

Graph-Based Automatic Dynamic Load Balancing for HPC Agent-Based Simulations

Claudio Márquez^(✉), Eduardo César, and Joan Sorribes

Computer Architecture and Operating Systems Department (CAOS),
Universitat Autònoma de Barcelona, 08193 Bellaterra, Spain
claudio.marquez@caos.uab.es, {eduardo.cesar, joan.sorribes}@uab.cat
<http://caos.uab.es/>

Abstract. The main problem of Agent-Based Modelling (ABM) simulations in High Performance Computing (HPC) is load imbalance due to a non-uniform distribution of the agents that may generate uneven computation and increase communication overhead, inhibiting the efficiency of the available computing resources. Moreover, the agents' behaviours can considerably modify the workload at each simulation step thereby affecting the workload progression of the simulation. In order to mitigate such problems, automatic mechanisms for dynamically adjusting the computation and/or communication workload are needed. For this reason, we introduce an Automatic Dynamic Load Balancing (ADLB) strategy to reduce imbalance problems as the simulation proceeds. The ADLB tunes the global simulation workload migrating groups of agents among the processes according to their computation workload and their message connectivity map modelled using a Hypergraph. This Hypergraph is partitioned using the Zoltan Parallel HyperGraph partitioner method (PHG). In addition, to prevent excessive all-to-all communications, the ADLB uses filtering routines to send message groups to specified recipient processes in a simple 3D grid-based structure. Our method has been tested with a biological ABM using the framework Flexible Large-scale Agent Modelling Environment (Flame), obtaining a significant impact on the application performance.

Keywords: Agent-based simulation · Graph partitioning · Message filtering · Load balancing · Performance tuning · HPC · SPMD · Flame

1 Introduction

Agent-Based Modelling (ABM) describes the system's behaviour through the interaction of a set of autonomous entities called agents in a virtual environment. The agents' behaviour is decided by sets of agent's internal operations

C. Márquez, E. César and J. Sorribes—This work has been partially supported by MICINN-Spain under contract TIN2011- 28689-C02-01 and TIN2014-53234-C2-1-R and GenCat-DIUe(GRR) 2014-SGR-576. This research was also partially performed by a collaboration agreement with the Department of Physiology and Medical Physics at the Royal College of Surgeons in Ireland (RCSI), under the project ANGIOPREDICT: European Union FP7-funded (<http://www.angiopredict.com>).

and information data exchange between agents. Depending on the complexity order of these operations and exchanges, the workload of the ABM simulation may progress irregularly, producing significant variations on the workload of the entire system. All these issues negatively affect the simulation time and, in most cases, they inhibit the efficiency of the computing resources available. Consequently, when a large number of agents with complex interaction rules are simulated in a High Performance Computing (HPC) infrastructure, to be aware of these issues becomes essential.

In HPC ABM simulations, a non-uniform distribution of the agents' workload may introduce uneven CPU computing and network communication workload that delay the simulation and may propagate across all processes. This load-imbalance problem is well known in Single Process Multiple Data (SPMD) programming paradigm, which consists of a unique code replicated in all processes running on different data sets. These SPMD applications are comprised by sets of computation and synchronisation phases. In time-driven cases, these phases are repeated in each simulation step. In the same way, the communication phases of the simulation become expensive as the number of processes increases. Besides, the uneven CPU workload will also impact negatively the completion time of the simulation steps. With the purpose of mitigate such problems, automatic mechanisms for dynamically adjusting the computation and/or communication load are needed.

According to the execution of the tuning decisions, the load balancing strategies for HPC applications can be developed using centralised/hierarchical and decentralised approaches [11]. The centralised/hierarchical approaches report a high computational cost and scalability problems. On the other hand, decentralised approaches can present problems regarding the quality balance because the neighbouring processes exchange incomplete information. Whilst many load-balancing solutions can be found such as: [12–15]; these have rarely been incorporated to a multi-purpose environment. For this reason, we have been progressively developing a load balancing strategy for multi-purpose environments of ABM simulations as can be seen in [6–8].

In this paper, we present a Graph-based Automatic Dynamic Load Balancing (ADLB) strategy that allows automatic and dynamic tuning decisions in terms of computation/communication workload. The ADLB tunes the global simulation workload migrating groups of agents among the processes using a Hypergraph perspective. This Hypergraph is partitioned using the Zoltan Parallel Hyper-Graph partitioner method (PHG) [4]. In order to achieve a suitable management of the agent communications, the ADLB uses message filtering routines for sending message groups to specified recipient processes in a simple 3D grid-based structure. Our method has been tested with a biological ABM using the framework Flexible Large-scale Agent Modelling Environment (Flame), obtaining a significant impact on the application performance.

This paper is arranged in five sections. First, Sect. 2 presents the algorithm for grouping agents in a 3D grid structure and the mechanism for filtering messages based in this structure. Next, the Graph-based partitioning method is explained,

and then the ADLB operation is described. The results section presents the ADLB operation for a cancer developing ABM. The final section includes the conclusions and future work.

2 3D Spatial Agent Organisation

In order to reduce the amount of agent messages in the system, a mechanism for avoiding broadcasting is needed. However, such mechanism can not rely in storing the location of each agent involved in a communication because storing this information becomes unmanageable when a large number of agents with complex interaction rules is simulated. An intermediate solution consists of grouping agents using for example a 3D grid-based structure that organises the agents' locations using a virtual division of the space to ease the message management by the simulation platform.

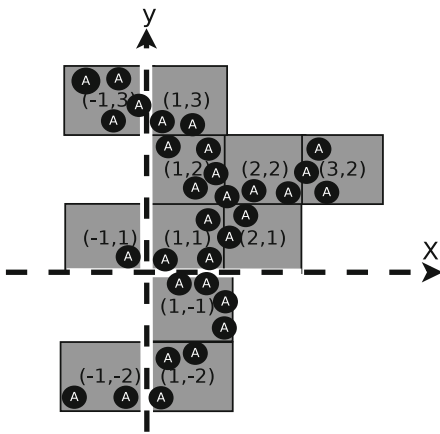


Fig. 1. Cubes covering agents' space.

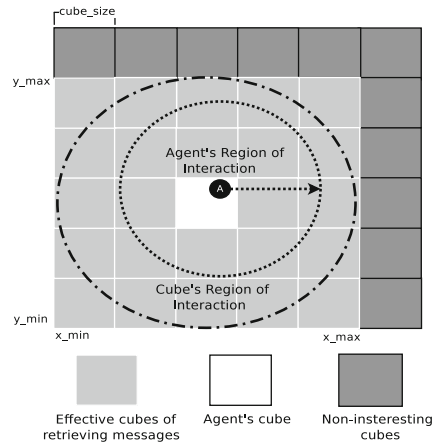


Fig. 2. Interaction ranges.

The 3D grid-based structure divides the whole space in 3D cubic regions and builds them using a *rasterisation* approach where the objects are characterised from its shape within a 2D/3D space [10]. In the same way, the cubes of our 3D structure are only created according to the space occupied by agents. Similar approaches have been designed in 2D particle simulations [2]. Thanks to this approach, it is possible to model an indefinitely large domain. Following Fig. 2, the *cube_size* defines cubic regions of dimension *cube_size* x *cube_size* x *cube_size*. The value of *cube_size* should be estimated in accordance with the *influence range* of the agents. This influence range is usually named *halo* in the literature [9], and it is explained later.

As **Grid Construction Algorithm**, each cube is built on the fly by a function that maps each agent coordinates to its container cube (algorithm 1 shows this operation, $\lceil \text{coordinate}/\text{cube_size} \rceil$). As a result, each cube identifier is composed of three integers that are a combination of the x -, y - and z -axis ids.

In this manner, every agent will be assigned to only one cube, and all cubes will have a positive agent counter. These cubes will be constructed along with the exploration of the existing agents across processes. The space covered by cubes is named *known space*, so new cubes will appear when an agent is located outside the known space. If an agent is located within the known space, the agent counter belonging to its agent's cube region is increased. Finally, only the agent's regions will be covered by multiple 3D cubes (Fig. 1 depicts this principle in a 2D space).

Algorithm 1. Grid Construction

```

c.size ← cube length
c.groupi ← cubes in parallel_processi
for each agent ∈ parallel_processi do
  xyz ← xyz-coordinate of agent
  cidx ←  $\text{ceil}(x / c\_size)$ 
  cidy ←  $\text{ceil}(y / c\_size)$ 
  cidz ←  $\text{ceil}(z / c\_size)$ 
  agent_cube ← {cidx, cidy, cidz}
  if agent_cube ∈ c.groupi then
    ++agent_counter of agent_cube
  else
    add agent_cube to c.groupi
  end if
end for

```

Algorithm 2. Cube Interaction

```

c.range ←  $\text{ceil}(\text{agent\_range}/c\_size)$ 
global_group ← cubes in all processes
for each agent ∈ parallel_processi do
  agent_cube ← {cidx, cidy, cidz}
  for each cube ∈ global_group do
    cx ← x-component of cube
    cy ← y-component of cube
    cz ← z-component of cube
    if agent_cube ∈ [cxyz ± c.range] then
      cube ∈ the interaction region
    end if
  end for
end for

```

Algorithm 2 shows the **Cube Interaction Algorithm**. This algorithm simplifies the access to the information of relations among the agents. Likewise, the recipient cubes of an agent message can be determined using its *halo*. Additionally, this algorithm helps to distinguish whether the recipients of an agent message are located in a cube belonging to another process or not; hence the required external network communications can be predicted. The global view of the cubes is defined gathering the cube information from every process. This global cube information contains the cube's *ternary ids* and the number of agents within each cube (the latter just for load balancing purposes). In the same way, the agent's *message connectivity map* can be built using the Cube Interaction Algorithm. The estimation of the interaction regions is performed through an Euclidean distance calculation (Algorithm 2 shows this operation). In Algorithm 2 the cube's *halo* (interval $[c_{xyz} - c_range, c_{xyz} + c_range]$) is obtained by dividing the agent interaction range by *cube_size* value. As explained in [8], the *halo* defines the range of the agents' messages, and hence, using this *halo*, the agents' messages can be filtered in order to avoid broadcast communications.

3 Graph-Based Agent Partitioning

The graph partitioning in HPC applications allows to represent the connectivity of different computational structures to decompose the computational domains for parallelisation. The goal is to divide the graph into equal sized sub-graphs while minimising the number of edges between different sub-graphs. In our case, these sub-graphs could represent a sub-domain of computing workload while the edges between them could represent the required communication network that need to be balanced by the ADLB. However, the communication representation offered by graph edges does not accurately represent the actual interaction of the agents as we need because agent messages may have more than one recipient. For this reason, the hypergraph partitioning approach fits better to agent interactions than graph approaches.

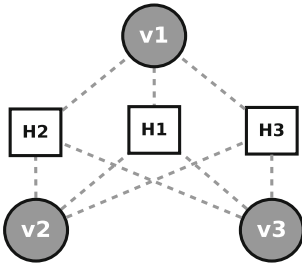


Fig. 3. Hyperedges representation.

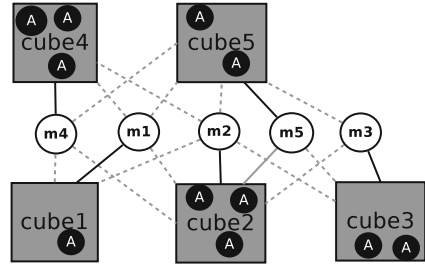


Fig. 4. Communication of cubes.

The hypergraphs contain hyperedges which connect two or more vertices as well as the agent communications having more than one recipient. Figure 3 shows an example of hyperedges denoted as H1-H2-H3 and their respective vertices are denoted as v1-v2-v3. On the other hand, hypergraph partitioning is computationally more expensive than graph partitioning, which is already NP-hard [5]. In order to reduce the excessive memory/communication requirements, we have considered the cubes as vertices and their communication connectivity as hyperedges. Additionally, the number of agents and the volume of messages per cube are added as vertices and hyperedges' weights respectively (Fig. 4 depicts this approach).

Having a proper representation of groups of agent interactions, it will help the ADLB not only to reduce the workload requirements, but also to have a proper spatial representation of the simulation as long as its workload develops variations. In ADLB, the hypergraph partitioning method has been included as the load balancing criterion.

4 Automatic Dynamic Load Balancing

As we mentioned before, agent-based applications workload can vary during the simulation due to issues related to the complexity of the model, and interaction patterns. The ADLB decides the global reconfiguration of the workload when the performance measures indicate imbalances according to an imbalance threshold value. The threshold is a value between 0.0 and 1.0 that represents the acceptable percentage of imbalance over/under the mean. Computing times and number of agents are monitored at each parallel process. This monitoring is repeated at the beginning of each iteration, then the measures are shared among the processes. Hence, each process knows the global workload situation and executes the algorithm with the same input. Consequently, all processes calculate the same reconfiguration of the workload without a central decision unit. The components of ADLB are described below.

The **automatic monitoring** component launches the load balancing procedure when the imbalance factor exceeds the given threshold. The monitoring is executed by all processes; hence each process needs to know the global load situation, thereby the processes workload measures are broadcasted along with the simulation synchronisation at the end of each iteration. In these cases, the computing time must be determined using the previous iteration results and the current number of agents because the current computing time is obtained at the end of the iteration. The predicted current computing time is described in Eq. 1. This information is broadcasted over all processes to have the global workload information and assess the imbalance degree in the activation mechanism.

$$comp_time_{iter} = \frac{comp_time_{iter-1} * num_agents_{iter}}{num_agents_{iter-1}} \quad (1)$$

In the **activation mechanism**, the process imbalance factors are calculated using the broadcasted current computing time. The imbalance factor represents the degree of imbalance according to the computing time mean. The tolerance establishes the range considered as balanced. Furthermore, depending on the tolerance range, workload imbalances can be detected (Eqs. 2 and 3 show the imbalance factor, the tolerance and the tolerance range). Consequently, as every process executes this analysis with the same inputs, the reconfiguration procedure is triggered across processes when an excessive imbalance appears.

$$ib_factor_i = \frac{comp_time_i}{avg_time} \quad (2) \quad \begin{aligned} tolerance &= avg_time * threshold \\ tolerance_range &= avg_time \pm tolerance \end{aligned} \quad (3)$$

In the **reconfiguration procedure**, ADLB decides the amount of agents and the place where the agents need to be reallocated. The reallocation is performed according to the Zoltan PHG decision [4], but at this point, the global cube connectivity map needs to be transformed into a hypergraph represented as a sparse matrix. Moreover, each hypergraph vertex id must have a unique global identifier represented as an unsigned integer. Due to the vertices depict spatial

cubes which, in turn, are set of agents, ADLB must be sure that the global cube ids are unique and each parallel process has unique cubes. To accomplish this, agents belonging to external cubes need to be migrated before, then sharing the cube’s location and the computing workload per cube in order to define unique global cube ids and vertex weights. Later, each parallel process stores its global cube connectivity map into a CSR sparse matrix format (CSC format is also allowed by PHG). Then, PHG performs the parallel hypergraph partitioning and returns the vertex ids that should be imported and exported. In the same way, the degree of hypergraph cutting accuracy could be set up using PHG configuration parameters. Finally, in accordance with the PHG results, ADLB will introduce *send* and *receive* agent requests across the parallel processes. After the agent migration occurs, the platform should take back the control of the simulation.

5 Experimental Results

In this section, we are focused on analysing the preliminary performance results of ADLB in a real parallel SPMD ABM platform. The experiments have been performed using the Flame platform with a biological ABM (this model is explained further on).

Flame [3] is a C/MPI-based parallel code generator that allows to run simulations on large HPC systems from an XML/C model definition (Fig. 5 shows the functional diagram). The Flame engine (xparser) parses the model definition and generates parallel code deployed under a SPMD paradigm, thereby, implies a unique code replicated among all the processes that performs a set of computing and communication phases. In Flame, the interaction between agents is handled by the message board library libmboard, which provides message memory management and message data synchronisation routines using MPI. Therefore, the agents only interact sending messages to the board library, so all messages have to be stored in a board before starting the board synchronisation

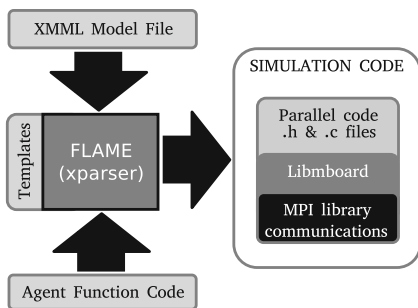


Fig. 5. Functional description of Flame.

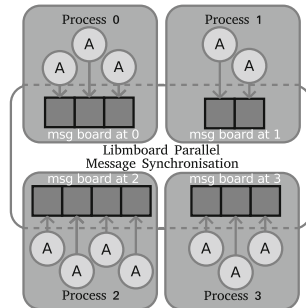


Fig. 6. Flame communication model

(see Fig. 6). Hence, libmboard organises the messages before being dispatched in such a way that the communication between processes is overlapped with computation via non-blocking MPI communications. Once the synchronisation has been completed, every parallel process will maintain a unified view of the message board. Finally, the agents should read their messages of interest from the synchronised board. However, Flame does not provide routines intended to reconfigure the computing and communication workload during the execution of the simulation. For this reason, the necessary modules to solve these requirements such as agent migration, message filtering, agent connectivity mapping and performance measurement management have been implemented in Flame. Once these functionalities are implemented, ADLB is capable of reconfiguring the global workload in accordance with the simulation workload measures.

The use case is an agent-based model that represents an expansive in-vivo tumour development behaviour [1]. This model is composed of tumour cells and tumour-associated endothelial cells which are implemented as agents. In each iteration, the agents interact computing the expansion coordinates of new cells, forces among the cells, amount of nutrients that comes from the vessels, amount of oxygen permeated through the cells as well as new coordinates using the resulting forces. Later, these procedures will determine either the growth or the death of each tumour or endothelial cell. Consequently, this model presents computing/communication workload imbalances as the simulation proceeds. Figure 7 shows a graphical representation of the model where the grey and black spheres represent tumour cell and endothelial cell agents respectively.

In terms of the initial partitioning methods, geometric and round-robin strategies are provided by Flame (FlameGeo and FlameRR respectively). The geometric approach divides the space into non-overlapping orthogonal regions depending on the spatial space simulation dimensions assigning agents according their spatial coordinates. And the round-robin approach randomly assigns agents across all processes in a way that each process stores approximately an even number of agents. Compared with the Zoltan PHG approach, the Flame partitioning methods offer worse partition quality as has been shown in [8]. Therefore, we have opted for implementing the Zoltan Hypergraph-based initial partitioning in order to ensure a better partitioning quality from the beginning

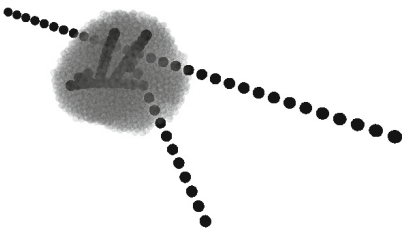


Fig. 7. Tumour development model.

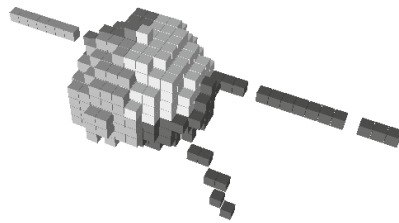


Fig. 8. PHG grid distribution.

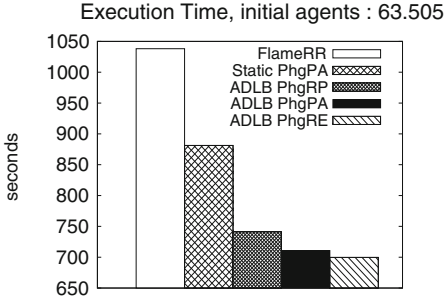


Fig. 9. Execution times.

Table 1. Flame times vs. ADLB times.

Approach	LB (sec)	Total (sec)
FlameGeo	-	26218.9
FlameRR	-	1038.1
Static PhgPA	-	881.3
ADLB PhgRP	83.3	741.3
ADLB PhgPA	63.4	710.7
ADLB PhgRE	56.9	699.7

of the simulation. In this way, we can assess the ADLB performance impacts starting from a better distribution of the agents and better message filtering.

Within the Zoltan PHG setup parameters, the hypergraph approach can be configured as *partition*, *repartition*, or *refine*. In the partition mode (PhgPA) the current vertices distribution is not taken into account (a partition from scratch). In the repartition mode (PhgRP) the current vertices distribution is considered to cut the hypergraph. Finally, the refine mode (PhgRE) refines the given distribution minimising the number of changes. Figure 8 shows an example of PhgPA partitioning over the resulting 3D grid of the tumour model. In addition, the PHG cut accuracy has been set up to 20% of imbalance deemed acceptable in order to reduce the required time to find an appropriate graph cut, and all the initial graph partitions have been performed using the PhgPA mode.

The following results correspond to 20 iterations of the tumour development model executed using 128 processors. The results were obtained using Flame 0.17.0, libmboard 0.3.1 and OpenMPI 1.6.4. The experiments were executed on an IBM Cluster with the following features: 32 IBM x3550 Nodes, 2xDual-Core Intel(R) Xeon(R) CPU 5160 @ 3.00 GHz 4MB L2 (2 × 2), 12 GB Fully Buffered DIMM 667 MHz, Hot-swap SAS Controller 160GB SATA Disk and Integrated dual Gigabit Ethernet. In these experiments, 63.505 cells are simulated and their interaction range is 25 microns. The *cube_size* is defined as 50 microns in order to reduce the number of vertices computed by PHG.

Figure 9 shows the execution times of the ADLB versions by comparing different PHG options with two static approaches (the FlameRR default option and initial PhgPA partitioning with message filtering). Here, the ADLB versions obtain better results than the static approaches FlameRR and static PhgPA. FlameGeo has been excluded from this figure because its execution time excessively surpasses the other times, however, its execution time is contained in Table 1. As FlameGeo parts the space into orthogonal rectangles, it creates uneven or empty partitions according to the agents' spatial locations. In the same way, the FlameRR randomly distributes the agents generating a similar number of agents per process. The PHG versions gain more than 30% over the FlameRR experiment in terms of execution time, even an initial PhgPA

partitioning improves the Flame times using messages filtering. Basically, the hypergraph partitioning methods in ADLB shows similar results and the main difference relies on the total PHG overhead time as shown in the second column of Table 1.

Table 2. Details of the ADLB options.

Hyper-graph method	Number ADLB calls	Average vertices per call	% Total messages internal/external	Cubes map construction (sec)	Zoltan time (sec)	Migration time (sec)	Total time (sec)
PhgRP	10	3746	95.95/4.05	10.3	70.1	2.9	83.3
PhgPA	10	3743	96.57/3.43	7.3	54.0	2.1	63.4
PhgRE	9	3745	96.53/3.47	8.8	45.1	3.0	56.9

Table 2 shows the ADLB development, its communication workload and its overhead composition summary. For all versions, the number of ADLB calls and average number of vertices is similar. The main difference is rooted in the Zoltan PHG time required for cutting the hypergraph. Furthermore, repartitioning the current partition (PhgRP) is expensive compared with partitioning from scratch (PhgPA), and refining the hypergraph (PhgRE) is the best approach for these experiments. Even so, ADLB reduces simulation time as well as the number of communications among the parallel processes. Additionally, Table 2 also shows the impact of the message filtering over the communication workload. As a matter of fact, the percentage of the amount of messages that are held in the sender process is higher than the messages dispatched to external parallel processes (internal/external respectively). In the same manner, dispatching a small amount of messages also impacts directly the performance of the recipient processes because these have to examine a smaller amount of messages and, later on, determine its significance. As a result, the ADLB strategy enhances the performance of the parallel simulation using agent migration, message filtering, agent connectivity mapping and performance measures monitoring.

6 Conclusion and Future Work

When the ABMs present irregular computing/communication patterns due to the complexity and the large number of agents, the HPC ABM simulation platforms need to be able to reallocate workload as the simulation proceeds. This is a difficult task because the platform should implement features such as agent migration, message filtering, agent connectivity mapping and performance measures monitoring. Moreover, the platform should decide the appropriate moment to launch the load balancing mechanism and decide the amount of workload to be reallocated. Therefore, to solve this need, we present the Automatic Dynamic

Load Balancing (ADLB) that implements the issues mentioned previously in order to improve the performance of the HPC ABM simulation platform.

In this paper, the ADLB is proven in a model with workload variability due to agent creation and elimination. Additionally, the ADLB decision to reconfigure the workload is tested using three methods of hypergraph partitioning. For these cases, our schema obtains good results reducing the simulation execution time up to 30%. All in all, our approach gives better performance than the standard Flame partitioning methods, and our results confirm the importance of introducing the components of ADLB presented in this paper. The overhead of ADLB is mostly given by the execution of the Zoltan PHG method. In spite of that, its benefits directly impact the efficiency of the message filtering mechanism and the workload reconfiguration. Also, the overhead results suggest that reducing the number of vertices by increasing the *cube_size* could reduce the Zoltan overhead.

As future work, we want to find a proper cube size that minimises the Zoltan PHG overhead and keeps good performance filtering messages. In the same way, it is planned to test ADLB using other ABMs.

References

1. Angiopredict: Predictive genomic biomarkers methods for combination bevacizumab (avastin) therapy in metastatic colorectal cancer ANGIOPREDICT. EU's Framework Programme Seven (FP7) under contract 306021 (2014). <http://www.angiopredict.com/>
2. Bithell, M., Macmillan, W.: Escape from the cell: spatially explicit modelling with and without grids. *Ecol. Model.* **200**(1–2), 59–78 (2007)
3. Coakley, S., Gheorghe, M., Holcombe, M., Chin, S., Worth, D., Greenough, C.: Exploitation of high performance computing in the flame agent-based simulation framework. In: 2012 IEEE 9th International Conference on High Performance Computing and Communication, 2012 IEEE 14th International Conference on Embedded Software and Systems (HPCC-ICCESS), pp. 538–545, June 2012
4. Devine, K., Boman, E., Heaphy, R., Bisseling, R., Catalyurek, U.: Parallel hypergraph partitioning for scientific computing. In: 20th International Parallel and Distributed Processing Symposium, IPDPS 2006, p. 10, April 2006
5. Lengauer, T.: *Combinatorial Algorithms for Integrated Circuit Layout*. Wiley, New York (1990)
6. Márquez, C., César, E., Sorribes, J.: A load balancing schema for agent-based SPMD applications. In: *Parallel and Distributed Processing Techniques and Applications* (2013)
7. Márquez, C., César, E., Sorribes, J.: Agent migration in HPC systems using FLAME. In: an Mey, D., Alexander, M., Bientinesi, P., Cannataro, M., Clauss, C., Costan, A., Kecskemeti, G., Morin, C., Ricci, L., Sahuquillo, J., Schulz, M., Scarano, V., Scott, S.L., Weidendorfer, J. (eds.) *Euro-Par 2013. LNCS*, vol. 8374, pp. 523–532. Springer, Heidelberg (2014)
8. Márquez, C., César, E., Sorribes, J.: Impact of message filtering on HPC agent-based simulations. In: *Proceedings of 28th European Simulation and Modelling Conference, ESM 2014*, pp. 65–72. EUROSIS (2014)

9. Parry, H.R., Bithell, M.: Large scale agent-based modelling: a review and guidelines for model scaling. In: Heppenstall, A.J., Crooks, A.T., See, L.M., Batty, M. (eds.) *Agent-Based Models of Geographical Systems*, pp. 271–308. Springer, Dordrecht (2012)
10. Petkovic, T., Loncaric, S.: Supercover plane rasterization - a rasterization algorithm for generating supercover plane inside a cube. In: *GRAPP (GM/R 2007)*, pp. 327–332 (2007)
11. Plastino, A., Ribeiro, C.C., Rodriguez, N.: Developing spmd applications with load balancing. *Parallel Comput.* **29**(6), 743–766 (2003). [http://dx.doi.org/10.1016/S0167-8191\(03\)00060-7](http://dx.doi.org/10.1016/S0167-8191(03)00060-7)
12. Solar, R., Suppi, R., Luque, E.: Proximity load balancing for distributed cluster-based individual-oriented fish school simulations. *Procedia Comput. Sci.* **9**, 328–337 (2012). *Proceedings of the International Conference on Computational Science, ICCS 2012*
13. Viguera, G., Lozano, M., Orduña, J.M.: Workload balancing in distributed crowd simulations: the partitioning method. *J. Supercomput.* **58**(2), 261–269 (2011)
14. Zhang, D., Jiang, C., Li, S.: A fast adaptive load balancing method for parallel particle-based simulations. *Simul. Model. Pract. Theor.* **17**(6), 1032–1042 (2009)
15. Zheng, G., Meneses, E., Bhatele, A., Kale, L.V.: Hierarchical load balancing for Charm++ applications on large supercomputers. In: *Proceedings of the 2010 39th International Conference on Parallel Processing Workshops, ICPPW 2010*, pp. 436–444. IEEE Computer Society, Washington, DC (2010)