulation*, *11*(4), 10. Available at: http://jasss.soc.surrey.ac.uk/11/4/10.html

Massaioli, F., Castiglione, F., & Bernaschi, M. (2005). OpenMP parallelization of agent-based models. *Parallel Computing, 31*, 1066–1081.

Mellott, L. E., Berry, M. W., Comiskey, E. J., & 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., & de Roos, A. M. (1992). The role of physiologically structured population models within a general individual based model perspective. In D. L. DeAngelis & L. J. Gross (Eds.), *Individual based models and approaches in ecology: Concepts and models* (pp. 88–111). New York: Routledge, Chapman and Hall.

Minson, R., & Theodoropoulos, G. K. (2008). Distributing RePast agent-based simulations with HLA. *Concurrency and Computation: Practice and Experience, 20*, 1225–1256.

Nagel, K., & Rickert, M. (2001). Parallel implementation of the TRANSIMS micro-simulation. *Parallel Computing, 27*, 1611–1639.

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

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

Pacheco, P. S. (1997). *Parallel programming with MPI*. San Francisco: Morgan Kauffman Publishers.

Parry, H. R. (2006). Effects of land management upon species population dynamics: A spatially explicit, individual-based model (Unpublished PhD thesis, University of Leeds, Leeds).

Parry, H. R. (2009). Agent based modeling, large scale simulations. In R. A. Meyers (Ed.), *Encyclopedia of complexity and systems science* (pp. 148–160). New York: Springer.

Parry, H. R., & Evans, A. J. (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., Evans, A. J., & Heppenstall, A. J. (2006a). Millions of agents: Parallel simulations with the Repast agent-based toolkit. In Trappl, R. (Ed.), Cybernetics and Systems 2006, *Proceedings of the 18th European Meeting on Cybernetics and Systems Research*. Vienna: Austrian Society for Cybernetic Studies.

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

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

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

Railsback, S. F., Lytinen, S. L., & Grimm, V. (2005). StupidModel and extensions: A template and teaching tool for agent-based modeling platforms. Available at: http://condor.depaul.edu/~slytinen/abm/StupidModelFormulation.pdf

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

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

Richmond, P., Coakley, S., & Romano, D. (2009a). A high performance agent-based modelling framework on graphics card hardware with CUDA. In K. Decker, J. Sichman, C. Sierra, and C. Castelfranchi (Eds.), *Proceeding of 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, 10–15 May 2009. Budapest

Richmond, P., Coakley, S., & Romano, D. (2009b). Cellular level agent-based modelling on the graphics processing unit. In *International Workshop on High Performance Computational Systems Biology HIBI'09*, 14–16 Oct 2009, Trento.

Scheffer, M., Baveco, J. M., DeAngelis, D. L., Rose, K. A., & van Nes, E. H. (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 GADGET-2. *Monthly Notices of the Royal Astronomical Society, 364*, 1105–1134.

Stage, A. R., Crookston, N. L., & Monserud, R. A. (1993). An aggregation algorithm for increasing the efficiency of population models. *Ecological Modelling, 68*, 257–271.

Standish, R. K., & Madina, D. (2008). Classdesc and graphcode: Support for scientific programming in C++, arXiv:cs.CE/0610120. Available from http://arxiv.org/abs/cs.CE/0610120

Stone, J. E., Phillips, J. C., Freddolino, P. L., Hardy, D. J., Trabuco, L. G., & Schulten, K. (2007). Accelerating molecular modelling applications with graphics processors. *Journal of Computational Chemistry, 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 Dec 2004. Heidelberg: Springer-Verlag.

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, T. Jakeman (Eds.), *iEMSs 2004 International Congress: Complexity and Integrated Resources Management*. Osnabrueck: International Environmental Modelling and Software Society.

Timm, I. J., & Pawlaszczyk, D. (2005). Large scale multiagent simulation on the grid. In *Proceedings of the Workshop on Agent-based Grid Economics (AGE 2005) at the IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*. Cardiff: Cardiff University

Wang, D., Gross, L., Carr, E., & Berry, M. (2004). Design and implementation of a parallel fish model for South Florida. In *Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS '04)*, 5–8 Jan 2004, Big Island: IEEE Computer Society

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

Wang, D., Berry, M. W., Carr, E. A., & Gross, L. J. (2006). A parallel fish landscape model for ecosystem modeling. *Simulation, 82*, 451–465.

Wang, D., Berry, M. W., & Gross, L. J. (2006). On parallelization of a spatially-explicit structured ecological model for integrated ecosystem simulation. *International Journal of High Performance Computing Applications, 20*, 571–581.

Wendel, S., & Dibble, C. (2007). Dynamic agent compression. *Journal of Artificial Societies and Social Simulation*, *10*(2), 9. Available at: http://jasss.soc.surrey.ac.uk/10/2/9.html

Wilkinson, B., & Allen, M. (2004). *Parallel programming: Techniques and applications using networked workstations and parallel computers* (2nd ed.). New Jersey: Pearson Prentice Hall.

Woods, J. D. (2005). The Lagrangian Ensemble metamodel for simulating plankton ecosystems. *Progress in Oceanography, 67*, 84–159.

Woods, J., & Barkmann, W. (1994). Simulating plankton ecosystems by the Lagrangian ensemble method. *Philosophical Transactions of the Royal Society of London Series B-Biological Sciences, 343*, 27–31.

Wu, Y. G., Sklar, F. H., Gopu, K., & Rutchey, K. (1996). Fire simulations in the Everglades Landscape using parallel programming. *Ecological Modelling, 93*, 113–124.