Accepted Manuscript

Colorectal tumour simulation using agent based modelling and high performance computing

Guiyeom Kang, Claudio Márquez, Ana Barat, Annette T. Byrne, Jochen H.M. Prehn, Joan Sorribes, Eduardo César

PII: S0167-739X(16)30072-3
DOI: http://dx.doi.org/10.1016/j.future.2016.03.026
Reference: FUTURE 3002

To appear in: *Future Generation Computer Systems*

Received date: 31 July 2015
Revised date: 15 December 2015
Accepted date: 28 March 2016

Please cite this article as: G. Kang, C. Márquez, A. Barat, A.T. Byrne, J.H.M. Prehn, J. Sorribes, E. César, Colorectal tumour simulation using agent based modelling and high performance computing, *Future Generation Computer Systems* (2016), http://dx.doi.org/10.1016/j.future.2016.03.026

Highlights (for review)

Colorectal Tumour Simulation using Agent Based Modelling and High Performance Computing

- Colorectal tumour modelling and simulation
- Parallel agent based modelling and simulation (ABMS)
- Load balancing of parallel ABMS using graph partitioning
- Extending FLAME with agent migration, output message filtering and dynamic load balancing

Colorectal Tumour Simulation using Agent Based Modelling and High Performance Computing

Guiyeom Kang[a], Claudio Márquez[b], Ana Barat[a], Annette T. Byrne[a], Jochen H.M. Prehn[a], Joan Sorribes[c], Eduardo César[c,*]

[a] *Centre for Systems Medicine and Physiology & Medical Physics Department, Royal College of Surgeons in Ireland, Dublin 2, Ireland*
[b] *Computer Applications in Science & Engineering, Barcelona Supercomputing Center, Barcelona, Spain*
[c] *Computer Architecture and Operating Systems Department, Universitat Autnoma de Barcelona, Barcelona, Spain*

Abstract

450,000 European citizens are diagnosed every year with colorectal cancer (CRC) and more than 230,000 succumb to the disease annually. For this reason, significant resources are dedicated to the identification of more effective therapies for this disease. However, classical assessment techniques for these treatments are slow and costly. Consequently, systems biology researchers at the Royal College of Surgeons in Ireland (RCSI) are developing computational agent-based models simulating tumour growth and treatment responses with the objective of speeding up the therapeutic development process while, at the same time, producing a tool for adapting treatments to patient-specific characteristics. However, the model complexity and the high number of agents to be simulated require a thorough optimisation of the process in order to execute realistic simulations of tumour growth on currently available platforms. We propose to apply the most advanced HPC techniques to achieve the efficient and realistic simulation of a virtual tissue model that mimics tumour growth or regression in space and time. These techniques combine extensions of the previously developed agent-

*Corresponding author
Email addresses: guiyeomkang@rcsi.ie (Guiyeom Kang), claudio.marquez@bsc.es (Claudio Márquez), anabarat@rcsi.ie (Ana Barat), annettebyrne@rcsi.ie (Annette T. Byrne), JPrehn@rcsi.ie (Jochen H.M. Prehn), Joan.Sorribes@uab.cat (Joan Sorribes), Eduardo.Cesar@uab.cat (Eduardo César)

based simulation software platform (FLAME) with autotuning capabilities and optimisation strategies for the current tumour model. Development of such a platform could advance the development of novel therapeutic approaches for the treatment of CRC which can also be applied to other solid tumours.

Keywords: Colorectal Cancer (CRC), Agent Based Modelling and Simulation (ABMS), High Performance Computing (HPC), Load Balancing

1. Introduction

Colorectal cancer (CRC) is common cancer in both males and females and is the third leading cause of cancer related mortality in both sexes [1]. It is a complex cellular disease [2] caused by sequential genetic mutations which trigger abnormal tumour cell proliferation rates and changes in metabolism, immunogenicity and cell death susceptibility. Moreover, tumour angiogenesis, i.e., the formation of new blood vessels, as well as metastasis leads to extensive disease dissemination. While a large amount of tumour related data is available at various temporal and spatial scales, data are not always able to explain the underlying diseade mechanisms, predict patient outcome, or identify patients who could benefit from targeted anti-cancer therapies. With the complex, multiscale process of tumour growth and dissemination continuously being investigated, computational modelling plays a powerful tool in helping to understand complex tumour biology mechanisms and responses to therapy.

Computational modelling of tumour growth can be implemented at different levels, i.e., molecular, microscopic, or macroscopic scale [3]. In recent studies it has been shown that computational modelling at the protein pathway and network scale may provide insights into cell processes relevant for responses to cancer therapy such as cell death activation, and may be used in the clinic to predict responses to chemotherapy [4, 5, 6]. Although these models advance our understanding of biological pathways and enable predictions, there are still many areas that require further exploitation through computational modelling.

Multiscale computational models incorporating hallmarks of cancer at differ-

ent scales in both space and time may provide a better understanding of tumour growth and may more accurately predict responses to therapy or suggest novel mechanistic hypotheses. Multiscale modelling has already been widely used for the study of tumour growth and response to therapy, and is though to provide more realistic tumour growth models [7, 8, 9].

However, multiscale modelling faces significant challenges if such models shall accurately predict tumour growth and therapeutic outcome. Since multiscale models include more biological processes than single scale models containing an element of multiscale models, more parameters and interactions are required that correspond to molecular, physiological and clinical data.

This leads to very complex models, resulting in a high computational burden with excessive usage of computational resources. Herein, agent-based modelling and simulation (ABMS) is chosen to simulate tumour growth and development. ABMS is one of the most powerful simulation modelling techniques and has the capacity to provide significant benefits for studying tumour biology process, and providing a possible solution for implementing more realistic tumour growth models. For example, in the study by Engelberg *et al.* [10], an agent-based analogue of *in vitro* tumour growth was presented using *in silico* axioms whereby simulation results were in close agreement with experimental data.

Nevertheless, simulating a complex ABMS system for realistic cases is only feasible in a reasonable time if the simulation is executed in parallel on a High Performance Computing (HPC) system. For example, Christley *et al* [11] previously described implementation of an agent-based cellular model for 3D epidermal development on GPUs, which accelerates the execution of the model up to 18x. However, in HPC ABMS simulations, a weak distribution of the agents workload may introduce uneven CPU computing and network communication overhead that delays the simulation and may propagate across all processes. With the purpose of mitigating this problem, 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 [12]. However, the centralised/hierarchical approaches report a high computational cost and scalability problems.

For this reason, we have developed a distributed Graph-based Dynamic Load Balancing (DLB) strategy that allows automatic and dynamic tuning decisions in terms of computation/communication workload. DLB tunes the global simulation workload migrating groups of agents among the processes using a Hypergraph perspective. This Hypergraph is partitioned using the Zoltan Parallel Hypergraph partitioner method (PHG) [13]. Moreover, in order to reduce agent communications, DLB uses message filtering routines to send message groups to specified recipient processes in a simple 3D grid-based structure.

We have implemented this load balancing approach on the framework Flexible Large-scale Agent Modelling Environment (FLAME), achieving a significant improvement in application performance. In order to integrate this DBL approach in FLAME, the framework was extended to allow for migration of agents, message filtering, collation of performance measurements, building the Hypergraph representation of the simulated system, and integrating Zoltan libraries.

This document is arranged in six sections. Section 2 presents the description of the cell model for simulating CRC tumours. Section 3 introduces a survey of ABMS frameworks, a more detailed description of FLAME, and details relating to the implementation of the CRC tumour cell model using this framework. Section 4 robustly discusses optimisations done to the implemented model, extensions implemented in FLAME and, in particular, the DLB method based on Hypergraph partitioning. Section 5 presents the CRC model validation and performance improvements obtained from the different optimisations implemented in FLAME. Finally, Section 6 presents our conclusion.

2. Tumour Cell Model

The tumour growth model implemented in this paper includes the basic biological properties of tumour cells (TC) such as cell growth, proliferation, and death located within a vascular network and further considers dependence on

nutrients (oxygen). Two tumour growth stages may be considered; the first is defined as avascular growth when tumours depend on simple diffusion for nutrient supply. The second stage involves vascular growth, a multistep physiological and biological process 'angiogenesis' which is initiated when tumour cells become increasingly hypoxic and, in response, secrete angiogenic factors such as vascular endothelial growth factor (VEGF). These diffuse and stimulate the existing vessels to form new sprouts, which migrate and connect to other sprouts or to the existing vascular network, forming new blood vessels. This results in an abnormal tumour vasculature which is leaky and tortuous.

The model developed here couples tumour growth with tumour angiogenesis. It is based on an extension of the vascular network model originally presented by Bartha and Rieger [14]. Each TC undergoes a fundamental cell cycle, as shown in Figure 1. The cell cycle is a set of events, including cell growth (G1) and division (M), occurring in order. In the model, each cell has an internal timer, and progresses on tick through the cycle at each iteration unless it is inhibited for cycling. Since cell division is dependent on the O_2, the oxygen concentration is also checked during the cell cycle. When the cell has divided, a daughter cell inherits all properties of the parent cell. A new blood vessel is branched from existing vessels where the local VEGF concentration exceeds a threshold. A threshold VEGF concentration prompts tumor endothelial cells (TEC) to proliferate, and the existing vessel stalk gradually forms new branches. A TEC can be divided either at a tip or at the wall of a vessel branch.

For the model simulations, the oxygen concentration $O_2(r)$ is implemented by a sum of the source strength of a vessel segment within a maximum oxygen diffusion radius, R_{O_2}. This is a modified version of the equation used in [14]:

$$O_2(r) = \sum_{all\ TECs \in R_{O_2}} \frac{leak_{factor}(R_{O_2} - d_{TEC\_TC})r_{TEC}^2}{(n_{min\_TEC}R_{O_2})r_{TC}^2} \tag{1}$$

where d_{TEC_TC} is the distance between TC and TEC, n_{min_TEC} is the optimum number of TECs, in which the TC is well oxygenated, and r_{TEC} and r_{TC} are the radii of the TC and TEC respectively. The contribution of each

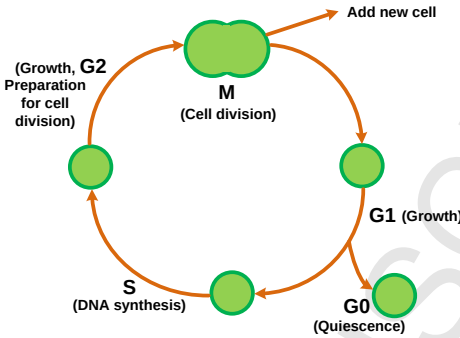


Figure 1: Schematic diagram of the simplified cell cycle. M = Mitosis, G1 = Gap 1, G0 = Gap 0 / Resting, S = Synthesis, G2 = Gap 2.

TEC is affected by $leak_{factor}$. A $leak_{factor} < 1$ indicates that the TEC is a part of a defective vessel, which is consequently poorly circulated.

Similarly, VEGF concentration $GF(r)$ is implemented using a sum of the source strength of a TC within a maximum VEGF diffusion radius, R_{VEGF}. This is also a modified version of the equation used in [14]:

$$GF(r) = \sum_{\text{all TCs} \in R_{O_2}} \frac{Prd_{VEGF(0-1)}(R_{VEGF} - d_{TEC\_TC})}{n_{min\_TC} R_{VEGF}} \tag{2}$$

where n_{min_TC} is the optimum number of TCs, in which the TEC can be divided. $GF(r)$ also depends on a production factor $Prd_{VEGF(0-1)}$ which characterises the ability of each TC to produce VEGF.

Figure 2 provides a flow chart of the model. The model specifies the initial locations of TCs and TECs in a 3D computational domain. The parameter values used for TC proliferation and vessel generation and collapse are also initialised. The model then computes oxygen concentration which affects the cell cycle. VEGF concentration is calculated after checking for cell division, death and movement. This induces tumour angiogenesis which initiates vessel growth. Lastly, vessels are checked for collapse. The entire process is repeated for a fixed duration of time.

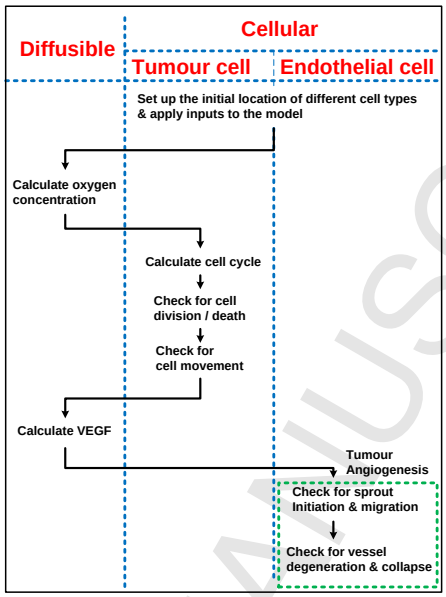


Figure 2: Flow chart of the multiscale model for tumour growth and development, illustrating the temporal sequence of the simulation.

3. Agent Based Modelling and Simulation

Agent based modelling and simulation (ABMS) is a type of computational modelling that simulates the actions and interactions of autonomous agents (both individual or collective entities such as organisations or groups) with the goal of assessing their effects on a system as a whole. ABMS belongs to a category of models known as discrete event simulations, which run with some set of starting conditions over some period of time, allowing the programmed agents to carry out their actions until some specified stopping criterion is satisfied, usually either a certain amount of time or a specified system state.

An agent is an autonomous, dynamic rule-based entity within a defined environment. The behaviour of the agents is encoded in algorithms, which may go from simple deterministic rules to sophisticated algorithms including learning and adaptive strategies. Agents determine the dynamics of the system as a whole by interacting with each other. Being able to communicate with each other, agents can influence the behaviour of other agents creating complex

interactions within a system.

Depending on the complexity of the model and the number of agents participating in the simulation, an ABMS application may consume a significant amount of computational resources. Consequently, in many cases simulations can take advantage of parallel techniques and HPC hardware. In addition, parallelising this kind of systems is usually straight forward because of the underlying autonomous behaviour of agents.

The CRC agent based model presented in this work is an ideal candidate for parallel implementation because of its complexity and the large amount of agents involved in a simulation. For this reason, we introduce in this section a survey of available parallel ABMS frameworks and, specially, a description of FLAME, which is the framework that has been used for developing our model. Finally, we also discuss the implementation in FLAME of the tumour cell model described in Section 2.

3.1. Parallel ABMS

Currently, several ABMS general frameworks for generating parallel simulations on HPC environments can be found.

Ecolab [15] is an object-oriented environment written in C++. Essentially, the user writes a class representing the entire model being simulated and instantiates an object, the variables and methods of this object are exposed to a Tool Command Language (TCL) interpreter which is used for running the experiments. When run in parallel, a TCL interpreter is launched in each used processor, facilitating the execution of any TCL command by use of the the `parallel` command, or declaring a method as `parallel`. Communication between processors can be implemented using Message Passing Interface (MPI) calls or a special wrapper class.

Repast HPC [16] was released in 2012, and written in C++ using MPI for parallel simulations. Agent types are implemented as C++ classes that are associated to contexts, which can be defined as a population of agents, and projections, which define the structure of the population contained in a context.

When run in parallel, each process is responsible for executing a set of local agents. Interactions between agents assigned to different processes are managed by copying and synchronising the interacting agents in the involved processes.

D-Mason [17] is a framework written in Java, based on a master/worker paradigm. D-Mason uses idle desktop workstations subdividing the workload among these heterogeneous machines. Communication between agents is accomplished by sharing channels between workers that share information. In addition, recent improvements of D-Mason provide a load balancing schema based on executing multiple workers on the most powerful nodes.

Pandora [18] is a framework developed in C++, OpenMP and MPI. Agents are implemented as C++ or Python classes as well as the environment the agents live in (called world). Parallelisation is achieved by distributing different parts of the world among the nodes participating in the simulation. Then, each node distributes the simulation of its assigned portion among the node cores using OpenMP. The frontiers of each world partition are automatically communicated to the neighbouring nodes in each simulation step using MPI.

Finally, *FLAME* [19] allows the production of automatic parallelisable code. FLAME is written in C, it uses MPI for communication, and agents are specified using an extension of XML plus C. Given that this is the framework used in this work, a more detailed description is given in the next subsection.

3.2. FLAME

FLAME [19] was developed at the University of Sheffield in collaboration with the Science and Technology Facilities Council (STFC). FLAME has been used to solve problems involving multiple domains such as economical, medical, biological and social sciences. This framework facilitates the writing of several agent models using a common simulation environment, and then perform simulations on different parallel architectures, including GPUs.

FLAME is not a simulator in itself, but a tool able to generate the necessary source code for the simulation. It automatically generates the simulation code in C through a template engine, which uses a set of template files (shown in

Table 1) and the user-provided specification to generate the simulation code.

Table 1: FLAME templates description.

Template	Description
low_primes.tmpl	prime numbers storage.
main.tmpl	main file of the simulator code.
memory.tmpl	agent's routines and structures.
messageboards.tmpl	structures and routines for message boards.
Makefile.tmpl	simulation code Makefile.
partitioning.tmpl	partition methods (R.Robin and Geometric).
rules.tmpl	input filtering rules
timing.tmpl	timing functions.
xml.tmpl	xml reading and writing functions.

The model specification is described by two types of files, XMML (X-Machine Markup Language) files, which is a dialect of XML, and the implementation of the agent functions contained in C files. This approach is similar to the one followed by Repast and Pandora, but in this case, instead of using an object oriented language, agents state and data are specified using XMML. Figure 3 schematically shows the inputs provided to FLAME and the output produced by this framework.

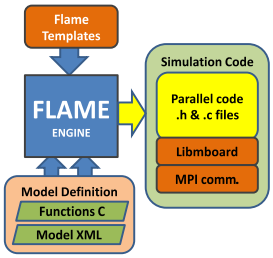


Figure 3: FLAME basic diagram.

The functionality of FLAME is based on finite state machines called X-machines, which consists of a finite set of states, transitions between states,

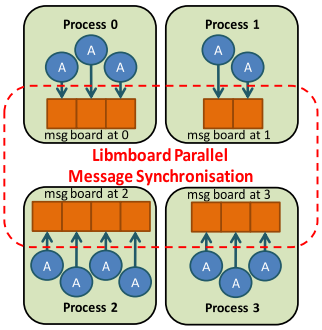


Figure 4: Parallel communication and synchronisation via libmboard.

messages between agents, and actions. To perform the simulation, FLAME holds each agent as an X-machine data structure, whose state is changed via a set of transition functions. Furthermore, transition functions may perform message exchanges between agents.

The transitions between the states of the agents are accomplished by keeping the X-machines in linked lists. The simulation environment has one linked list for each state of a specific kind of agent. During the simulation, all agents' X-machines are inserted into the list associated to their initial state. Next, the corresponding transition function is applied to each X-machine, and they are moved to the list associated to the agents' next state. This process is repeated until all agents reach the last state, which determines the end of the iteration.

When FLAME generates parallel code, this structure is replicated in all the nodes participating in the simulation and the agents are distributed among these nodes. In addition, a communication library called `libmboard`, which is build on MPI, is used for managing communication between agents assigned to different nodes. This library sends all messages to external agents through a coordinated communication mechanism between different MPI processes as shown in Figure 4. In this way, FLAME provides a general communication mechanism that allows any pair of agents to interchange messages without needing any replication of agents in different nodes.

FLAME also has some drawbacks, the main one is that its current version

does not include any mechanism to enable the movement of agents between processes. Thus, the workload in each process will depend on the evolution of the model from its initial population of agents. In addition, the centralised communication scheme based on libmboard limits the scalability of the generated simulators. These drawbacks do not depend on the particular model being defined and simulated. For this reason, this work also includes general proposals, independent of the CRC model, for improving the performance of any simulator generated with this framework.

3.3. CRC Model Implementation

There are two types of agents in the model of tumour growth presented in this paper: *Cell* agent and *Helper* agent. Cell agent represents both TCs and TECs. The TC agent consists of a group of TCs, while the TEC agent indicates one TEC. The Helper agent is used to compute the last cell Id and is run only once for each iteration. In FLAME, the agents perform actions according to predefined rules, iteration by iteration. The iteration in the model is defined as 30 minutes in real time. The predefined rules on memory variables of the agents are defined for updating the position of the agents, cell cycle, oxygen concentration, VEGF concentration and variables related to angiogenesis. The position of the agents are computed based on physical rules defined in [20]. The number of functions the agents have is also introduced to perform tumour growth coupled with angiogenesis, such as output_location, resolve_forces, update_rel_c_oxy, update_rel_c_gf, cycle, collapse_TECs and update_last_cell_id_helperagent. Part of the model stage graph with function layers is shown in Figure 5.

4. Optimisation Strategies

Several optimisations are needed in order to execute realistic simulations of the CRC model described in Section 2 in a reasonable time.

On the one hand, in terms of performance, the implementation of the model introduced in Section 3.3 has a clear bottleneck because of the Helper agent.

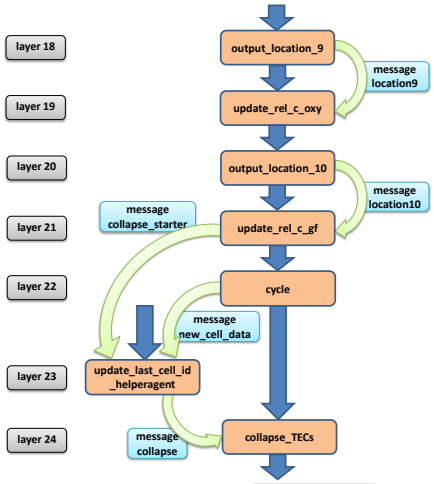


Figure 5: State graph with function layers in the ABMS to implement tumour growth and development in vasculature.

During the execution of the simulation, there is only one instance of this agent, which is responsible for creating every new tumour cell. Therefore, a specific optimisation has been devised for reducing the negative impact of this agent on the simulation performance.

On the other hand, the CRC model simulation is likely to present load imbalances because of the creation and death of tumour cells. However, this problem is not exclusive of this particular model and can occur in any ABMS where agents may appear and disappear. Consequently, the general optimisations introduced in FLAME for reducing load imbalance are presented.

4.1. Model Performance Optimisation

The model presented in Section 3.3 needs to assign unique id numbers to each new TC. Given that FLAME lacks features to supply these ids, developers implemented the Helper agent as the responsible for creating new TC with unique agent id numbers for each TC.

However, the implementation of this agent produces important performance problems in the parallel simulation. First, whenever a TC cycle determines cell-division, the parent-cell sends an agent creation request to the Helper agent.

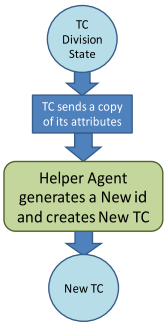


Figure 6: Original Helper agent diagram for TC replication.

This request is a message that contains a copy of the parent-cell information in order to perform a cell replication. Then, the Helper reads all the agent creation requests to perform the cell replications assigning consecutive id numbers (Figure 6 shows the TC replication process). This means that all messages, including all the necessary information for creating a TC agent, have to be processed by the same agent and that all new TC agents are created in the node where the Helper is located. This implementation quickly produces uneven workloads and represents a bottleneck because the agent creation and creation requests are centralised in one process.

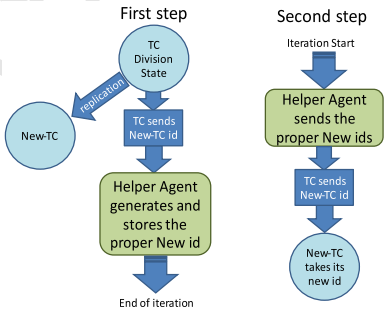


Figure 7: Helper agent operation in two step for TC replication.

In order to solve this problem, the TC replication process has been redesigned as a two-step operation which is depicted in Figure 7. The first step consists in creating the new TC by the respective parent-cell with a temporal negative

integer id, then requesting the Helper agent for a new id to replace the temporal id. The Helper agent processes the requests replacing each temporal id with the proper id. In the second step, performed at the beginning of the next iteration, the Helper sends messages with the pairs {temporal id, final id} to all the processes in the simulation. These messages are recovered by the new TC agents, which substitute their temporal ids by the final ones before starting their life cycle simulations.

This modification reduces the amount of data communicated because only the temporal id is sent to the Helper, and, most important, this reduces the load imbalance because the new TC agents are created in the same node of their parent cells.

4.2. FLAME Performance Optimisation

Section 3.2 introduced the main characteristics of FLAME and also its main drawbacks. This framework does not include a load balancing mechanism and its centralised communication scheme limits the scalability of the generated simulators.

This section describes how FLAME has been extended with mechanisms for (1) automatically and dynamically balancing the simulator load and (2) for decentralising communication between the nodes participating in the simulation. Different parts of this work have been previously published in [21], [22], and [23] but here we present the general policy that dynamically balances the computational load of the simulation considering also the amount of communication among the compute nodes.

For implementing a load balancing policy, it is necessary to be able to move agents from one node to another, and, for decentralising communications, it is necessary to provide some control mechanism over the messages sent by the agents. In this Section, we first discuss the extensions done to FLAME for agent migration and output message filtering and, next, we describe the design and implementation of the general load balancing policy.

4.2.1. Agent Migration

An agent migration mechanism is necessary to implement policies for solving load/communication imbalance problems. Consequently, FLAME has been enhanced for automatically generating efficient routines for migrating agents.

In order to deliver this new feature, we have added a new template for generating the migration routines. It is *migration.tmpl*, which generates variables, data structures and algorithms to develop the migration process.

The template engine has been modified to process this template to obtain the information about the agent types, the variables (properties) of the agents, and the size of each agent variable.

Internally, the template engine uses this information for generating migration routines alongside the simulation code as shown in Figure 8. Once the simulation code has been created, the migration routines can be used for moving agents between simulation processes. The migration process can be subdivided into two procedures: *dispatching agents* and *acquiring agents*.

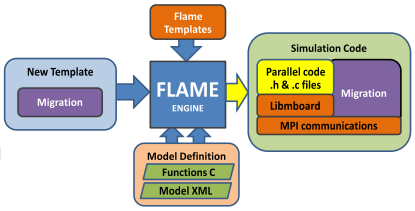


Figure 8: Base-diagram of the FLAME framework with migration routines.

The *dispatching* procedure consists of removing, packing (serialising) and sending the selected agents in the sender processes. This procedure holds a migration list for each target process and type of agent. Then, the agents to be migrated are extracted from the simulation process X-machine list associated to its current state and inserted in the corresponding migration list. Once all migrating agents have been inserted in the appropriated list, they are serialised in a set of contiguous memory buffers to be packed, using the corresponding MPI functions, in order to be sent in a single message to a specific receiving process. Finally, the message is asynchronously sent to the receiving process

for overlapping the creation of the next message with the communication of the previous one.

The *acquiring* procedure consists of receiving, unpacking (deserialise) and adding the agents in the recipient processes.

The messages with packages of agents arrive to the recipient process in buffers that must be unpacked using the corresponding MPI functions. Once the agents X-machines have been unpacked, they are inserted in the list associated with their state alongside the other agents in the recipient process.

The migration routines are specifically generated for each type of agent in the model, and it is possible to perform migrations after any transition. The following list introduces the main migration routines. The suffix NAME indicates the name of a specific type of agent.

- *Pop_NAME*: moves agents X-machines to a specific linked list and removes them from the current process.
- *Pack_NAME*: packs (serialises) all agents X-machines kept in the linked lists in contiguous memory buffers, one buffer for each recipient.
- *Send_NAME/Recv_NAME*: prototypes to define how to send and receive sets of packet agents.
- *Unpack_NAME*: unpack (deserialise) agents X-machines from the buffer to the appropriated object.
- *Push_NAME*: add an agent to the current process inserting the received agent into the adequate X-machine list.

4.2.2. Output Message Filtering

FLAME uses broadcast for implementing its board approach, leaving the recipient agents the possibility of using input filters for choosing which messages to read. However, FLAME does not publicise routines intended to send messages to specific processes, although its communication library `libmboard` is able to do this task through the MB Filter*-family functions shown in Listing 1.

These functions facilitate targeting of a set of specific processes for sending the data of local message boards (outgoing messages). For doing so, a filter function must be provided by the user. This filter must receive two arguments: a pointer to the message and the rank of an MPI process, and it must return 1 if the message has to be sent to the process with the given rank or 0 otherwise. Using these filters, each simulation process can create separate buffers for each remote MPI process sending only the relevant messages to each process, avoiding global communications at the cost of creating more buffers and messages.

Listing 1: "Board and Filter initialisation"

```
/* Create an MB Board object */
MB_Create(&board_msg, sizeof(msg));
/* Create an MB Filter object */
MB_Filter filterG;
/* Link Code-2 to the filter */
MB_Filter_Create(&filterG, &isTargetPid);
/* Assign that filter to the board */
MB_Filter_Assign(board_msg, filterG);
```

The reason why FLAME does not publicise these functions is that using them requires users to know which agents are assigned to each simulation process, which is clearly in a lower abstraction level with respect to using input filters.

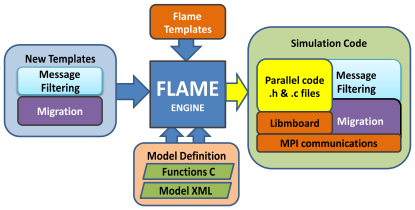


Figure 9: Base-diagram of the FLAME framework with message filtering and migration routines.

Nevertheless, we have implemented a new template in order to generate output messages filtering routines using the MB Filter* functions as shown in

Figure 9. The new template is *mapfiltering.tmpl*, which generates the message filters according to the agent's message phases.

4.2.3. Load Balancing Mechanism

Including the possibility of generating migration routines for the agents and message filters for outgoing messages give experienced users the ability to control placement of the agents and the amount of communications during the simulation process. However, we have gone beyond these extensions designing and implementing a load balancing mechanism that relies on them and the Zoltan graph partition library [13], which is, to the best of out knowledge, the only well established partition library supporting the possibility of being used at runtime.

This mechanism is based on representing the entire agent system as a weighted and directed graph where vertices represent agents computation time and edges communication volume between agents. Using this structure it is possible to achieve two important goals:

1. Agent locations can be known for automatically generating the parameters for the *output message filters* described in Section 4.2.2. This way, broadcast communication can be minimised reducing the *synchronisation and communication overhead*.
2. The graph can be provided to Zoltan functions for determining an appropriated partition for *balancing the simulator load*. With the output produced by Zoltan some agents will be migrated using the migration routines described in Section 4.2.1.

However, the amount of memory and time needed to build a graph, where each particular agent is a vertex and each message an edge, is unaffordable for any simulation involving thousands or millions of agents. Consequently, an efficient mechanism for building a suitable structure has been devised and integrated into FLAME. This mechanism consists in clustering agents using a grid-based structure and a rasterisation approach similar to the one presented in [24]. The grid-based structure characterises spatial regions during the simulation and the rasterisation approach allows to explore only the space occupied

by agents, making it is possible to model an indefinitely large domain. It is worth noticing that this approach is likely to collapse several communications on the same edge, so, a graph structure does not accurately represent the actual interaction of the agents because these edges may have more than one recipient. Consequently, we have used a different structure, also supported by Zoltan, called the *hypergraph* which contains hyperedges that can be used to connect two or more vertices.

Creating the Hypergraph. We have designed a 3D-Grid Construction Algorithm (shown in Algorithm 1) that builds each cube on the fly mapping each agent coordinates to its container cube. Each cube is identified by three integers indicating the x -, y- and z -axis origins of the cube.

In this way, cubes will be constructed along with the exploration of the existing agents across processes, every agent will be assigned to only one cube, and all cubes will have a positive agent counter. The space covered by cubes is named the *known space*, so new cubes will appear when agents are created or moved outside the known space. If a new agent appears within the known space, the agent counter belonging to its corresponding agent cube is increased.

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 [25] and it can be defined as the maximum distance an agent message can reach. Consequently, cube_size should be a function of the agents' halo, searching for a compromise between minimising the number of neighbours of each cube (cube_size = halo), which minimises communication, and getting the proper number of cubes, for having enough cubes to be able to balance the load without creating an unmanageable hypergraph. Basically, if the halo of an agent is large then cube_size should be a fraction of the halo, while, if it is small then cube_size should be a multiple of the halo.

Nevertheless, the second step after defining the cubes (hypergraph vertices) consists in defining the hyperedges. For doing so, we use algorithm 2.

Algorithm 1 Grid Construction

$c\_group_i \leftarrow cubes \in parallel\_process_i$
for all $agent \in parallel\_process_i$ **do**
  $xyz \leftarrow x, y, z - coordinates$ of $agent$
  $cid_x \leftarrow ceil(x \ / \ cube\_size)$
  $cid_y \leftarrow ceil(y \ / \ cube\_size)$
  $cid_z \leftarrow ceil(z \ / \ cube\_size)$
  $agent\_cube \leftarrow \{cid_x, cid_y, cid_z\}$
  if $agent\_cube \in c\_group_i$ **then**
    $++agent\_counter$ of $agent\_cube$
  else
    add $agent\_cube$ to c_group_i
  end if
end for

Algorithm 2 Cube Interaction

$c\_range \leftarrow ceil(agent\_range/cube\_size)$
$global\_group \leftarrow$ cubes in all processes
for all $agent \in parallel\_process_i$ **do**
  $agent\_cube \leftarrow \{cid_x, cid_y, cid_z\}$
  for all $cube \in global\_group$ **do**
    $c_x \leftarrow x$-component of $cube$
    $c_y \leftarrow y$-component of $cube$
    $c_z \leftarrow z$-component of $cube$
    if $agent\_cube \in [c_{xyz} \pm c\_range]$ **then**
      $cube \in$ interaction region
    end if
  end for
end for

This algorithm simplifies the access to the information of relations among

agents. The recipient cubes of an agent message are 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 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 id*s and the number of agents within each cube. In the same way, the agent's *message connectivity map* can be built using the Cube Interaction Algorithm. The euclidean distance is used for estimating the interaction regions of each cube. Algorithm 2 obtains the cube's *halo* (interval $[c_{xyz} - c\_range, c_{xyz} + c\_range]$) by dividing the agent interaction range by *cube_size*. This halo is then used for filtering the agents' messages in order to avoid broadcast communication.

Hypergraph-based Partitioning. Agent based applications workload can vary during the simulation due to issues related to the complexity of the model and interaction patterns. We have implemented in FLAME a *dynamic load balancing* mechanism (DLB), which decides the global reconfiguration of the workload when performance measurements 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 in each iteration of the simulation and shared among all 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.

FLAME has been extended to launch the load balancing mechanism when the imbalance factor exceeds the given threshold. The monitoring is executed locally by all processes and the processes workload measurements are broadcasted along with the simulation synchronisation at the end of each iteration. The computing time is 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 computing time for the current iteration, described by Equation 3, is considered to be the same as of the previous iteration weighted by the variation rate of the number of agents.

$$comp\_time_{iter} = \frac{comp\_time_{iter-1} \times num\_agents_{iter}}{num\_agents_{iter-1}} \tag{3}$$

Next, the process *imbalance factor* is calculated using the broadcasted current computing time. This factor, calculated with equation 4, represents the degree of imbalance according to the computation time mean. Then, *tolerance*, which establishes the range considered as balanced, and *tolerance range*, which is used to detect imbalances, are calculated using equations 5 and 6.

$$ib\_factor_i = \frac{comp\_time_i}{avg\_time} \tag{4}$$

$$tolerance = avg\_time \times threshold \tag{5}$$

$$tolerance\_range = avg\_time \pm tolerance \tag{6}$$

If $ib\_factor_i > tolerance\_range$, DBL must reallocate agents for reducing the load imbalance. The reallocation is performed according to the Zoltan Parallel Hypergraph and Graph partitioning (PHG) decision [26].

At this point, the hypergraph built using algorithms 1 and 2 needs to be transformed into a hypergraph represented as a sparse matrix. This implies that each hypergraph vertex must have a unique global identifier represented as an unsigned integer. Given that each vertex represents a spatial cube containing a set of agents, DLB is responsible for generating unique global cube ids and for assuring that each cube is assigned to only one parallel process.

To accomplish this, an initial migration of agents belonging to the same cube but assigned to different processors is done. Then, using broadcasted cube's location and computation workload, unique global cube ids are defined and vertex weights assigned. Later, each parallel process stores this global hypergraph into

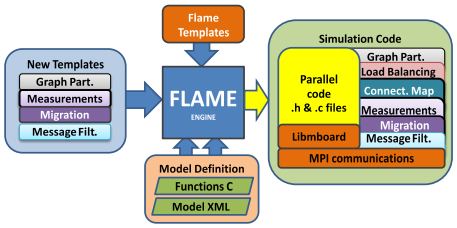


Figure 10: Base-diagram of the FLAME framework with message filtering and migration routines.

a CSR sparse matrix format. Next, PHG performs the parallel hypergraph partitioning and returns the vertex ids that should be moved to other processors in order to balance the load. Finally, in accordance with the PHG results, DLB will introduce the necessary migration calls across the simulation processes. After the agent migration occurs, the simulation will resume normally.

Part of these extensions have been implemented adding to FLAME the new templates (1) and (2), while the rest have been implemented through modifications to original FLAME templates (3), (4) and (5) :

1. *measures.tmpl*, which generates the measurement points for monitoring the application performance.
2. *zoltanmap.tmpl*, transforms the connectivity map into the graph data structures required by the hypergraph partitioner.
3. *main.tmpl*, addition of the hypergraph data structure initialisation, gathering of performance data, and calls to measurement functions.
4. *memory.tmpl*, initialisation of the output message filtering data structures.
5. *messageboards.tmpl*, addition of counters of received messages.

Finally, Figure 10 illustrates the final block diagram of FLAME including all the extensions presented in this work.

4.2.4. Comparison with other Load Balancing Proposals

Several proposals have been published for solving the load balancing problem. In this section, we discuss the differences between some, to the best of our knowledge, relevant proposals and ours.

Cosenza et al. [27] present a distributed load balancing mechanism for ABMS based on modifying the boundaries of a global space assigned to neighbouring processors. This is a low overhead mechanism triggered in each simulation step. However, its low overhead is mainly due to the fact that it is assumed that few agents will be moved between processors. In addition, it is also assumed that all agents are of the same type, which makes it easier to decide new boundaries. The load balancing scheme we are presenting can deal successfully with different kind of agents and it is not constrained to neighbouring processors, at the cost of a slightly higher overhead, which is compensated by the fact that the mechanism is only triggered when the imbalance threshold is exceeded.

Toh Da-Jun et al. [28] present an ABMS platform for multicellular biological systems, which, similarly to our proposal, incorporates a load balancing mechanism including migration of cells. However, apart for being a specialised platform (FLAME is far more generalist), this load balancing mechanism is centralised and, in consequence, not scalable to nowadays systems.

Xu et al. [29] introduce a dynamic load balancing mechanism for an ABMS platform for traffic simulation, which presents many similarities with our proposal. They also use a graph partitioning mechanism triggered when a certain imbalance threshold is exceeded, producing the redistribution of agents among computation nodes. However, there are significant differences between both proposals. First, our approach is completely distributed , while theirs uses a master-worker approach where the master is responsible for detecting the imbalance and computing the new partitioning, and the workers are responsible for taking measurements and doing the redistribution. Clearly, for larger simulations the master may become a bottleneck. Second, our proposal optimises agent migration by packing agents, while in their system, agents are migrated one at a time. Finally, although it can be easily generalised, this proposal is specific for ABMS based traffic simulations, while ours has been implemented in a general framework.

5. Experimental Assessment

This section focuses on analysing the preliminary performance results of the enhancements over the tumour model definition and the new capabilities of FLAME working together. The optimisations were tested 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 @ 3.00GHz 4MB L2 (2x2), 12 GB Fully Buffered DIMM 667 MHz, and Integrated dual Gigabit Ethernet. The simulations were commenced with an initial population of 63.505 agents, which is an adequate workload for the cluster used for running the experiments.

For the model performance enhancements experiments, 64 processes were used, while for the platform performance optimisations, 128 processes were used. In both cases, the results correspond to 20 iterations of the tumour development model, which are enough for highlighting the effect of the optimisations.

5.1. Model Behaviour

The results obtained from the model showing a tumour growth and development associated with the vasculature are presented in Figure 11. The computational domain size is 1200 um and time step is 30 minutes. A small tumour was placed at $t = 0$ with two interconnected parent vessels.

Initially, every TC is quiescent, secrets VEGF which stimulates the existing vessels, and proliferates. Increasing VEGF concentration, after a certain period of time, results in new sprout formation from the parent vessels whereby new vessels grow towards the hypoxic region of the tumour. Widespread quiescent cells in the area with no vasculature appear due to insufficient oxygen supply.

5.2. Model Optimisations Experiments

The first experimental objective was to analyse the model optimisations described in section 4.1. Tables 2 and 3 show the total number of bytes communicated to create new cells. It can be seen that the new id request in two steps significantly reduces (80%) the amount of bytes communicated to spawn a new

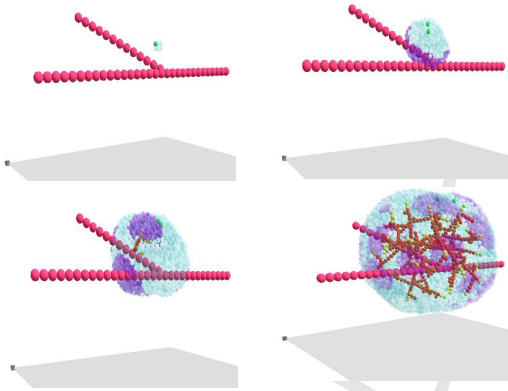


Figure 11: Tumour growth associated with angiogenesis. TCs which has a smaller oxygen level than the threshold are displayed in cyan, and TCs in proliferation with a oxygen level greater than the threshold in purple. TECs are displayed in red. TECs which have VEGF levels over the threshold and hence can divide are shown in yellow.

cell. In order to minimise the impact of having two messages instead of one, the new id request message is sent at the end of the iteration and the new id response is sent at the beginning of the next iteration, along with the simulation step synchronisation. In addition, this improvement also contributes to have a much better distribution of new cells during the simulation.

Message	new cell data	
num msgs	size/msg(bytes)	total(bytes)
9484	184	1745056

Table 2: Total messages of the first approach of the model for 20 iteration in 64 processes.

Message	new id request		new id response	
num msgs	size/msg(bytes)	total(bytes)	size/msg(bytes)	total(bytes)
9484	20	189680	16	151744

Table 3: Total messages of the enhanced version of the model for 20 iteration in 64 processes.

Figures 12 and 13 depict the new agents distribution across the parallel

processes for both approaches. Here, red bars show the number of agents per process after a round-robin distribution when FLAME starts the simulation; yellow bars show the agent increment per process after 20 iterations. Figure 12 depicts an excessive increase in agents in the process where the Helper agent is performing all cell replications, while the two-step approach evenly distributes the new cells among all processes as shown in Figure 13. This occurs because, in this approach, each parent cell is responsible for creating the new cell and cloning its information in the division phase. In this way, the model optimisation helps the platform to manage the imbalances.

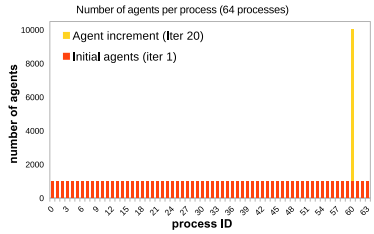


Figure 12: Tumour development model.

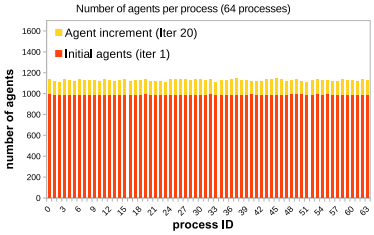


Figure 13: Enhanced Tumour model.

5.3. Platform Optimisations Experiments

The experiments, shown below, present the performance gains using the FLAME extensions described in Section 4.2. The extensions, working together, allow the global reconfiguration of the workload by means of the dynamic load balancing mechanism (DLB).

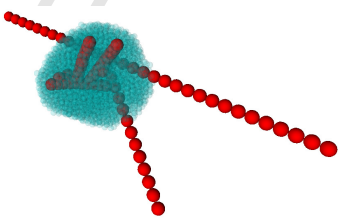


Figure 14: Tumour development model.

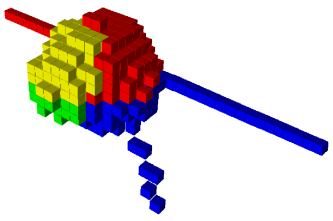


Figure 15: PHG distribution (4 processes).

Zoltan Parallel Hypergraph and Graph partitioning (PHG) can be configured as *partition*, *repartition*, or *refine*. The *partition* mode (PhgPA) does not take into account the current vertices distribution (a partition from scratch). The *repartition* mode (PhgRP) considers the current vertices distribution for repartitioning the hypergraph. Finally, the *refine* mode (PhgRE) refines the given distribution minimising the number of changes. Figure 14 shows a graphical representation of the model where the green and red spheres represent TC and TEC agents respectively. Figure 15 shows an example of the resulting 3D grid from PhgPA for 4 processes.

In order to reduce the time to find an appropriate graph partition, the PHG accuracy has been set up to 0.2 (20%) of imbalance deemed acceptable and the *cube_size* is defined as 50 microns. Also, all the initial graph partitions use the PhgPA mode with the exception of the FLAME default methods (FlameGeo and FlameRR, geometric and round-robin respectively).

Method	DLB Calls	Avg vertices Call	msgs internal/external
PhgRP	10	3746	95.95/4.05
PhgPA	10	3743	96.57/3.43
PhgRE	9	3745	96.53/3.47

Table 4: Details of the DLB options.

Table 4 shows DLB communication performance. For all versions, the number of DLB calls and average number of vertices are similar, and the communication workload is reduced. The percentage of the amount of messages held in each process is higher than the messages dispatched to external processes (internal/external respectively). Dispatching less messages also improves the performance of other processes because they have to examine a significantly smaller amount of incomming messages.

Table 5 shows the execution times of the DLB versions by comparing different PHG options with three static approaches (FlameRR, FlameGeo and initial PhgPA with message filtering). DLB versions obtain much better results than

Approach	DLB overhead(sec)	Simulation time(sec)
FlameGeo	-	26218.9
FlameRR	-	1038.1
Static PhgPA	-	881.3
DLB PhgRP	83.3	741.3
DLB PhgPA	63.4	710.7
DLB PhgRE	56.9	699.7

Table 5: Total execution time.

the static approaches. FlameGeo leads to the highest time because it divides the space into orthogonal rectangles, creating uneven or empty partitions according to the agents' spatial locations, while FlameRR randomly distributes the agents generating a similar number of agents per process. The PHG versions gain more than 30% over the FlameRR case in terms of execution time, even an initial Static PhgPA partitioning improves the FLAME times because it can use message filtering. In addition, the hypergraph DLB options show similar results and the main difference relies on the total PHG overhead time as shown in the second column of Table 5. Nevertheless, repartitioning the current partition (PhgRP) is expensive compared to partitioning from scratch (PhgPA), and refining the hypergraph (PhgRE) is the best approach for these experiments.

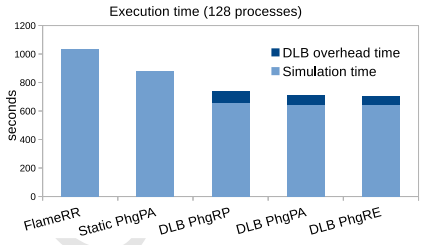


Figure 16: Execution times.

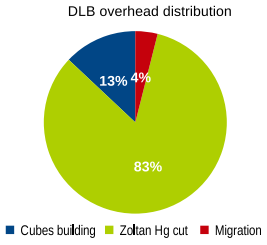


Figure 17: DLB Average overhead.

Figure 16 depicts the execution time and the overhead shown in Table 5. For all versions, DLB significantly reduces the execution time. FlameGeo has been

excluded because of its excessive execution time. Figure 17 shows that most of the DBL overhead comes from partitioning the hypergraph using Zoltan. This result suggests that reducing the number of vertices by increasing the *cube_size* could reduce the Zoltan overhead. As a result, the DLB strategy enhances the performance of the parallel simulation using agent migration, message filtering, agent connectivity map and performance measures monitoring.

6. Conclusions

A comprehensive, 3D model of tumour growth and its dependence on angiogenesis has been developed in the present study. It represents a multiscale model which includes vasculature at the tissue scale, cell-to-cell interaction at the cellular scale, and cell cycle and VEGF production at the sub/intracellular scale. TC and TEC were modelled using an ABMS approach, incorporating continuous modelling of oxygen and VEGF concentrations. ABMS is widely used to simulate tumour growth in cancer research, but imposes significant demands on computational resources and is often very time consuming.

Consequently, parallel ABMS arises as a promising way for achieving realistic simulations in a reasonable time. However, parallel ABMS simulations usually present load imbalances and excessive communication problems.

We have designed the Dynamic Load Balancing (DLB) mechanism that includes all the features (agent migration, message filtering, agent connectivity mapping and performance measurement monitoring) needed to improve the performance of parallel ABMS simulations. This mechanism has been implemented in FLAME and used on the CRC simulation model presented in this work.

DLB obtains good results, significantly reducing the simulation execution up to 48%. Our approach leads to better performance than the standard FLAME partitioning methods, and our results confirm the importance of introducing the components presented in this paper. This dynamic approach introduces an overhead in the simulation, but its benefits are significantly greater and directly impact the efficiency of the message filtering and load balancing mechanisms.

Finally, our approach will facilitate the development of ABMS for tumour growth and therapy responses that more realistically incorporate complex biology at a spatial and/or temporal scale.

Acknowledgements

This work has been partially supported by MICINN-Spain under contract TIN2011- 28689-C02-01 and TIN2014-53234-C2-1-R and GenCat-DIUiE(GRR) 2014-SGR-576. This research was also funded by the European Community's Framework Programme Seven (FP7) Programme under contract No. 278981 AngioPredict and supported by the DJEI/DES/SFI/HEA Irish Centre for High-End Computing (ICHEC).

References

[1] L. A. Torre, F. Bray, R. L. Siegel, J. Ferlay, J. Lortet-Tieulent, A. Jemal, Global cancer statistics, 2012., CA Cancer J Clin 65 (2) (2015) 87–108.

[2] H. M. Byrne, T. Alarcon, M. R. Owen, S. D. Webb, P. K. Maini, Modelling aspects of cancer dynamics: a review., Philos Trans A Math Phys Eng Sci 364 (1843) (2006) 1563–1578.

[3] T. S. Deisboeck, Z. Wang, P. Macklin, V. Cristini, Multiscale cancer modeling., Annu Rev Biomed Eng 13 (2011) 127–155.

[4] E. Passante, M. L. Wrstle, C. T. Hellwig, M. Leverkus, M. Rehm, Systems analysis of apoptosis protein expression allows the case-specific prediction of cell death responsiveness of melanoma cells., Cell Death Differ 20 (11) (2013) 1521–1531.

[5] A. U. Lindner, C. G. Concannon, G. J. Boukes, M. D. Cannon, F. Llambi, D. Ryan, K. Boland, J. Kehoe, D. A. McNamara, F. Murray, E. W. Kay, S. Hector, D. R. Green, H. J. Huber, J. H. M. Prehn, Systems analysis of bcl2 protein family interactions establishes a model to predict responses to chemotherapy., Cancer Res 73 (2) (2013) 519–528.

[6] M. Rehm, H. J. Huber, H. Dussmann, J. H. M. Prehn, Systems analysis of effector caspase activation and its control by x-linked inhibitor of apoptosis protein., EMBO J 25 (18) (2006) 4338–4349.

[7] L. Tang, A. L. van de Ven, D. Guo, V. Andasari, V. Cristini, K. C. Li, X. Zhou, Computational modeling of 3d tumor growth and angiogenesis for chemotherapy evaluation., PLoS One 9 (2014) e83962.

[8] H. Perfahl, H. M. Byrne, T. Chen, V. Estrella, T. Alarcn, A. Lapin, R. A. Gatenby, R. J. Gillies, M. C. Lloyd, P. K. Maini, M. Reuss, M. R. Owen, Multiscale modelling of vascular tumour growth in 3d: the roles of domain size and boundary conditions., PLoS One 6 (4) (2011) e14790.

[9] M. Welter, H. Rieger, Physical determinants of vascular network remodeling during tumor growth., Eur Phys J E Soft Matter 33 (2) (2010) 149–163.

[10] J. Engelberg, G. Ropella, C. A. Hunt, Essential operating principles for tumor spheroid growth, BMC Systems Biology 2 (1) (2008) 110.

[11] S. Christley, B. Lee, X. Dai, Q. Nie, Integrative multicellular biological modeling: a case study of 3d epidermal development using gpu algorithms, BMC Systems Biology 4 (1) (2010) 107.

[12] A. Plastino, C. C. Ribeiro, N. Rodriguez, Developing spmd applications with load balancing, Parallel Computing 29 (6) (2003) 743–766.

[13] E. G. Boman, U. V. Catalyurek, C. Chevalier, K. D. Devine, The Zoltan and Isorropia parallel toolkits for combinatorial scientific computing: Partitioning, ordering, and coloring, Scientific Programming 20 (2) (2012) 129–150.

[14] K. Bartha, H. Rieger, Vascular network remodeling via vessel cooption, regression and growth in tumors, Journal of theoretical biology 241 (4) (2006) 903–918.

[15] R. K. Standish, R. Leow, Ecolab: Agent based modeling for C++ programmers, CoRR cs.MA/0401026.

[16] N. Collier, M. North, Parallel agent-based simulation with repast for high performance computing, Simulation 89 (10) (2013) 1215–1235.

[17] G. Cordasco, R. De Chiara, A. Mancuso, D. Mazzeo, V. Scarano, C. Spagnuolo, Bringing together efficiency and effectiveness in distributed simulations: the experience with d-mason, Simulation 89 (10) (2013) 1236–1253.

[18] X. Rubio-Campillo, Pandora: A versatile agent-based modelling platform for social simulation, in: Proceedings of SIMUL 2014, The Sixth International Conference on Advances in System Simulation, IARIA Publishing, 2014, pp. 29–34.

[19] S. Coakley, M. Gheorghe, M. Holcombe, S. Chin, D. Worth, C. Greenough, Exploitation of high performance computing in the flame agent-based simulation framework, in: High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on, 2012, pp. 538–545.

[20] T. Sun, P. McMinn, S. Coakley, M. Holcombe, R. Smallwood, S. MacNeil, An integrated systems biology approach to understanding the rules of keratinocyte colony formation, Journal of the Royal Society Interface 4 (17) (2007) 1077–1092.

[21] C. Márquez, E. César, J. Sorribes, Agent migration in hpc systems using flame, in: Euro-Par 2013: Parallel Processing Workshops, Vol. 8374 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2014, pp. 523–532.

[22] C. Márquez, E. César, J. Sorribes, A load balancing schema for agent-based spmd applications, in: International Conf. on Parallel and Distributed Processing Techniques and Applications, PDPTA, 2013, pp. 62–69.

[23] C. Márquez, E. César, J. Sorribes, Impact of message filtering on hpc agent based simulations, in: European Simulation and Modelling Conference 2014, 2014, pp. 62–72.

[24] T. Petkovic, S. Loncaric, Supercover plane rasterization - a rasterization algorithm for generating supercover plane inside a cube, in: GRAPP 2007, Proceedings of the Second International Conference on Computer Graphics Theory and Applications, Barcelona, Spain, March 8-11, 2007, Volume GM/R, 2007, pp. 327–332.

[25] H. Parry, M. Bithell, Large scale agent-based modelling: A review and guidelines for model scaling, in: Agent-Based Models of Geographical Systems, Springer Netherlands, 2012, pp. 271–308.

[26] K. Devine, E. Boman, R. Heaphy, R. Bisseling, U. Catalyurek, Parallel hypergraph partitioning for scientific computing, in: Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International, 2006, pp. 10 pp.–.

[27] B. Cosenza, G. Cordasco, R. De Chiara, V. Scarano, Distributed load balancing for parallel agent-based simulations, in: Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on, 2011, pp. 62–69.

[28] T. Da-Jun, F. Tang, T. Lee, D. Sarda, A. Krishnan, A. Goryachev, Parallel computing platform for the agent-based modeling of multicellular biological systems, in: Parallel and Distributed Computing: Applications and Technologies, Springer, 2005, pp. 5–8.

[29] Y. Xu, W. Cai, H. Aydt, M. Lees, Efficient graph-based dynamic load-balancing for parallel large-scale agent-based traffic simulation, in: Proceedings of the 2014 Winter Simulation Conference, WSC '14, IEEE Press, Piscataway, NJ, USA, 2014, pp. 3483–3494.

*Biographies (Text)

Guiyeom Kang is currently a Postdoctoral researcher at the Centre for Systems Medicine, Royal College of Surgeons in Ireland. She was awarded an Irish Research Council's Embark Postgraduate Scholarship to pursue Ph.D. for 3 years and received her Ph.D. in biomedical engineering from the University College Dublin in 2014. Her research interests include developing computational and mathematical models to help clinical planning to provide personalised therapies for patients. In particular, her research focuses on multiscale modelling of tumour growth and anti-cancer drug delivery and neural activity including intrinsic properties and network dynamics in healthy and disease states.

Claudio Márquez got his BS degree in computer science in 2007 from University of Magallanes (Chile) and his MSc in High Performance Computing in 2011 from Universitat Autònoma de Barcelona (Spain). From 2011 onward, he is pursuing a Ph.D. in High Performance Computing at the same university. Currently, he is working at Barcelona Supercomputing Center (BSC) in the joint research project with Repsol for developing software systems for seismic imaging (BSIT). His main research interests include the design and development of optimisations techniques for parallel computing applications, dynamic partitioning techniques, application tuning, simulations and algorithms.

Ana Barat is a Postdoctoral Scientist and Data Base Manager under the FP7 Angiopredict Project at the Centre for Systems Medicine, Royal College Surgeons in Ireland since 2013. She received her Ph.D. in Computational Biology in Dublin City University in 2007. During 2007-2009, she was awarded an Irish Research Council for Science, Engineering and Technology - EMBARK Award, and from 2010-2013 she received a Marie Curie International Mobility Fellowships INSPIRE Award, focusing on the treatment, analysis and modelling of cancer-related biomolecular data. She is a member of the Marie Curie Fellows Association and Association France-Moldavie.

Annette Byrne is Senior Lecturer (Physiology), Principal Investigator Tumour Biology/Molecular Imaging at the Royal College of Surgeons, Ireland. In 1999 following completion of her Ph.D., she was awarded the prestigious UCSF John Kerner Fellowship in gynaecologic oncology. She transitioned to industry in 2001 as a pre-clinical cancer drug development Scientist. On her return to academia in 2005 (University College Dublin), she established Ireland's first multimodality molecular imaging facility. Currently, she is Coordinator of Angiopredict, a major European Commission funded research platform in the translational oncology space, whose focus is towards an improved precision medicine strategy in colorectal cancer.

Jochen H.M. Prehn is the Director for the Centre of Systems Medicine and Chair of Physiology at the Royal College of Surgeons in Ireland. He pioneered the translation of dynamic systems models of apoptosis signalling into clinical settings, delivering novel predictive tools for therapy responses in cancer patients. He was appointed as Ireland's first Science Foundation Ireland Research Professor and is currently an SFI Investigator. He is an expert on medical systems biology and cell survival and death signalling and has 190 peer-reviewed publications indicating the high productivity of his research group.

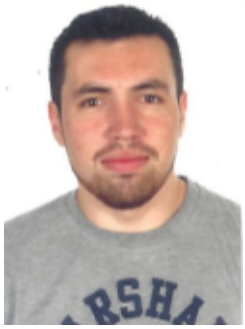
Joan Sorribes got his MSc in Physics in 1981 and the Ph.D. in Computer Engineering in 1987 from Universitat Autònoma de Barcelona (UAB). Actually is an associate professor from 1989 in the Computer Architecture Department (UAB). He has worked multiprocessing in distributed systems. At present his main research interest is High Performance Computing, specially in models to tune real parallel and distributed applications, large-scale parallel computing, dynamic tuning, scientific computing on the cloud, and big data processing

Eduardo César got his BS degree in computer science in 1992 from University Simón Bolívar (Venezuela). He got the MSc in computer science in 1994 and in 2006 the Ph.D. in computer science, both from Universitat Autònoma de Barcelona (Spain). Since 1998 his investigation is related to parallel and distributed computing. He is currently involved in Spanish national projects and the European project Autotune. His main interests are focused on high performance parallel applications, automatic performance analysis and dynamic tuning. He has been involved in the definition of performance models for automatic and dynamic performance tuning on cluster environments.

***Biographies (Photograph)**

Guiyeom Kang

Claudio Márquez



Ana Barat

Annette Byrne

Jochen Prehn

Joan Sorribes

Eduardo César