

The QosCosGrid project: Quasi-Opportunistic Supercomputing for Complex Systems Simulations. Description of a general framework from different types of applications

M. Charlot¹, G. De Fabritis¹, A.L. García de Lomana¹, A. Gómez-Garrido¹, D. Groen², L. Gulyas^{3,4,*}, A. Hoekstra², M. Johnston¹, G. Kampis¹, S. Portegies Zwart², S. Robinson⁵, M. Strathern⁶, M. Swain⁵, G. Szemes⁴, and J. Villà-Freixa^{1,*}

¹ Research Group on Biomedical Informatics (GRIB), Institut Municipal d'Investigació Mèdica and Universitat Pompeu Fabra, C/Doctor Aiguader, 88 08003 Barcelona, Catalunya, Spain;

² Universiteit van Amsterdam, Faculty of Science, Kruislaan 404, 1098 SM Amsterdam;

³ Collegium Budapest, Szentthromsg u. 2 H-1014 Budapest, Hungary;

⁴ AITIA, Czetz Janos u. 48-50., 1039 Budapest;

⁵ University of Ulster, Coleraine, Northern Ireland;

⁶ University of Cranfield, Bedfordshire, United Kingdom

lgulyas@aitia.ai, jvilla@imim.es

WWW home page: <http://www.qoscosgrid.eu>

Abstract. In this minireview we summarize the characteristics of 9 different use cases prototypical of complex systems simulations. Problems like social behavior, gene regulatory networks, molecular simulations and supply chain all share a common structure. They can be classified according to their computational needs in a way suitable for implementation in a quasiopportunistic environment as the one provided by the QosCosGrid project. Thus, a common communication patterns framework is extracted from the individual use cases to help identifying the relevant aspects of parallelization of complex systems on such a particular grid environment.

1 Introduction

Many real-world systems involve large numbers of highly interconnected heterogeneous elements. These complex systems (CS), typically exhibit non-linear behavior and emergent properties determining the high level functioning and behavior of such systems[2]. Possibly, the most striking common denominator of all complex systems is the fact that the classic experimental methodologies and research approaches plainly fail to allow prediction and control of their behavior. Accordingly, for many real-world systems, computational modelling and simulation (sometimes replacing, sometimes complementing experimentation) is

therefore the only practical method of developing an understanding of their properties at the cost of considerable computational resources such that only supercomputers can render.

Currently there is no grid technology with the capability to harness the available grid resources and provide a computationally equivalent to a supercomputer service. The EC funded STREP QosCosGrid (Quasi Opportunistic Supercomputing for Complex Systems in Grid Environments) was conceived to address this problem. The aim of the project is to develop core grid technology capable of providing quasi-opportunistic supercomputing grid services and technology. These grid middleware and services will provide complex systems researchers with access to computational and storage resources of near supercomputing capacity based on a grid infrastructure that will be somewhat better than opportunistic. This so called quasi-opportunistic supercomputing will open the way to the truly ubiquitous and cost-effective access to the grid needed by the CS research community and industrial sectors. Existing grid execution environments and programming grid interfaces providing large-scale multi-domain grids do so in a relatively simple, static, inefficient and often insecure way. The QosCosGrid project is developing a quality of service (QoS) and service oriented system designed to support application adaptation, resource reservation, co-allocation and QoS/SLA (service level agreement) performance monitoring together with some economic-based grid middleware extensions.

The test bed used to evaluate this quasi-supercomputing grid technology consist of a number of diverse CS modelling applications operating as part of complex problem-solving environments. Examples of complex systems include ant-hills, nervous systems, cell and living things, human economies and societies. Despite the variety of CS applications, the key IT requirements for computational modelling and simulation are essentially the same and include:

1. Integration of heterogeneous and large volumes of data and information, which are often geographically widely dispersed.
2. Design and execution of computationally intensive, storage- and memory-intensive computational models.
3. Considerable volumes of data generated as output from the underlying simulations. These data need to be managed, shared and analysed using varied computational methodologies, such as data mining and database management systems.
4. Sharing of the developed models, model outputs, and model output analysis results.

The provision of a reliable, extensible but inexpensive computing infrastructure that facilitates the execution of very large-scale complex systems simulations is the challenge of the QosCosgrid project. These different CS applications and how they benefit from the grid infrastructure are described in the first part of this article. The second part of the paper presents the abstraction of all the use cases into a simplified set of communication paradigms that can be used to develop specific opportunistic frameworks for supercomputing on the grid.

2 Use Case Description

Because of the universal view taken in the QosCosGid project, a large set of applications, covering the whole spectra of CS types, from a totally opportunistic to an ideally parallel implementation, has been chosen. The different use cases (UCs) have been collected into three main general paradigms for computation in CS simulations (CSS): Living Simulations (referring to systems where different computational paradigms are needed through a single computational run), Evolutionary Modelling (where biological evolution is mimicked to perform global optimizations in complex functions) and Coevolutionary Agent Based Modelling (a paradigmatic example of CSS). Each category comprises three UCs.

2.1 Living Simulations

UC1: Stellar dynamics and evolution In this UC we perform high precision dynamic simulations of self-gravitating N -body systems, including both the dynamics of all interacting stars, the evolution of the individual stars and the computation of the details of stellar collisions [18]. The stellar dynamics computations are continuous in time and are performed using state-of-the-art N -body integration routines [17]. Setting up a grid simulation that alternatively executes stellar dynamics on multiple grid sites with dedicated hardware and stellar evolution (and optionally hydrodynamic models) on massively parallel computers improves the efficiency of the calculation, allowing the simulation in excess of a million bodies over more than a relaxation time.

UC2: Protein folding and conformational changes in PPI The aim here is to compute at different level of approximations the degree of interactions between two or a set of proteins. The UC makes use of several different methodologies and applications including the molecular simulations package Adun[12]. The grid environment will enable to extend the physical-chemistry methodologies such as making use of information from multiple independent runs. An example of this can be found in the recent BOINC-based project PS3GRID[7]. In the future, this will allow to harness more computer power than it is possible today on computer clusters and extend the accuracy of the prediction of binding affinities for protein-protein interactions.

UC3: Spatial aware gene regulatory networks using cellular automata This type of CSS deals with cellular automata (CA) simulations to model gene regulatory networks (GRN). At present GRNs are modelled with ordinary differential equations (ODE). However, these models are often unsatisfactory, as they do not take into account the spatial relationships of the component parts or the effect of the noise that exists in such systems. The aim is to contrast the existing model with a CA model in which the gene products migrate throughout the cellular arena and interact with other genes and their products. While small CA models can be executed on just one node, as the size of the arena being

modelled grows, bigger models benefit from extensive parallelization implementations. The parallelisation requirements are such that there will be an optimum degree of distribution between nodes and a load sharing that will compensate for the areas of the arena that have intense activity. [3, 4]

2.2 Evolutionary computation

UC4: Topology of gene regulatory networks This UC runs Genetic Algorithms (GA)[10] to discover parameters for gene regulatory networks. The GA operates on populations of individuals, where each individual is an instance of a gene regulatory network. Instances of the populations, living on separate islands, are evolved through many generations in order to find an optimal set of parameters. The interest in a grid architecture is to add communication between islands, i.e. information exchange so that individuals migrate between island populations. The aim is also to scale up the computation by increasing the size and complexity of the genetic networks: this implies that the evaluation of a single individual will increase from 10 milliseconds by many orders of magnitude.

UC5: Parameter Estimation in Systems of Highly Coupled Differential Equations The study of the dynamical behavior of biochemical networks involve the numerical integration of systems of highly coupled differential equations that try to fit a given set of experimental data. These simulations depend on a big number of unknown parameters, which typically need to be optimized to yield the best fit. Global searches of the sets of parameters underlying the simulation are needed, and combinations of GA and local searches are commonly used in programs like ByoDyn[6]. As before, the grid improvements are directed to the ability of launching islands of populations. The particularity of this case, communication between islands would ensure the synchronization among populations, even if the islands are not running the exact same generation.

UC6: Evolutionary algorithms toolbox The GA Toolkit (GAT) will provide general tools to discover and optimise parameters for complex system simulations in a parallel environment[20]. A number of different GA implementations and CS simulations will be used. In particular, the results of this toolkit will be initially tested in the previous two UCs. The aim is so to implement different evolutionary algorithms (not just GA) with different patterns of communication, and some of those patterns will require frequent communication (on the millisecond scale). GAs are quite robust and can perform with changing numbers of individuals or populations. This is a property that the GAT should exploit within the QosCosGrid vision of dynamic resource usage. In the island implementation, for example, populations on islands may be required to go extinct or new island populations may be added to the system, depending on the available resources; similarly the size and topology of the lattice in the cellular implementation may also change in order to exploit the available resources.

2.3 Coevolutionary agent models

UC7: Business supply chain evolutionary dynamics The purpose is to simulate a real logistics network using historical data and bootstrapped data to discover rules of operation that are better than those used at present. The simulation works on a model of a small part of the whole of the logistics supply chain and only a limited set of possible rules has been explored. The aim is to look for rules that are not merely better under particular historical regimes of activity but that are robust to changes in activity.

UC8: Agent based modelling of ecological evolution The studies involve experiments on an experimental high-end system called FATINT, for realizing ecological evolutionary simulations based on phenotype-to-phenotype interactions in order to study emergent evolutionary phenomena such as speciation, niche construction, competitive exclusion, and segregation viz. isolation. The model uses a leading edge agent based simulation development platform, RePast[16], written in a Java based environment. These simulations are computationally highly demanding due to the large number of complex agents and the length of the evolutionary periods to be studied (typically, these are several thousands of overlapping generations). Tests performed also include extended sensitivity analysis using parameter sweeps. The current resources for the model do not permit developing and running full blown ecosystems, i.e. where there are both consumers and producers, and an open ended niche structure[8, 21].

UC9: Social influence and discrete choice This model studies the dynamics of social choices made by humans subject to social influences. The bounded rational decision makers are facing discrete choices like the selection of a particular brand in the market, or a particular political movement during the elections. However, the existing models typically assumed independent decision makers and focused on the modelling of the individual decision making process. One need supercomputing capabilities to be able to study more realistic networks, with more realistic decision makers and also to study social choices on dynamic networks. Here, the social interaction network does not remain static over the course of the simulation, as is the case with current works. [5]

3 General Framework of Complex systems simulations

3.1 Common and different characteristics of CS

The arsenal of existing CSS methodologies, technologies and applications is huge and varies widely in terms of the underlying formal methods, programming models and languages, and software and hardware platforms. A general view of Complex Systems Simulations (CSS) is needed, motivated by the urge being present in contemporary efforts aiming to speed up such simulations by parallelizing them and distributing the computational load over a network of processors. An

initial dissection of the main characteristics of all the selected UCs have been made. Table 1 resumes their different characteristics: the way the CS are represented and mathematically formalized, the approach taken to solve the specific questions they face (being the bottom-up, top-down and embedded approaches directly related with the different parallelization paradigms -intra-model, inter-model or both) and the challenges the UCs imply.

The global analysis of the UCs emphasizes common features and computing requirements [15]:

1. CSS are often encoded as a large set of the relatively simple sub-components so the middleware of the envisaged quasi-opportunistic supercomputer needs to map the computing objects of CSSs and their structure to the topology of the underlying grid and ensure sufficient levels of performance and reliability of both the grid/computing nodes and the communication among them.
2. Complex Systems Simulations needs to have a flexible and efficient communication environment to meet the demanding communication requirements of CSS applications.
3. CSS represents a very large number of processes which need to be executed in parallel to achieve the intended function and to improve overall performance, i.e., minimize execution time. Also, computation results may need to be collected and stored in suitable locations.
4. CCSs can be highly dynamic as the computational components of a CSS application may need to be created, destroyed and updated at a high rate in order to achieve the desired behavior and temporal resolution. To support CSS applications on a QOS system, the underlying grid middleware needs to be able to manage the constantly changing execution topology in a flexible and dynamic fashion.
5. The dynamic character of a CSS may requires the migration of several components from one execution environment to another. Therefore, the underlying middleware should be able to preserve the systems state, and enable restoration of the latest state in case of process migrations, failures or resubmissions.
6. During a large-scale CSS it is often possible for users to tell whether or not the simulation will succeed and intervene. Thus, the underlying grid middleware needs to support such interactive scenarios, and allow users to interact with the CSS in response to the simulation progress.

Beside those common features, the different UCs also show specific traits that make them special cases in general CSS. As a main specificity, they usually vary in the terms they use, or in what they define as the basic unit of the parallelization. More precisely in the description of UC1 and UC3 there is a need only for parallelizing the simulation itself (in-run level). For some cases, a set of simulations needs to be distributed among processors, referring to the inter-run level parallelization (UC7). Most of the cases seem to be available for both in-run and inter run level parallelization (UC: 2,4,8,9). Finally, in UC5 and UC6 one more abstract level appears (“inter-inter-run” parallelization), where isolated sets of simulations need to be distributed. Table 2 summarizes the 9 UCs selected and the way those UCs are covering the complete range of levels of complexity.

Table 1. Dissection of the different characteristics for each UC, classified in the three main categories

		Representation type				Approach			Challenge			
		Networks	Systems / non linear dyn.	Cellular Automata	Agent-Based	Statistical / Markov	Bottom Up	Top Down	Embedded systems	Heterogenous comp.	Missing or uncertain data	Changing State Space
Living Simulations	Stellar Dynamics Evolution		x					x	x			x
	Protein Folding and P-P interactions		x					x				x
	Spatial Model Gene Regulation			x			x	x				
Evolutionary Computation	Gene Regulation Networks	x				x		x		x	x	x
	Highly Coupled ODEs		x			x		x				x
	Genetic Evolutionary Algorithms ToolBox					x		x		x		
Agent Based Modelling	e-Business supply chain				x		x			x	x	x
	Ecological Evolution Models				x		x			x		x
	Social Influence on discrete choices	x			x		x			x	x	

Table 2. Classification of the 9 use cases

		<i>Major source of complexity</i>		
		<i>Number of entities</i>	<i>Structure entities</i>	<i>Topology</i>
+ Major Benefit of the Grid -	Inter-Model Parallelisation (Parameter space search, Model fitting, Sensitivity analysis)	Genetic/ Evolutionary Algorithms Toolbox	Gene Regulation Networks	Parameter Estimation in Highly coupled ODEs
	Intra-Model Parallelisation (Spatial segregation, Multiscale modelling, etc)	Stellar Dynamics Evolution	Protein Folding and P-P Interactions	Gene Regulation (spatial model)
	Inter-and Intra model Parallelisation (agent-based models)	Ecological Evolution Modes	Business Supply Chain Dynamics	Social Influence on Discrete Choices

3.2 Communication Templates for Parallelization of Complex System Simulations

For being able to handle all the use-cases together, we have introduced a general framework for CSS so that one CSS will be suitable for every parallelization level of a UC. Then, using the terms of the framework computational cost and communication, it will be possible to study the possibilities of parallelization of a CSS.

General description According to CSS terminology, the relevant constituents of Complex Systems are called components, and the dynamics of the system is specified via a description of the behaviour and interaction of these components. A component will be considered as having a state set and an updating function defined on it. When all components of a CSS share the same state set and updating function, we can refer to it as a homogeneous system, while in other cases we call it a heterogeneous system.

Time, Temporal behaviour Having defined the components, the next step in describing a CSS is to introduce the time-dynamics of the system. We mainly deal with discrete-time systems, although continuous-time systems are also possible, but not straightforward. For this purpose, we introduce the index $t > 0$ for states referring to simulated discrete time. Moreover, we specify a special vector of states, holding the states of the components at $t = 0$. This is called the initial state of the system. When the initial state and the updating function of the CSS are given, the temporal dynamic of the CSS, whether deterministic or stochastic, can be defined by simple iterations. When the number of components changes over time, we talk about a dynamic population otherwise it is a static population.

Communication The key to the dynamics of complex systems is interaction. The definition of communication can be described thus; when one component needs information about the state of another in order to update its own state, we say that the component in question communicates with or depends on the other at time t .

Parallelization of CSS The aim of the parallelization of any code is to distribute the whole simulation over several computer processors in order to shorten the time necessary for completing a simulation. To solve the problem of parallelization of CSS, a rule is needed for partitioning the set of the components into partitions, then mapping these partitions onto the set of processors. To do it efficiently, we need on one hand to make the partitioning unbalancing the load assigned to each processor to be in proportion with the performance of the processor. On the other hand, we need to minimize the communication among components of different partitions to restrict the domain of the components updating functions. To be able to do this a general framework of Complex Systems

Simulations (CSS) has been defined and communication patterns identified, to help classifying the CSS of our different cases.

The communication templates classification The communication-based classification presented there might serve as a natural basis of parallelization issues of CSS. That is, that the communication patterns introduced below should reflect the communicational dependencies within CSS and the inner structures of Complex Systems Simulations and thus also should naturally suggest those parallelization methods. The communication-based classification comprises six templates:

- T0** The communication graph does not have an edge. Thus, the execution of the agent's updating functions can be asynchronous (e.g., in *parameter sweep*). Usually, we need to execute the CSS repeatedly, with different initial conditions, to explore the parameter space. Using the results of previously executed runs, the aim of heuristic parameter-sweeps is to filter out the less relevant initial parameter settings.
- T1** The communication graph is static in time -the components have the same neighbourhood and therefore communication dependencies all the time. Thus, the partition a component can contact with is well-defined (and constant) throughout the whole simulation.
- T2** The communication graph is updated at a certain time scale. This time scale of updating the communication graph it and that of updating the states of the components are the same, or at least comparable.
- T3** The communication between components decreases as the spatial distance increases.
- T4** By a given metric and sensory range, one function is enough to describe communication dependencies (usually static in time-called neighbourhood) for every component. The best example of this communication template is that of cellular automata.
- T5** This is the case, when CSS interaction patterns are a priori unknown (e.g. no space or static/dynamic communication graph is given), or the communication graph (static or dynamic) is explicitly given, but it is equivalent to a complete graph (e.g. where all components communicate with everybody else).In this case, we should deal again with the relative proportions of the communication cost and the computational time of the single components. When the communication cost is negligible, the parallelization falls under the case of T0. On the other hand, when the communication cost has a decisive contribution, the general approach is to shift the partitioning problem into a higher level - the whole original CSS will be the component of a higher level CSS, and a set of this component will be a new CSS. From this point of view, the parallelization template of the new CSS will be equivalent with T0.

The parallelization methods We can find examples in the literature that show how the two orthogonal dimensions(i.e. minimizing the communication among agents of different clusters and balancing the computational time of clusters on the given set of processors) are treated in practice. Those approaches

fit into our framework much easily that deal with them separately, i.e. who optimize first according to one dimension, and after that according to the other. However application of cases where the two dimensions are treated together to grid architectures is not straightforward, but it might be possible.

However, in Static Graph Partitioning the two dimensions of optimizing are handled together, the results of Global Methods are usually balanced partitions (balancing the computational time), and there are some local improve methods available (minimizing the communication) to make the results better. Graph repartition algorithms work with a given partition and try to improve that in accordance with both of the optimization dimension. In the Linear Programming and the Multilevel repartitioning approaches, this is done separately (first improving the balance, and then improving the cut-size keeping the balance fixed).[13, 11, 14, 9]

In spatially embedded communication patterns, if we can presume, that the components are distributed smoothly in the space, providing a balanced partitioning is straightforward (since the time necessary for the computing phase is proportional to the partition size). To decrease the communication cost between processors, we should make the (mutual) borderline of these areas smaller. With a rectangular 2D map used as space the task is even easier. CAs are the most specific cases here, where the components are homogeneous, and a uniform distribution is insured. The so-called dynamic load-balancing method solves the optimization of the two dimensions in a different order: first partitioning the system according to the communication patterns and then optimizing the balance.[1, 4, 19]

4 Applications to the use cases

The different CSS levels of the UCs are identified and a communication template for every level, which exist is given.

UC1 This UC has three different layer of simulations: stellar dynamics, stellar evolution and hydrodynamic simulation (optional). These layers need to communicate with each other; this can be described as a special case of inter-run communication. At this level the CSS consist in two or three heterogeneous components that communicate infrequently compared with the computational time needed by the main stellar dynamics layer. This therefore falls under the case of T1. The main layer (stellar dynamics) deals with the inter-run parallelization level. At this level of CSS, components are stars, time is continuous and components communication (stars interaction) is defined by the gravity. In the default case the communication graph is complete, because all the stars weight acts to every other body.

UC2 This UC deals with both inter-run and in-run parallelization (CSS) level. In the in-run level, components are atoms and the communication graph depends on the studied proteins. Where the protein is large a large communication graph

is needed. In this level communication topology is unknown, thus this level falling under the case of T5. At the inter-run level, components are whole simulations and in the default case, there is no communication between them (T1). Another type of activity requires some communication between the components, but very infrequently (quantum leap forward). As described in T5, this falls under the case of T1 again.

UC3 It deals only with in-run parallelization level. Here, the components are genes, and the communication is the migration of their products. The communication topology described as a Dynamic Cellular Automata falls under the T4 (Static Locations) and seems to be equivalent with the General Cellular Automata case described.

UC4 This UC deals with in-run and inter-run level. At the in-run level, each component is an individual of a genetic regulatory network. Communication and systems dynamic are defined by the operations of the genetic algorithms (GAs) (T5). The CSS of the inter-run level consists of a set of individuals (genetic regulatory networks) living on separated islands as components. In the basic case there is no communication between them (T1). In a second step the communication will be equivalent to migration of individual simulation result among islands. This communication is infrequent, asynchronous and its topology is unknown (T5).

UC5 This UC deals with both inter-run and “inter-inter-run” parallelization levels. At the in-run level the CSS components are ODEs, and the aim is to evaluate the numerical integration of the system of ODEs for the calculation of a given number of squared distances. In the inter-run level the model uses a simple GA technique combined with a local optimization method based on the so-called Gradient Search technique. Thus, components are heterogeneous simulations and therefore different execution times are needed. The communication and system dynamics are defined by the operators of the GA algorithm. At the “inter-inter-run” level CSS a component means a population of simulations, that is, following in the model’s terminology, an island. The communication pattern consists of the migration of the parameter vectors among these islands. To select the simulations that are allowed to migrate, GA’s fitness-value of local islands is measured (T5). When no communication is needed (Basic Case), we face the simple parameter sweep described in T1.

UC6 This UC may be taken as a generalized variant of UC4. UC6 deals with the inter-run and the inter-inter-run level parallelization of a simulation. In the inter-run level, components are individual simulations and the communication-need among them is defined by the GA’s operations. Different evolutionary algorithms with different patterns of communications will be implemented (all Templates). In the inter-inter-run level an isolated group of simulations representing islands, creates a component. Communication among islands means the migration of simulation results among them. Like in UC4 and UC5, this falls under the case of T5.

UC7 Here there is no requirement for in-run parallelization, but a deeper look into its inner structure foreshadows the opportunity of using the Static or the Dynamic Net approach at this level (T1, T2). A Static Net can be useful for modelling an existing chain, ie. a machine in a factory and the supply chain from the productions of that machine. A Dynamic Net could be used to measure how robust that chain is to the changes in activity. At this level of representation, components are heterogeneous objects of business supply chain. In the level of inter-run parallelization, the components are complete individual simulations. One can envisage a progress in several steps: i) no communication among components (T1); ii) components communicate infrequently; and iii) increased need for interaction. Because no communication pattern is a priori given here, these steps fall under T5.

UC8 In UC8 there is a need for both inter-run and in run parallelization. At the in-run level, components are phenotypes, and in the basic case, the communication defined by random pairwise fully connected interactions (T5). There are plans to development a 2 or 3 dimensional space (T3). Also there is a possibility to support a layer in which a dynamic network partitioning algorithm seamlessly allocates and reallocates the changing mobile population to cluster nodes according to the available resources, communication patterns and the populations dynamic (T2). The inter-run level consists of whole simulations as components, and in the basic case there is no communication between them (parameter-sweep, heuristic parameter-sweep - T1).

UC9 In this UC, independent decision making agents inherently require both in-run and inter-run parallelization. At the in run level, components are the agents within a changeable or static communication graph. The inter run level about simple and heuristic Parameter-sweeps falls under the case of T1.

5 Conclusion

Complex systems (CS) refer to a multi-disciplinary research using a variety of methodologies and tools. In this paper, nine CSs use cases selected for the QosCosGrid project are presented, emphasizing their main characteristics, their computational requirements, and how the applications could benefit from the grid technology. The aim of the project is to provide a grid infrastructure able both to be generic and sector-independent, encompassing the large variety of domains in which CS are researched and also to be able to take into account of the specificity of each application. Here, we provide a general framework for solving this parallelization problem by identifying communication patterns in CS simulations (CSS). This framework is motivated by the urge being present in contemporary efforts aiming to speed up such simulations by parallelizing them and distributing the computational load over a network of processors. The classification, by reflecting the communicational dependencies within CSS, emphasizes some natural parallelization methods that are suggested by the inner

structure of CSS. Some known methods and collected work are examined for parallelizing the cases falling under our communication templates. The result of this classification methodology is to provide the requirements for casting each UC into quasi-opportunistic framework and to be able to identify some ways to perform its practical implementation.

Acknowledgements This work has been funded by the EC STREP project QosCosGrid (contract number 033883). This work was also supported in part by the Dutch Science Foundation (NWO) via grant #643.200.503 and by the Spanish Ministry of Science and Technology (MCYT) via grant BQU2003-04448.

References

1. S. Bandini, G. Mauri, and R. Serra. Cellular automata: From a theoretical parallel computational model to its application to complex systems. *Parallel Computing*, 27:539–553, 2001.
2. Albert-Laszlo Barabasi. *Linked*. Perseus Publishing, 2002.
3. A. Beckers and T. Worsch. A perimeter-time ca for the queen bee problem. *Parallel Computing*, 27:555–569, 2001.
4. R. Cappuccio, G. Cattaneo, G. Erbacci, and U. Jocher. A parallel implementation of a cellular automata based model for coffee percolation. *Parallel Computing*, 27:685–717, 2001.
5. B. G. W. Craenen and B. Paechter. Peer-to-peer networks for scalable grid landscapes in social agent simulations. *In B. Edmonds, N. Gilbert, S. Gustafson, D. Hales and N. Krasnogor (Eds.), Proceedings of the Socially Inspired Computing Joint Symposium (AISB)*, pages 64–71, 2005.
6. A.L. Garcia-Lomana, A. Gómez, and J. Villà-Freixa. Byodyn; <http://cdbl.imim.es/byodyn>.
7. G. Giupponi, M.J. Harvey, J. Villà-Freixa, and G. De Fabritiis. Ps3grid.net: a distributed computing environment for molecular simulations on the cell processor. *International Conference for High Performance Computing, Networking, Storage and Analysis*, submitted.
8. K. Glass, M. Livingston, and J. Conery. Distributed simulation of spatially explicit ecological models. *In Proceedings of the 11th Workshop on Parallel and Distributed Computing, Vienna, Austria (PADS97)*, pages 60–63, 1997.
9. F. Glover. Tabu search part ii. *ORSA J. Comput.*, 2:4–32, 1990.
10. David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.
11. B. Hendrickson and R. Leland. An improved spectral graph partitioning algorithm for mapping parallel computations. *SIAM J. Sci. Comput.*, 16:452–469, 1995.
12. M.A. Johnston, I. Fdez. Galván, and J. Villà-Freixa. Framework based design of a new all-purpose molecular simulation application: The Adun Simulator. *J. Comput. Chem.*, 26:1647–1659, 2005.
13. B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical J.*, pages 291–307, 1970.
14. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220, pages 671–680, 1983.

15. V. Kratsov, D. Carmeli, A. Schuster, B. Yoshpa, A. Orda, and W. Dubitzky. Quasi-opportunistic supercomputing for complex systems in grid environments. *submitted to ICPP*, 0000.
16. Michael J. North, Nicholson T. Collier, and Jerry R. Vos. Experiences creating three implementations of the repast agent modeling toolkit. *ACM Trans. Model. Comput. Simul.*, 16(1):1–25, January 2006.
17. S. F. Portegies Zwart, S. L. W. McMillan, P. Hut, and J. Makino. Star cluster ecology - IV. Dissection of an open starcluster: photometry. *Monthly Notices of the Royal Astronomical Society*, 321:199–226, February 2001.
18. S. F. Portegies Zwart, S. L. W. McMillan, and J. Makino. Star cluster ecology - VII. The evolution of young densestar clusters containing primordial binaries. *Monthly Notices of the Royal Astronomical Society*, 374:95–106, January 2007.
19. R. Serra, M. Villani, and A. Salvemini. Continuous genetic networks. *Parallel Computing*, 27:663–683, 2001.
20. Bruce A. Shapiro, Jin Chu Wu, David Bengali, and Mark J. Potts. The massively parallel genetic algorithm for RNA folding: MIMD implementation and population variation. *Bioinformatics*, 17(2):137–148, 2001.
21. D. Wang, M. W. Berry, and L. J. Gross. On parallelization of a spatially-explicit structured ecological model for integrated ecosystem simulation. *International Journal of High Performance Computing Applications*, 20:571–581, 2006.